UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM TEXCOCO

INGENIERÍA EN COMPUTACIÓN

EJERCICIOS DE

PROGRAMACIÓN ORIENTADA A OBJETOS

IRENE AGUILAR JUÁREZ

JOEL AYALA DE LA VEGA

Septiembre del 2016

C O N T E N I D O

C O N T E N I D O *2*
Compendio y Presentación: *5*
Unidad 1 Principios básicos de la POO *6*

Ejercicio 1: Modelado en UML **6**
Objetivo: 6
Introducción Teórica 6
Procedimiento 11
Evaluación: 12

Ejercicio 2: Codificación de los conceptos básicos de la POO (clase, objeto, polimorfismo, encapsulamiento) **13**
Objetivo 13
Introducción Teórica: 13
Ejemplo de codificación de una clase: 15
Procedimiento del ejercicio. 17
Evaluación: 18

Ejercicio 3: Herencia **19**
Objetivo: 19
Introducción Teórica: 19
Procedimiento 24
Evaluación 24

Ejercicio 4: Interfaces y su implementación **25**
Objetivo: 26
Introducción Teórica 26
Procedimiento 30
Evaluación: 30
Unidad 2 Sintaxis de JAVA *31*

Ejercicio 5 Tipos de datos y sintaxis básica en JAVA **31**
Objetivo 31
Introducción Teórica: 31
Ejemplo de Operadores 38
Ejemplo de Tipos de Datos primitivos 38
Procedimiento 38
Evaluación: 39

Ejercicio 6 Clases, constructores, setters y getters **40**
Objetivo: 40
Introducción Teórica 40
Ejemplo de atributos de Instancia y de clase 47
Ejemplo Clase String 47
Procedimiento 47

Ejercicio 48
Evaluación: 49

Ejercicio 7 Clases para entrada y salida de datos 50
Objetivo: 50
Introducción Teórica: 50
Ejemplo de lectura de teclado 53
Procedimiento 54
Evaluación: 54

Ejercicio 8 Clase Object y sus métodos 55
Objetivo: 55
Introducción Teórica: 55
Ejemplo de método equals 56
Procedimiento 57
Evaluación 57

Unidad 3 Estructuras de Control 58

Ejercicio 9 Flujo de control; instrucciones condicionales y ciclos 58
Objetivo: 58
Introducción Teórica: 58
Procedimiento 62
Evaluación: 62

Ejercicio 11 Flujo de control; Excepciones 63
Objetivo: 63
Introducción Teórica: 63
Procedimiento: 66
Evaluación: 67

Unidad 4 Clases para estructuras de datos 68

Ejercicio 12 Clases utilitarias 68
Objetivo: 68
Introducción teórica: 68
Procedimiento: 73

Ejercicio 13 Arreglos en JAVA 74
Objetivo: 74
Introducción Teórica: 74
Ejemplo Arreglos 79
Ejemplo de Arreglo de Objetos 80
Procedimiento 81
Evaluación: 81

Ejercicio 16 Colecciones en java; Vector 82
Objetivo: 82
Introducción teórica 82
Procedimiento 83
Evaluación: 83

Ejercicio 17 Clases Arraylist, Treeset, Map 84

Objetivo: 84
Introducción teórica 84
Procedimiento: 88
Modelo UML de las clases necesarias y codificación 88
Evaluación: 88
Unidad 5 Interfaz Gráfica en JAVA 89
Ejercicio 18 Interfaz Gráfica en java; componentes y eventos 89
Objetivo: 89
Introducción teórica 89
Eventos 91
Procedimiento: 94
Evaluación: 94
Ejercicio 19 Aplicación web; clase Applet 95
Objetivo: 95
Introducción teórica 95
Ejemplo para visualizar en un Applet un formulario 99
Ejemplo Reloj 100
Procedimiento del Ejercicio: 101
Evaluación: 101
Ejercicio 20 Aplicaciones de escritorio; clase JFrame 102
Objetivo: 102
Introducción teórica: 102
Ejemplo de JFrame 102
Procedimiento 104
Evaluación: 105
Ejercicio 21 Componentes complejos; clase JTable 106
Objetivo 106
Introducción teórica 106
Procedimiento: 108
Evaluación: 109
Cuestionarios 110
Actividades Complementarias 112
Bibliografía 117

Compendio y Presentación:

Las tareas de un Ingeniero en computación son variadas pero es indudable que las tareas de programación son inherentes a su desempeño laboral, las organizaciones se enfrentan a problemas en su vida cotidiana que pueden ser resueltos con apoyo del Ingeniero y las aplicaciones informáticas adecuadas.

Las computadoras son herramientas que automatizan el procesamiento de la información, el manejo de los lenguajes de programación permite que los Ingenieros programen aplicaciones sencillas pero eficientes pensadas y generadas a medida de las organizaciones.

Este material aborda las primeras y más complejas competencias que el alumno debe adquirir en un curso de programación, consiste en 20 apartados dedicados a temas importantes de la programación orientada a objetos. Los temas fueron seleccionados para que durante el curso, se le facilite al alumno y al docente el tratamiento de los conceptos básicos de programación orientada a objetos, el material de estos apartados han sido seleccionados de varias fuentes de información y están encaminados a facilitar la adquisición de las habilidades de programación en el alumno; el objetivo es ayudar a los estudiantes a programar soluciones computacionales, las habilidades que se alcanzarán son las siguientes:

1. **Conocer los conceptos básicos de la POO, la UML y las plataformas de desarrollo**
2. **Sintaxis básica de un lenguaje Orientado a Objetos**
3. **Conceptos, estructuras, y objetos que permiten controlar el flujo de la POO**
4. **Conocer las estructuras que permiten el manejo de datos en un lenguaje OO**
5. **Operar los componentes que conforman la interfaz gráfica del usuario**

A gran detalle los ejercicios acompañan a alumno en una revisión de algunos conceptos, se realiza un análisis de problemas, se muestran ejemplos que solucionan algún problema o que ejemplifican situaciones útiles a conocer sobre el uso de componentes o clases particulares. También se muestra la especificación de algunas clases con notación UML. Posteriormente se le dan a conocer las reglas de sintaxis del lenguaje de programación Java y se le guía paso a paso en la codificación de programas usando sentencias secuenciales, condicionales y cíclicas. Los ejercicios están acompañados de ejemplos y de las reglas de evaluación para ser aplicables en la evaluación continua del alumno.

Se recomienda usar simultáneamente los ejercicios de la siguiente forma:

- Unidad 1 (conceptos) con Unidad 2 (sintaxis de java)
- Unidad 3
- Unidad 4 (estructuras de datos) con Unidad 5 (interfaz gráfica)

Se anexan actividades complementarias útiles al profesor como alternativa en el trabajo de evaluación de los alumnos o para que sea usada como autoevaluación del alumno; además, el programa de la Unidad de Aprendizaje es muy amplio por lo que se anexa un listado de libros que son fuente importante de información para usarse como base para tareas de investigación o para el autoestudio de los jóvenes.

Unidad 1 Principios básicos de la POO

Ejercicio 1: Modelado en UML

U. Competencia 1 **Conocer los conceptos básicos de la POO, la UML y las plataformas de desarrollo**

Duración Estimada: 1 sesión de 50 minutos

Objetivo:

El alumno representará las clases y sus relaciones con las especificación de UML

Introducción Teórica

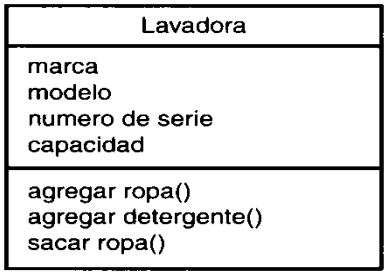
El UML (Unified Model Lenguaje) es una herramienta en la Ingeniería del Software muy útil para definir el comportamiento de un Sistema de Información, antes de él, la definición de los sistemas se realizaba con notaciones que cada analista o programador diseñaba para especificar el comportamiento de sus productos. El UML está compuesto de elementos gráficos que al usarse bajo ciertas normas se forman diagramas que especifican la estructura de un sistema tanto en su forma estática como en su forma dinámica.

UML es particularmente útil en la industria del software por su enfoque Orientado a Objetos que permite generar modelos fáciles de usar y comprender por analistas, desarrolladores e incluso clientes. La orientación a objetos es un paradigma muy aceptado en la industria pues entre otras cosas permite implementar componentes que pueden reusarse cuantas veces sea necesario; cuando un sistema de Información requiere actualizaciones o ampliaciones el paradigma permitirá mantener al sistema mediante el desarrollo de nuevos componentes o con el reúso (o adaptación) de componentes ya existentes. La Programación Orientada a objetos se basa en ocho principios básicos

- **la abstracción:** Se refiere a identificar y usar sólo las propiedades y las acciones de un objeto que son necesarios; el análisis debe identificar en qué momento o casos se puede ampliar un objeto o clase si es necesario.
- **la herencia:** la extensión de clases en subclases; es decir el reúso de acciones comunes entre objetos en una clase general y la especificación de acciones particulares en las subclases.
- **el polimorfismo**: el uso de varios métodos diferentes entre sí (en comportamiento), respondiendo todos ellos bajo un mismo nombre.

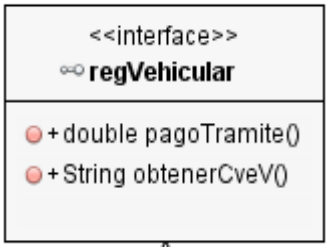
- **el encapsulamiento**: el control de acceso a métodos y atributos para mantener oculto el funcionamiento de los objetos.
- **el envío de mensajes**: se refiere a la comunicación entre objetos para compartir datos, e invocar acciones.
- **las asociaciones**: se refiere a la identificación de las relaciones y su multiplicidad entre los objetos, las relaciones pueden ser "tiene un", "es un", "usa un" y la multiplicidad puede ser "uno a uno", "uno a muchos", "cero a uno", "muchos a muchos".

Para el programador es de particular importancia comprender el diagrama de clases que representa la composición de los elementos base de los sistemas (los objetos y sus relaciones). Para la notación de UML una clase representa una categoría que tiene atributos (propiedades que caracterizan a un objetos) y métodos (operaciones que realiza un objeto) similares.

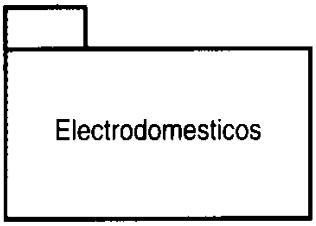


Gráficamente se representa mediante un rectángulo dividido en tres secciones verticales la primera sección se dedica a definir el nombre de la clase, la segunda sección sirve para definir los atributos de la clase (estado) y la tercera sirve para definir los métodos (comportamiento).

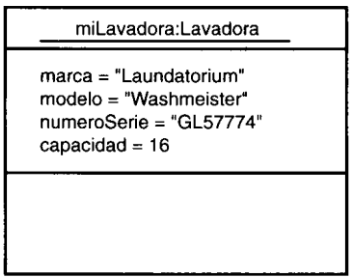
En la imagen se puede observar un ejemplo de una clase "Lavadora", en este caso una lavadora que tiene cuatro atributos (marca, modelo, numero de serie, y capacidad); también tiene tres métodos (agregar ropa, agregar detergente, sacar ropa).



Una interface es una clase que realiza operaciones y generalmente no tiene atributos, es un conjunto de acciones que será necesario repetir en varias clases pero que cada una de ellas podrá operar de diferente forma, el símbolo para representar una interface se muestra a continuación.



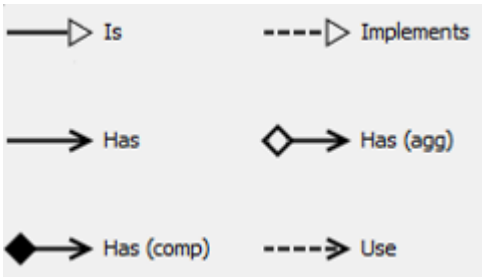
Otro símbolo importante en la notación de clases es el paquete, el cual simboliza la agrupación de las clases relacionada en un ámbito semejante, en el sistema de archivos se implementa como un directorio. En el ejemplo de la lavadora se observa que podemos agrupar la clase Lavadora junto con algunas clases más en un paquete llamado "Electrodomésticos".



Cuando se instancia un objeto de una clase se tiene un ejemplar de esa clase con datos concretos en cada atributo, en UML un objeto se representa con un rectángulo semejante al de la clase, sólo que ahora los atributos se indican con el estado del objetos.

Asociaciones

Los objetos se asocian entre sí de diferentes formas y en diferente multiplicidad (cantidad de objetos que se relacionan con un objeto), esta situación de la vida real se representa en UML con los siguientes símbolos:

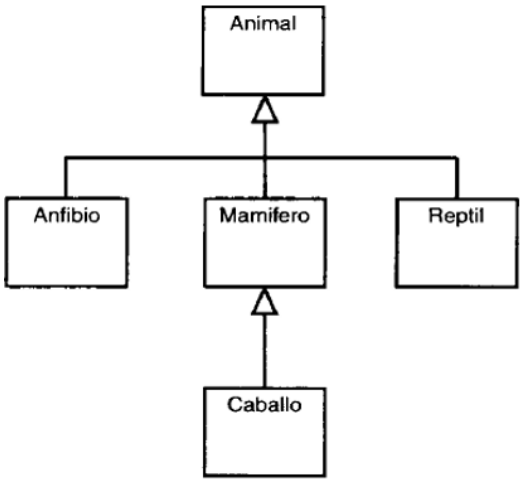


Herencia y realización: generalización y especialización

Agregación: Unión de objetos con independencia entre ellos

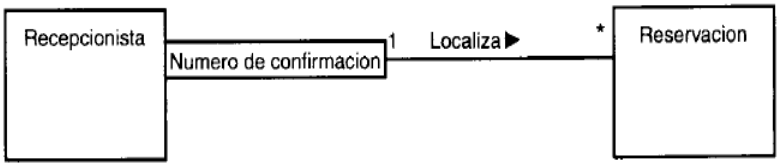
Composición: Unión de objetos con dependencia de uno de ellos de sus componentes.

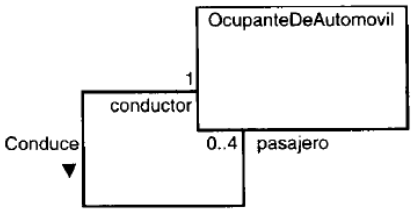
Las clases se relacionan por herencia en cuyo caso el diagrama recibe el nombre de jerarquía de clases. En la imagen se observa una jerarquía de clases formada por cuatro clases, la clase más general es la clase Animal, la superclase se observará siempre en el nivel superior. En la codificación en java se debe iniciar con la clase de mayor jerarquía, después con las del siguiente nivel y si hay un nivel más se codificará en tercer lugar. En el ejemplo de la imagen el orden de codificación será:



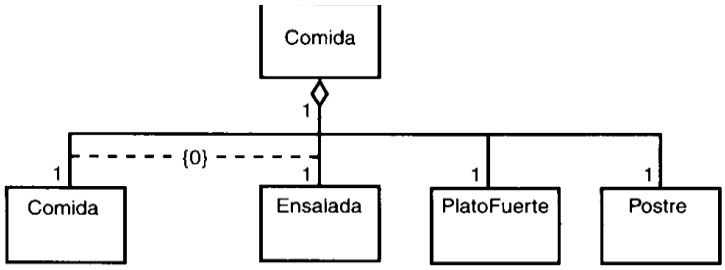
1. Clase Animal
2. Clase Anfibio, Mamífero y/o Reptil
3. Clase Caballo (después de Mamífero, antes se señalaría un error de sintaxis)

También se pueden representar relaciones calificadas, en estas relaciones se indicará con un identificador, en el ejemplo de la siguiente imagen la relación se denomina “localiza” y se requiere un dato el “número de confirmación”



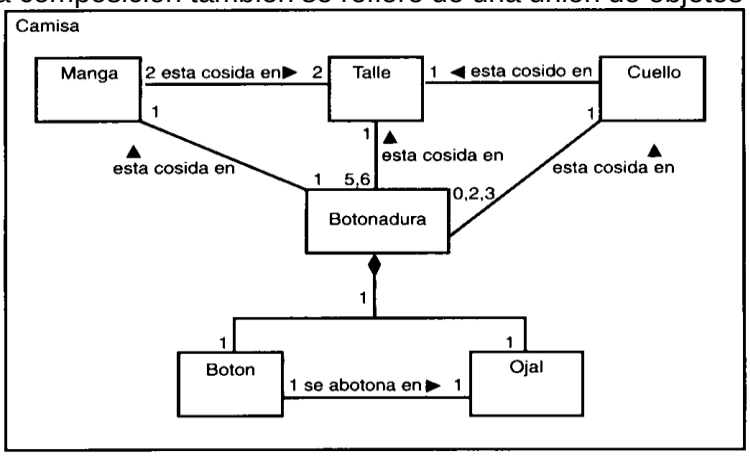


Las relaciones también pueden ser reflexivas es decir que un objeto se relaciona con un objeto del mismo tipo por ejemplo un OcupanteDeAuto (conductor) se relaciona con otro OcupanteDeAuto (pasajero) la relación recibiría en nombre de conduce.



La relación de agregación se presenta cuando los objetos se unen con cierta independencia entre ellos por ejemplo en diagrama se muestra que la comida se forma con la unión de ensalada, postre, plato fuerte pero no es obligatorio que todos los objetos estén presentes en la relación. En un restaurante se podría solicitar como comida sólo una ensalada, sin obligación de pedir el plato fuerte o el postre.

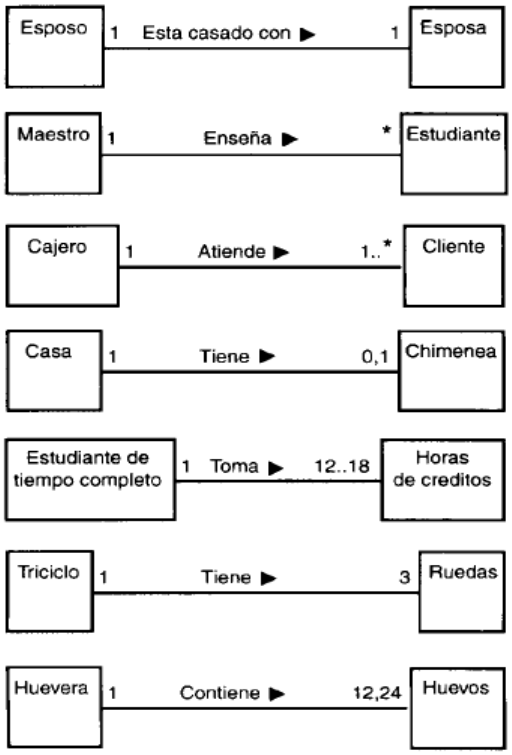
La composición también se refiere de una unión de objetos pero a diferencia de la agregación en la composición hay una dependencia del objeto hacia sus componentes en el ejemplo siguiente se observa la composición de una camisa, la camisa se compone de varias piezas (objetos), la manga, el talle, el cuello, los botones, etc., si faltara uno de ellos no sería una camisa completa. Obsérvese que la composición se representa con un rombo negro relleno y la agregación se representa con un rombo sin rellanar o blanco



En código la composición se implementa considerando a los componentes como atributos del objeto, la agregación se puede manejar usando arreglos o colecciones en las que se agreguen esos objetos en una unidad.

Las relaciones se presentan con multiplicidad, es decir un objeto se relaciona con un número posible de objetos mediante dicha relación por ejemplo un triciclo debe tener tres ruedas entonces la relación es de composición (la rueda es necesaria para que exista el triciclo) en una relación de uno a tres (un triciclo por tres ruedas).

En la siguiente imagen se muestran algunos ejemplos de la multiplicidad en las relaciones:



La multiplicidad es uno a uno

Uno a muchos

Uno a uno o más

Uno a ninguno o uno

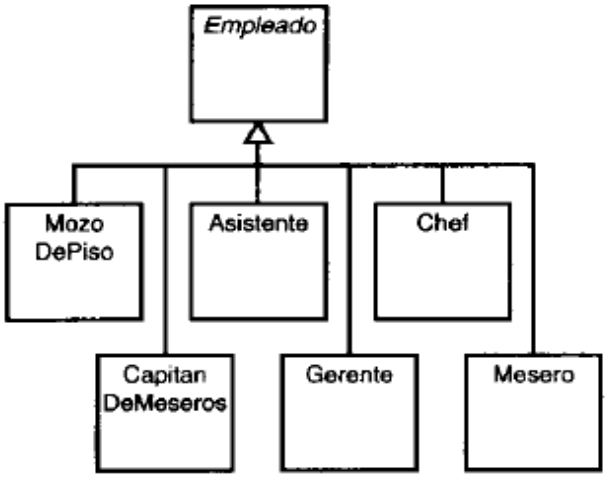
En caso de que las relaciones tengan un número específico de objetos también se puede indicar en el diagrama

En este caso la relación es uno a tres

Relación de uno a 12 o 24

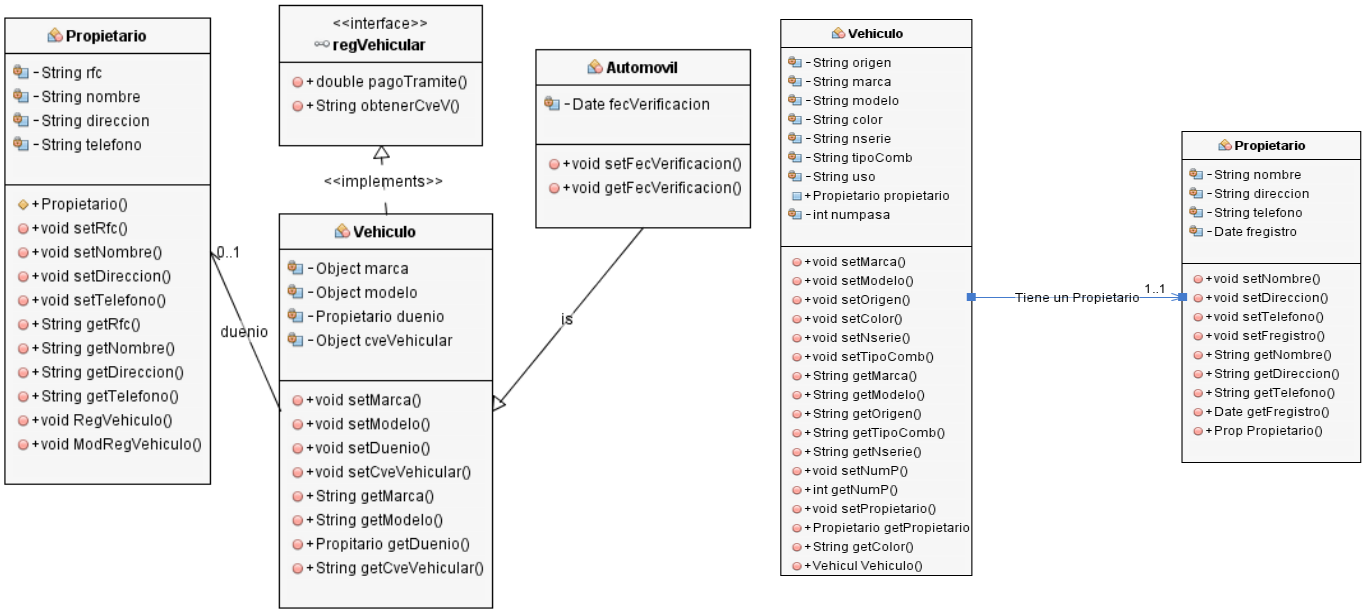
Diseño de una jerarquía de clases

Para diseñar una jerarquía de clases se debe identificar en el discurso del problema todos los sustantivos, después se reconocen las relaciones y se identifican los atributos de cada clase. Por ejemplo para solucionar las operaciones de administración en un restaurante se deberán identificar primero los actores participantes, se deben identificar las relaciones entre ellos, será importante identificar la generalización (herencia), composición y la agregación.



En siguiente ejemplo se muestra la jerarquía de clases que ayudaría a representar las clases necesarias para administrar la información de los empleados de un restaurante. Habría una jerarquía más para representar los alimentos, otra para la gestión de pagos.

También se deben identificar los atributos y métodos de cada clase en la imagen se muestra una jerarquía completa



Procedimiento

Con base en el discurso de la siguiente situación diseñe un diagrama de Clases necesario para representar la información, desarrolle los atributos y métodos de

Campeonatos de esquí

El campeonato consta de una serie de pruebas. En cada una se inscribe un conjunto de participantes. Hay pruebas individuales y por equipos.

Un esquiador puede participar en varias pruebas a título individual o sino formando parte de un equipo (pero NO unas veces individualmente y otras en equipo).

Por cada esquiador se desea guardar su DNI, nombre, fecha de nacimiento y edad. A cada participante en una prueba (equipo, para pruebas por equipos o esquiador si es individual) se le asigna un código formado por el nombre de la prueba y un dorsal.

Por cada equipo se tiene su código, entrenador, sus esquiadores y cuántos son. No todos los esquiadores de un equipo deben participar en cada prueba donde se ha inscrito.

Hay varias federaciones de esquí. De cada una se conoce su nombre y nº de federados. Cada esquiador pertenece a una única federación. No se admite la participación de esquiadores no federados. Cada federación puede administrar estaciones de esquí. Toda estación se administra al menos por una federación, aunque puede serlo por varias.

Cada estación de esquí dispone de un código identificativo. Tiene un nombre, personas de contacto, dirección, teléfono, nº total de kilómetros esquiables y nº de pistas. Cada pista se identifica por el código de la estación a la que pertenece y un número correlativo. Se guarda su longitud en kilómetros y su nivel de dificultad (según un código de colores).

Algunas pruebas de largo recorrido utilizan varias pistas de una misma estación como si fuesen una sola pista compuesta de varias subpistas. Cada prueba se realizará en las pistas de una única estación. Puede durar varios días. Se almacenan las fechas en las que tiene lugar.

Los participantes podrán competir en diferentes pruebas y en diferentes pistas. Se registrará la fecha o fechas en las que cada participante compite en cada prueba así como el tiempo empleado y la posición obtenida. Cada prueba se identifica por un nombre, será de un tipo (fondo, slalom, salto, …) tendrá unas fechas previstas de realización y se registrará el vencedor y el tiempo empleado por éste

Evaluación:

Criterio	
1. Identificó todas las clases de la jerarquía	SI NO
2. Identificó las relaciones correctas entre las clases	SI NO
3. Uso los símbolos correctos	SI NO
4. Identificó los atributos y métodos de las clases solicitadas	SI NO
Puntaje Total_____	0 1 2 3 4

Ejercicio 2: Codificación de los conceptos básicos de la POO (clase, objeto, polimorfismo, encapsulamiento)

U. Competencia 1: **Conocer los conceptos básicos de la POO, la UML y las plataformas de desarrollo**

Duración Estimada: 2 sesiones de 50 minutos

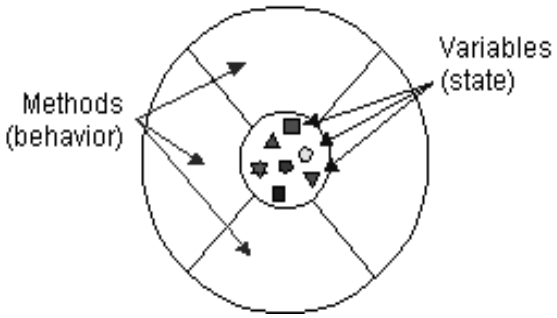
Objetivo

El alumno aplicará los conceptos básicos de la POO (clase, objeto, polimorfismo, encapsulamiento).

Introducción Teórica:

Las Clases

Una clase está constituida por unos datos y unos métodos que operan sobre esos datos.



Cada objeto, es decir cada ejemplar concreto de la clase, tiene su propia copia de las variables miembro. Las variables miembro de una clase (también llamadas campos) pueden ser de tipos primitivos (boolean, int, long, double,…) o referencias a objetos de otra clase (composición).

Un aspecto muy importante del correcto funcionamiento de los programas es que no haya datos sin inicializar. Por eso las variables miembro de tipos primitivos se inicializan siempre de modo automático, incluso antes de llamar al constructor (false para boolean, el carácter nulo para char y cero para los tipos numéricos). De todas formas, lo más adecuado es inicializarlas también en el constructor.

Una clase puede tener variables propias de la clase y de cada objeto. A estas variables se les llama variables de clase o variables static. Las variables static se suelen utilizar para definir constantes comunes para todos los objetos de la clase.

Si no se les da valor en la declaración, las variables miembro static se inicializan con los valores por defecto para los tipos primitivos (false para boolean, el carácter nulo para char y cero para los tipos numéricos), y con null si es una referencia.

Las variables miembro static se crean en el momento en que pueden ser necesarias: cuando se va a crear el primer objeto de la clase, en cuanto se llama a un método static o en cuanto se utiliza una variable static de dicha clase. Lo importante es que las variables miembro static se inicializan siempre antes que cualquier objeto de la clase

Las clases son el centro de la Programación Orientada a Objetos (OOP- Object Oriented Programming). Algunos de los conceptos más importantes de la POO son los siguientes:

1. **Encapsulación**. Las clases pueden ser declaradas como públicas (public) y como package (accesibles sólo para otras clases del package). La variable miembro y los métodos pueden ser public, private, protected y package. De esta forma se puede controlar el acceso y evitar un uso inadecuado.

2. **Herencia.** Una clase puede derivar de otra (extends), y en ese caso hereda todas sus variables y métodos. Una clase derivada puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados.

3. **Polimorfismo**. Los objetos de distintas clases pertenecientes a una misma jerarquía o que implementan una misma interface pueden tratarse de una forma general e individualizada, al mismo tiempo. Esto, facilita la programación y el mantenimiento del código.

A continuación se enumeran algunas características importantes de las clases en java:

1. Todas las variables y funciones de Java deben pertenecer a una clase. No hay variables y funciones globales.
2. Si una clase deriva de otra (extends), hereda todas sus variables y métodos.
3. Java tiene una jerarquía de clases estándar de la que pueden derivar las clases que crean los usuarios.
4. Una clase sólo puede heredar de una única clase (en Java no hay herencia múltiple). Si al definir una clase no se especifica de qué clase deriva, por defecto la clase deriva de Object. La clase Object es la base de toda la jerarquía de clases de Java.
5. En un fichero se pueden definir varias clases, pero en un fichero no puede haber más que una clase public. Este fichero se debe llamar como la clase public que contiene con extensión *.java. Con algunas excepciones, lo habitual es escribir una sola clase por fichero.
6. Si una clase contenida en un fichero no es public, no es necesario que el fichero se llame como la clase.
7. Los métodos de una clase pueden referirse de modo global al objeto de esa clase al que se aplican por medio de la referencia this.
8. Las clases se pueden agrupar en packages, introduciendo una línea al comienzo del fichero (package packageName;). Esta agrupación en packages está relacionada con la jerarquía de directorios y ficheros en la que se guardan las clases.

Una clase es una agrupación de datos (variables o campos) y de funciones (métodos) que operan sobre esos datos. La definición de una clase se realiza en la siguiente forma:

```
[public] class Classname {

// Definición de variables y métodos

...

}
```

Donde la palabra public es opcional: si no se pone, la clase tiene la visibilidad por defecto, esto es, sólo es visible para las demás clases del package. Todos los métodos y variables deben ser definidos dentro del bloque {...} de la clase.

Un objeto (en inglés, instance) es un ejemplar concreto de una clase. Las clases son como tipos de variables, mientras que los objetos son como variables concretas de un tipo determinado.

```
Classname unObjeto;

Classname otroObjeto;
```

Ejemplo de codificación de una clase:

Planteamiento: La Dirección de Tránsito necesita un sistema para cobro de placas y tenencia; en este caso, el objeto a modelar es el vehículo. Hablaremos de la clase **Vehículo** para considerar todos los vehículos posibles y no uno de ellos en particular. Las características esenciales a considerar de la clase **Vehículo** son: **origen** (nacional o extranjero), **marca** (Por ejemplo nissan), **modelo** (Por ejemplo Tsuru, Sentra), **color** (negro, plata), **nserie, tipVehículo** (automóvil, camión de carga, camión de pasajeros, camioneta, etc.), **tipCombustible** (gasolina, diesel, etc.), **uso** (particular, público), **númpasajeros** (2, 5, 11), **propietario.** Las operaciones que podemos realizar en la clase serán**: Constructor, ModificarMarca, ModificarModelo.**

Como se observa se requiere definir una clase Propietario en la que se procese la información del dueño del vehículo: nombre, dirección, teléfono y fecha de registro.

Clase Vehiculo

```
package ejercicios; //paquete dónde se almacena la clase
import java.util.Date; //Agrega el código de una clase fuera del paquete origen

public class Vehiculo {  //Define la clase vehículo
   // atributos
   String origen, marca, modelo, color, nserie, tipComb, uso;
   Propietario prop;
   int numpas;

//Constructor
   public Vehiculo(String origen, String marca, Propietario prop, int numpas) {
      this.origen = origen;       this.marca = marca;       this.prop = prop;
      this.numpas = numpas;
   }
//métodos Getters y Setters
   public String getOrigen() {       return origen;    }
   public void setOrigen(String origen) {       this.origen = origen;    }
```

```
    public String getMarca() {       return marca;    }
    public void setMarca(String marca) {       this.marca = marca;    }
    public String getModelo() {       return modelo;    }
    public void setModelo(String modelo) {       this.modelo = modelo;    }
    public String getColor() {       return color;    }
    public void setColor(String color) {       this.color = color;    }
    public String getNserie() {       return nserie;    }
    public void setNserie(String nserie) {       this.nserie = nserie;    }
    public String getTipComb() {       return tipComb;    }
    public void setTipComb(String tipComb) {       this.tipComb = tipComb;    }
    public String getUso() {       return uso;    }
    public void setUso(String uso) {       this.uso = uso;    }
    public Propietario getProp() {       return prop;    }
    public void setProp(Propietario prop) {       this.prop = prop;    }
    public int getNumpas() {       return numpas;    }
    public void setNumpas(int numpas) {       this.numpas = numpas;    }
}
```

Clase Propietario

```
package ejercicios;  //indica el paquete dónde se almacena la clase
import java.util.Date; //Agrega el código de una clase fuera del paquete origen

public class Propietario { //define una clase
   //atributos que definen el estado de la clase
    String nom, dir, telefono;
    Date dreg;

//constructor (Inicializa los valores de los atributos)
   public Propietario(String nom, String dir, String telefono, Date dreg) {
      this.nom = nom;       this.dir = dir;
      this.telefono = telefono;       this.dreg = dreg;
   }
   //getters  (sirven para mostrar los valores de los atributos)
   public String getNom() {       return nom;    }
   public String getDir() {       return dir;    }
   public String getTelefono() {       return telefono;    }
   public Date getReg() {       return dreg;    }
   //Setters (sirven para modificar los valores de los atributos)
   public void setNom(String nom) {       this.nom = nom;    }
   public void setDir(String dir) {       this.dir = dir;    }
   public void setTelefono(String telefono) {       this.telefono = telefono;    }
   public void setReg(Date reg) {       this.dreg = reg;    }
    }
```

Clase TestVehiculo

```
package ejercicios;
import java.util.Date;
public class TestVehiculo {
   public static void main(String[] args) {
      //instanciación de un propietario
      Propietario duenio = new Propietario("Juan López Ruiz","Lomas Altas #27, Texcoco", "5958424556", new Date());
      //instanciación de un vehiculo
      Vehiculo miauto = new Vehiculo("nacional", "Mercedes",duenio,5 );
      //asignación de nuevos datos
      miauto.setNserie("HY87865");
      miauto.setTipComb("gasolina");
      //muestra de datos almacenados
      System.out.println("Dueño del auto"+miauto.prop.nom);
      System.out.println("Marca: "+miauto.marca);
      System.out.println("Número de serie:"+miauto.nserie);
      System.out.println("Marca:"+miauto.marca);
      System.out.println("Num de pasajeros:"+miauto.numpas);
   }
}
```

Procedimiento del ejercicio.

Observe la imagen e identifique al menos 5 objetos que sean representados mediante una clase en java por ejemplo lámpara, ventana, alfombra, cuadros, etc.

Identifique los atributos de las clases y codifique 5 ejemplos de Clase una clase Test que use varias instancias de ellos. Entregue al docente su código en libreta y demuestre la ejecución en el laboratorio.

Número de Ejemplo Clase	Atributos	Test
1. Ejemplo 1:		
2. Ejemplo 2:		
3. Ejemplo 3:		
4. Ejemplo 4:		
5. Ejemplo 5:		

Evaluación:

Los alumnos entregaran la identificación de las clases y el código ejecutándose en el laboratorio

Criterio	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5
1. Cumplió con la identificación de atributos	SI NO	SI NO	SI NO	SI NO	SI NO
2. Cumplió con la codificación correcta y completa de la clase	SI NO	SI NO	SI NO	SI NO	SI NO
Puntaje (marque los puntos obtenidos y sume) Total_____	0 1 2	0 1 2	0 1 2	0 1 2	0 1 2

Ejercicio 3: Herencia

U. Competencia 1 **Conocer los conceptos básicos de la POO, la UML y las plataformas de desarrollo**

Duración Estimada: 2 sesión de 50 minutos

Objetivo:

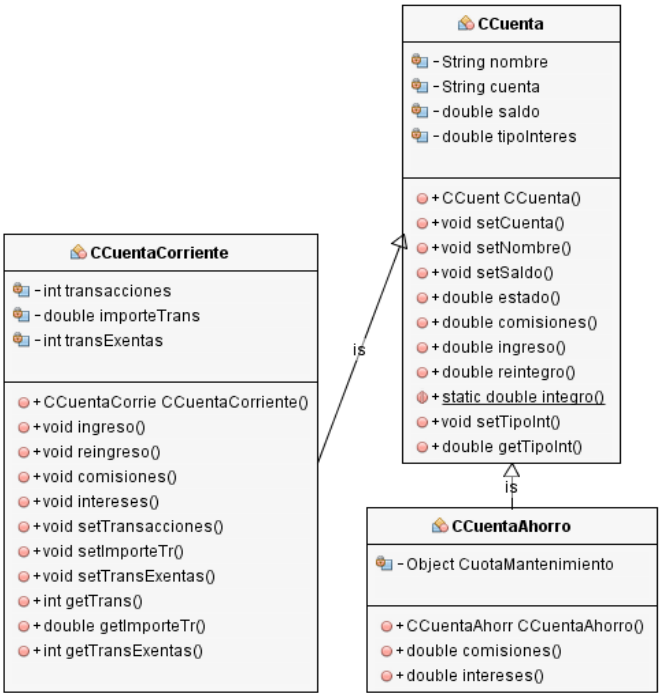
El alumno identificará los conceptos y beneficios sobre de la Herencia

Introducción Teórica:

Ésta provee el mecanismo más simple para especificar una forma alternativa de acceso a una clase existente, o para definir una nueva clase que añada nuevas características a una clase existente. Esta nueva clase se denomina subclase o clase derivada de la clase existente, superclase o clase base. Las clases que están en la parte inferior en la jerarquía heredan de las clases que están en la parte superior de la jerarquía.

En una jerarquía de clases, una clase es tanto más especializada cuanto más alejada esté de la raíz. Y es tanto más genérica cuando más cerca esté de la raíz. Por ejemplo, la clase Object es la raíz de la jerarquía de las clases de Java y sólo define los atributos y comportamientos comunes a todas las clases.

Generalmente, la razón de la existencia de una clase genérica es proporcionar los atributos y comportamientos que serán compartidos por todas las subclases. En la jerarquía de la página siguiente se observan los atributos y métodos comunes a cualquier tipo de cuenta:



Ahora bien, una cuenta de ahorro tiene las características aportadas por un objeto del tipo CCuenta, y además algunas otras; por ejemplo, un atributo que especifique el importe que hay que pagar mensualmente por el mantenimiento de la misma.

Esto significa que se tiene que diseñar una nueva clase, CCuentaAhorro, que tenga las mismas capacidades de CCuenta, pero a las que se tienen que añadir otras que den solución a las nuevas necesidades.

CCuenta	
Atributos	**Significado**
nombre	Dato de tipo String que almacena el nombre del propietario.
cuenta	Dato de tipo String que almacena el número de cuenta
saldo	Dato de tipo double que almacena el saldo de la cuenta.
tipoInteres	Dato de la clase de tipo double que almacena el tipo de
Método	**Significado**
CCuenta	El constructor de la clase. Inicia los datos nombre, cuenta, saldo y tipoInteres.
setNombre	Permite asignar el dato nombre.
getNombre	Retorna el dato nombre.
setCuenta	Permite asignar el dato cuenta.
getCuenta	Retorna el dato cuenta
setTipoInt	Método que permite asignar el dato tipoInteres
getTipoInt	Método que retorna el dato tipoDeInteres
estado	Retorna el saldo de la cuenta
comisiones	Es un método abstracto sin parámetros que será redefinido en las subclases. Se ejecutará los días uno de cada mes para cobrar el importe del mantenimiento de una cuenta.
ingreso	Es un método que tiene un parámetro "cantidad" tipo doble que añade la cantidad especificada al saldo actual de la cuenta.
reintegro	Es un método que tiene un parámetro "cantidad" de tipo doble que resta la cantidad especificada del saldo actual de la cuenta
interés	Método abstracto. Calcula los intereses producidos

A través de la herencia, la POO permite definir la clase CCuentaAhorro como una extensión de CCuenta

Los siguientes puntos resumen las reglas a tener en cuenta cuando se define una subclase:

1. Una subclase hereda todos los miembros de su superclase, excepto los constructores. Una consecuencia inmediata de esto es que la estructura interna de datos de un objeto de una subclase, estará formada por los atributos que ella define y por los heredados de su superclase. Una subclase no tiene acceso directo a los miembros privados de su superclase. Una subclase si puede acceder directamente a los miembros públicos y protegidos de su superclase, también puede acceder a los miembros predeterminados

2. Una subclase puede añadir sus propios atributos y métodos. Si el nombre de alguno de estos miembros coincide con el de un miembro heredado, este último queda oculto para la subclase, que se traduce en que la subclase ya no puede acceder directamente a ese miembro.

3. Los miembros heredados por una subclase pueden, a su vez, ser heredados por más subclases de ella. A esto se le llama programación de herencia.

Los método de una subclase no tiene acceso a los miembros privados de su superclase, pero sí lo tiene a sus miembros protegidos y públicos, y si la subclase pertenece al mismo paquete que la superclase, también tiene acceso a sus miembros predeterminados. Por ejemplo, el método comisiones de la clase CuentaAhorro no puede acceder al atributo "saldo" de la clase CCuenta porque es privado, pero sí puede acceder a su método público "reintegro". Esta restricción permite imponer la encapsulación. Si una subclase tuviera acceso a los miembros privados de su

superclase, entonces cualquiera podría acceder a los miembros privados de una clase, simplemente derivando una clase de ella. Consecuentemente, si una subclase quiere acceder a los miembros privados de su subclase, debe hacerlo a través de la interfaz pública, protegida, o predeterminada en su caso, de dicha superclase.

Constructores de las subclases.

Cuando se tiene una relación subclase-superclase, al momento de construir un objeto de una subclase, se invoca a su constructor, que a su vez invoca al constructor sin parámetros de la superclase, que a su vez invoca al constructor de su superclase, y así sucesivamente.

Ahora bien, si se han definido constructores con parámetros tanto en subclases como en superclases, tal vez se desee construir un objeto de la subclase iniciándolo con unos valores predeterminados. Esto permite que el constructor de subclase invoque explícitamente al constructor de la superclase. Esto se hace utilizando la palabra reservada “super”.

```
public nombre_subclase (lista de parámetros){
        Super(lista de parámetros);
                //cuerpo del  constructor de la subclase.
}
```

Cuando desde un constructor de una subclase se invoca al constructor de su superclase, esta línea tiene que ser la primera. La sintaxis y los requerimientos son análogos a los utilizados con “this” cuando se llama a otro constructor de la misma clase.

Si la superclase no tiene un constructor en forma explícita o tiene uno que no requiere parámetros, no se necesita invocarlo explícitamente, ya que Java lo invocará automáticamente mediante “super()” sin argumentos.

Por ejemplo, en la clase CCuentaAhorro se tiene el constructor:

super(nom, cue, sal, tipo);

Esta línea llama al constructor de CCuenta, superclase de CCuentaAhorro. Por lo que CCuenta debe tener un constructor con cuatro parámetros del tipo de los argumentos especificados

Cuando se crea un objeto de una subclase, por ejemplo “cliente01” o “cliente02”, primero se construye la porción del objeto correspondiente a su superclase y a continuación la porción del objeto correspondiente a la subclase.

CCuentaCorriente	
Atributos	**Significado**
transacciones	Dato de tipo "int" que almacena el número de transacciones efectuadas sobre esa cuenta
importeTrans	Dato de tipo "double" que almacena el importe que la entidad bancaria cobrará por cada transacción
transExentas	Dato de tipo "int" que almacena el número detransacciones gratuitas
Método	**Significado**
CCuentaCorriente	Constructor de la clase
decrementarTransacciones	Decrementa en uno en número transacciones
setTrans	Establece el número de transacciones
getTrans	Devuelve el número de transacciones
setImporteTrans	Establecer el importe por transacción
getImporteTrans	Devuelve el importe por transacción
setTransExentas	Establecer el número de transacciones exentas
getTransExentas	Devuelve el número de transacciones exentas
ingreso	Añade la cantidad específica al saldo actual de la cuenta e incrementa el número de transacciones
reingreso	Resta la cantidad específica del saldo actual de la cuenta e incrementa el número de transacciones
comisiones	Se ejecuta el día uno de cada mes para cobrar El importe de las transacciones efectuadas y no estén exentas y pone el número de transacciones a cero
intereses	Se ejecuta los días uno de cada mes para calcular el importe correspondiente a los a los intereses por mes y se añade al saldo

// Clase CCuentaCorriente derivada de CCuenta

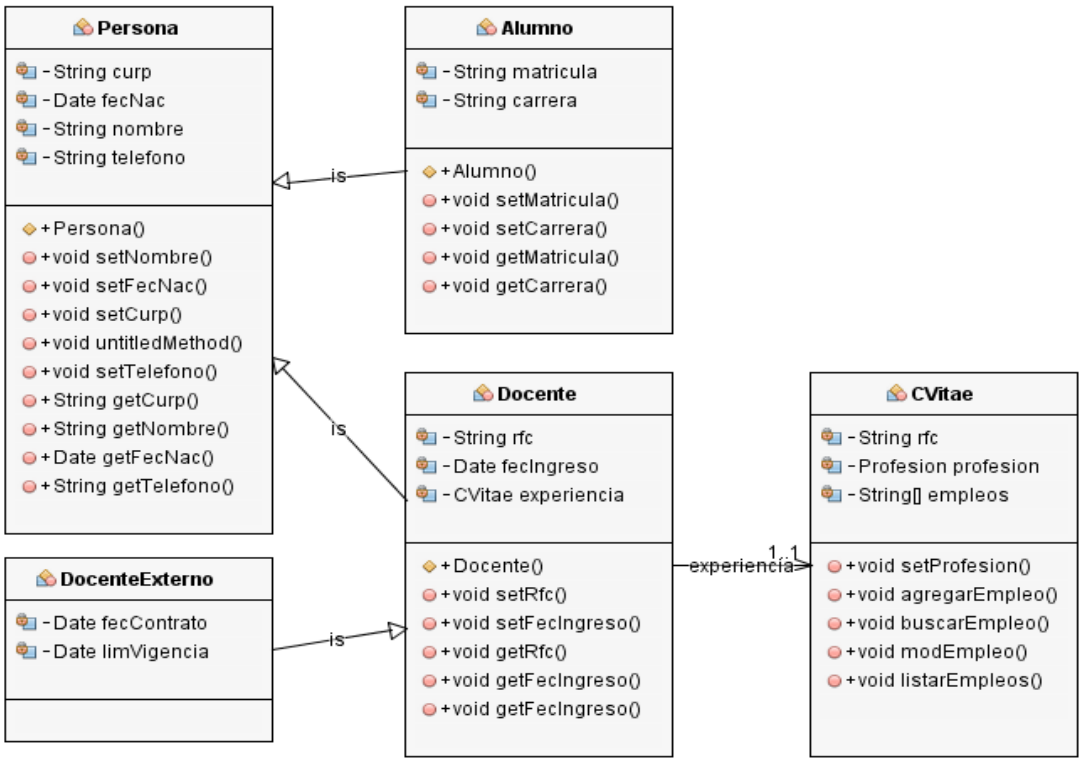
```
import java.util.*;
public class CCuentaCorriente extends
CCuenta{
  // Atributos
private int transacciones;
private double importeTrans;
private int transExentas;
// Métodos
public CCuentaCorriente() {} // constructor sin parámetros
```

```
public CCuentaCorriente(String nom, String cue, double sal,
                double tipo, double imptrans, int transex){

  super(nom, cue, sal, tipo);    // invoca al constructor CCuenta obsérvese el uso de super
   transacciones = 0;              // inicia transacciones
   setImporteTrans(imptrans); // inicia importePorTrans
   setTransExentas(transex);     // inicia transExentas
 }

 public void decrementarTransacciones(){    transacciones--;  }

 public void asignarImportePorTrans(double imptrans){
    if (imptrans < 0)    {      System.out.println("Error: cantidad negativa");      return;    }
   importePorTrans = imptrans;
 }

 public void asignarTransExentas(int transex){
   if (transex < 0){      System.out.println("Error: cantidad negativa");      return;
   }    transExentas = transex;
 }

 public double obtenerImporteTrans(){    return importeTrans;  }
 public int getTransExentas(){    return transExentas;  }

 public void ingreso(double cantidad){    super.ingreso(cantidad);    transacciones++;  }

 public void reintegro(double cantidad){    super.reintegro(cantidad);    transacciones++;  }

 public void comisiones(){
   // Se aplican mensualmente por el mantenimiento de la cuenta
   GregorianCalendar fechaActual = new GregorianCalendar();
   int día = fechaActual.get(Calendar.DAY_OF_MONTH);
   if (día == 1){      int n = transacciones - transExentas;
    if (n > 0) reintegro(n * importePorTrans);
    transacciones = 0;
   } }

                    public double intereses(){
                      GregorianCalendar fechaActual = new GregorianCalendar();
                      int día = fechaActual.get(Calendar.DAY_OF_MONTH);
                      if (día != 1) return 0.0;
                    // Acumular los intereses por mes sólo los días 1 de cada mes
                      double interesesProducidos = 0.0;
                    // Hasta 3000 pesos al 0.5%. El resto al interés establecido.
                      if (estado() <= 3000)
                        interesesProducidos = estado() * 0.5 / 1200.0;
                      else{
                        interesesProducidos = 3000 * 0.5 / 1200.0 +
                            (estado() - 3000) * obtenerTipoDeInterés() / 1200.0;
```

```
    }
    ingreso(interesesProducidos);
  // Este ingreso no debe incrementar las transacciones
    decrementarTransacciones();
 // Devolver el interés mensual por si fuera necesario
    return interesesProducidos;
  }
}
```

Procedimiento

Tomando las consideraciones anteriormente explicadas codifica de la siguiente jerarquía de clases:



Evaluación

Los alumnos entregarán la codificación de las clases y una clase main ejecutándose en el laboratorio

Criterio	Clase Persona	Clase Alumno	Clase Docente	Clase DocenteExterno	Clase CVitae
1.Cumplió con la codificación correcta de los atributos	SI NO	SI NO	SI NO	SI NO	SI NO

2. Codificó correctamente la herencia de las clases	SI NO	SI NO	SI NO	SI NO	SI NO
3.Cumplió con la codificación correcta de los constructores	SI NO	SI NO	SI NO	SI NO	SI NO
Puntaje (marque los puntos obtenidos y sume) Total_____	0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3

Ejercicio 4: Interfaces y su implementación

U. Competencia 1 **Conocer los conceptos básicos de la POO, la UML y las plataformas de desarrollo**

Duración Estimada: 2 sesiones de 50 minutos

Objetivo:

El alumno usará las interfaces en java como estrategia para resolver la herencia múltiple

Introducción Teórica

Una interfaz se utiliza para definir un protocolo de conducta que puede ser implementado por cualquier clase en una jerarquía de clases. La utilidad que esto pueda tener puede resumirse en los siguientes puntos:

- Captar similitud entre clases no relacionadas sin forzar entre ellas una relación artificial. Una acción de este tipo permitirá incluso, definir una matriz de objetos de esas clases y aplicar, si fuera necesario, la definición de polimorfismo.
- Declarar métodos que una o más clases deben implementar en determinadas situaciones. Suponga que se ha diseñado una clase de objetos que pueden tener un comportamiento especial siempre que implemente unos determinados métodos. Por ejemplo, cuando aprenda sobre Applets y subprocesos comprobará que usar un subproceso en un Applet implica que la clase de éste implemente la interfaz Runnable.
- Publicar la interfaz de programación de una clase sin descubrir cómo está implementada.

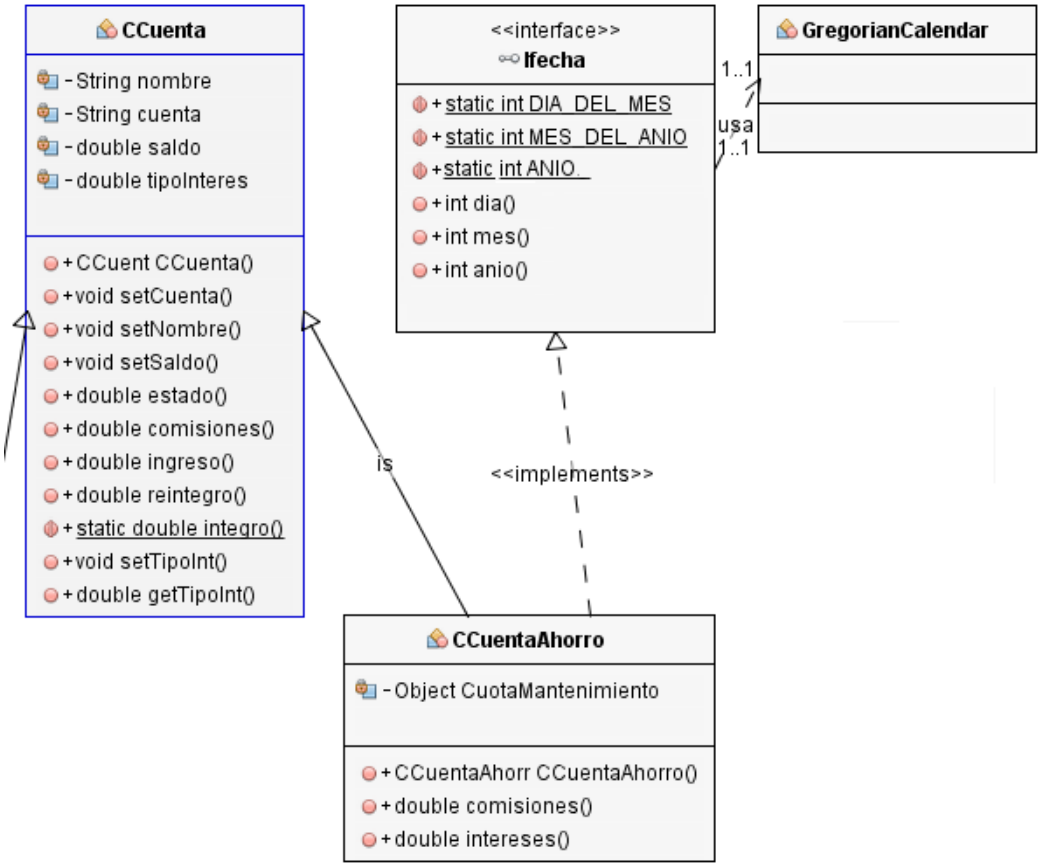
En otro caso, otros desarrolladores recibirían la clase compilada y la interfaz correspondiente

Las interfaces, al igual que las clases y métodos abstractos, proporcionan las plantillas de comportamiento que se espera sean implementados por otras clases. Esto es, una interfaz Java declara un conjunto de métodos, pero no los define (sólo aporta los prototipos de los métodos). También puede incluir definiciones de constantes.

Como ejemplo, se realizará una interfaz IFecha que interactúe entre dos clases “GregorianCalendar” y “CCuenta”. La clase GregorianCalendar es un proveedor de servicios; en nuestro ejemplo, notificará el día a los objetos derivados de CCuenta cuando intenten ejecutar sus métodos comisiones o intereses. Para ello, como se muestra a continuación, se proporciona el método get que devuelve el tipo de dato (día, mes, etc) solicitado.

Cualquier objeto derivado de CCuenta que quiera utilizar un objeto GregorianCalendar debe implementar el método día proporcionado por la interfaz IFecha. Este método es el medio utilizado por el objeto GregorianCalendar para notificar al objeto derivado de CCuenta el día actual. La interfaz IFecha tendrá el siguiente aspecto:

La jerarquía de clases se representará así:



La codificación de la Interface Ifecha será así:

```
import java.util.*;
public interface IFecha{
   //atributos de la interface que deb ser statics y final
  public final static int DIA_DEL_MES = Calendar.DAY_OF_MONTH;
  public final static int MES_DEL_ANIO = Calendar.MONTH;
  public final static int ANIO = Calendar.YEAR;
  //métodos de la interface
  public abstract int día();
  public abstract int mes();
  public abstract int año();
}
```

Observe que una interfaz sólo declara los métodos, no los define, y además puede definir constantes. Todo método declarado en una interfaz es implícitamente público y abstracto (public y abstract); y todas las constantes son implícitamente públicas, finales y estáticas (public, final y static). Cualquier clase puede tener acceso a las constantes de la interfaz a través del nombre de la misma. Por ejemplo:

IFecha.ANIO

En cambio, una clase que implemente la interfaz puede tratar las constantes como si las hubiesen heredado; esto es, accediendo directamente a su nombre. Para utilizar una interfaz hay que añadir el nombre de la misma precedido por la palabra clave **implements** a la definición de la clase. La palabra clave **implements** sigue a la palabra clave **extends**, si existe.

La calse CCuentaAhorro quedaría de la siguiente forma:

```
import java.util.*;
public class CCuentaAhorro extends CCuenta implements Ifecha{
 // Atributos
 private double cuotaMantenimiento;

 // Métodos
 public CCuentaAhorro() {  } // constructor sin parámetros

 public CCuentaAhorro(String nom, String cue, double sal,
              double tipo, double mant){
  super(nom, cue, sal, tipo); // invoca al constructor CCuenta
  asignarCuotaManten(mant);   // inicia cuotaMantenimiento
 }

 public void asignarCuotaManten(double cantidad){    if (cantidad < 0){    System.out.println("Error: cantidad negativa");
    return;    }
  cuotaMantenimiento = cantidad;  }

 public double obtenerCuotaManten(){     return cuotaMantenimiento;  }
```

```
// Se aplican mensualmente por el mantenimiento de la cuenta
public void comisiones(){    if (día() == 1) reintegro(cuotaMantenimiento);  }

public double intereses(){
  if (día() != 1) return 0.0;
  // Acumular los intereses por mes sólo los días 1 de cada mes
  double interesesProducidos = 0.0;
  interesesProducidos = estado() * obtenerTipoDeInterés() / 1200.0;
  ingreso(interesesProducidos);
   // Devolver el interés mensual por si fuera necesario
  return interesesProducidos;
}

// la implementación de los métodos de la interfaz IFecha es obligatorio para las clase CCuentaAhorro
public int día(){
  GregorianCalendar fechaActual = new GregorianCalendar();
  return fechaActual.get(DIA_DEL_MES);  }
// los siguientes métodos no se necesitarán en esta clase  pero deben implementarse; son reglas del lenguaje java
public int mes() { return 0; }
 public int año() { return 0; } // no se necesita
}
```

Como una interfaz sólo aporta declaraciones de métodos abstractos, **es nuestra obligación definir todos los métodos en cada una de las clases que utilice la interfaz. No podemos elegir y definir sólo aquellos métodos que necesitemos. De no hacerlo, Java obligaría a que la clase fuera abstracta. Además, el acceso a las constantes es directo.**

Si una clase implementa una interfaz, todas sus subclases heredarán los nuevos métodos que se hayan implementado en la superclase, así como las constantes definidas por la interfaz.

Una interfaz es un tipo de datos; un tipo referenciado. Por lo tanto, el nombre de una interfaz se puede utilizar en cualquier lugar donde pueda aparecer el nombre de cualquier otro tipo de datos.

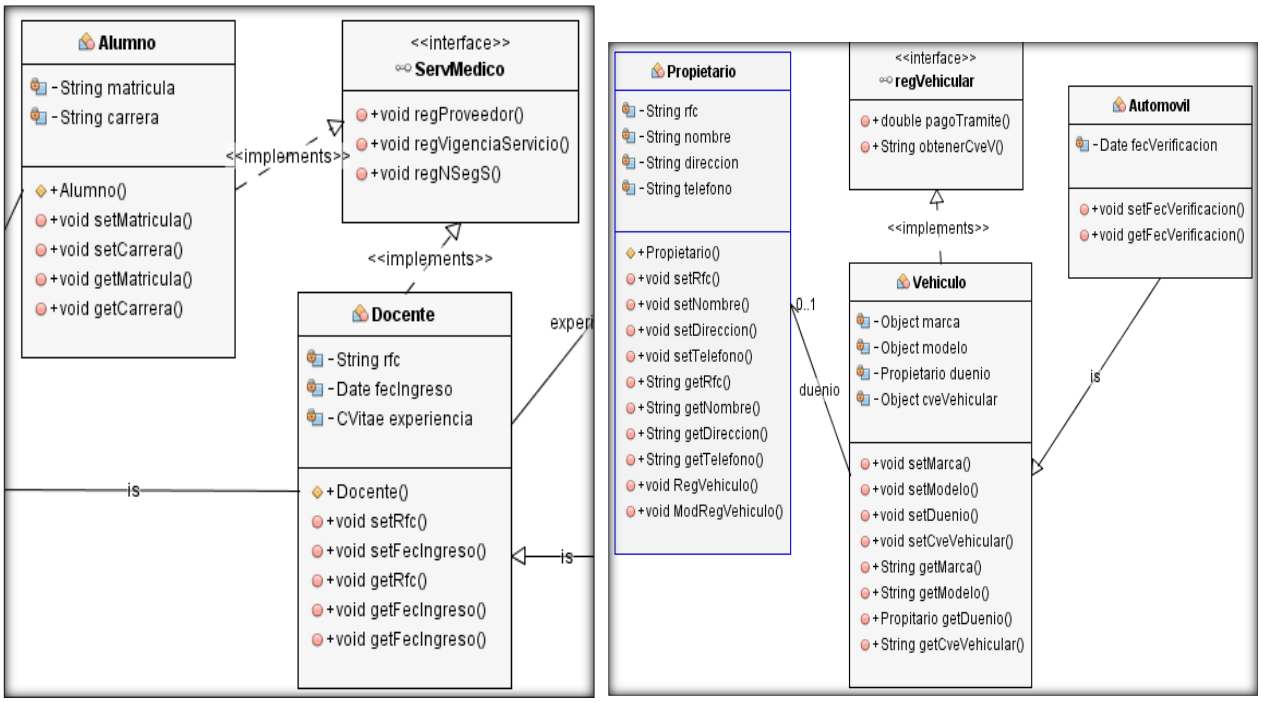
Por ejemplo, se puede declarar una matriz clientes que sea del tipo IFecha y asignar a cada elemento un objeto de alguna de las subclases de CCuenta:

```
//Ejemplo
IFecha[ ] clientes = new IFecha[3];
CCuentaAhorro cliente0=newCCuentaAhorro();
clientes[0]=cliente0;
((CCuentaAhorro)clientes[0]).asignarNombre(“cliente0);
System.out.println(cliente0.obtenerNombre());
// . . .
```

Una variable del tipo interfaz espera referenciar un objeto que tenga implementada dicha interfaz, de lo contrario el compilador de Java mostrará un error

Procedimiento

Codifique las dos jerarquías siguientes y muestre la ejecución mediante una clase main para cada caso



Evaluación:

Criterio	Caso 1 Escuela	Caso 2 Vehiculos
1. Cumplió con la codificación correcta de todas las clases de la jerarquía	SI NO	SI NO
2. Codificó correctamente la implementación de las clases	SI NO	SI NO
Puntaje Total_____	0 1 2	0 1 2

Unidad 2 Sintaxis de JAVA

Ejercicio 5 Tipos de datos y sintaxis básica en JAVA

U. Competencia 2 **Sintaxis básica de un lenguaje Orientado a Objetos**

Duración Estimada: 1 sesión de 50 minutos

Objetivo

El alumno utilizará los elementos básicos de java para la codificación de algoritmos

Introducción Teórica:

Cuando se programa en Java, se coloca todo el código en métodos, no se permite código fuera de un método o fuera de una clase.

Letras permitidas.
A-Z, a-z, 0-9, y cualquier carácter especial de unicode arriba de 00C0 como_ , $, á, e´, í, ó, ú.

Caracteres en blanco.	ASCII	SP
Tabulador horizontal.	ASCII	HT
Avance de página	ASCII	FT
Nueva línea	ASCII	LF
Retorno de carro	ASCII	CR o CR LF equivale a “\n”

Caracteres especiales y signos de puntuación:

, . ; : ¿ ‘ “ ( ) { } [ ] < ! / | \ + % & * - = > ~ ^

Secuencias de escape.

SECUENCIA	ASCII	DEFINICIÓN
\n	CR + CF	Retorno de carro más salto de línea.
\t	HT	Tabulador horizontal.
\b	BS	Retroceso.
\r	CR	Retorno de carro.
\f	FF	Alimentación de página (impresora)
\’	‘	Comilla simple
\”	“	Comilla doble
****	\	Barra invertida
\ddd		Carácter ASCII representación octal
\udddd		Carácter ASCII representación unicode
\u0007	Bell	Sonido
\u000b	VT	Tabulación vertical (impresoras).

Comentarios

En Java hay tres tipos de comentarios:

// comentarios para una sola línea
/* comentarios de una o más líneas ***/**
/** comentario de documentación, de una o más líneas ***/**

Los dos primeros tipos de comentarios son los que todo programador conoce y se utilizan del mismo modo. Los comentarios de documentación, colocados inmediatamente antes de una declaración (de variable o función), indican que ese comentario ha de ser colocado en la documentación que se genera automáticamente cuando se utiliza la herramienta de Java, javadoc. Dichos comentarios sirven como descripción del elemento declarado permitiendo generar una documentación de nuestras clases escrita al mismo tiempo que se genera el código.

En este tipo de comentario para documentación, se permite la introducción de algunos tokens o palabras clave, que harán que la información que les sigue aparezca de forma diferente al resto en la documentación.

Identificadores

Los identificadores nombran variables, métodos, clases y objetos; cualquier cosa que el programador necesite identificar o usar. En Java, un identificador comienza con una letra, un subrayado (_) o un símbolo de dólar (**$**). Los siguientes caracteres pueden ser letras o dígitos. Se distinguen las mayúsculas de las minúsculas y no hay longitud máxima.

Reglas para la definición de identificadores

- El primer carácter tiene que ser una letra, o un guion bajo o $.
- Es sensible al contexto.
- Pueden ser de cualquier tamaño.
- NO se usan las palabras reservadas:

abstract	**default**	**if**	**private**	**throw**	
boolean	**do**	**implements**	**protected**	**trhows**	
break	**double**	**import**	**public**	**transient**	
byte	**else**	**instanceof**	**return**	**try**	
case	**extends**	**int**	**short**	**void**	
catch	**final**	**interface**	**static**	**volatile**	
char	**finally**	**long**	**super**	**while**	
class	**float**	**native**	**switch**		
const	**for**	**new**	**synchronized**		
continue	**goto**	**package**	**this**		

Las palabras reservadas siempre se escriben en minúscula. Con base en las reglas anteriores serían identificadores válidos:

identificador
nombre_usuario

```
Nombre_Usuario
_variable_del_sistema
$transaccion
```

y su uso sería, por ejemplo:

```
int  contador_principal;
char  _lista_de_ficheros;
float  $cantidad_en_Ptas;
```

Literales

Un valor constante en Java se crea utilizando una representación literal de él. Java utiliza cinco tipos de elementos: enteros, reales en coma flotante, booleanos, caracteres y cadenas, que se pueden poner en cualquier lugar del código fuente de Java. Cada uno de estos literales tiene un tipo correspondiente asociado con él.

Enteros:

byte	8 bits	complemento a dos
short	16 bits	complemento a dos
int	32 bits	complemento a dos
long	64 bits	complemento a dos
Por ejemplo:	**21 077 0xDC00**	

Reales en coma flotante:

float	32 bits	IEEE 754
double	64 bits	IEEE 754
Por ejemplo:	**3.14 2e12 3.1E12**	

Booleanos: **True false**

Caracteres: por ejemplo**: a \t \u ???? [????] es un número unicode**

Cadenas: por ejemplo: **"Esto es una cadena literal"**

Declaración asignación de variables.

Tipo de dato o clase del que será el objeto	Nombre de variable	Signo de Igualdad que indica referencia a	Palabra que indica la construcción de un objeto	Constructor necesario para generar el objeto
String	Cad1	=	new	String();
Alumno	Alum1	=	new	Alumno(“Pedro”, “12/02/89”);
int	Sueldo;	=	564;	
float	Perímetro	=	123.50;	

Operadores:

Aritméticos. + - * // % //igual que en C.

Las conversiones son:
int / int = int
int + float = float //El resultado se convierte al tipo más alto.

Relación. **< > <= >= !=**

Ejemplo:
int x=10 ,y=0;
bolean r;
r = x==y; // r= false
r = x>y;
r = x != y;

Lógicos. && AND || OR ! NOT ^ XOR

Unitarios
~ Complemento a uno. (Cambia los unos por ceros y los ceros por unos). El operando tiene que ser entero. (ASCII 126)
- Se realiza el complemento a dos (cambia el signo del operando) El operando debe ser entero.

Ejemplo:
int a=2, b=0, c=0;
c = -a; //c toma el valor de -2
c= ~ b; //c toma el valor de -1

A nivel bit.
& AND a nivel bit
| OR a nivel bit (ASCII 124).
^ XOR a nivel bit (ASCII 94).
<< Desplaza a la izquierda rellenando de ceros la derecha.
>> Desplaza a la derecha rellenando con bits de signo por la izquierda.
>>> Desplaza a la derecha rellenando con ceros la izquierda

Ejemplos:
int a=255, r=0, m=32;
r= a & 017 ; //r = 15 . El octal 017 = 15 en decimal
a = 11111111
b = 00001111
00001111

r = r | m; // r = 47
r = 15 en decimal => 00001111 en binario
m = 32 en decimal => 00100000 en binario
00101111

r = a & ~ 07; //r= 248

octal 07 = 00000111 en binario
~07 = 11111000
Decimal 255 = 11111111 en binario

11111111
11111000
11111000

r = a >> 7; //a se desplaza 7 bits a la derecha Por lo tanto, a = 1,
r = m <<1; // r = 64. Equivale a r= m*2
r = m >> 1; //r = 16. Equivale ar r=m/2

De asignación. ++ -- = *= /= %= += -=
<<= Desplazamiento a la izquierda
>>= Desplazamiento a la derecha rellenando con bit de signo a la izquierda.
>>>= Desplazamiento a la derecha rellenando con ceros a la izquierda.
&= Operación AND a nivel bit
|= Operación OR a nivel bit
^= Operación XOR a nivel bit

Ejemplo:
n >>= 1: //Equivale a n = n>>1;
x = ++n; // Se realiza primero el incremento y después la asignación.
x= n++; //Primero se realiza la asignación y después el incremento.

Condicional. operador1 **?** operador2 **:** operador3
La expresión operador1 tiene que ser booleana. La ejecución se realiza de la siguiente forma:
Si el resultado de la evaluación del operador1 es true, el resultado de la expresión condicional es operador2.
Si el resultado de la evaluación del operador1 es false, el resultado de la expresión condicional es operador 3.

Ejemplo:
double a = 10.2, b=20.5, mayor =0;
mayor = (a>b) ? a : b ;

Los operadores de Java son muy parecidos en estilo y funcionamiento a los de C. En la siguiente tabla aparecen los operadores que se utilizan en Java, por orden de precedencia:

Precedencia de Operadores en Java

operadores sufijo	[] . (params) expr++ expr--
operadores unarios	++expr --expr +expr -expr ~ !
creación o tipo	new (type)expr
multiplicadores	* / %
suma/resta	+ -
desplazamiento	<< >> >>>
relacionales	< > <= >= instanceof
igualdad	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
AND lógico	&&
OR lógico	||
condicional	? :
asignación	= += -= *= /= %= ^= &= |= <<= >>= >>>=

Los operadores numéricos se comportan como esperamos:

int = int + int

Los operadores relacionales devuelven un valor booleano.

Para las cadenas, se pueden utilizar los operadores relacionales para comparaciones además de + y += para la concatenación:

String nombre = "nombre" + "Apellido";

El operador **=** siempre hace copias de objetos, marcando los antiguos para borrarlos, y ya se encargará el garbage collector de devolver al sistema la memoria ocupada por el objeto eliminado.

Sólo hay un par de secuencias con otros caracteres que pueden aparecer en el código Java; son los separadores simples, que van a definir la forma y función del código. Los separadores admitidos en Java son:

( ) (paréntesis) Para contener listas de parámetros en la definición y llamada a métodos. También se utiliza para definir precedencia en expresiones, contener expresiones para control de flujo y rodear las conversiones de tipo.

{} (llaves) Para contener los valores de matrices inicializadas automáticamente. También se utiliza para definir un bloque de código, para clases, métodos y ámbitos locales.

[ ] (corchetes) Para declarar tipos matriz. También se utiliza cuando se referencian valores de matriz.

; (punto y coma) Separa sentencias.

, (coma) Separa identificadores consecutivos en una declaración de variables. También se utiliza para encadenar sentencias dentro de una sentencia for.

. (punto) Para separar nombres de paquete de subpaquetes y clases. También se utiliza para separar una variable o método de una variable de referencia.

Declaración de una clase.

```
class CElementosJava{ //definición de una clase
        //definición de los atributos
        short dia, mes, año;
       //definición de métodos
        void  Test(){
                int contador =0;
                String Nombre =””,Apellidos=””;
                dia = 20;
                Apellidos = “Ceballos”;
                }
        //más métodos
}
```

Las variables dia, mes y año son accesibles desde todos los métodos no static de la clase. Se les conoce como variables miembro de la clase.

Las otras variables están declaradas dentro del cuerpo del método Test. Por lo que son locales al método. Una variable local no puede ser del tipo static. Las variables miembro de una clase son iniciadas por omisión por el compilador. Los números se inicializan en cero, los caracteres se inicializan en ‘\0’. Las referencias a cadenas y el resto de referencias a otros objetos se inicializan con “null”. Las variables locales no se inicializan, por lo que es obligación del usuario inicializarlas, de lo contrario el compilador visualizará un mensaje de error en todas las sentencias que hagan referencia a esas variables.

Ejemplo:

```
Class CElementosJava {
        short var1;
        void Test() {
                int var2;
                System.out.println(var2);  //error pues no ha sido inicializada al ser variable de método
                System.out.println(var1);  //correcto pues como es atributos de objeto fue inicializada por JVM
        }
}
```

Ejemplo de Operadores

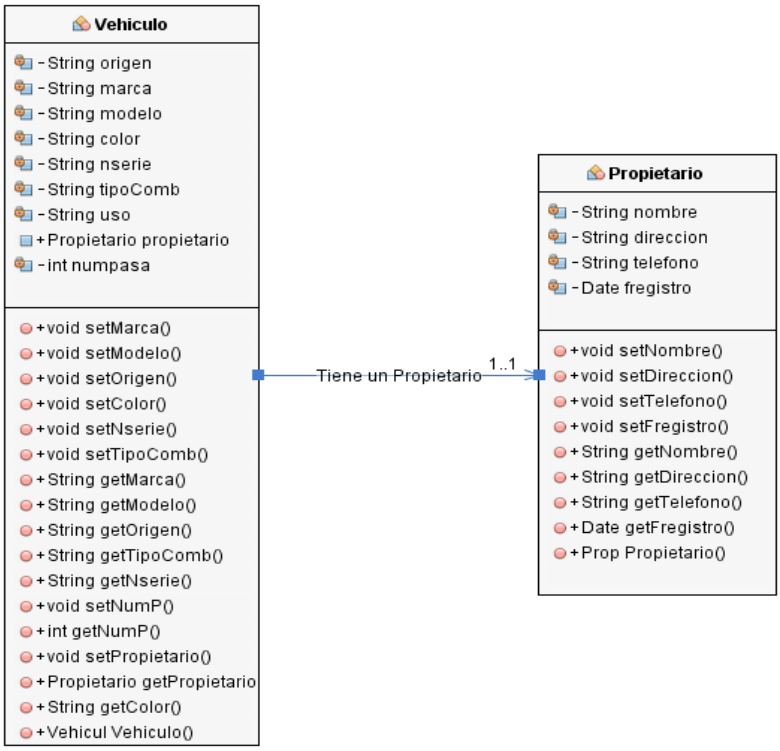
```
1.  public class IncrementoDecremento
2.  {
3.  public static void main(String[] args)   {
4.  int variable = 10;
5.  System.out.println(variable);
6.  System.out.println(++variable);
7.  System.out.println(variable++);
8.  System.out.println(variable--);
9.  System.out.println(--variable);
10. System.out.println(variable);
11. }     }
```

Ejemplo de Tipos de Datos primitivos

```
1.  public class TiposPrimitivos   {
2.  public static void main(String[] args)   {
3.  byte b = 127;
4.  char c = 65; // Representa el carácter 'A'
5.  short s = 32767;
6.  // Conversiones implícitas
7.  int i = b;
8.  long l = s;
9.  float f = i;
10. double d = f;
11. } }
```

Procedimiento

Codifique las clases del siguiente diagrama



Evaluación:

Criterio	
1. Cumplió con la codificación correcta en el identificador de clase	SI NO
2. Cumplió con la codificación correcta en el constructor	SI NO
3. Cumplió con la codificación correcta en los métodos getter	SI NO
4. Cumplió con la codificación correcta en los métodos setter	SI NO
Puntaje Total_____	0 1 2 3 4

Ejercicio 6 Clases, constructores, setters y getters

U Competencia 2: **Sintaxis básica de un lenguaje Orientado a Objetos**

Duración Estimada: 1 sesión de 50 minutos

Competencia 2:

Objetivo:

El alumno conocerá las reglas de sintaxis elementales para codificar una clase simple en el lenguaje java e identificará la correspondencia entre el diagrama UML de la clase con la estructura del código

Introducción Teórica

Clases

Las clases son la base Java. Todo en Java forma parte de una clase, es una clase o describe cómo funciona una clase. El conocimiento de las clases es fundamental para poder entender los programas Java.

Todas las acciones de los programas Java se colocan dentro del bloque de una clase o un objeto. Todos los métodos se definen dentro del bloque de la clase, Java no soporta funciones o variables globales. Así pues, el esqueleto de cualquier aplicación Java se basa en la definición de una clase.

Todos los datos básicos, como los enteros, se deben declarar en las clases antes de hacer uso de ellos. Son pocas las sentencias que se pueden colocar fuera del bloque de una clase. La palabra clave `import` puede colocarse al principio de un fichero, fuera del bloque de la clase. Sin embargo, el compilador reemplazará esa sentencia con el contenido del fichero que se indique, que consistirá, como es de suponer, en más clases.

Tipos de Clases

Hasta ahora sólo se ha utilizado la palabra clave public para calificar el nombre de las clases que hemos visto, pero hay tres modificadores más. Los tipos de clases que podemos definir son:

abstract Una clase abstract tiene al menos un método abstracto. Una clase abstracta no se instancia, sino que se utiliza como clase base para la herencia.

final Una clase final se declara como la clase que termina una cadena de herencia. No se puede heredar de una clase final. Por ejemplo, la clase Math es una clase final.

public Las clases public son accesibles desde otras clases, bien sea directamente o por herencia. Son accesibles dentro del mismo paquete en el que se han declarado. Para acceder desde otros paquetes, primero tienen que ser importadas.

synchronizable Este modificador especifica que todos los métodos definidos en la clase son sincronizados, es decir, que no se puede acceder al mismo tiempo a ellos desde distintos threads (hilos de ejecución); el sistema se encarga de colocar los flags necesarios para evitarlo. Este

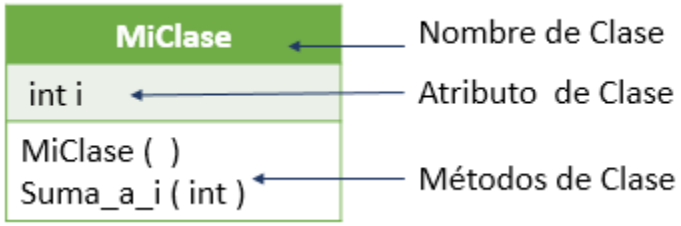
mecanismo hace que desde threads diferentes se puedan modificar las mismas variables sin que haya problemas de que se sobre escriban.

Variables y Métodos de Instancia

Una clase en Java puede contener variables y métodos. Las variables pueden ser tipos primitivos como int, char, etc. Los métodos son funciones.

Por ejemplo, en el siguiente trozo de código podemos observarlo y la imagen muestra el modelo UML:

```
public MiClase {
   int i;
   public MiClase() {
      i = 10;
      }
   public void Suma_a_i( int j ) {
      i = i + j;
      }
   }
```



La clase MiClase contiene una variable (`i`) y dos métodos, MiClase que es el constructor de la clase y Suma_a_i(int j).

Ámbito de una variable

Los bloques de sentencias compuestas en Java se delimitan con dos llaves. Las variables de Java sólo son válidas desde el punto donde están declaradas hasta el final de la sentencia compuesta que la engloba. Se pueden anidar estas sentencias compuestas, y cada una puede contener su propio conjunto de declaraciones de variables locales. Sin embargo, no se puede declarar una variable con el mismo nombre que una de ámbito exterior. El siguiente ejemplo intenta declarar dos variables separadas con el mismo nombre. En C y C++ son distintas, porque están declaradas dentro de ámbitos diferentes. En Java, esto es ilegal.

```
Class Ambito {
   int i = 1;       // ámbito exterior
      {          // crea un nuevo ámbito
      int i = 2;    // error de compilación
      }
}
```

Métodos y Constructores

Los métodos son funciones que pueden ser llamadas dentro de la clase o por otras clases. El constructor es un tipo específico de método que siempre tiene el mismo nombre que la clase.

Cuando se declara una clase en Java, se pueden declarar uno o más constructores opcionales que realizan la inicialización cuando se instancia (se crea una ocurrencia) un objeto de dicha clase.

Utilizando el código de ejemplo anterior, cuando se crea una nueva instancia de MiClase, se crean (instancian) todos los métodos y variables, y se llama al constructor de la clase:

```
MiClase mc;

mc = new MiClase();
```

La palabra clave **new** se usa para crear una instancia de la clase. Antes de ser instanciada con new no consume memoria, simplemente es una declaración de tipo. Después de ser instanciado un nuevo objeto `mc`, el valor de `i` en el objeto `mc` será igual a 10. Se puede referenciar la variable (de instancia) `i` con el nombre del objeto:

```
mc.i++; // incrementa la instancia de i de mc
```

Al tener `mc` todas las variables y métodos de MiClase, se puede usar la primera sintaxis para llamar al método Suma_a_i() utilizando el nuevo nombre de clase mc:

```
mc.Suma_a_i( 10 );
```

y ahora la variable `mc.i` vale 21.

Finalizadores

Java no utiliza destructores (al contrario que C++) ya que tiene una forma de recoger automáticamente todos los objetos que se salen del alcance. No obstante proporciona un método que, cuando se especifique en el código de la clase, el reciclador de memoria (garbage collector) llamará:

```
// Cierra el canal cuando este objeto es reciclado
protected void finalize() {
   close();       }
```

Alcance de objetos y reciclado de memoria

Los objetos tienen un tiempo de vida y consumen recursos durante el mismo. Cuando un objeto no se va a utilizar más, debería liberar el espacio que ocupaba en la memoria de forma que las aplicaciones no la agoten (especialmente las grandes).

En Java, la recolección y liberación de memoria es responsabilidad de un thread llamado automatic garbage collector (recolector automático de basura). Este thread monitoriza el alcance de los objetos y marca los objetos que se han salido de alcance. Veamos un ejemplo:

```
String s;               // no se ha asignado todavía
s = new String( "abc" );  // memoria asignada
s = "def";              // se ha asignado nueva memoria
```

Más adelante veremos en detalle la clase String, pero una breve descripción de lo que hace esto es; crear un objeto String y rellenarlo con los caracteres "abc" y crear otro (nuevo) String y colocarle los caracteres "def".

En esencia se crean dos objetos:

Objeto String "abc"; Objeto String "def"

Al final de la tercera sentencia, el primer objeto creado de nombre `s` que contiene "abc" se ha salido de alcance. No hay forma de acceder a él. Ahora se tiene un nuevo objeto llamado `s` y contiene "def". Es marcado y eliminado en la siguiente iteración del thread reciclador de memoria.

Variables Finales

Una variable de un tipo primitivo declarada como final no puede cambiar su valor a lo largo de la ejecución del programa. Puede ser considerada como una constante, y equivale a la palabra const de C/C++.

Java permite separar la definición de la inicialización de una variable final. La inicialización puede hacerse más tarde, en tiempo de ejecución, llamando a métodos o en función de otros datos.

La variable final así definida es constante (no puede cambiar), pero no tiene por qué tener el mismo valor en todas las ejecuciones del programa, pues depende de cómo haya sido inicializada.

Además de las variables miembro, también las variables locales y los propios argumentos de un método pueden ser declarados final. Declarar como final un objeto miembro de una clase hace constante la referencia, pero no el propio objeto, que puede ser modificado a través de otra referencia. En Java no es posible hacer que un objeto sea constante.

Métodos de objeto

Los métodos son funciones definidas dentro de una clase. Salvo los métodos static o de clase, se aplican siempre a un objeto de la clase por medio del operador punto (**.**). Dicho objeto es su argumento implícito. Los métodos pueden además tener otros argumentos explícitos que van entre paréntesis, a continuación del nombre del método.

La primera línea de la definición de un método se llama declaración o header; el código comprendido entre las llaves {…} es el cuerpo o body del método. El header consta del cualificador de acceso (public, en este caso), del tipo del valor de retorno, del nombre de la función y de una lista de argumentos explícitos entre paréntesis, separados por comas. Si no hay argumentos explícitos se dejan los paréntesis vacíos.

Los métodos tienen visibilidad directa de las variables miembro del objeto que es su argumento implícito, es decir, pueden acceder a ellas sin cualificarlas con un nombre de objeto y el operador punto (.). De todas formas, también se puede acceder a ellas mediante la referencia this, de modo discrecional o si alguna variable local o argumento las oculta.

El valor de retorno puede ser un valor de un tipo primitivo o una referencia. En cualquier caso no puede haber más que un único valor de retorno (que puede ser un objeto o un array). Se puede devolver también una referencia a un objeto por medio de un nombre de interface. El objeto devuelto debe pertenecer a una clase que implemente esa interface.

Se puede devolver como valor de retorno un objeto de la misma clase que el método o de una sub-clase, pero nunca de una super-clase. Los métodos pueden definir variables locales. Su visibilidad llega desde la definición al final del bloque en el que han sido definidas. No hace falta inicializar las variables locales en el punto en que se definen, pero el compilador no permite utilizarlas sin haberles dado un valor. A diferencia de las variables miembro, las variables locales no se inicializan por defecto.

Metodos setters y getters

Como medida importante de encapsulamiento se recomienda definir los atributos privados, de esta forma solo los métodos propios de la clase podrá usarlos. Sin embrago será necesario algún mecanismo que permita a los métodos de otras clases que deban estar autorizados el acceso a dichos atributos. La estrategia se implementa mediante métodos públicos que devuelven los valores de los atributos y métodos que modifican dichos atributos; estos métodos reciben el nombre de setter para los que permiten modificar el valor del atributo y getter para los métodos que devuelven el valor de los atributos, se deberá codificar un par de métodos por cada atributo privado

La sintaxis de los métodos es la siguiente:

```
public void setNombreAtributo ( tipoDAtributo  valorNuevo){   atributo=valorNuevo;}

public tipoDAtributo getNombreAtributo (){ return atributo; }
```

Constructores

Un punto clave de la Programación Orientada Objetos es el evitar información incorrecta por no haber sido correctamente inicializadas las variables. Java no permite que haya variable miembro que no esté inicializadas. Ya se ha dicho que Java inicializa siempre con valores por defecto la variable miembro de clases y objetos. El segundo paso en la inicialización correcta de objetos es el uso de constructores. **Un constructor es un método que se llama automáticamente cada vez que se crea un objeto de una clase**. La principal misión del constructor es reservar memoria e inicializar la variable miembro de la clase.

Los constructores no tienen valor de retorno (ni siquiera void) y su nombre es el mismo que el de la clase. Su argumento implícito es el objeto que se está creando.

De ordinario una clase tiene varios constructores, que se diferencian por el tipo y número de sus argumentos (son un ejemplo típico de métodos sobrecargados). Se llama constructor por defecto al constructor que no tiene argumentos. El programador debe proporcionar en el código valores iniciales adecuados para toda la variable miembro.

Un constructor de una clase puede llamar a otro constructor previamente definido en la misma clase por medio de la palabra this. En este contexto, la palabra this sólo puede aparecer en la primera sentencia de un constructor.

El constructor de una sub-clase puede llamar al constructor de su super-clase por medio de la palabra super, seguida de los argumentos apropiados entre paréntesis. De esta forma, un constructor sólo tiene que inicializar por sí mismo las variables no heredadas.

El constructor es tan importante que, si el programador no prepara ningún constructor para una clase, el compilador crea un constructor por defecto, inicializando las variables de los tipos primitivos a su valor por defecto, y los Strings y las demás referencias a objetos a null. Si hace falta, se llama al constructor de la super-clase para que inicialice las variables heredadas.

Al igual que los demás métodos de una clase, los constructores pueden tener también los modificadores de acceso public, private, protected y package. Si un constructor es private, ninguna otra clase puede crear un objeto de esa clase. En este caso, puede haber métodos public y static (factory methods) que llamen al constructor y devuelvan un objeto de esa clase. Dentro de una clase, los constructores sólo pueden ser llamados por otros constructores o por métodos static. No pueden ser llamados por los métodos de objeto de la clase.

Inicializadores

Java todavía dispone de una tercera línea de actuación para evitar que haya variables sin inicializar correctamente. Son los inicializadores, que pueden ser static (para la clase) o de objeto.

Inicializadores static

Un inicializador static es un algo parecido a un método (un bloque {…} de código, sin nombre y sin argumentos, precedido por la palabra static) que se llama automáticamente al crear la clase (al utilizarla por primera vez). También se diferencia del constructor en que no es llamado para cada objeto, sino una sola vez para toda la clase.

Los tipos primitivos pueden inicializarse directamente con asignaciones en la clase o en el constructor, pero para inicializar objetos o elementos más complicados es bueno utilizar un inicializador (un bloque de código {…}), ya que permite gestionar excepciones3 con try…catch.

Los inicializadores static se crean dentro de la clase, como métodos sin nombre, sin argumentos y sin valor de retorno, con tan sólo la palabra static y el código entre llaves {...}. En una clase pueden definirse varios inicializadores static, que se llamarán en el orden en que han sido definidos.

Los inicializadores static se pueden utilizar para dar valor a las variables static. Además se suelen utilizar para llamar a métodos nativos, esto es, a métodos escritos por ejemplo en C/C++ (llamando a los métodos System.load() o System.loadLibrary(), que leen las librerías nativas).

Destrucción de objetos (liberación de memoria)

En Java no hay destructores como en C++. El sistema se ocupa automáticamente de liberar la memoria de los objetos que ya han perdido la referencia, esto es, objetos que ya no tienen ningún nombre que permita acceder a ellos, por ejemplo por haber llegado al final del bloque en el que habían sido definidos, porque a la referencia se le ha asignado el valor null o porque a la referencia se le ha asignado la dirección de otro objeto. A esta característica de Java se le llama garbage collection (recogida de basura).

En Java es normal que varias variables de tipo referencia apunten al mismo objeto. Java lleva internamente un contador de cuántas referencias hay sobre cada objeto. El objeto podrá ser borrado

cuando el número de referencias sea cero. Como ya se ha dicho, una forma de hacer que un objeto quede sin referencia es cambiar ésta a null, haciendo por ejemplo:

ObjetoRef = null;

En Java no se sabe exactamente cuándo se va a activar el garbage collector. Si no falta memoria es posible que no se llegue a activar en ningún momento. No es pues conveniente confiar en él para la realización de otras tareas más críticas.

Se puede llamar explícitamente al garbage collector con el método System.gc(), aunque esto es considerado por el sistema sólo como una "sugerencia" a la JVM.

Finalizadores

Los finalizadores son métodos que vienen a completar la labor del garbage collector. Un finalizador es un método que se llama automáticamente cuando se va a destruir un objeto (antes de que la memoria sea liberada de modo automático por el sistema). Se utilizan para ciertas operaciones de terminación distintas de liberar memoria (por ejemplo: cerrar ficheros, cerrar conexiones de red, liberar memoria reservada por funciones nativas, etc.). Hay que tener en cuenta que el garbage collector sólo libera la memoria reservada con new. Si por ejemplo se ha reservado memoria con funciones nativas en C (por ejemplo, utilizando la función malloc()), esta memoria hay que liberarla explícitamente utilizando el método finalize().

Un finalizador es un método de objeto (no static), sin valor de retorno (void), sin argumentos y que siempre se llama finalize(). Los finalizadores se llaman de modo automático siempre que hayan sido definidos por el programador de la clase. Para realizar su tarea correctamente, un finalizador debería terminar siempre llamando al finalizador de su super-clase.

Tampoco se puede saber el momento preciso en que los finalizadores van a ser llamados. En muchas ocasiones será conveniente que el programador realice esas operaciones de finalización de modo explícito mediante otros métodos que él mismo llame.

Resumen del proceso de creación de un objeto

El proceso de creación de objetos de una clase es el siguiente:

1. Al crear el primer objeto de la clase o al utilizar el primer método o variable static se localiza la clase y se carga en memoria.

2. Se ejecutan los inicializadores static (sólo una vez).

3. Cada vez que se quiere crear un nuevo objeto:

- se comienza reservando la memoria necesaria
- se da valor por defecto a las variables miembro de los tipos primitivos
- se ejecutan los inicializadores de objeto
- se ejecutan los constructores

Ejemplo de atributos de Instancia y de clase

```
1.  public class DeClaseDeInstancia     {
2.  public static void main(String[] args)    {
3.  Ayuda a1 = new Ayuda();
4.  Ayuda a2 = new Ayuda();
5.  Ayuda a3 = new Ayuda();
6.  System.out.println(a1.deClase);
7.  System.out.println(a1.deInstancia);
8.  }
9.  }
10. class Ayuda
11. {
12. public static int deClase;
13. public int deInstancia;
14. public Ayuda()
15. {
16. deClase++;
17. deInstancia++;
18. }
19. }
```

Ejemplo Clase String

```
1.  public class EjemploString {
2.  public static void main (String []args){
3.  String nombre =new String();
4.  String Apellido = new String ("Aguilar");
5.  nombre="Irene";
6.  int longitud =nombre.length();
7.  String Mayusculas = nombre.toUpperCase();
8.  String Nomcompleto =nombre.concat(" "+Apellido);
9.  String remplazo= Nomcompleto.replace('u','F');
10. String subcadena =Nomcompleto.substring(4,10);
11. System.out.println("Nombre = "+ nombre);
12. System.out.println("Apellido =" +Apellido);
13. System.out.println("Longitud de nombre =" +longitud);
14. System.out.println("Nombre en mayúsculas =" + Mayusculas);
15. System.out.println("Nombre Completo = "+ Nomcompleto);
16. System.out.println("Nombre Remplazado = "+ remplazo);
17. System.out.println("Subcadena de Nombre= "+ subcadena);
18. }   }
```

Procedimiento

En el ambiente de desarrollo Netbeans y con las indicaciones de tu profesor codifica, almacena y compila las siguientes clases:

```
public class Circulo {
        //Atributos
   private double r;
      /** Constructor */
   public Circulo( double r) {
      this.r=r;
   }
//métodos
   public double area(){
      return (Math.PI*r*r);
   }
   public double perimetro(){
      return(2*Math.PI*r);
   }
}
```

```
public class Rectangulo {
   //atributos
   private double b;
   private double h;

   //constructor por default o de base
   public Rectangulo(){
      b=3;
      h=2;
   }
// metodos
   public double area(){
   double a;
   a=b*h;
   return (a);
   }
  public double perimetro(){
   return(2*b+2*h);}

}
```

```
public class TestRectanguloCirculo {
public static void main(String[] args) {
Rectangulo rec1 =new Rectangulo();
double res;
res=rec1.area();
System.out.println("Area= "+ res);
System.out.println("Perimetro="+ rec1.perimetro());
Circulo circ= new Circulo(3);
System.out.println("Area=" + circ.area());

System.out.println("Perimetro="+ circ.perimetro());  }
}
```

Ejercicio

Una vez que comprendiste el uso de cada uno de los elementos de código de las clases anteriores diseña, codifica, almacena y compila tres clases más que permitan la obtención del área y perímetros de algunas figuras geométricas debes usar los métodos set… y get…

Entrega a tu profesor (a) como resultado de la práctica el diseño, el código y muestra la ejecución de las clases del ejercicio:

Diseño de clase	Código de Clase

Evaluación:

Criterio	
1. Cumplió con el diseño de las clases	SI NO
2. Cumplió con la codificación correcta en el identificador de clase	SI NO
3. Cumplió con la codificación correcta en el constructor	SI NO
4. Cumplió con la codificación correcta en los métodos getter	SI NO
5. Cumplió con la codificación correcta en los métodos setter	SI NO
Puntaje Total_____	0 1 2 3 4 5

Ejercicio 7 Clases para entrada y salida de datos

U Competencia 2: **Sintaxis básica de un lenguaje Orientado a Objetos**

Duración Estimada: 1 sesión de 50 minutos

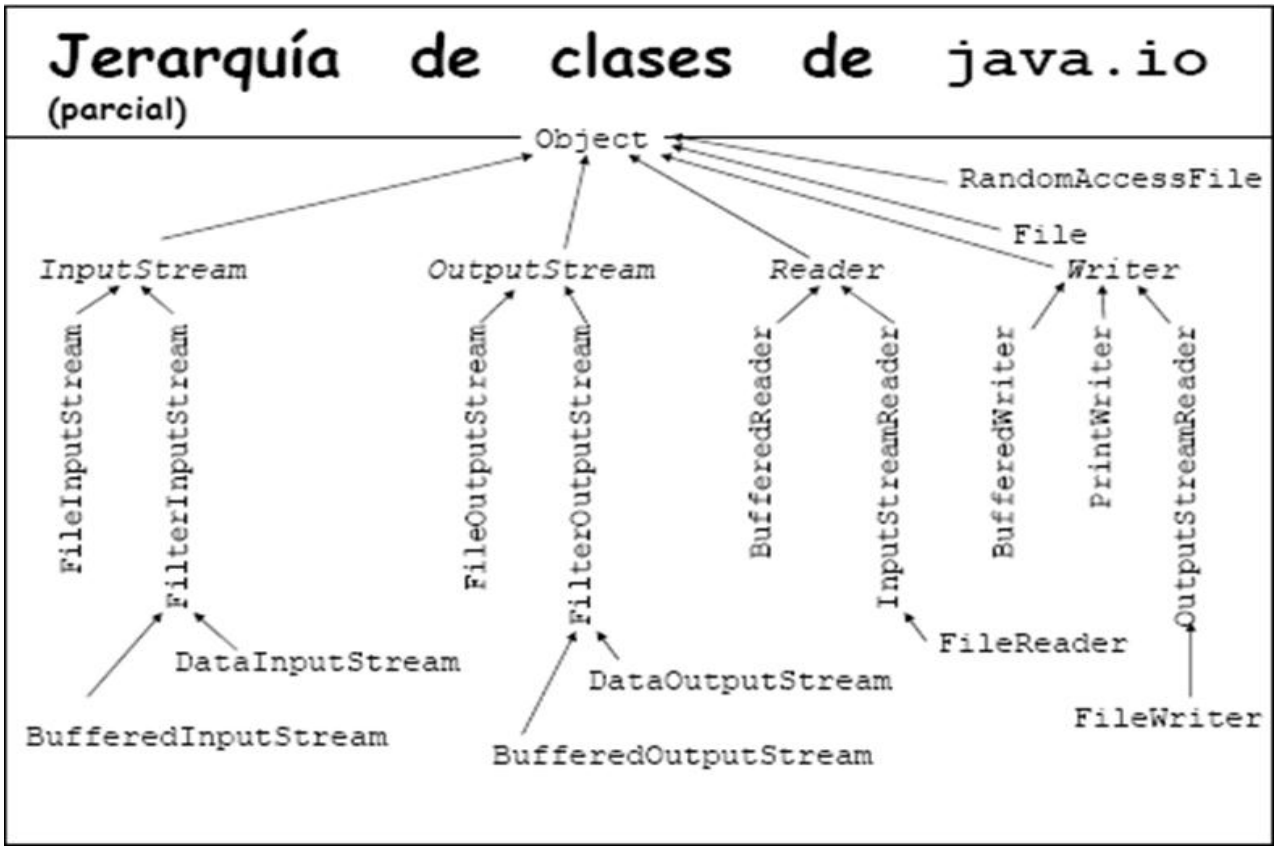
Objetivo:

El alumno usará las reglas de sintaxis para usar las clases necesarias para leer y mostrar datos en el lenguaje java

Introducción Teórica:

En el paquete java.io se almacenan las clases necesarias para usar flujos de datos los cuales permiten transferir datos entre aplicaciones y dispositivos, los flujos se clasifican en tres categorías:

1. Clases de entrada de datos orientadas a caracteres y a bytes
2. Clases de salida de datos orientadas a uso de caracteres y a bytes
3. Clases orientadas a archivos



Flujos estándar.

La biblioteca de Java proporciona tres flujos estándar, manipulados por la clase System del paquete java.lang y son automáticamente abiertos al iniciar un programa y cerrados cuando éste finaliza.

a) System.in Referencia a la entrada estándar del sistema (teclado).
b) System.out Referencia a la salida estándar (monitor).
c) System.err Referencia a la salida estándar de error (monitor).

Si se desea leer un byte o un elemento tipo char, se tiene:

```
b=(byte)System.in.read( );    O    c=(char) System.in.read( );
```

BufferedInputStream

Esta clase se deriva indirectamente de InputStream por lo tanto hereda todos los miembros de ésta; por ejemplo el método “read”.

Esta clase no aporta métodos nuevos, pero apoya con un buffer que actúa como memoria intermedia para lecturas futuras.

Cuando una aplicación ejecuta una sentencia de entrada (solicitud de datos) los datos obtenidos de origen se depositan en el buffer en bloques mayores a los solicitados. Ejemplo: Al leer de disco, se lee el bloque completo. Esto aumenta la velocidad de ejecución ya que la siguiente vez que se requiera algún dato no se tiene que esperar ya que está en el buffer. Por otra parte, cuando se trata de una operación de salida, los datos no serán enviados al destino hasta que esté lleno el buffer, lo que reduce el número de accesos a un dispositivo físico.

BufferedReader

Bajo un flujo de la clase BufferedReader subyace otro flujo de caracteres (objeto de la clase derivada Reader) o de bytes (objeto de la clase derivada inputStream). Esto es, cada petición de lectura hecha a un flujo de la clase BufferedReader es dirigida a otro flujo de bytes o caracteres subyacente.

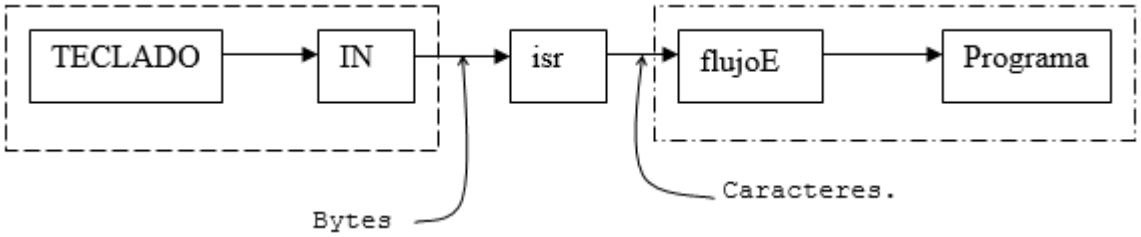
La figura anterior traducida a código dependiente de los flujos puede interpretarse como:

```
BufferedReader flujoE = new BufferedReader(isr);
```

El código anterior indica que flujoE dirigirá todas las invocaciones de sus métodos al flujo subyacente isr, en el caso de que el origen sea el teclado (dispositivo vinculado con System.in), convertirá los bytes leídos del teclado en caracteres. De esta forma flujoE suministrará un flujo de caracteres al programa destino de los datos. Para ello se define el flujo denominado como “isr”.

```
InputStreamReader isr = newInputStreaReader(System.in);
```

InputStream establece un puente para pasar flujos de bytes a flujos de caracteres.



BufferedReader proporciona métodos análogos a BufferedInputStream.

Método readLine

Lee una línea de texto y devuelve un objeto de la clase String.

Se lee una cadena hasta encontrar "\r" o "\n" o ambos; estos caracteres son leídos pero no almacenados. Ejemplo: Lectura de una cadena:

```
import java.io.*;
public class LeerUnaCadena {
  public static void  main(String args[]){
    // Definir un flujo de caracteres de entrada: flujoE
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader flujoE = new BufferedReader(isr);
    PrintStream flujoS = System.out;
    String sdato; // variable para almacenar una línea de texto
    try {
      flujoS.print("Introduzca un texto: ");
      sdato = flujoE.readLine(); // leer una línea de texto
      flujoS.println(sdato);     // escribir la línea leída
    }
    catch (IOException ignorada) { }
  }
    // Definir una referencia al flujo estándar de salida: flujoS
}
```

PrintStream

Se deriva indirectamente de OutputStream, por lo que hereda todos los miembros de ésta.
Los métodos print y println son lo mismo con la única diferencia de que println añade "\n" al final de la cadena. Por lo que:

System.out.print("hola\n"); Equivale a System.out.println("hola");

Los argumentos para print y println pueden ser primitivos o referenciados.

Object, String, char[], int, long, float, double, bolean. Ejemplo:

```
public class TestTiposDatos{
  // Tipos de datos
  public static void main(String[] args){
    String sCadena = "Lenguaje Java";
    char[] cMatrizCars = { 'a', 'b', 'c' };   // matriz de caracteres
    int dato_int = 4;
    long dato_long = Long.MIN_VALUE;    // mínimo valor long
    float dato_float = Float.MAX_VALUE; // máximo valor float
    double dato_double = Math.PI;       // 3.1415926
    boolean dato_boolean = true;
    System.out.println(sCadena);
    System.out.println(cMatrizCars);
    System.out.println(dato_int);
    System.out.println(dato_long);
    System.out.println(dato_float);
    System.out.println(dato_double);
    System.out.println(dato_boolean);
  }
}
```

En print y println se puede imprimir un objeto. Por ejemplo, se puede imprimir un objeto del tipo String. La cadena de un objeto “Class” imprimirá una cadena de caracteres correspondiente al nombre de la clase del objeto que estuviera siendo inspeccionado.

PrintStream. El buffer de salida es vaciado automáticamente.

PrintWriter. En este caso, no se vacía el buffer de salida, por lo que para vaciar el buffer se usa el método “flush”. Ejemplo:

```
PrintWriter flujoS = new PrintWriter(Sysrem.out);
Int idato=4;
flujoS.println(idato);
flujoS.flush();
```

Ejemplo de lectura de teclado

```
1.  import java.io.*;
2.  public class TestCirculo {
3.  public static void main(String[] args) throws IOException{
4.  System.out.println("Proporciona el radio");
5.  //Se establece el  flujo vinculado al flujo estándar de entrada
6.  BufferedReader entrada = new BufferedReader (new InputStreamReader(System.in));
7.  String cadradio= entrada.readLine();
8.  double radio;
9.  //La clase Double es una clase wrapper que tiene un metodo que convierte un dato tipo String a un double
10. radio=Double.parseDouble(cadradio);
11. Circulo co= new Circulo(radio);
12. System.out.println("Area "+ co.area());
13. System.out.println("Perimetro "+ co.perimetro());
14. }
15. }
```

Procedimiento

Con base en el ejemplo anterior codifica las clases Test de las clases diseñadas en el ejercicio anterior, incluya las clases que facilite la interacción con el usuario mediante flujos de entrada, tenga cuidado en la conversión de datos. Entrega a tu profesor (a) como resultado de la práctica el código desarrollado y muestra la ejecución de las clases del ejercicio:

Clase Test de la Clase

Evaluación:

Criterio	
1. Cumplió con el uso del flujo de entrada	SI NO
1. Cumplió con el uso de la conversión de datos	SI NO
2. Cumplió con la ejecución del código	SI NO
Puntaje Total_____	0 1 2 3

Ejercicio 8 Clase Object y sus métodos

U Competencia 2: **Sintaxis básica de un lenguaje Orientado a Objetos**

Duración Estimada: 1 sesión de 50 minutos

Objetivo:

El alumno usará los métodos heredados de la Clase Object

Introducción Teórica:

Se sitúa en la parte más alta del árbol de jerarquía. La clase Object del paquete java.lang es la clase raíz de la jerarquía de clases java.

Cada clase es un descendiente, directo o indirecto, de la clase Object. Cada clase que usa o escribe hereda los métodos de instancia de Object. No necesita usar ninguno de estos métodos, pero, si lo desea, puede necesitar sobrepasarlos con el código que es específico a su clase. Los métodos heredados de Object son:

Métodos	Descripción
protected void finalize() throws Throwable	Llamado por el recolector de basura sobre un objeto cuando se convierte en basura cuando la colección basura determina que ya no hay más referencias al objeto
public boolean equals(Object obj)	Indica si algún otro objeto es "igual a" este.
public final Class getClass()	Devuelve la clase de tiempo de ejecución de un objeto
public int hashCode()	Devuelve un valor identificador de código (hash) para el objeto.
public String toString()	Devuelve una representación de cadena del objeto.

Los métodos de ObjetoClass permitirán obtener información acerca de la clase de objeto referenciado por cualquier objeto. Por ejemplo, el método getName devuelve una cadena correspondiente al nombre de la clase; getMethods devuelve una matriz de la clase Method con los nombres de todos los métodos.

```
import java.io.*;
class ClaseDeUnObj {
  public static void main(String[] args){
    int n;
try {
      System.out.print("Dato: ");
      n = System.in.read(); // leer un carácter desde el teclado
      System.out.println((char)n); // visualizar el carácter
```

```
// Investigamos la instanciación de la clase
                    Class ObjetoClass; // objeto Class
                    ObjetoClass = System.in.getClass();
                    System.out.println("Clase de in:  " + ObjetoClass.getName());
                    ObjetoClass = System.out.getClass();
                    System.out.println("Clase de out: " + ObjetoClass.getName());
                    ObjetoClass = System.err.getClass();
                    System.out.println("Clase de err: " + ObjetoClass.getName());
            }
                    catch(IOException e){       System.err.println("Error: " + e.getMessage());    }
                  }
                 }
```

Ejemplo de método equals

```
1.   public class MyDate {
2.                 private int dia ;
3.                private int mes;
4.                private int anio;
5.
6.     public MyDate(int dia , int mes, int anio) {
7.       this.dia  = dia;
8.       this.mes = mes;
9.       this.anio  = anio;
10.    }
11.    public boolean equals(Object o) {
12.      boolean resultado = false;
13.      if ( (o != null) && (o instanceof MyDate) ) {
14.
15.  //           Aplicamos casting
16.      MyDate d = (MyDate) o;
17.        if ( (dia == d.dia) && (mes == d.mes)  && (anio == d.anio) ) {
18.          resultado = true;      }
19.      }
20.      return resultado;   }
21.
22.    public int hashCode() {
23.      return (day ^ month ^ year);
24.    }
25.  }
```

```
1.   Test de ejemplo de  igualdad
2.   public class TestEquals {
3.     public static void main(String[] args) {
4.
5.       MyDate  date1 = new MyDate(14, 3, 1976);    MyDate  date2 = new MyDate(14, 3, 1976);
6.
7.       if ( date1 == date2 ) {
8.         System.out.println("fecha 1 es idéntica a la fecha 2");
9.       } else {       System.out.println("fecha 1 no es idéntica a la fecha 2");    }
10.
11.      if ( date1.equals(date2) ) {
12.        System.out.println("fecha 1 es igual a la fecha 2");
```

```
13.     } else {      System.out.println("fecha 1 no es igual a la fecha 2");    }
14.
15.     System.out.println("set date2 = date1;");
16.     date2 = date1;
17.
18.     if ( date1 == date2 ) {
19.       System.out.println("fecha 1 es idéntica a la fecha 2");
20.     } else {       System.out.println("fecha 1 no es idéntica a la fecha 2");    }
21.   }
22. }
```

Procedimiento

Resuelva los siguientes problemas

1. Realice un programa que sobreescriba el método toString
2. Realice un programa que sobreescriba el método equals
3. Realice un programa que genere instancias de varias clases y que el programa permita al usuario mediante un menú la clase de la que se instanciaron las variables

Evaluación

Criterio	
1. Cumplió satisfactoriamente con el programa 1	SI NO
2. Cumplió satisfactoriamente con el programa 2	SI NO
3. Cumplió satisfactoriamente con el programa 3	SI NO
Puntaje Total_____	0 1 2 3 4

Unidad 3 Estructuras de Control

Ejercicio 9 Flujo de control; instrucciones condicionales y ciclos

U Competencia **3 Conceptos, estructuras, y objetos que permiten controlar el flujo de la POO**

Duración Estimada: 1 sesión de 50 minutos

Objetivo:

El alumno usará las estructuras de control de flujo del programa en programas en java

Introducción Teórica:

Para tener control del flujo del programa en java se implementaron muchas de las sentencias de control del flujo del programa del lenguaje C y por lo tanto funcionan de la misma forma, como resumen se define el funcionamiento y ejemplos de estas estructuras.

Las estructuras básicas de control son: secuenciales, condicionales y cíclicas en este apartado se describirán las estructuras condicionales y cíclicas

Sentencias Condicionales

```
if/else
        if( Boolean ) {
           sentencias;
           }
        else {
           sentencias;
        }
```

Ejemplo:
if ((diasemana>=1) && (diasemana<=5)**) { **trabajar = **true; }**
else { trabajar = **false; }**

operador ?
Este operador (conocido como **if** de una línea) permite ejecutar una instrucción u otra según el valor de la expresión. Sintaxis:

```
Expresionlogica ? valorSiVerdadero : valorSiFalso ;
```

Ejemplo: paga= (sueldo>1800)**?**6000**:**3000**;**

En este caso si la variable sueldo es mayor de 1800, la paga será de 6000, sino será de 3000.

Se evalúa una condición y según es cierta o no se devuelve un valor u otro. Nótese que esta función ha de devolver un valor y no una expresión correcta. Es decir, no funcionaría:

(sueldo>1800)? paga=6000: paga=3000; ***/ERROR!!!!***

Condicional múltiple

```
Switch
   switch( expr1 ) {
      case expr2:        sentencias;        break;
      case expr3:        sentencias;        break;
      default:        sentencias;        break;
  }
```

Sentencias Iterativas

Ciclo For Sintaxis:

```
   for( expr1 inicio; expr2 test; expr3 incremento ) {
      sentencias;
      }
```

El siguiente trocito de código Java que dibuja varias líneas en pantalla alternando sus colores entre rojo, azul y verde. Este fragmento sería parte de una función Java (método):

```
int contador;
for( contador=1; contador <= 12; contador++ ) {
   switch (contador % 3 ) {
      case 0:  setColor ( Color.red );          break;
      case 1:  setColor ( Color.blue );          break;
      case 2:  setColor ( Color.green );          break;
      }
   g.drawLine( 10,contador*10,80,contador*10 );
   }
```

En los bucles for También se soporta el operador coma (,)

```
for( a=0,b=0;a<7; a++,b+=2) //se inicializa la variable a y b ; a se incrementa de 1 en 1 y b de 2 en 2
```

Ciclo while

```
while( Boolean ) {
      sentencias;
      }
```

Ciclo do/while

```
do {
   sentencias;
   }while( Boolean );
```

Control General del Flujo

```
break [etiqueta]

continue [etiqueta]

return expr;

etiqueta: sentencia;
```

En caso de que nos encontremos con bucles anidados, se permite el uso de etiquetas para poder salirse de ellos, por ejemplo:

```
uno: for( )   {
   dos: for( )   {
        continue;          // seguiría en el bucle interno
        continue uno;      // seguiría en el bucle principal
        break uno;         // se saldría del bucle principal
        }
}
```

En el código de una función siempre hay que ser consecuentes con la declaración que se haya hecho de ella. Por ejemplo, si se declara una función para que devuelva un entero, es imprescindible que se coloque un return final para salir de esa función, independientemente de que haya otros en medio del código que también provoquen la salida de la función. En caso de no hacerlo se generará un Warning, y el código Java no se puede compilar con Warnings.

```
int func()    {
if( a == 0 )       return 1;
return 0;    // es imprescindible porque se debe retornar un entero
}
```

Ejemplo:

En el siguiente ejemplo se muestra un programa en el que se usan las estructuras de control condicionales y cíclicas

```
public class TestVehiculo {
   static Object registro[]= new Object[10];  //Contenedor para 10 registros
   static Scanner leer = new Scanner (System.in);  //objeto para leer de teclado

   public static void main(String[] args) { //Método Principal almacena registros según elección del usuario
```

```
int sel, ej=0;
do{
   sel=menu();
   switch(sel){
      case 1:registro[ej]=leerProp(); break;
      case 2:registro[ej]=leerVehiculo(); break;
      case 3:mostrarReg();break;
      case 4:System.out.println("Programa Terminado");break;
      default: System.out.println("Opción inválida");
   }
   ej++;
}while(sel!=4 && ej<10);       }
```

```
public static int menu(){  // obtiene la selección del usuario
   int op;
   System.out.println("Indique su opción \n 1: "
         + "Registrar Propietario \n 2: Registrar Vehículo \n "
         + "3: Mostrar registros \n 4: Salir \n");
        op=leer.nextInt();
   return op;
}
```

```
public static void mostrarReg(){  //método que muestra los registros almacenados en el arreglo
   int i=0;
   while (registro[i]!=null){
      System.out.println("****** Registro número "+ i +"************");
      if (registro[i]instanceof Propietario){ // si el registro es un Propietario
         Propietario due=(Propietario)registro[i]; // casteamos el objeto a Propietario
        System.out.println("Dueño del auto"+due.nom); //mostramos nombre
        System.out.println("telefono: "+due.telefono); //mostramos teléfono
        System.out.println("dirección:"+due.dir);  //mostramos dirección
        System.out.println("fecha de registro:"+due.dreg); //mostramos fecha de registro
      }
      if (registro[i]instanceof Vehiculo){  // si el registro es un Vehículo
        Vehiculo auto=(Vehiculo)registro[i]; // casteamos el objeto a Vehículo
        System.out.println("Dueño del auto"+auto.prop.nom); //mostramos nombre de propietario
        System.out.println("Marca: "+auto.getMarca()); //mostramos marca de auto
        System.out.println("Origen:"+auto.getOrigen());  //mostramos dirección
        System.out.println("Numero de pasajeros:"+auto.getNumpas()); //mostramos fecha de registro
      }
      i++;      }    }
```

```
public static Propietario leerProp(){ // método que lee datos de propietario
   System.out.println("Introduzca los datos que se le piden");
   System.out.println("Dueño del auto");
   String nom=leer.nextLine();
```

```
System.out.println("Teléfono: ");String ntel=leer.nextLine();
System.out.println("Dirección:");String dir=leer.nextLine();
Propietario nuevo= new Propietario(nom,ntel,dir, new Date());
return nuevo;   }

public static Vehiculo leerVehiculo(){  //Método que lee datos de Vehículo
 System.out.println("Introduzca los datos que se le piden");
System.out.println("Origen del auto");
String org=leer.nextLine();
System.out.println("Marca del auto"); String mar=leer.next();
System.out.println("Numero de pasajeros"); int  npas=leer.nextInt();
//Datos de un propietario a modificar
Propietario nuevo= new Propietario("a determinar","000000","debe actualizar", new Date());
Vehiculo reg= new Vehiculo(org,mar,nuevo,npas);
return reg;
 }
```

Procedimiento

Con base en el código anterior, codifique en un entorno de desarrollo y describa el significado de cada línea de instrucción y haga un seguimiento de valores en las variables (prueba de escritorio). Verifique que ha comprendido el uso de cada estructura de control. Resuelva los siguientes problemas con la codificación de una o más clases en Java:

1. Genere dos arreglos con 15 números aleatorios, después genere un tercer Vector con la suma del contenido de los dos arreglos anteriores
2. Genere un programa en Java que le permita almacenar los datos personales de 10 empleados de una empresa apóyese en un arreglo
3. Genere una aplicación que permita conocer la información del problema anterior mediante un Menú que permita seleccionar entre modificar datos, ver datos , y borrar datos
4. Genere un arreglo de alumnos con sus respectivas calificaciones, genere la aplicación con los métodos necesarios para clasificar a los alumnos en tres arreglos diferentes de acuerdo a la calificación si es menor a 6, si es mayor a 6 y menor a 8 y si es mayor a 8 y menor a 10. El usuario podrá conocer los nombres de los alumnos en las diferentes categorías

Evaluación:

Criterio	
1. Cumplió con la codificación correcta del programa	SI NO
2. Cumplió la entrega de la prueba de escritorio	SI NO
3. Cumplió con la codificación de las clases necesarias	SI NO
4. Cumplió los cuatro problemas	SI NO SI NO
Puntaje Total_____	

Ejercicio 11 Flujo de control; Excepciones

U Competencia **3 Conceptos, estructuras, y objetos que permiten controlar el flujo de la POO**

Duración Estimada: 1 sesión de 50 minutos

Objetivo:

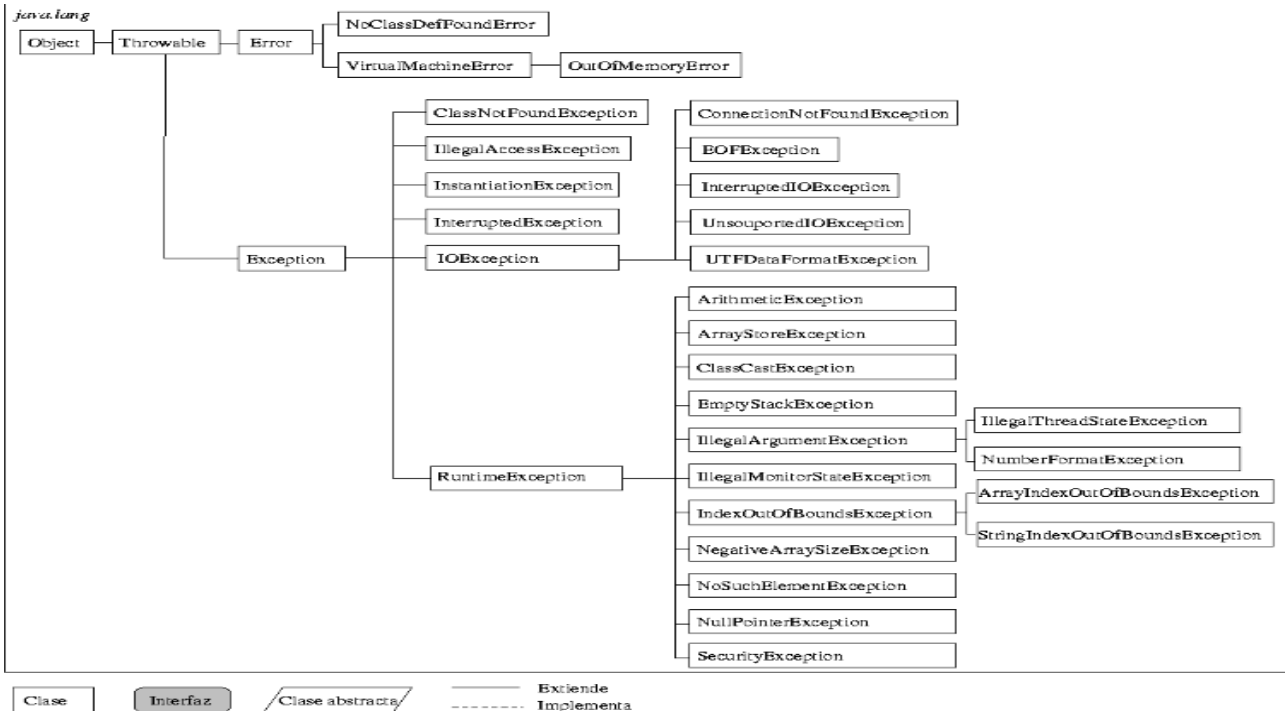
El alumno usará el manejo de excepciones

Introducción Teórica:

Excepciones estándar java

La clase Throwable de Java describe todo aquello que se pueda lanzar en forma de excepción. Hay dos tipos generales de objetos Throwable ("tipos de" = "heredados de"). Error representa los errores de sistema y de tiempo de compilación que uno no se preocupa de capturar (excepto en casos especiales). Exception es el tipo básico que puede lanzarse desde cualquier método de las clases de la biblioteca estándar de Java y desde los métodos que uno elabore, además de incidencias en tiempo de ejecución. Por tanto, el tipo base que más interesa al programador es Exception.

En Java hay muchos tipos de excepciones (de operaciones de entrada y salida, de operaciones irreales. El paquete java.lang.Exception y sus subpaquetes contienen todos los tipos de excepciones. La jerarquía de excepciones se observará en la siguiente imagen



Cuando se produce un error se genera un objeto asociado a esa excepción. Este objeto es de la clase Exception o de alguna de sus herederas. Este objeto se pasa al código que se ha definido para manejar la excepción. Dicho código puede manipular las propiedades del objeto Exception.

Hay una clase, la java.lang.Error y sus subclases que sirven para definir los errores irrecuperables más serios. Esos errores causan parada en el programa, por lo que el programador no hace falta que los manipule. Estos errores los produce el sistema y son incontrolables para el programador. Las excepciones son fallos más leves, y más manipulables

La idea básica es que el nombre de la excepción representa el problema que ha sucedido; de hecho se pretende que el nombre de las excepciones sea autoexplicativo. Las excepciones no están todas ellas definidas en java.lang; algunas están creadas para dar soporte a otras bibliotecas como util, net e io. Así, por ejemplo, todas las excepciones de E/S se heredan de java.io.IOException.

Las sentencias que tratan las excepciones son try y catch. La sintaxis es:

```
try {
    sentencias;
} catch( Exception ) {      sentencias;    }
```

Puede haber más de una sentencia catch para un mismo bloque try. Ejemplo:

```
try {
readFromFile("arch");
...
}
catch(FileNotFoundException e) {
//archivo no encontrado
...
}
catch (IOException e) {
...
}
```

Java implementa excepciones para facilitar la construcción de código robusto. Cuando ocurre un error en un programa, el código que encuentra el error lanza una excepción, que se puede capturar y recuperarse de ella. Java proporciona muchas excepciones predefinidas.

Manejo de excepciones

Siempre se debe controlar una excepción, de otra forma nuestro software está a merced de los fallos. En la programación siempre ha habido dos formas de manejar la excepción:

• Interrupción. En este caso se asume que el programa ha encontrado un error irrecuperable. La operación que dio lugar a la excepción se anula y se entiende que no hay manera de regresar al código que provocó la excepción. Es decir, la operación que dio pies al error, se anula.

• Reanudación. Se puede manejar el error y regresar de nuevo al código que provocó el error.

La filosofía de Java es del tipo interrupción, pero se puede intentar emular la reanudación encerrando el bloque try en un while que se repetirá hasta que el error deje de existir. Ejemplo:

```
boolean indiceNoValido=true;
int i; //Entero que tomará nos aleatorios de 0 a 9

String texto[]={“Uno”,”Dos”,”Tres”,”Cuatro”,”Cinco”};
while(indiceNoValido){
   try{
          i=Math.round(Math.random()*9);
System.out.println(texto[i];
indiceNoValido=false;
      }catch(ArrayIndexOutOfBoundsException exc){
      System.out.println(“Fallo en el índice”);
} //fin de catch
} // fin de while
```

En el código anterior, el índice i calcula un número del 0 al 9 y con ese número el código accede al array texto que sólo contiene 5 elementos. Esto producirá muy a menudo una excepción del tipo ArrayIndexOutOfBoundsException que es manejada por el catch

throws

Al llamar a métodos, ocurre un problema con las excepciones. El problema es, si el método da lugar a una excepción, ¿quién la maneja? ¿El propio método?¿O el código que hizo la llamada al método?

Con lo visto hasta ahora, sería el propio método quien se encargara de sus excepciones, pero esto complica el código. Por eso otra posibilidad es hacer que la excepción la maneje el código que hizo la llamada.

Esto se hace añadiendo la palabra throws tras la primera línea de un método. Tras esa palabra se indica qué excepciones puede provocar el código del método. Si ocurre una excepción en el método, el código abandona ese método y regresa al código desde el que se llamó al método. Allí se posará en el catch apropiado para esa excepción.

Ejemplo: void usarArchivo (String archivo) throws IOException, InterruptedException {...

En este caso se está indicando que el método usarArchivo puede provocar excepciones del tipo IOException y InterruptedException. Esto significará, además, que el que utilice este método debe preparar el catch correspondiente para manejar los posibles errores.

finally

La cláusula finally está pensada para limpiar el código en caso de excepción. Su uso es:

```
try{
...
}catch (FileNotFoundException fnfe){
...
}catch(IOException ioe){
...
}catch(Exception e){
...
}finally{  ...//Instrucciones de limpieza   }
```

Las sentencias finally se ejecutan tras haberse ejecutado el catch correspondiente. Si ningún catch capturó la excepción, entonces se ejecutarán esas sentencias antes de devolver el control al siguiente nivel o antes de romperse la ejecución. Hay que tener muy en cuenta que las sentencias finally se ejecutan independientemente de si hubo o no excepción. Es decir esas sentencias se ejecutan siempre, haya o no excepción. Son sentencias a ejecutarse en todo momento. Por ello se coloca en el bloque finally código común para todas las excepciones (y también para cuando no hay excepciones

Procedimiento:

Explique el código de las dos clases que se muestran a continuación y codifique lo necesario para que se ejecuten (mostrar al profesor el código generado y una corrida de escritorio)

```
1.  import java.io.*;
2.  public class Test{
3.  public static void main(String args[]){
4.  // Definir un flujo de caracteres de entrada: flujoE
5.  InputStreamReader isr = new InputStreamReader(System.in);
6.  BufferedReader flujoE = new BufferedReader(isr);
7.  // Definir una referencia al flujo estándar de salida: flujoS
8.  PrintStream flujoS = System.out;
9.  String sdato;     float precio = 0.0F;
10. try{
11. flujoS.print("Precio: ");
12. sdato = flujoE.readLine();
13. precio = (sdato != null) ? (new Float(sdato)).floatValue()   : Float.NaN;
14. }     catch (IOException ignorada){ }
15. flujoS.println(precio);
16. flujoS.println("Continua la aplicación");      }
17. }
```

```
1. package areas;
2. import java.io.*;
3. import java.util.*;
4. public class Areas {
5. public static void main(String[] args) {
6. try{
```

```
7.  Vector figuras= new Vector();
8.  BufferedReader ent= new BufferedReader(new InputStreamReader(System.in));
9.  String cadBase,cadAlt;
10.System.out.println("Dime la base de tu rectángulo");
11.cadBase=ent.readLine();
12.System.out.println("Dime la altura de tu rectángulo");
13.cadAlt=ent.readLine();
14.float b,a;
15.b=Float.parseFloat(cadBase);    a=Float.parseFloat(cadAlt);
16.Rectangulo r2= new Rectangulo(b,a);
17.figuras.addElement(r2);
18.Rectangulo r1= new Rectangulo();
19.figuras.addElement(r1);
20.Circulo c1= new Circulo(78);
21.c1.imprime();
22.figuras.addElement(c1);
23.Object n[] = figuras.toArray();
24.for(int i=0;i<n.length;i++){
25.if (n[i] instanceof Rectangulo)
26.{ Rectangulo ob= (Rectangulo)n[i];
27.System.out.println("Objeto Rectángulo "+i+" tiene :" +   ob.alt +" de alto y " +ob.base);
28.}
29.if (n[i] instanceof Circulo)  { Circulo ob= (Circulo)n[i];
30.System.out.println("Objeto Circulo"+i+"  :" ); ob.imprime();    }
31.}
32.}  //fin de try
33.catch( Exception e){  System.out.println("Se presentó la exception: "+ e.getMessage());    }
34.}
35.}
```

Evaluación:

Criterio	
1. Cumplió con la codificación correcta de los ejemplos	SI NO
2. Cumplió con la explicación correcta de las clases necesarias	SI NO
3. Cumplió con la ejecución de la prueba de escritorio	SI NO
Puntaje Total_____	0 1 2 3

Unidad 4 Clases para estructuras de datos

Ejercicio 12 Clases utilitarias

U Competencia **4 Conocer las estructuras que permiten el manejo de datos en un lenguaje OO**

Duración Estimada: 1 sesión de 50 minutos

Objetivo:

El alumno usará las clases del paquete útil para agregar funcionalidad a sus programas

Introducción teórica:

En el paquete java.util y java.lang se almacenan múltiples Interfaces y clases que son muy útiles en la codificación de programas; en java.util las principales interfaces son: Collection, Comparator, Iterator, List, Map, Set, SortedSet, SortedMap.

Las clases más usadas en el paquete útil son: Calendar, Date, HashMap, HashSet, LikendList, StringTokenizer, TreeMap, TreeSet.

En esta sección se describirán algunas clases de este paquete:

Calendar

Es una clase abstracta que nos obliga a implementar diversos métodos para crear un calendario, pero normalmente se utiliza la subclase de esta GregorianCalendar para utilizar un calendario gregoriano, que es el calendario de uso más habitual en la mayoría de países.

Date

Es una clase que permite representar un instante específico en el tiempo con precisión de milisegundos. En las versiones más recientes de Java muchos de los métodos de esta clase están "deprecated", es decir, obsoletos y de uso no recomendado. Para reemplazar funcionalidades de esta clase aparece el paquete java.time. Este paquete proporciona funcionalidades sobre fechas y calendarios, y es el que debemos usar para el manejo de fechas y tiempo. ¿Por qué se menciona aquí la clase Date? Porque ha sido una clase muy usada en el pasado y nos podemos encontrar con código que usa versiones anteriores o que sigue usando la clase Date del paquete java.util. Algunos de los métodos de la clase Date son los siguientes:

Método	Descripción
Date()	Inicializa el objeto con la fecha y hora actual
Date(año,mes,día)	Establecerá la hora a las 00:00:00 (medianoche) del día especificado.
Date(año,mes,día,horas,minutos)	Establecerá la fecha y hora, dejando los segundos a 0.
Date(año,mes,día,horas,minutos,segundos)	Establecerá la hora exacta.

Date(String)	Este constructor de Date, analizará una representación tipo cadena con la fecha y hora y la convertirá en objeto.
Date("Fri Oct 3 21:23:45 GMT 1997")	Date convierte muchos formatos de fecha tipo cadena, pero debería seguirse la sintaxis de fecha de tipo:
Date(long)	Establece la hora exacta a partir del número de milisegundos que han pasado desde el 1 de Enero de 1970, a las 00:00:00 GMT.
getTime()	Devuelve el número de milisegundos transcurrido desde el 1 de Enero de 1970, a las 00:00:00 GMT.
setTime(long)	Fija la fecha a partir del número de milisegundos transcurrido desde el 1 de Enero de 1970, a las 00:00:00 GMT
before(Date)	Comprueba si una fecha es anterior a la especificada.
after(Date)	Comprueba si una fecha es posterior a la especificada.
equals(Object)	Compara dos fechas. El resultado es true, si y sólo si el argumento no es nulo y los objetos coinciden a nivel de milisegundos
toString()	Crea la representación canónica de la fecha, de la forma "Fri Oct 3 21:23:45 GMT 1997“
getDate()	Devuelve la fecha actual

Math

Math es una clase del paquete java.lang tiene métodos que devuelven los resultados de operaciones matemáticas necesarias en cálculos económicos o científicos, los métodos de la clase Math son:

Método	Descripción
Static double E	Valor numérico e
Double PI	Valor numérico de pi
Double ceil (double a)	Valor más pequeño mayor a “a” sin decimales
Tipo abs(tip a)	Valor absoluto de a.
Tipo max(tipo a, tipo b)	Valor mayor de a y b. El tipo es igual
Double floor(double a)	Valor más grande menor a “a” sin decimales
Tipo min(tipo a, tipo b)	Valor menor de a y b. El tipo es igual
Double random()	Valor aleatorio entre 0.0 y 1.0
Double rint(double a)	Valor double más cercano a “a” sin decimales
Long round(double a)	Valor long más cercano a “a”.
Int round (float a)	Valor int más cercano a “a”.
Double sqrt(double a)	Raíz cuadrada de “a”, a>=0.
Double exp(double a)	Valor de a
Double log(double a)	Logaritmo a
Double ln(a).	Logaritmo natural de a
Double pow(double a, double b)	Potencia b de a

Double IEEEremainder (Double f1, double f2)	Resto de una división entre números) reales. C=f1/f2, siendo c el valor entero más cercano al valor real f1/f2.
Double acos(double a)	Arco, de 0.0 a pi, cuyo coseno es a
Double asin(double a)	Arco, de –pi/2 a pi/2, cuyo seno es a
Double atan(double a)	Arco, de –pi/2 a pi/2, cuya tangente es a
Double atan(double a, Double b)	Convierte las coordenadas rectangulares (b,a) a polares (r,angulo)
Double sin(double a)	seno de a radianes
Double cos(double a)	coseno de a radianes
Double tan(double a)	tangente de a radianes
Double toDegrees(double Rads)	Convierte radianes a grados
Double toRadians(doube Grados)	Convierte de grados a radianes

String

La clase String se almacena en el paquete java.lang tiene los métodos necesarios para realizar operaciones en cadenas de caracteres como concatenar cadenas, quitar espacios innecesarios, dividir cadenas en subcadenas, comparar cadenas, conocer la letra en determinada posición o buscar un carácter. Los objetos de la clase String tienen una buena cantidad de métodos para realizar muchas cosas interesantes.

Método	Descripción
indexOf (carácter, desde)	Devuelve la posición de la primera vez que aparece el carácter indicado por parámetro en un String. Si no encuentra el carácter en el String devuelve -1. El segundo parámetro es opcional y sirve para indicar a partir de qué posición se desea que empiece la búsqueda.
charAt (indice)	Devuelve el carácter que hay en la posición indicada como índice. Las posiciones de un string empiezan en 0.
lastIndexOf (carácter, desde)	Busca la posición de un carácter exactamente igual a como lo hace la función indexOf pero desde el final en lugar del principio.
replace(substring_a_buscar, nuevoStr)	El método no reemplaza en el String, sino que devuelve un resultante de hacer ese reemplazo. Acepta expresiones regulares como substring a buscar.
toUpperCase()	Pone todos los caracteres de un String en mayúsculas.
toString()	Este método lo tienen todos los objetos y se usa para convertirlos en cadenas.

Iterator

Nos define los métodos que tenemos que implementar para hacer un iterador sobre una colección de elementos. Esto en parte fue visto anteriormente cuando vimos la implementación de la interfaz del paquete java.lang Iterable. Este nos obligaba a tener un método llamado iterator(), que nos devolvía un objeto de la interfaz Iterator.

StringTokenizer

Es una de las clases más utilizadas dentro del paquete java.util. Esta clase permite partir un String en tokens (un token es una porción de un String más grande al cual le decimos por qué carácter debemos de partirlo. Por ejemplo si tenemos el String “Un coche rápido y rojo” y deseamos obtener los tokens separados por un espacio en blanco, tendríamos como resultado 5 tokens que son: “Un”, “coche”, “rápido”, “y”, “rojo”).

Ejemplo de String

```
1.  public class Comparaciones {
2.  public static void main( String[] args ) {
3.  int resultado;
4.  int num, num2;
5.  num=3;
6.  num2=3;
7.  String cadena1 = new String("Hola");
8.  String cadena2 = new String("Hola");
9.  String cadena3 =new String("hola");
10. //las cadenas tienen el mismo valor pero se generaron con dos new distintos
11. //por lo tanto tienen diferente referencia en memoria y el operador ==
12. if (cadena1 == cadena2) {System.out.println("Somos iguales");}
13. else {System.out.println("no somos iguales");}
14. System.out.println("_____");
15. //los valores son iguales
16. if (num == num2) {System.out.println("Somos iguales");}
17. else {System.out.println("no somos iguales");}
18. System.out.println("_____");
19. if (cadena1.equals(cadena3)==true) {System.out.println("Somos iguales");  }
20. else {System.out.println("no somos iguales");}
21. System.out.println("_____");
22. resultado = cadena1.compareTo(cadena2);
23. if (resultado == 0) {System.out.println("Somos iguales");}
24. if (resultado < 0){System.out.println("debo colocarme antes de..");    }
25. if (resultado > 0){System.out.println("debo colocarme después de..");  }
26. else {System.out.println(resultado);}
27. } }
```

```
1. import java.util.*;
2. public class TPAviso implements TareaPeriodica {
3. int periodoSegs;
4. Date ultimaEj;
5. boolean activa;
```

```
6.  String msg;
7.
8.  public TPAviso(String aMsg, int aPeriodoSegs) {
9.  periodoSegs = aPeriodoSegs;
10. ultimaEj = new Date ();
11. activa = true;
12. msg = aMsg;   }
13.
14. public boolean necesitaEjecucion () {
15. if (!activa)      return false;
16. Calendar calProximaEj = new GregorianCalendar ();
17. calProximaEj.setTime (ultimaEj);
18. calProximaEj.add (Calendar.SECOND, periodoSegs);
19. Calendar calAhora = new GregorianCalendar ();
20. return (calProximaEj.before (calAhora));
21. }
22. public void activar () { activa = true; }
23. public void desactivar () { activa = false; }
24. public void ejecutarTarea () {
25. System.out.println ("ATENCIÓN AVISO: " + msg);    desactivar ();
26. }
27. }
```

Ejemplo Date

```
1.  import java.text.*;
2.  import java.util.*;
3.  class Fecha{
4.  public static void main(String[] args){
5.              Date hoy=new Date();
6.              String sFecha,sHora;
7.              DateFormat formato;
8.              formato=DateFormat.getDateInstance();
9.              sFecha=formato.format(hoy);
10.             formato=DateFormat.getTimeInstance();
11.             sHora=formato.format(hoy);
12.             System.out.println("Fecha:"+sFecha);
13.             System.out.println("Hora:"+sHora);        }
14. }
```

```
1.      import java.util.*;
2.      class CStoken{
3.              public static void main(String[] args){
4.                      StringTokenizer cadena;
5.                      cadena=new StringTokenizer("uno, dos, tres y cuatro");
6.                      while(cadena.hasMoreTokens())
7.                              System.out.println(cadena.nextToken());    }   }
```

Procedimiento:

Realice un programa que permita mostrar un menú que permita al usuario seleccionar entre las siguientes opciones:

1. Solicitar tres fechas y mostrar la diferencia en días de cada una de las fechas, muestra la diferencia entre los años ordenando de menor a mayor las fechas
2. Mostrar las funciones trigonométricas de un ángulo solicitado en grados
3. Solicitar tres cadenas al usuario y mostrar los datos de tamaño de las tres cadenas, mostrar las cadenas ordenadas de menor a mayor, separar las cadenas en tokens.
4. Realice un programa que reciba mediante el teclado varias cadenas del teclado y con ellas demuestre los resultados de al menos 15 métodos de la clase String
5. Realice un programa que ejemplifique al menos 6 métodos de la clase Date
6. Realice un programa que muestre varias opciones sobre cálculos matemáticos y que se muestre el resultado de usar al menos 10 métodos de la clase Math y Random
7. Genere un programa que simule una jugada de dados con dos usuarios, el programa debe indicar las veces que gano el jugador 1 el jugador 2 y el número de empates

Evaluación:

Criterio	
1. Cumplió con la codificación correcta de los ejemplos	SI NO
2. Cumplió con la codificación correcta de las clases necesarias	SI NO
3. Cumplió con la entrega de la prueba de escritorio	SI NO
Puntaje Total_____	0 1 2 3

Ejercicio 13 Arreglos en JAVA

U Competencia **4 Conocer las estructuras que permiten el manejo de datos en un lenguaje OO**

Duración Estimada: 1 sesión de 50 minutos

Objetivo:

El alumno usará arreglos de tipos primitivos o de objetos

Introducción Teórica:

Un arreglo es una estructura de datos homogéneo, la forma de declarar un arreglo en java es:
TipoDeDato [ ] nombre; (todas las variables serán arreglos del tipo específico) O
TipoDeDato nombre [ ]; (Sólo la variable nombre es un arreglo)

Ejemplo:

```
int [ ] m;
float [ ] temp;
COrdenador [] ordenador; //Siendo COrdenador una clase.
```

Observe que la declaración no especifica el tamaño del arreglo. El tamaño se especificará cuando se crea el arreglo durante la ejecución del programa. Crear el arreglo significa reservar el espacio necesario de memoria.

Ejemplo: int nombre=new int [10];

La declaración y la creación se pueden hacer en una misma linea.

Ejemplo: int m[]=new int[10]; o int [] m=new int [10];

Como se observará, el tamaño de la matriz se especifica en tiempo de ejecución y no necesariamente en tiempo de compilación como sucede en un lenguaje estático (como pascal) o semiestático (como c++).

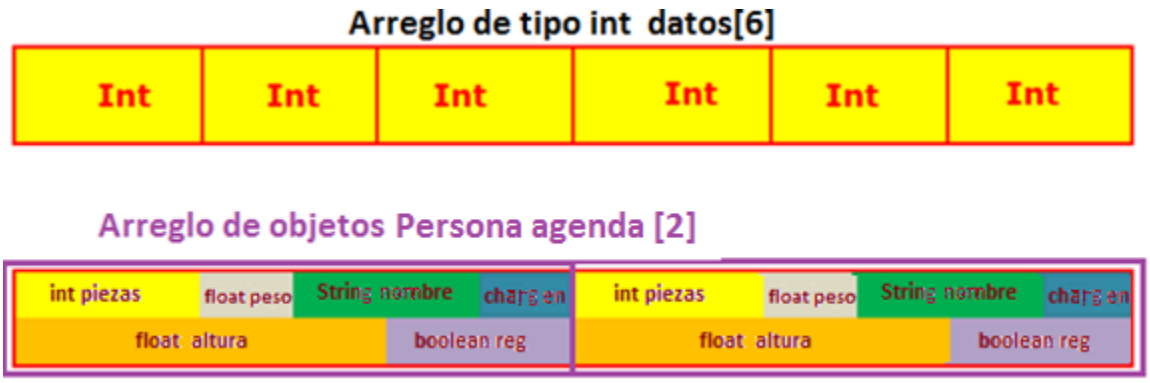
Ejemplo:

```
int nElementos;
Sistem.out.print("# de elementos del vector:");
nElementos=leer.datoInt();
int [] m=new int [nElementos];
```

Por lo que el arreglo es un objeto y como tal, cuando se crea, sus elementos son automáticamente inicializados. Cuando el arreglo es numérico, sus elementos con cero, si no son numéricos, con valores análogos al cero. Por ejemplo, los caracteres se inicializan con '\u000', un booleano con false y una referencia a objeto con null. Si se desea iniciar con valores diferentes se pede realizar lo siguiente:

```
float temp[]= {10.5f,12.6f,3.4f};
```

La siguiente imagen muestra un ejemplo de cómo se visualizarían dos arreglos uno de tipo entero con seis elementos y otro arreglo de tipo Persona con dos objetos



Si se intenta acceder a un elemento con subíndice menor que cero o mayor que el # de elementos del arreglo menos uno, java lanzará una excepción del tipo ArrayIndexOutBoundException.

Para saber el tamaño máximo de la matriz se puede realizar lo siguiente:

```
int n=m.length;
```

Métodos de un arreglo

La clase genérica “Array” proporciona un conjunto de métodos que se ha heredado de la clase Object del paquete java.lang. Entre ellos se encuentran.

```
equals   (bolean equals (Object obj))     y     clone (oject clone());
```

Equals verifica si dos referencias se refieren al mismo objeto y el segundo duplica a un objeto.

Ejemplo:

```
int [] m1={10, 20, 30, 40};
int m2[]=(int{})m1.clone();
if (m1.equals(m2)) {System.out.println(“m1 y m2 se refieren a la misma matriz”);}
else {System.out.println(“m1 y m2 se refieren a matrices diferentes”);   }
```

En este caso la evaluación dentro del if es falsa ya que m1 y m2 hacen referencia a dos matrices diferentes. Se mostrarán algunos ejemplos sencillos de aplicaciones con matrices.

Ejemplo: Se obtiene el promedio de calificaciones de “n” alumnos.

```
public class CMatrizUnidimensional    {     // Trabajar con una matriz unidimensional
 public static void main(String[] args)    {
  int nAlumnos;    // número de alumnos (valor no negativo)
  do    {
   System.out.print("Número de alumnos: ");
   nAlumnos = Leer.datoInt();
  }     while (nAlumnos < 1);

  float[] nota = new float[nAlumnos]; // crear la matriz nota
  int i = 0;       // subíndice
  float suma = 0F; // suma total de las notas medias
  System.out.println("Introducir las notas medias del curso.");
  for (i = 0; i < nota.length; i++)    {
   System.out.print("Nota media del alumno " + (i+1) + ": ");
   nota[i] = Leer.datoFloat();    }
  // Sumar las notas medias
  for (i = 0; i < nota.length; i++)      suma += nota[i];
  // Visualizar la nota media del curso
  System.out.println("\n\nNota media del curso: " + suma / nAlumnos);    }     }
```

Ejemplo de lectura y escritura de una matriz unidimensional:

```
public class CMatrizUnidimensional{
 // Creación de una matriz unidimensional
 public static void main(String[] args)
 {
  int nElementos;
  System.out.print("Número de elementos de la matriz: ");
  nElementos = Leer.datoInt();
  int[] m = new int[nElementos]; // crear la matriz m
  int i = 0; // subíndice

  System.out.println("Introducir los valores de la matriz.");
  for (i = 0; i < nElementos; i++)
  {
   System.out.print("m[" + i + "] = ");
   m[i] = Leer.datoInt();
  }

  // Visualizar los elementos de la matriz
  System.out.println();
  for (i = 0; i < nElementos; i++)
```

```
        System.out.print(m[i] + " ");
        System.out.println("\n\nFin del proceso.");
    }
}
```

Ejemplo. Lee “n” elementos numéricos e indica cual es el menor y cual es el mayor.

```
public class CValoresMaxMin{
  // Obtener el máximo y el mínimo de un conjunto de valores
  public static void main(String[] args){
    int nElementos;    // número de elementos (valor no negativo)
    do{
      System.out.print("Número de valores que desea introducir: ");
      nElementos = Leer.datoInt();
    }
    while (nElementos < 1);
    float[] dato = new float[nElementos]; // crear la matriz dato
    int i = 0;       // subíndice
    float max, min;  // valor máximo y valor mínimo
    System.out.println("Introducir los valores.\n" +  "Para finalizar pulse [Entrar]");
    for (i = 0; i < dato.length; i++){
      System.out.print("dato[" + i + "]= ");
      dato[i] = Leer.datoFloat();
      if (Float.isNaN(dato[i])) break; // salir del bucle
    }
    nElementos = i; // número de valores leídos
    // Obtener los valores máximo y mínimo
    if (nElementos > 0){
      max = min = dato[0];
      for (i = 0; i < nElementos; i++){
        if (dato[i] > max)
          max = dato[i];
        if (dato[i] < min)
          min = dato[i];
      }
      // Escribir los resultados
      System.out.println("\nValor máximo: " + max);
      System.out.println("Valor mínimo: " + min);
    }
    else
      System.out.println("\nNo hay datos.");
  }
}
```

Ejemplo. Muestra la frecuencia en el que se dan los caracteres “a”-“z” en un texto dado. No se incluye la “ñ” y la “ll”.

```
                    import java.io.*;
                    public class CMatrizAsociativa{
// Frecuencia con la que aparecen las letras en un texto.
                     public static void main(String[] args){
// Crear la matriz c con 'z'-'a'+1 elementos.   Java inicia los elementos de la matriz a cero.
                      int[] c = new int['z'-'a'+1];
                      char car; // subíndice
                      final char eof = (char)-1;
// Entrada de datos y cálculo de la tabla de frecuencias
                      System.out.println("Introducir un texto.");
                      System.out.println("Para finalizar pulsar [Ctrl][z]\n");
                      try  {
// Leer el siguiente carácter del texto y contabilizarlo
                        while ((car = (char)System.in.read()) != eof){
 // Si el carácter leído está entre la 'a' y la 'z'       incrementar el contador correspondiente
                          if (car >= 'a' && car <= 'z')
                            c[car - 'a']++;     }    }
                      catch (IOException ignorada) {}
// Mostrar la tabla de frecuencias
                      System.out.println("\n");
 // Visualizar una cabecera "a b c ... "
                      for (car = 'a'; car <= 'z'; car++)
                      System.out.print(" " + car);
                      System.out.println("\n ----------------------------------");
// Visualizar la frecuencia con la que han aparecido los caracteres
                      for (car = 'a'; car <= 'z'; car++)
                        System.out.print(" " + c[car - 'a']);
                      System.out.println();    }          }
```

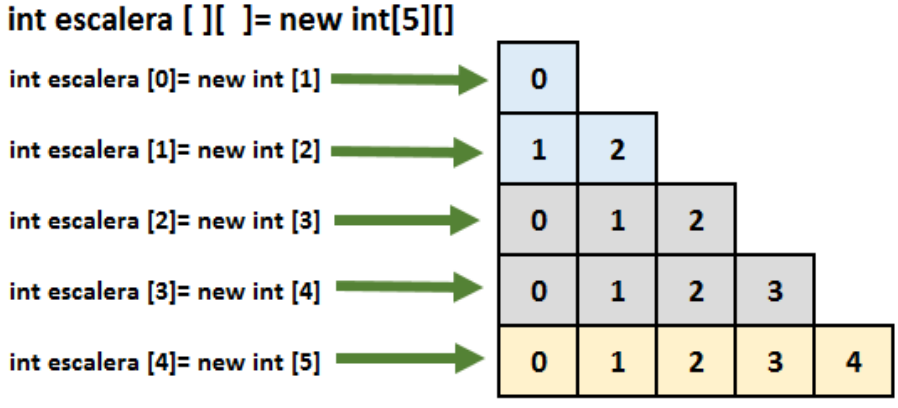
Ejemplo. Suma de filas de una matriz:

```
                    public class CMatrizMultidimensional{
                     // Creación de una matriz multidimensional.
                     // Suma de las filas de una matriz de dos dimensiones.
                     public static void main(String[] args){
                      int nfilas, ncols;     // filas y columnas de la matriz
                      do{
                       System.out.print("Número de filas de la matriz:    ");
                       nfilas = Leer.datoInt();
                      }
                      while (nfilas < 1);    // no permitir un valor negativo
                      do{
                       System.out.print("Número de columnas de la matriz: ");
```

```
    ncols = Leer.datoInt();
  }while (ncols < 1);     // no permitir un valor negativo
  float[][] m = new float[nfilas][ncols]; // crear la matriz m
  int fila = 0, col = 0; // subíndices
  float sumafila = 0;    // suma de los elementos de una fila
  System.out.println("Introducir los valores de la matriz.");
  for (fila = 0; fila < nfilas; fila++){
    for (col = 0; col < ncols; col++){
      System.out.print("m[" + fila + "][" + col + "] = ");
      m[fila][col] = Leer.datoFloat();     }    }
 // Visualizar la suma de cada fila de la matriz
  System.out.println();
  for (fila = 0; fila < nfilas; fila++){
    sumafila = 0;
    for (col = 0; col < ncols; col++)
      sumafila += m[fila][col];
    System.out.println("Suma de la fila " + fila + ": " + sumafila);
  }
  System.out.println("\nFin del proceso.");
 }      }
```

Ejemplo Arreglos

```
1.  public class Escalera {
2.  public static void main(String[] args) { int escalera[ ] [ ]=new int[5][];
3.  escalera[0]= new int[1];
4.  escalera[1]= new int[2];
5.  escalera[2]= new int[3];
6.  escalera[3]= new int[4];
7.  escalera[4]= new int[5];
8.  for (int i=0; i<escalera.length;i++)
9.  for (int j=0;j<escalera[i].length;j++)    escalera[i][j]=i+j;
10. for (int i=0; i<escalera.length;i++){
11. System.out.println();
12. for (int j=0;j<escalera[i].length;j++)
13. System.out.print(escalera[i][j]+"\t");
14.     }
```



15.

Ejemplo de Arreglo de Objetos

```
1.  public class ArregloAlumnos {
2.  public static void main(String[] args) {
3.  Alumno a[]= new Alumno[5];
4.  a[0]=new Alumno ("Juan Simon","9/09/65");
5.  a[1]= new AlumnoExterno("Irene Perez","8/12/63","trrgghjuh","Sistemas operativos");
6.  a[2]= new AlumnoInterno("Sonia Lopez","05/02/96","Ing. Mecanica","5676899");
7.  a[3]= new AlumnoExterno("Julieta Lopez","9/01/68","trftr5e4f67h","Sistemas  operativos");
8.  a[4]= new AlumnoInterno("Juan Mendoza","7/04/73","Ing. Computacion","89754687");
9.  for (int i=0; i<a.length;i++){
10. if (a[i] instanceof AlumnoExterno)   // pertenece a un clase en particular
11. {   a[i].imprime();}
12. }
13. System.out.println("Total de Alumnos " + Alumno.ca);
14. }
15. }
```

La clase Array, esta clase del paquete java.util contiene varios métodos del tipo “static” para manipular matrices. Estos métodos son: binarySearch, equals, fill, sort.

binarySearch

Permite buscar un valor en una matriz que esté ordenada en forma ascendente utilizando el algoritmo de búsqueda binaria. La sintaxis es.

int binarySearch(tipo[] m, tipo clave)

Donde clave es el valor que se desea buscar. El valor devuelto es un entero correspondiente al índice del elemento que coincide con el valor buscado. Si no se localiza, retorna un negativo que representa el punto de inserción.

Ejemplo:

```
import java.util.*;
class Test{
        public static void main(String[] ar){
                double[] a={10,15,20,25,30,35,40,45,50,55,60};
                 int [] a2={8,7,2,9,3,5,10,1};
                int [] b={8,7,2,-10,-30,-50,10,1};
                int i;
                System.out.print("Da un valor:");
                i=Leer.datoInt();
                System.out.println("Lugar="+ Arrays.binarySearch(a,i));
                Arrays.sort(a2);
                Arrays.sort(b,2,6);
                for (int i=0;i<a.length;i++)  System.out.println("a2["+i+"]="+a2[i]);
                for (int i=0;i<b.length;i++)  System.out.println("b["+i+"]="+b[i]);
        }
}
```

Procedimiento

Describe línea a línea las sentencias de los programas de esta lección, además realiza los siguientes ejercicios.

1: Realizar un programa que lea una lista de valores introducidos por el teclado. A continuación permitir que el usuario seleccione entre buscar los valores máximos, mínimos y escribirlos.

2: Escribir un programa que de cómo resultado la frecuencia con la que aparece cada una de las parejas de letras adyacentes de un texto introducido por el teclado. No se hará diferencia entre minúsculas y mayúsculas. El resultado se presentara en forma de tabla. Ejemplo.

```
          A b c d e f . . . z
A         0 4 0 2 1 0     1
B         8 0 0 0 3 1     0
C               .
D               .
E               .
F
. . .
Z
```

En la tabla anterior dice que la pareja de letras “ab” ha aparecido 4 veces.

3: Realizar un programa que lea los pesos de los alumnos e imprima un histograma correspondiente. Suponer que los pesos están entre los valores de 10 a 100 kgs. En el histograma solo aparecerán los valores que tengan un niño o más. Ejemplo:

```
          Peso          Número de niños.
21        **
22        **********
23        ******
24        **
25        *
```

4: Realizar un programa que lea una cadena de n caracteres e imprima el resultado que se obtiene cada vez que se realiza una rotación de un carácter a la derecha sobre dicha cadena. El proceso finalizara cuando se haya obtenido nuevamente la cadena de caracteres original. Por ejemplo:

```
HOLA  AHOL        LAHO        OLAH        HOLA
```

Evaluación:

Criterio	Ejercicio 1	Ejercicio 2	Ejercicio 3	Ejercicio 4
1. El alumno describió los programas	SI NO	SI NO	SI NO	SI NO
2. Cumplió con la codificación correcta del ejercicio	SI NO	SI NO	SI NO	SI NO
Puntaje Total_____	0 1 2	0 1 2	0 1 2	0 1 2

Ejercicio 16 Colecciones en java; Vector

U Competencia **4 Conocer las estructuras que permiten el manejo de datos en un lenguaje OO**

Duración Estimada: 1 sesión de 50 minutos

Objetivo:

El alumno usará la clase Vector

Introducción teórica

Una limitante de usar arreglos es que debemos conocer antes el número de localidades que usaremos, para ese caso una solución sería crear un array cuya dimensión sea más grande que el número de elementos que necesitamos guardar. Otra solución es usar un objeto tipo Vector que tiene la propiedad de crecer y decrecer de acuerdo a las necesidades del programa. La clase Vector es parte del paquete java.util de la librería estándar de clases de Java y en ella se pueden almacenar todo tipo de objetos.

- Ofrece un servicio similar a un arreglo, ya que se pueden almacenar y accesar valores y referencias a través de un índice. A diferencia de un arreglo, un Vector no está declarado para ser de un tipo particular.
- Un elemento puede insertarse y eliminarse de una posición específica a través de la invocación de un sólo método.
- Un objeto de tipo Vector maneja una lista de referencias a la clase Object, así no pueden almacenarse tipos de datos primitivos.
- Para usar la clase Vector hemos de poner al principio del archivo del código fuente la siguiente sentencia import **import java.util.Vector;**

El constructor de la clase se invoca de la siguiente forma **Vector vector=new Vector();**

Método	Descripción
Vector ( )	Constructor: crea un vector inicialmente vacío
void addElement (Objet obj)	Inserta el objeto especificado al final del vector
void setElementAt (Object obj, int indíce)	Inserta el objeto especificado en el vector en la posición especificada
void removeElementAt (int indíce)	Elimina el objeto especificado en el índice del vector
void clear ( )	Elimina todos los objetos del vector
Object elementAt (int indíce)	Regresa el componente en el índice especificado
boolean isEmpty ( )	Regresa verdadero si el vector no contiene elementos
int size ( )	Regresa el número de elementos en el vector

En el siguiente código se muestra un ejemplo del uso de esta clase

Ejemplo **Vector**

```
1.  import java.io.*;
2.  import java.util.Vector;
3.
4.  public class EjemploVector {
5.  public static void main(String[] args) throws IOException{
6.  Vector datos=new Vector();
7.  BufferedReader ent = new BufferedReader(new InputStreamReader(System.in));
8.  int i,b,t;
9.  String cadena= new String();
10. String nomdato= new String();
11. for ( i=0; i<3; i++){ // introducimos los datos que el usuario desea
        a.  System.out.println("Nombre " +i);
        b.  cadena = ent.readLine();
        c.  datos.addElement(cadena);
        d.  }
12. datos.addElement("Javier");  //Agregamos datos fijos
13. t=datos.size();    //conocemos el tamaño del vector
14. nomdato= (String)datos.elementAt(2); // Obtenemos el objeto de una posición
15. b=datos.indexOf("Susana"); //Conocemos la posición de “Susana” si existe
16. System.out.println(datos);
17. System.out.println("tamaño del Vector" + t);
18. System.out.println("Posicion 2 "+nomdato);
19. if (b<0)  //Si encontramos a “Susana”
20. System.out.println("Susana no está en el vector");
21. else
22. System.out.println("Susana está en la posición "+b);
23. }
24. }
```

Procedimiento

Realice un programa que maneje un menú en el que el usuario seleccione entre insertar elementos, eliminar elementos, conocer el número de elementos, obtener y mostrar los datos de una posición en un Vector y modificar los elementos en un Vector

Evaluación:

Criterio	Ejercicio 4
1. El alumno codificó el programa	SI NO
2. Cumplió con la ejecución correcta del ejercicio	SI NO
Puntaje Total_____	0 1 2

Ejercicio 17 Clases Arraylist, Treeset, Map

U Competencia **4 Conocer las estructuras que permiten el manejo de datos en un lenguaje OO**

Duración Estimada: 8 sesiones de 50 minutos

Objetivo:

El alumno conocerá y usará las reglas para la construcción de algoritmos, los codificará, y los implementará en otras máquinas

Introducción teórica

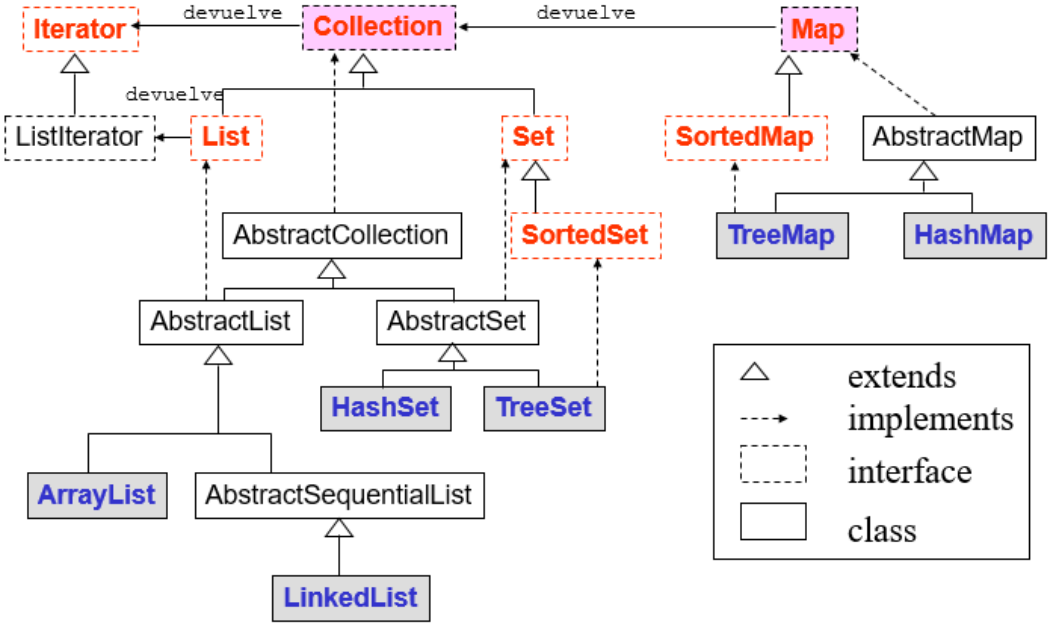
Generalmente, los elementos en una colección representan datos de una agrupación específico de objetos, como una colección de personas, casas. En una colección se pueden realizar operaciones sobre los objetos que están almacenados en su interior, así podemos almacenar, manipular, obtener y comunicar entre estos objetos.

En java se tienen clases e interfaces para facilitar la gestión de datos organizados en distintas colecciones. Las interfaces son representaciones abstractas sobre las colecciones, estas son implementadas en clases particulares que el programador podrá usar. Las clases tinene métodos que permiten gestionar la información de los objetos almacenados, dan servicio sobre operaciones de búsqueda, ordenamiento, inserción y eliminación.

En la interface Collection se especifican los atributos y los métodos generales necesarios para manipular los contenedores:

- int size() : devuelve el tamaño de una colección
- boolean empty() : devuelve un valor verdadero si la colección esta vacia
- boolean contains(Object elem): devuelve un valor boleano si la colección contiene un objeto dado
- Iterator iterator(): devuelve el objeto que permite recorrer y obtener los objetos almacenados en la colección
- Object[] toArray(), Object[] toArray(Object dest[]) devuelve el contenido de la colección en un arreglo de objetos
- boolean add(Object elem): método que permite inserter un nuevo objeto en la colección
- boolean remove(Object elem): método que elimina de la colección un objeto dado
- void clear(): método que elimina todos los objetos almacenados en la colección,

La imagen siguiente muestra las clases e interacciones que son útiles en java para almacenar objetos.



Las Interfeces que especifican el uso de las colecciones son:

Collection: Es la raíz de la jerarquía. Representa un grupo de objetos. Esta interfaz es el último denominador común que todas las colecciones implementan y es usada cuando se desea manipularlas con el máximo de generalidad deseado**..**
Set: una colección que no puede tener elementos duplicados. Un ejemplo son las cartas de un naipe**.**
List: Una colección ordenada. Las listas pueden contener elementos duplicados. Generalmente se puede tener control sobre donde cada elemento es insertado y se puede acceder a cada elemento por su índice (posición).
Queue: Una colección utilizada para guardar varios elementos por prioridad. Provee operaciones adicionales de inserción, extracción e inspección. Cada implementación de Queue debe especificar sus propiedades de ordenamiento**.**
Map: Un objeto que trabaja por parejas siendo sus componentes clave y valor. Map no permite tener claves duplicadas, cada clave debe corresponder al menos a un valor**.**
SortedSet: Mantiene sus elementos en un orden ascendente**.**
SortedMap: Mantiene sus mapeos en un orden ascendente por clave**.**

Las clases que permiten concretar el uso de las colecciones, todas son Cloneable y Serializable son:

LinkedList – Una implementación de una lista doblemente enlazada. La modificación es poco costosa para cualquier tamaño, pero el acceso aleatorio es lento. Útil para implementar colas y pilas. getFirst( ), getLast( ), removeFirst( ), removeLast( ), addFirst ( ), addLast ( ).

ArrayList – Una lista implementada utilizando un array de dimensión modificable. Es costoso añadir o borrar un elemento cerca del principio de la lista si ésta es grande, pero es relativamente poco costoso de crear y rápido para acceso aleatorio.

HashSet – Un Set implementado mediante una tabla hash. Es una buena implementación de propósito general por lo que la búsqueda, la adición y eliminación son insensibles al tamaño de los contenidos.

TreeSet – Un SortedSet implementado utilizando un árbol binario equilibrado. Es más lento para buscar o modificar que un HashSet, pero mantiene los elementos ordenados. Asume que los elementos son comparables si no se le ha pasado un comparator en el constructor.

```
class Estadistico {
   public static void main( String args[] ) {

HashMap tabla = new HashMap();
      for(int i=0; i < 10000; i++) {
         // Generar un número entre 0 y 20
         Integer num = new Integer((int)(Math.random()*20));
               if(tabla.containsKey(num))
                        //Incrementamos el contador asociado al número
            ((Contador)tabla.get(num)).incrementar();
         else
                        //Añadimos nuevo par: numero-contador
            tabla.put(num, new Contador());
         }
      System.out.println(tabla);
   }
}
```

Practica usando ArrayList

En el siguiente ejemplo se muestra el código necesario para almacenar en un contenedor tipo ArrayList llamado mercado

ArrayList mercado									
0		1		2		3		4	
Producto		Producto		Producto		Producto		Producto	
String nombre	int cantidad	String nombre	int cantidad	String nombre	int cantidad	String nombre	int cantidad	String nombre	int cantidad
pera	12	manzana	4	limón	45	sandía	23	leche	3

```
1.  class Producto{
2.  String nombre;
3.  int cantidad;
4.  Producto(String s, int i) {
5.  nombre = s;
6.  cantidad = i;
7.  }
8.   }
```

```
1.  import java.util.*;
2.  public class Mercado {
3.  public static void main(String args[]) {
4.  // Definir 5 instancias de la Clase Producto
5.  Producto m = new Producto("pera ", 12);
6.  Producto n = new Producto("manzana ", 4);
7.  Producto o = new Producto("limón ", 45);
8.  Producto p = new Producto("sandía", 23);
9.  Producto q = new Producto("lecha", 3);
10. // Definir un ArrayList
11. ArrayList mandado = new ArrayList();
12. // Colocar Instancias de Producto en ArrayList
13. mandado.add(m);
14. mandado.add(n);
15. mandado.add(o);
16. mandado.add(p);
17. // Indica el indice de inserción
18. mandado.add(1, q);
19. mandado.add(q);
20. // Imprimir contenido de ArrayLists
21. System.out.println(" - Lista de mandado con " + mandado.size() + " elementos");
22.  // Definir Iterator para extraer/imprimir valores
23. for( Iterator it = mandado.iterator(); it.hasNext(); ) {
24. Producto x = (Producto)it.next();
25. System.out.println(x.nombre + " : " + x.cantidad);
26. }
27. // Eliminar elemento de ArrayList
28. mandado.remove(2);
29. System.out.println(" - Lista de mandado con " + mandado.size() + " elementos");
30.  // Definir Iterator para extraer/imprimir valores
31. for( Iterator it2 = mandado.iterator(); it2.hasNext();) {
32. Producto x = (Producto)it2.next();
33. System.out.println(x.nombre + " : " + x.cantidad);
34. }
35. // Eliminar todos los valores del ArrayList
36. mandado.clear();
37. System.out.println(" - Lista de mandado final con " + mandado.size() + " elementos");
38. }
39.
40.  }
```

Procedimiento:

Con base en el ejemplo anterior genere una aplicación que permita almacenar y gestionar los datos de Personas, los datos que se almacenarán son nombre, curp, fecNac, teléfono, Tcredito. Debe generar una aplicación que permita almacenar, buscar, modificar, mostrar y eliminar los datos de las Personas. Los usuarios podrán seleccionar las acciones mediante un menú

Modelo UML de las clases necesarias y codificación

Evaluación:

Criterio	Ejercicio 4
1. El alumno codificó la clase de Persona	SI NO
2. El alumno codifico la clase main	SI NO
3. Cumplió con la ejecución correcta del ejercicio	SI NO
Puntaje Total_____	0 1 2 3

Unidad 5 Interfaz Gráfica en JAVA

Ejercicio 18 Interfaz Gráfica en java; componentes y eventos

U Competencia **5 Operar los componentes que conforman la interfaz gráfica del usuario, con el uso de archivos**

Duración Estimada: 4 sesiones de 50 minutos

Objetivo:

El alumno usará las clases que le permitan desarrollar interfaces graficas para el usuario

.

Introducción teórica

Java proporciona una serie de paquetes estándar que nos facilitan la labor de crear los Interfaces Gráficos de Usuario (GUI). Estos paquetes se localizan englobados dentro de Java Foundation Classes (JFC), que proporciona.

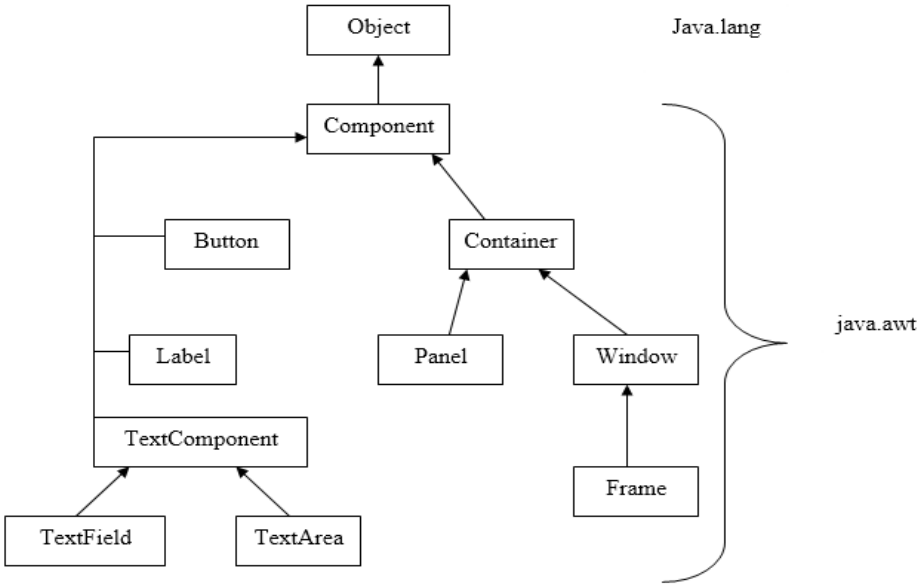
- Soporte de componentes entrada/salida a través de AWT (Abstract Window Toolkit) y Swing.
- Imágenes 2D
- Servicio de impresión
- Soporte para funciones Drag-and-Drop (arrastra y pega)
- Funciones de accesibilidad, etc.

De los paquetes anteriores, los más usados son AWT y Swing. AWT es el primero en aparecer y proporciona los componentes básicos de entrada/salida; Swing recoge la mayor parte de la funcionalidad de AWT, aportando clases más complejas y especializadas y otras de nueva creación.

El componente más básico que suele emplearse en un interfaz gráfico de usuario es la ventana. AWT proporciona la clase Window, aunque, para simplificar los desarrollos, se acudirá a la clase Grame derivada de Window y por lo tanto, más especializada.

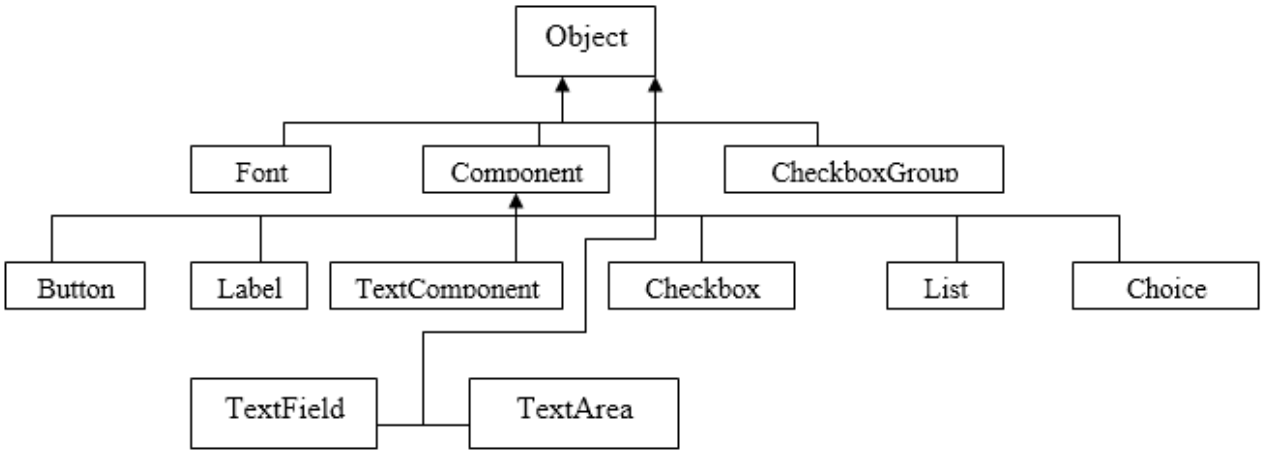
La clase abstracta Component proporciona una representación gráfica susceptible de ser representada en un dispositivo de salida.

La clase abstracta Container añade la funcionalidad de poder contener otros componentes AWT (más botones, caja de texto, etc). La clase Window proporciona una ventana gráfica sin bordes no posibilidad de incluir barras de menú. Frame no tiene esas restricciones. JFrame es una versión de Frame proporcionada por Swing. Y proporciona posibilidades más amplias que las implementadas por sus superclases de AWT, aunque también presenta una mayor complejidad de uso.



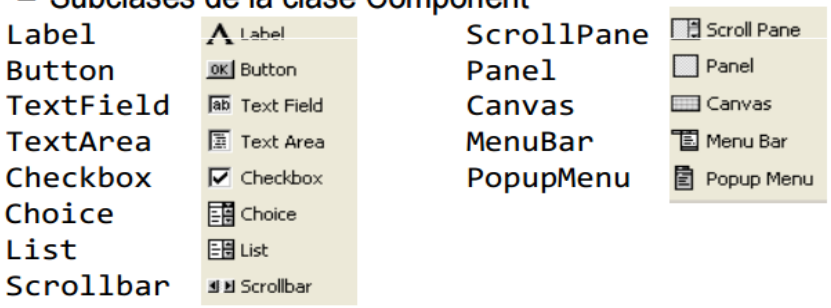
Entre los componentes habituales en el diseño de GUI se encuentran los botones, etiquetas, campos de texto y áreas de texto, todos ellos previamente estudiados. Ahora nos centraremos en las cajas de verificación (Checkbox), los botones de radio (CheckboxGroup), listas (List) y listas desplegables (Choice).

El siguiente diagrama sitúa todas estas clases en la jerarquía que proporciona Java



Visulmente se pueden reconocer los componentes gráficos con la siguiente imagen:

– Subclases de la clase Component



Para organizar los componentes en las ventanas gráficas se usan los Layout (manejadores de composición), en java se tienen varias opciones :

- BorderLayout (predeterminado)
- FlowLayout
- BoxLayout
- GridLayout
- GridBagLayout
- SpringLayout

En la imagen se muestra un ejemplo sobre el uso de BorderLayout

```
import java.awt.*;
class BorderLayoutDemo extends Frame {
  public static void main(String args[]) {
    BorderLayoutDemo bld = new BorderLayoutDemo();
    bld.setLayout(new BorderLayout(10, 10));
    bld.add(new Button("NORTE"), BorderLayout.NORTH);
    bld.add(new Button("SUR"), BorderLayout.SOUTH);
    bld.add(new Button("ESTE"), BorderLayout.EAST);
    bld.add(new Button("OESTE"), BorderLayout.WEST);
    bld.add(new Button("CENTRO"),BorderLayout.CENTER);
    bld.setSize(200, 200);
    bld.setVisible(true);
  }
}
```

Eventos

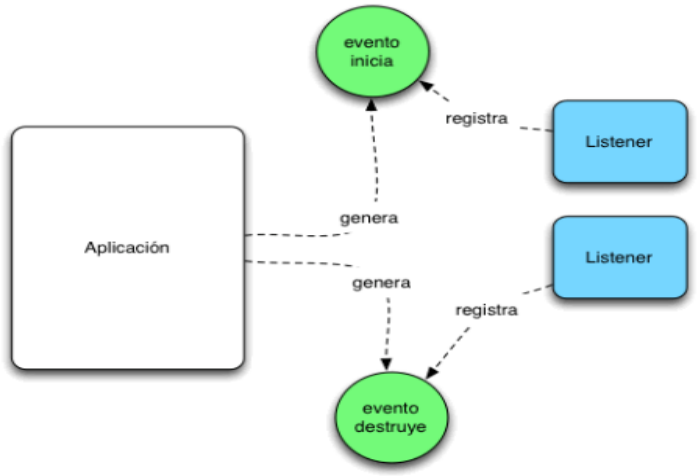
Java es un lenguaje orientado a eventos, puesto que nos proporciona todos los elementos necesarios para definirlos y utilizarlos; los eventos son objetos fundamentales en el desarrollo de la mayor parte de las aplicaciones.

Cuando llega un evento, en ocasiones nos interesa tratarlo (por ejemplo, la pulsación de un número en una aplicación calculadora) y, otras veces, no deseamos tratar el evento con ninguna acción (por ejemplo, cuando el usuario pulsa con el ratón sobre un texto al que no hemos asignado ninguna información complementaria). Los eventos más comunes que se pueden producir en la ejecución de una aplicación que proporcione un interfaz gráfico de usuario (GUI) son:

- Pulsación de ratón

- El puntero de ratón “entra en” (se sitúa sobre) un componente gráfico (por ejemplo, sobre un botón o una etiqueta)
- El puntero del ratón “sale de” un componente gráfico.
- Se pulsa una tecla
- Se cierra una ventana

Los eventos tienen el siguiente ciclo de vida: se generan del exterior de la aplicación, se registra en el Listener, se responde según se establece en la clase implementadora del Listener y posterioemente se destruye el evento.



Los eventos, habitualmente, se originan desde el exterior de la aplicación, producidos por el usuario que hace uso de ella. El simple movimiento del ratón genera multitud de eventos que pueden ser tratados por nuestros programas Java.

Tipo de Componente	Eventos generados	Hechos que los generan
Button	ActionEvent	El usuario hace un clic sobre el botón.
Checkbox	ItemEvent	El usuario selecciona o deselecciona el interruptor (Checkbox)
CheckboxMenuItem	ItemEvent	El usuario selecciona o deselecciona el interruptor (Checkbox)
Choice	ItemEvent	El usuario selecciona o deselecciona un elemento de la lista
Component	ComponentEvent	El componente se mueve, cambia de tamaño, se esconde o se exhibe
	FocusEvent	El componente gana o pierde el foco
	KeyEvent	El usuario pulsa o suelta una tecla
	MouseEvent	El usuario pulsa o suelta un botón del ratón, el cursor del ratón entra o sale o el usuario mueve o arrastra el ratón
Container	ContainerEvent	Se agrega o se quita un componente al contenedor
List	ActionEvent	El usuario hace doble clic en un elemento de la lista
	ItemEvent	El usuario selecciona o deselecciona un elemento de la lista
MenuItem	ActionEvent	El usuario selecciona un elemento del menú

Scrollbar	AdjustmentEvent	El usuario mueve la barra de desplazamiento
TextComponent	TextEvent	El usuario hace un cambio en el texto
TextField	ActionEvent	El usuario termina de editar el texto (hace un intro)
Window	WindowEvent	La ventana se abre, se cierra, se minimiza, se reestablece o se cierra.

El mecanismo básico de eventos de Java se basa en la existencia de clases de eventos y los interfaces *Listeners* (escuchadores). Los Listeners son interfaces que debemos implementar, colocando las acciones deseadas en sus métodos.

AWT proporciona una amplia gama de eventos que pueden ser recogidos por los métodos existentes en las implementaciones que hagamos de los Listeners (o las clases que especialicemos a partid de los adaptadores). La siguiente tabla muestra los listener y los métodos que se deben implementar para responder ante los eventos

Interfaz	Métodos
ActionListener	actionPerformed(ActionEvent)
AdjustmentListener	adjustmentValueChanged(AdjustementEvent)
ComponentListener	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
ItemListener	itemStateChanged(ItemEvent)
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
TextListener	textValueChanged(TextEvent)
WindowListener	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

Ejemplo sobre la gestión de eventos

```
import java.awt.*;
```

```
public class PruebaRaton{
	public static void main(String[] args){
		Frame MiFrame=new Frame("Esquema de eventos");
		Panel MiPanel=new Panel();
		Button Hola=new Button("Saludos");
		Button Adios=new Button("Despedida");
		MiPanel.add(Hola);
		MiPanel.add(Adios);
		MiFrame.add(MiPanel);
		MiFrame.setSize(200,100);
		MiFrame.show();
		Hola.addMouseListener(new EsquemaRaton());
		Adios.addMouseListener(new EsquemaRaton());	}	}
//Clase que recoge los eventos de raton mediante sus métodos
//mousePressed, mouseReleased, mouseClicked, etc. com parámetros
//"MouseEvent"

import java.awt.event.*;
public class EsquemaRaton extends Object implements MouseListener{
	public void mouseClicked(MouseEvent EventoQueLlega){
		System.out.println("Click de raton");	}

	public void mousePressed(MouseEvent EventoQueLlega){
		System.out.println("Presión de ratón");	}

	public void mouseReleased(MouseEvent EventoQueLlega){
		System.out.println("Se ha levantado el botón del ratón");	}

	public void mouseEntered(MouseEvent EventoQueLlega){
		System.out.println("'Focus' de ratón");	}
```

Procedimiento:

Genere una aplicación que genere un Applet que muestre un pequeño formulario, que permita insertar el nombre del usuario, el Applet generará un mensaje de saludo con el nombre insertado

Evaluación:

Criterio	Ejercicio 4
1. El alumno codificó el Applet	SI NO
2. El alumno insertó componentes para un formualrio	SI NO
3. Cumplió con la ejecución correcta del ejercicio (se mostró el saludo)	SI NO
Puntaje Total_____	0 1 2 3

Ejercicio 19 Aplicación web; clase Applet

U Competencia **5 Operar los componentes que conforman la interfaz gráfica del usuario, con el uso de archivos**

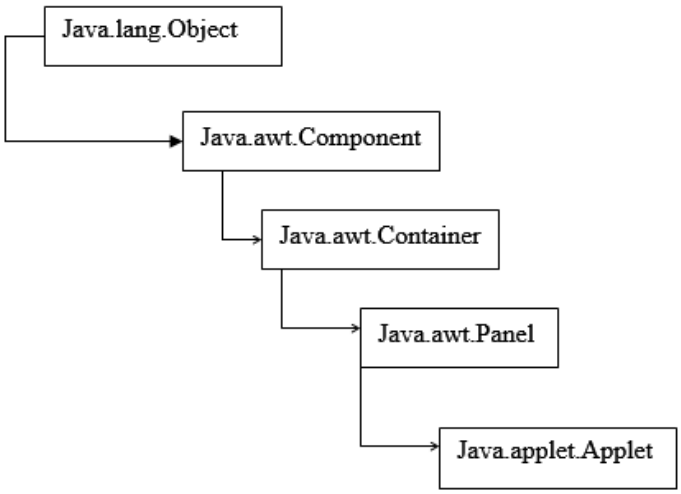
Duración Estimada: 4 sesiones de 50 minutos

Objetivo:

El alumno conocera y desarrollará aplicaciones en java para la web, específicamente los Applets

Introducción teórica

Para crear un applet únicamente se necesita escribir una clase derivada de la clase Applet del paquete java.applet. La clase Applet tiene que ser la superclase de cualquier applet que se construya. La siguiente figura muestra dónde se sitúa esta clase en la jerarquía de clases de la biblioteca Java.



Para poder ejecutar un applet utilizando HTML se requieren los siguientes pasos:

Paso 1. Escribir el programa. Es con mucho la parte más difícil del proceso.

Paso 2. Compilar el programa. El paso siguiente es producir el código de bytes enviando el código fuente de Java a un compilador Java (como jdk1.3 por ejemplo).

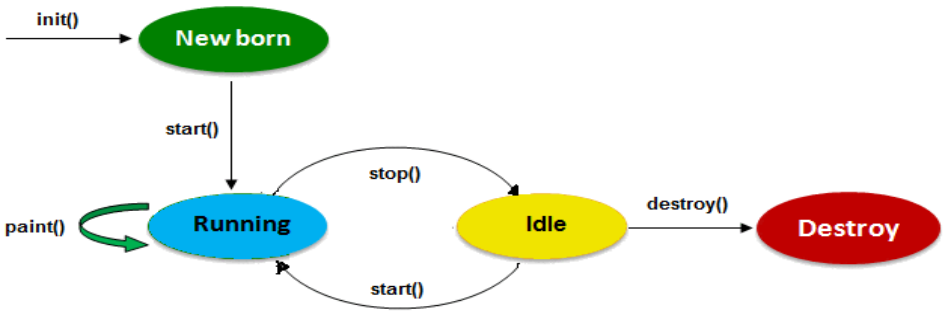
Paso 3. Indicar al navegador donde se localiza el código de bytes. A tal efecto, hay que generar un documento HTML que haga referencia al archivo .class que produjo el compilador Para ello, se necesita un editor de texto. Basta cualquier procesador de texto; no tiene que ser complejo o costoso, ya que sólo se requiere texto sencillo, sin características de estilo. Se escribirá el texto HTML siguiente:

```
<HTML>  <HEAD>  <TITLE> Ejemplo de subprograma  </HEAD>
<BODY>  Esta es la llamada al subprograma
<APPLET code = "Hola.class" width = 60 height = 60> </APPLET>
<HR>    </BODY>
</HTML>
```

Cada vez que visitamos una página Web que contiene una referencia a un applet, es el explorador el que se encarga de cargarlo. Una vez cargado, se crea un objeto de su clase, esto es, se crea un applet. A continuación el applet se inicia a si mismo (ejecutando su método init) y comienza su ejecución (ejecuta el método star).

Una vez que el applet está en ejecución puede ocurrir que visitemos otra página. En este caso, el applet detendrá su ejecución (ejecuta su método stop) que la reanudará en el momento en el que se visitada su página de nuevo (vuelve a ejecutarse su método start). Algunos exploradores no se comportan de la manera expuesta, sino que descargan el applet cuando visitamos otra página y lo vuelve a cargar cuando retornamos a la página que lo referencia.

Cuando un applet es descargado, por ejemplo, porque salimos del explorador, libera todos los recursos que hubiera adquirido y detiene su ejecución invocando al método stop y después el método destroy. Cuando el explorador muestra una página Web con un applet y se pulsa el botón actualizar del mismo, el applet es descargado y vuelto a cargar.



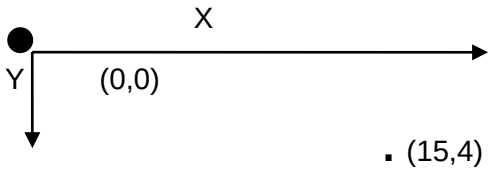
Entre los métodos proporcionados por esta clase cabe destacar los siguientes:

public void init()	Este método se carga una sola vez cuando se carga el applet. La definición predeterminada de este método no hace nada, por eso la clase que da lugar al Applet debe redefinirlo siempre que sea necesario realizar una operación de iniciación; por ejemplo, definir el tamaño del Applet, cargar imágenes y el sonido, lanzar hilos que sean necesarios, etc.
public void start()	Es llamado cada vez que se inicie la ejecución del applet. Esto es, después de que se ejecute el método **init** y cada vez que se vuelva a visitar la página Web que muestra el applet. La definición predeterminada de este método no hace nada, por eso la clase que da lugar a nuestro applet debe redefinirlo siempre que sea necesario realizar alguna operación cada vez que se muestre la página Web; por ejemplo, un applet animado puede utilizar este método para iniciar dicha animación cada vez que se visite la página que lo muestra.
public void paint (Graphics g)	La clase Applet hereda este método de la clase Container. Es llamado cada vez que es necesario repintar el contenedor del applet, esto es, su área de dibujo. La definición predeterminada de este método pinta una ventana

	gráfica de color gris, por eso la clase que da lugar a nuestro applet debe redefinirlo siempre que sea necesario mostrar una interfaz gráfica adecuada a nuestras necesidades. El parámetro g de la clase Graphics pertenece al paquete java.awt y hace referencia a la ventana gráfica. Previamente a este método, siempre es invocado update que se define exactamente igual, y la ejecución de update puede ser forzada invocando al método repaint.
public void stop()	Este método es llamado para informar al applet que su ejecución va a ser detenida. Esto ocurre siempre que la página que muestra el applet es reemplazada por otra página, y justo antes de que sea destruido. La definición predeterminada de este método no hace nada, por eso la clase que da lugar a nuestro Applet debe redefinirlo siempre que sea necesario realizar alguna operación cada vez que la página deje de ser visible; por ejemplo, un applet anidado puede querer detener la animación.
public void destroy()	Este método es llamado para informar al applet de que va a ser destruido con el fin de que pueda liberar cualquier recurso que tenga asignado; previamente a este método, siempre es invocado **stop**. La definición predeterminada de este método no hace nada, por eso la clase que da lugar a nuestro applet debe redefinirlo siempre que sea necesario realizar alguna operación previa a la destrucción

El paquete java.awt contiene 61 clases e interfaces, todas ellas dedicadas de una u otra forma a la producción de subprogramas y aplicaciones provistos de interfaz gráfica de usuario (GUI). En esta sección se explicará en forma breve el método paint().

Antes de iniciar, se debe mencionar la forma en que Java mide las posiciones para fines de trazado. Cada píxel de la pantalla se describe con coordenadas de enteros, uno horizontal (frecuentemente llamado x), que se mide desde el borde izquierdo del marco del subprograma (o el del objeto que se esté trazando) y otro vertical (que suele llamarse y), medido desde el margen superior del marco componente del subprograma, como se ilustra en la siguiente figura:

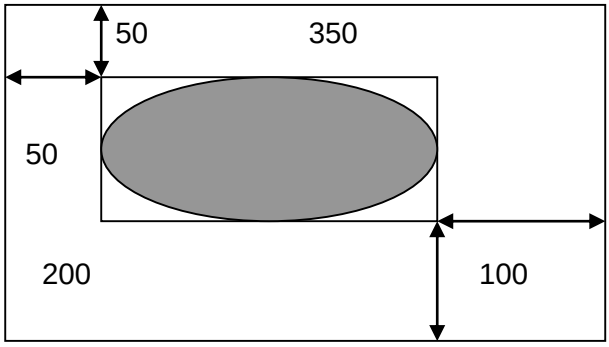


Observe que el origen, o sea el punto (0,0), está en la esquina superior izquierda y las coordenadas verticales aumentan, en vez de disminuir, conforme se desciende.

El Applet que se muestra enseguida permite mostrar algunas clases anidadas para crear un subprograma que traza un dibujo sencillo en el monitor. El subprograma genera un torbellino azul sobre un fondo negro, acompañado de un mensaje de texto en blanco con sombra caida. Para que quede agradable a la vista se decide utilizar diez óvalos de diferente tamaño y diferentes tonos de azul.

La única parte con truco fue decidir cómo ajustar el tamaño de los óvalos. Se partió de un subprograma de 500 por 350 pixeles y se decidió dejar los márgenes izquierdo y superior en 50, y

el inferior y derecho en 100. Ello permitió tener un óvalo externo de 350 de anchura y 200 de altura, en la posición (50,50), como se muestra en la siguiente figura



Los siguientes óvalos se alinearon en la misma coordenada derecha y estuvieran centrados verticalmente. Después de algunas pruebas, se llegó a la conclusión de que cada óvalo interno debería estar 30 pixeles a la derecha y 10 pixeles arriba y abajo del óvalo exterior inmediato. Creándose el siguiente cuadro:

X	Y	Whidth	Height
50	50	350	200
80	60	320	180
110	70	290	160
140	80	260	140
170	90	230	120
200	100	200	100
230	110	170	80
260	120	140	60
290	130	110	40
320	140	80	20

Se especifica el Color antes de la llamada al método fillOval(), definiéndose en cada caso un Color distinto. También en cada paso se aumentó el valor de los componentes rojo y verde en 10, y el descomponente azul en 25, con lo cual se obtuvieron tonos de azul cada vez más brillantes. El mensaje de texto “¡Hola Mundo!” se trazó al hacerlo primero con la sombra, en el punto (23,301), y luego trazar el texto blanco encima de la sombra, desplazado hacia arriba y a la derecha tres píxeles.

Ejemplo de Applet dibujando con el objeto Graphics

```
import java.applet.*;
import java.awt.*;
public class Torbellino extends Applet{
        public void paint(Graphics g){
                //Se coloca en negro el fondo
                g.setColor(Color.black);            g.fillRect(0,0,500,350);
                //Se dibujan 10 óvalos anidados

               //Se establece un color                    //Se dibuja un óvalo
```

```
                    g.setColor(new Color(0,0,25));          g.fillOval(50,50,350,200);
                    g.setColor(new Color(10,10,50));        g.fillOval(80,60,320,180);
                    g.setColor(new Color(20,20,75));        g.fillOval(110,70,290,160);
                    g.setColor(new Color(30,30,100));       g.fillOval(140,80,260,140);
                    g.setColor(new Color(40,40,125));       g.fillOval(170,90,230,120);
                    g.setColor(new Color(50,50,150));       g.fillOval(200,100,200,100);
                    g.setColor(new Color(60,60,175));       g.fillOval(230,110,170,80);
                    g.setColor(new Color(70,70,200));       g.fillOval(260,120,140,60);
                    g.setColor(new Color(80,80,225));       g.fillOval(290,130,110,40);
                    g.setColor(new Color(90,90,250));       g.fillOval(320,140,80,20);
                    //Se dibuja un texto, primero se establece color, tipo de letra y se dibuja
                    g.setColor(Color.blue);
                    g.setFont(new Font("TimesRoman", Font.BOLD,36));
                    g.drawString("¡Hola Mundo...!",23,301);
                    //Se dibuja un texto por segunda vez para similar la sombra,
                    g.setColor(Color.white);
                    g.drawString("¡Hola Mundo...!",20,298);
                }
            }
```

Ejemplo para visualizar en un Applet un formulario

```
  package formulario;
  import java.applet.Applet;
  import java.awt.*;
  import java.awt.event.*;
  public class Ventana3 extends Applet implements
ActionListener {
        Label et;       Label et2;
        Label et3;       Label et4;
        TextField t1;       TextField t2;
        Button b1;       Button b2;
        Button b3;
        Panel p1,p2,p3,p4;
        public void init() {
        et= new Label("Este programa muestra una serie
de componentes");
        et2= new Label ("Hola dame tu nombre");
        et3= new Label(" Soy");
        et4= new Label("        ");
        t1 =new TextField("",20);
        t2=new TextField("",20);
        b1=new Button("Aceptar");
        b2= new Button("Cancelar");
        b3= new Button("Continuar");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
```

```
        p1= new Panel();       p2= new Panel();
        p3= new Panel();       p4=new Panel();
        p1.setBackground(Color.CYAN);
        p1.setSize(400, 300);
        p2.setBackground(Color.GREEN);
        p1.setLayout(new GridLayout(1,1));
        p2.setLayout(new GridLayout(3,2));
        p3.setLayout(new GridLayout(3,1));
        p4.setLayout(new GridLayout(1,3));
        p1.add(et);       p2.add(et2);
        p2.add(t1);       p2.add(et3);
        p2.add(t2);       p2.add(et4);
        p4.add(b1);       p4.add(b2);
        p4.add(b3);       p3.add(p1);
        p3.add(p2);       p3.add(p4);
        this.add(p3);
  }
  public void actionPerformed(ActionEvent e){
     if (e.getActionCommand()=="Aceptar"){
        t2.setText(t1.getText());
        et4.setText("Hola "+t1.getText()+ " te doy la
bienvenida");         repaint();
     }
     if (e.getActionCommand()=="Cancelar"){
        et4.setText("Operacion Cancelada");
        repaint();    }
```

```
      if (e.getActionCommand()=="Continuar"){
         t1.setText("");
         t2.setText("");
         et4.setText("");
         repaint();
      }          }    }
```

Ejemplo Reloj

```
  import java.applet.Applet;
  import java.applet.*;
  import java.awt.*;
  import java.util.*;
  import java.text.DateFormat;
  public class aReloj extends Applet implements Runnable {
   private Thread hilo = null;
    private Font fuente;
    private String horaActual = "00:00:00";

    public void init(){
     fuente = new Font("Arial", Font.BOLD, 24);
    }
    public void start(){
     if (hilo == null){
       hilo = new Thread(this, "Reloj");
       hilo.start();
     }
    }
    public void run(){
            Thread hiloActual = Thread.currentThread();
            while (hilo == hiloActual){
           // Obtener la hora actual
       Calendar cal = Calendar.getInstance();
       Date hora = cal.getTime();
       DateFormat df = DateFormat.getTimeInstance();
       horaActual = df.format(hora);
       repaint();
       try{
       Thread.sleep(1000);              }
                    catch (InterruptedException e){ }
             } }
    public void paint(Graphics g){
     // Dibujar un rectángulo alrededor del contenedor
     g.draw3DRect(1,          1,          getSize().width-3,
getSize().height-3, false);
      // Establecer la fuente
            g.setFont(fuente);
      // Mostrar la hora
      g.drawString(horaActual, 14, 40);
    }
   public void stop(){
     hilo = null;
    }
   }
```

Procedimiento del Ejercicio:

Realice un Applet que mediante la Clase Graphics dibuje en la pantalla una casa con base en figuras geométricas como rectángulos, cuadrados, círculos, líneas, triángulos. Use la imagen siguiente como guía, no debe ser exactamente igual solo es un ejemplo

Evaluación:

Criterio	Ejercicio 4
1. El alumno codificó la clase Applet	SI NO
2. El alumno codifico círculos de color	SI NO
3. El alumno codifico rectángulos de color	SI NO
4. El alumno codifico líneas de color	SI NO
5. El Applet se visualiza en una página web	SI NO
6. En el Applet se dibuja una casa en dos dimensiones	SI NO
Puntaje Total_____	0 1 2 3 4 5 6

Ejercicio 20 Aplicaciones de escritorio; clase JFrame

U Competencia **5 Operar los componentes que conforman la interfaz gráfica del usuario, con el uso de archivos**

Duración Estimada: 2 sesiones de 50 minutos

Objetivo:

El alumno generará aplicaciones usando la clase JFrame

Introducción teórica:

Cuando un JFrame tiene cierta complejidad, lo más correcto y elegante es extender esta clase añadiendo todas las características que necesitemos El constructor puede utilizarse para la creación de componentes, gestores de distribución, etc.

Los componentes pueden conectarse a referencias que sean atributos para tener acceso a ellos en múltiples operaciones de la clase se usara este esquema con las clases JFrame, JInternalFrame o JDialog, cuando contengan un número relevante de componentes con una gestión más o menos compleja, incluyendo captura y procesamiento de eventos

En estos casos la nueva clase definida, además de ser en sí un componente de la interfaz, tiene una segunda función muy importante de coordinación, gestión y procesamiento de la información que llega a través de sus componentes hijos.

Ejemplo de JFrame

```
import java.util.*;
import java.io.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.ImageIcon;

public class f1 extends JFrame
{
    MenuBar menu=new MenuBar(); //crea una barra de menu
   Menu menu1=new Menu("menu1",true);//Submenu
    Menu menu2=new Menu("menu2",true);//Submenu
    Image icon;//Declara un objeto de imagen tipo icono
    int maxX,maxY;//Determina las coordenadas maximas del frame
    boolean actualiza=false;//Bandera para actualizar frame

    public static void main(String[] args) throws IOException    {
            System.out.println("La practica frame-1 está corriendo...");
            f1 i=new f1();//se llama al constructor del frame
    }
```

```
	public f1()throws IOException{ //crear el constructor del frame
		super("Frame1");//Asi se llama al constructor de una clase extendida
		maxX=800;
		maxY=600;
		resize(maxX,maxY);//Redibuja el frame con las dimensiones maxx y maxy
		setMenuBar(menu);//Agrega la barra de menu al frame
		menu.add(menu1);//agrega los menus 1 y 2 a la barra de menu
		menu.add(menu2);
		menu1.add("menu1-1");// agrega submenus
		menu1.add("menu1-2");// agrega submenús
		menu1.add("menu1-3");// agrega submenús
		menu2.add("menu2-1");
		menu2.add("menu2-2");
		menu2.add("menu2-3");
		menu2.add("menu2-4");
//crear un icono con el archivo de entrada
		icon=new ImageIcon("mia.jpg").getImage();
		setIconImage(icon); //Establece la imagen como icono
		show();//muestra la imagen
		move(100,50);//mueve el frame en la posición 100,50
	}
	public void paint(Graphics g)//Refresca la pantalla
	{   if (actualiza==true){
		g.clearRect(0, 0, maxX,maxY);
		actualiza=false;}
	}

public void update(Graphics g){paint(g);}//actualiza los cambios
//manejador de eventos
public boolean action(Event ev, Object arg)
//recibe un objeto y un evento
{
if (ev.target instanceof MenuItem)
//si se eligio un elemento de la barra de menús
{
if(arg.equals("menu1-1")){ //si se eligió el menu1-1
try{
//crea una caja de mensajes con el título "Escribe en el cuadro"
MsgBox msg=new MsgBox("Escribe en el cuadro");
//hace visible la caja de mensajes
msg.setVisible(true);
//si se dio click en el boton aceptar de la caja de mensajes
//se imprime el texto capturado
if (msg.status()==true){
System.out.println("Escribiste el texto "+msg.getMessage());
}

}catch(Throwable e){System.err.println("***Error al ejecutar menu1-1***");}   //variable de error
```

```
                        }
if(arg.equals("menu1-2")){  //si se eligio el menu1-2
//crea una ventana de dialogo de archivos en modo lectura
FileDialog f=new FileDialog(this,"Abrir Archivo de Vinculos",0);//Se usa el #0 para abrir
f.show();//muestra la ventana de dialogo
//obtiene direccion del archivo
                String directory = f.getDirectory();
//obtiene nombre del archivo
                String file = f.getFile();
                System.out.println("Abriendo el archivo "+directory+file);
                //llama automáticamente al método update()
repaint();
                        }
                }
        return true;
        }
//manejador de eventos de ventanas Windows
        public boolean handleEvent(Event e) {
        //en caso de dar clic en el tache de la ventana finaliza la aplicación
                if(e.id == Event.WINDOW_DESTROY)
                        System.exit(0);
                return super.handleEvent(e);
        }
};
```

Procedimiento

Genere un formulario que permita insertar datos de varias computadoras, que se almacenen en un ArrayList y que mediante botones se puedan visualizar los registros alamacenados en el ArrayList

Evaluación:

Criterio	Ejercicio 4
1. El alumno codificó las clases necesarios	SI NO
2. El alumno codificó el formulario	SI NO
3. Funcionó el almacenamiento de registros en el ArrayList	SI NO
4. Funcionaron los botones de exploración	SI NO
Puntaje Total_____	0 1 2 3 4

Ejercicio 21 Componentes complejos; clase JTable

U Competencia **5 Operar los componentes que conforman la interfaz gráfica del usuario, con el uso de archivos**

Duración Estimada: 2 sesiones de 50 minutos

Objetivo

El alumno usará la calse JTable para gestionar información en las aplicaciones

Introducción teórica

Una tabla representa una de las formas más comunes de mostrar un conjunto de datos relacionados; por ejemplo, los registros de una base de datos. Este componente presenta la información distribuida en filas y columnas.

La biblioteca de Java incluye una clase denominada JTable en el paquete javax.swing para permitir el acceso a un componente que puede manipular cualquier tipo de tabla.

Una tabla se añade a una ventana en la misma forma que se añade otro componente. Construimos un objeto JTable, establecemos sus propiedades y lo añadimos al contenedor adecuado. El constructor de la clase JTable tiene un primer argumento de tipo Object [ ][ ] que permite almacenar en una matriz a la tabla, y un segundo argumento del tipo String[] que permite almacenar en un vector la cabecera de las columnas.

En una tabla se pueden tener dos clases de filas o columnas: editables y no editables. El color por omisión de las filas o columnas no editables es gris, y el de las editables es blanca. Cuando el número de filas y columnas es superior a la superficie de la tabla, es posible utilizar barras de desplazamiento utilizando un panel de desplazamiento. La barra de desplazamiento horizontal sólo aparecerá si la propiedad “autoResizeMode” esta puesta en OFF.

Los elementos de una tabla pueden ser del tipo Object, String, Bolean, Integer, Byte, Short, Long, Float o Double. Generalmente la lista de datos se genera a partir de una matriz del tipo Object que se pasa como primer argumento al constructor JTable, y la lista de cabecera, a partir de una matriz de tipo String que se ha pasado como segundo argumento. Si estos argumentos se omiten, la tabla queda inicializada con cero filas y cero columnas. También es posible crear una tabla sin datos, pero con un número de filas y de columnas.

Una tabla consta de los siguientes modelos:

•TableModel. Define el modelo de datos de una tabla, proporcionando información acerca de las filas, columnas y valores de las celdas. La biblioteca JFC proporciona una implementación estándar de la interfaz TableModel bajo la clase DefaultTableModel derivada de AbstractTableModel.

•TableColumnModel. Modelo para almacenar varias columnas como una colección. La biblioteca JFC proporciona una implementación estándar de la interfaz Table ColumnModel bajo la clase DefaultTableColumnModel.

Otra forma de iniciar una tabla es utilizando el modelo que este componente implementa por omisión. Esto es, su propiedad "model" hace referencia a un objeto de la clase DefaultTableModel, implementación estándar de la interfaz TableModel y derivada de AbstractModel, que contiene los datos de la tabla. También, si lo deseamos, podemos crear nuestro propio objeto y asignárselo a la tabla invocando su método setModel. Por ejemplo,

```
            jScrollPanel1 = new javax.swing.JScrolPane( );
            jTabla1 = new javax.swing.JTable();
// se modifica el modelo de la tabla con datos que es un arreglo de objetos y cabecera es un arreglo de String
            jTablesetModel(new javax.swing.table.DefaultTableModel(datos,cabecera);
            jscrollPanel1.setViewportView(jTable1);
```

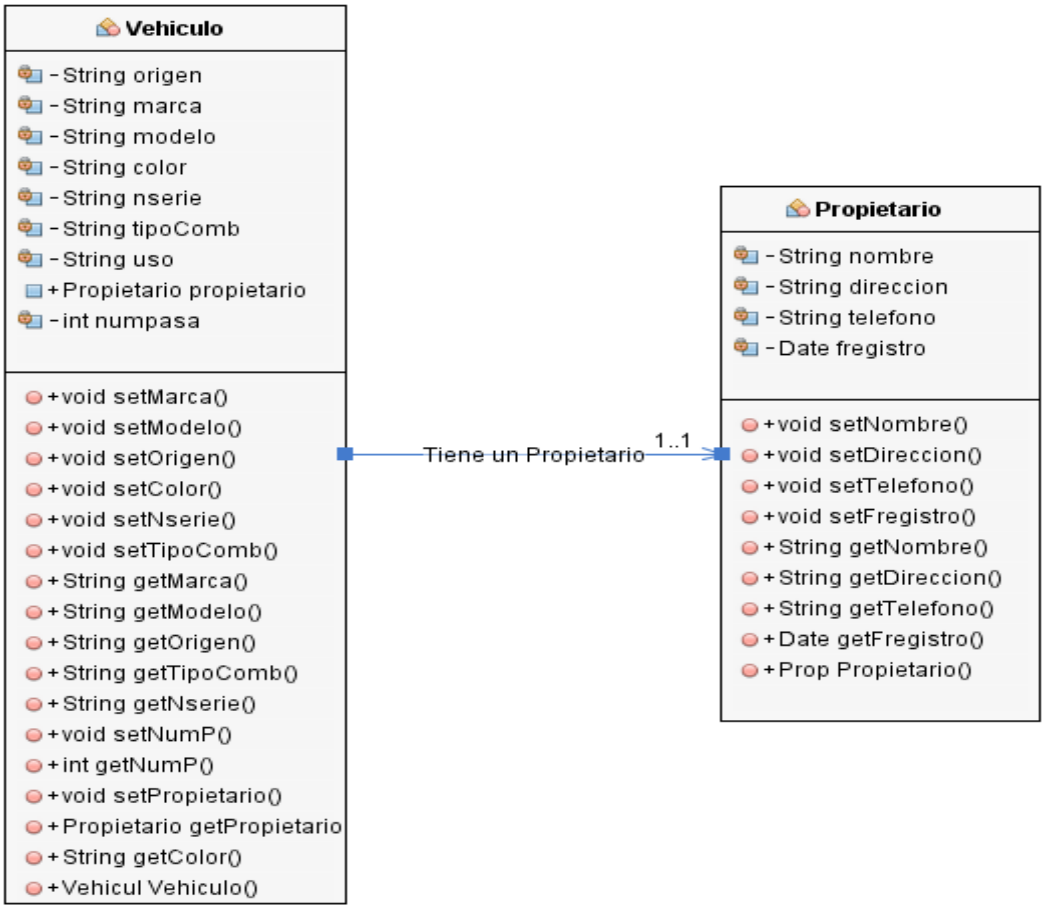
En el siguiente ejemplo se muestra como generar un formulario que incluye una tabla con los datos de varios teléfonos

```
public class TablaTfnos extends javax.swing.JFrame{
 /** Crear un formulario TablaTfnos */
 public TablaTfnos(){
  initComponents();
  setTitle("Teléfonos");
  setSize(500, 300);
 }
  /* Iniciar los componentes */
 private void initComponents(){
  // Datos para la tabla
  Object[][] datos =  new Object[][]{
   {"Alfons González Pérez", "Argentona, Barcelona", new Long(933333333), new Boolean(true)},
   {"Ana María Cuesta Suñer", "Gijón, Asturias", new Long(984454545), new Boolean(false)},
   {"Elena Veiguela Suárez", "Pontevedra", new Long(986678678), new Boolean(false)},
   {"Pedro Aguado Rodríguez", "Madrid", new Long(912804574), new Boolean(true)}    };

  // Cabeceras para las columnas de la tabla
  String[] cabeceras = new String[] {"Nombre", "Dirección", "Teléfono", "Casado"};
  jScrollPanel1 = new javax.swing.JScrollPane();
  jTabla1 = new javax.swing.JTable();
  jTabla1.setModel(new javax.swing.table.DefaultTableModel(datos, cabeceras){
   Class[] tipoColumn = { java.lang.String.class,
                java.lang.String.class,
                java.lang.Long.class,
                java.lang.Boolean.class };

   boolean[] editColum = { false, true, true, true };
   public Class getColumnClass(int indColum){
    return tipoColumn[indColum];     }
```

```
    public boolean isCellEditable(int indFila, int indColum){
      return editColum[indColum];     }
   });

   javax.swing.table.TableColumn colum = null;
   for (int i = 0; i < jTabla1.getColumnCount(); i++)    {
    colum = jTabla1.getColumnModel().getColumn(i);
    if (i < 2)       colum.setPreferredWidth(110);
    else       colum.setPreferredWidth(40);
   }

   jTabla1.addMouseListener(new java.awt.event.MouseAdapter()    {
    public void mouseClicked(java.awt.event.MouseEvent evt)     {       jTabla1MouseClicked(evt);     }    });

   jScrollPanel1 = new javax.swing.JScrollPane(jTabla1);
   addWindowListener(new java.awt.event.WindowAdapter()    {
    public void windowClosing(java.awt.event.WindowEvent evt)     {       exitForm(evt);     }
   });
   getContentPane().add(jScrollPanel1, java.awt.BorderLayout.CENTER);   }//fin del método initComponents
   private void jTabla1MouseClicked(java.awt.event.MouseEvent evt){
   java.lang.Object datoCelda =jTabla1.getValueAt(jTabla1.getSelectedRow(), jTabla1.getSelectedColumn());
System.out.println(datoCelda);  }
  /** Salir de la aplicación */
  private void exitForm(java.awt.event.WindowEvent evt){
    System.exit(0);  }

  public static void main(String args[]){
    new TablaTfnos().setVisible(true);  }
  // Declaración de variables
  private javax.swing.JScrollPane jScrollPanel1;
  private javax.swing.JTable jTabla1;
}
```

Procedimiento:

Con base en el ejemplo anterior genere una aplicación que permita almacenar en una tabla los datos de varios objetos tipo Vehiculo, junto con los datos de sus propietarios



Evaluación:

Criterio	Ejercicio 4
1. El alumno codificó las clases necesarias	SI NO
2. El alumno codificó el formulario	SI NO
3. Funcionó el almacenamiento de registros en la tabla	SI NO
Puntaje Total_____	0 1 2 3

Cuestionarios

1. ¿Qué significa la palabra null?
2. ¿Para qué se usa la palabra new?
3. ¿Qué es una clase envoltorio?
4. ¿Para qué sirve el "recolector de basura"?
5. ¿Qué es un dato primitivo?
6. ¿Cómo reconocer a un dato de tipo primitivo?
7. ¿Para qué se usa la palabra static?
8. ¿Cuál es la función de javadoc?
9. ¿para que se usan las etiquetas de los comentarios?
10. Explica el uso de las siguientes etiquetas:
    a. @see
    b. @author
    c. @exception
    d. @since
    e. @param

    package areas;
11. ¿Cómo se crea un botón?
12. ¿Qué es un jButton?
13. ¿Para qué sirve el método addActionListener?
14. Lugar donde se puede escribir texto o puede ser modificado por el programa jTextfield
15. De dos ejemplos de aplicaciones que proporcionan objetos de retorno que contienen el valor introducido por el usuario.
16. Cuál es la diferencia entre un sistema basado en el uso de recursos y java?
17. Para que sirve jTabbedPane en el uso de paneles tabulados.
18. Dentro del entorno de ventanas que se utiliza para el dialogo mensaje y el dialogo de confirmación.
19. Que se tiene que hacer si se quiere poner un Array de Strings en una jList?
20. Para que sirve un jTabbedPane?
21. Describe los métodos de jTextField.
22. Cuál es la diferencia entre las cajas jComboBox y jList.
23. En una jList permite la selección múltiple? Explica porque.
24. Cuál es la característica principal de Buttongroup?
25. Cuál es la intensión del TextPane()
26. Cuál es el funcionamiento de las casillas de verificación y con que método se hacen.
27. Para que sirve un setBorder()
28. Funcionamiento de un jButton.
29. Funcionamiento del abstractButton.
30. Para qué sirve el método get constructor()
31. Menciona la diferencia entre las cajas de java y de windows.
32. Como se crea una casilla de verificación.
33. Funcionamiento de jScrollPane
34. ¿Cuál es la forma directa de implementar un jPopupMenu?
35. ¿Qué biblioteca permite la generación de dibujos de manera razonablemente sencilla?
36. ¿Cuál es el enfoque más directo cuando se desea obtener una superficie de dibujo?
37. ¿Qué sucede cuando se invoca el método paintComponent( )?
38. ¿Qué es lo primero que hay que hacer al superponer paintCompnent( )?
39. ¿Qué es una caja de dialogo?
40. ¿Cuál es el propósito de una caja de dialogo?
41. ¿Cómo se crea una caja de dialogo?
42. ¿Cómo funciona un jDialog?
43. Mencione una diferencia significativa al llamar a windowClosing( ) y como funciona.
44. ¿Por qué no es frecuente ver *Applets* que hagan uso de cajas de dialogo?
45. ¿Qué sucede cuando los statics solo pueden estar en el nivel externo de la clase?
46. ¿Qué operaciones encapsula el jFilechooser de Java?
47. ¿Qué diálogo se invoca para abrir archivos?
48. ¿Qué diálogo se invoca en el caso de salvar archivo?
49. ¿Es posible que un componente que pueda tomar texto, también pueda tomar texto HTML?
50. ¿Qué es necesario para poder tomar texto HTML?
51. ¿En dónde se puede usar texto HTML?
52. ¿Cuál es la función de un deslizador?
53. ¿Cuál es la diferencia entre el jProgressBar y el jSlider?

Relaciones las columnas

Preguntas	Respuestas
1. ( ) Elemento de un objeto que determina su comportamiento 2. ( ) Concepto que facilita la reutilización de código 3. ( ) Método que se hereda de Object 4. ( ) Tipo de clase que no puede heredar a otras clases 5. ( ) Tipo de atributo que no puede ser modificado 6. ( ) Tipo de atributo del conjunto de objetos y no de cada objeto 7. ( ) Característica de la POO que oculta procesos y datos 8. ( ) Elemento del objeto que determina su estado 9. ( ) Tipo de Clases que agregan comportamiento a los datos primitivos 10. ( ) Si un atributo es privado, solo se puede usar desde otras clases con los métodos ... 11. ( ) Se le llama así a los métodos que nos permiten modificar el estado de los objetos. 12. ( ) Palabra reservada usada para referirse a los datos y métodos de la superclase 13. ( ) Palabra reservada que sirve para entregar un dato de un método a otro método 14. ( ) Método único en la clase que recibe como argumento un arreglo de String 15. ( ) Clase que permite convertir un dato tipo String a int	A. envoltorios B. final C. super D. Integer E. Herencia F. accesor G. Encapsulamiento H. Interfaces I. Boolean J. abstract K. objetos L. static M. mutator N. atributos O. polimorfismo P. return Q. Float R. this S. equals T. clases U. Int V. main W. void X. métodos Y.

Preguntas	Respuestas
1. ( ) ¿Qué son los Applets?	A. componentRisized
2. ( ) Es una restricción de las Applets	B. addXXXListener() y removeXXXListener
3. ( ) Es una ventaja de los Applets	C. Son pequeños programas que se ejecutan dentro del navegador web.
4. ( )¿Cómo se construyen los Applets?	D. No hay aspectos de instalación. Tiene interdependencia completa de la plataforma.
5. ( ) Componente que soporta el evento ActionListener	E. AdjustmentEvent
6. ( ) Se invoca cada vez que se visualiza un Applet en el navegador para permitirle empezar sus operaciones normales (especialmente las que se apagan con stop ()).	F. JMenuItem y sus derivados
7. ( ) Que evento soporta el componente JScrollBar	G. No puede tocar el disco local: esto significa escribir o leer

8.	( ) ¿Qué incluyen todos los componentes swing?	H. Utilizando un marco de trabajo de aplicación. Se hereda de JApplet y se superpone los métodos apropiados
9.	( ) Método de la interfaz ComponentListener	I. start
10.	( ) Método de la interfaz ContainerAdapter	J. componentRemoved

Actividades Complementarias

Instrucciones:
Codifica el siguiente ejemplo de herencia y entrega al profesor:
El diagrama de clases de la jerarquía
La explicación línea a línea de la clase Main
Explica el papel de la variable estática numreg (para qué se usa)

```
package ejherencia;
import java.util.*;
import java.text.*;
public class Persona {
//Inicializando a las variables estaticas o de clase
static {numreg=0;}
   public static int numreg;
   private String nom;
   private Date fechaNac;
   private int ife;
   private int numCredencial;

   public Persona(){
   nom="Maria Perez";
   fechaNac=new Date();
   ife=0;
   numreg++;
   }
   public Persona(String n, Date fn,int ife){
      nom=n;
      fechaNac=fn;
      this.ife=ife;
      numreg++;
   }

   public void setNombre(String n){
      nom=n;
   }
   public String getNombre(){
      return nom;
   }
   public void setFechNac(Date n){
      fechaNac=n;
   }
   public Date getFechNac(){
      return fechaNac;
   }
   public void setIfe(int ife){
      this.ife=ife;
   }
   public int getIfe(){
      return ife;
   }
   public void setCredencial(int cre){
      numCredencial=cre;
   }
   public int getCredencial(){
      return numCredencial;
   }

   public void imprime(){
      DateFormat df=  DateFormat.getDateInstance();
      System.out.println("Nombre:"+nom);
      System.out.println("fecha
Nacimiento:"+df.format(fechaNac));
      System.out.println("Folio IFE:"+ ife);
   }
}
```

```
package ejherencia;
```

```
import java.util.*;
public class Alumno extends Persona implements
Registro {
   private int mat;
   private int sem;
   private String lic;

   public Alumno(String nom,Date fc, int ife,int mat,int
sem, String lic){
      super(nom,fc,ife);
      this.mat=mat;
      this.sem=sem;
      this.lic=lic;
      tCredencial();
   }

   public String verdatos(){
      String yo = new String(this.getNombre()+"
"+this.getFechNac() +" " + this.getIfe()+""+ this.mat+""
+ this.sem +" "+ this.lic);
   return yo;
   }
public void tCredencial(){
System.out.println("Tramitando credencial de alumno
");
setCredencial(Persona.numreg);
//Envio información a registro de becas
}
}
```

```
package ejherencia;
public interface Registro {
   public void tCredencial();
 }
```

```
package ejherencia;
import java.util.*;
public class Empleado extends Persona implements
Registro{
   private String dep;
   private String rfc;
   public Empleado(String nom, Date fe, int ife, String
dep){
      super(nom,fe,ife);
      this.dep=dep;
      tCredencial();
   }

    public void setDep(String d){
      dep=d;
   }
   public String getDep(){
      return dep;
   }
    public void setRfc(String r){
      rfc=r;
   }
   public String getRfc(){
      return rfc;
   }

   public void tCredencial(){
   System.out.println("Tramitando Credencial de
Empleado");
   setCredencial(Persona.numreg);
   //Envio datos a registro de IMSS o ISEMYM
   }
   @Override
   public void imprime(){
   super.imprime();
   System.out.println("RFC:"+rfc);
   System.out.println("Departamento"+dep);
   }
}
```

```
package ejherencia;
import java.util.*;
public class Administrador extends Empleado{
   private int nContrato;
   private Date vigencia;
   public Administrador(String nom, Date fc, int ife,
String dep,  int nContrato, Date vigencia){
      super(nom, fc, ife, dep);
      this.nContrato = nContrato;
      this.vigencia = vigencia;
   }

   public void setnContrato(int nContrato){
   this.nContrato = nContrato;
   }
   public int getnContrato(){
   return nContrato;
   }
   public void setVigencia(Date vigencia){
   this.vigencia = vigencia;
   }

   public Date GetVigencia(){
   return vigencia;
```

```
  }    }
```

```
package ejherencia;
import java.util.*;
import java.text.*;
public class Profesor extends Empleado {
   private Date feIn;
   private int hrs;

   public Profesor(String nom, Date fec,int ife,String
dep, Date feI,int hr){
      super(nom,fec,ife,dep);
      feIn=feI;
      hrs=hr;
   }
   public void setFeI(Date fI){
      feIn=fI;
   }
   public Date getFeI(){
      return feIn;
   }
   @Override
   public void imprime(){

   super.imprime();
   DateFormat df=
DateFormat.getDateInstance(DateFormat.MEDIUM);
   DateFormat df2=
DateFormat.getDateInstance(DateFormat.LONG);
   System.out.println("Fecha de
Ingreso"+df.format(feIn));
   System.out.println("Fecha de
Ingreso"+df2.format(feIn));
   System.out.println("Numero de Horas Clase"+hrs);
   }     }
```

```
package ejherencia;
import java.io.*;
import java.util.*;
import java.text.*;
public class Main {
   public static void main(String[] args) {
      try{
         BufferedReader leer = new
BufferedReader(new InputStreamReader(System.in));
         String nom,lic;
         Date fec=new Date("1985/04/12");
         Date fM;
         int op1,op2,opA=1;
         Alumno inscritos[]= new Alumno[100];
         Empleado nomina[][] = new Empleado[2][];
         int alumnos=0;
         nomina[0]= new Profesor[15];
         nomina[1]= new Administrador[5];
do{
   System.out.println("Que Dato desea Almacenar");
   System.out.println("1 Alumnos \n 2 Empleados \n 3
Salir");
   op1=Integer.parseInt(leer.readLine());
   switch(op1){
      case 1: System.out.println("Ha seleccionado
almacenar datos de Alumnos ");
            do{
         if(alumnos<100){
         System.out.println("Introduce el nombre de un
Alumno");
         nom=leer.readLine();
         System.out.println("Introduce la fecha de
nacimiento del Alumno");
         fM=new Date(leer.readLine());
         Date feIn= new Date();
         inscritos[alumnos] = new
Alumno(nom,fM,6876768,6768,5,"Computación");
         System.out.println("deseas continuar 1:SI
0:NO");
         opA=Integer.parseInt(leer.readLine());
         alumnos++;}
               else { System.out.println("Su almacén de
datos está lleno");}
            }while (opA!=0);
      for (int i=0;i<100;i++)
      {
         if (inscritos[i]!=null){
         System.out.println("Alumno:"+i);
       inscritos[i].imprime();}
         else
            break;       }
      break;
      case 2: System.out.println("Que Desea
almacenar? \n 1: Profesores \n 2: Administradores \n
3:salir");
            op2=Integer.parseInt(leer.readLine());
            switch (op2){   case 1: System.out.println("Ha
seleccionado almacenar datos de Profesores");break;
 case 2: System.out.println("Ha seleccionado
almacenar datos de Administradores");break;
  default:  System.out.println("Opcion no
valida");break;
```

```
        }
        break;
    case 3:  System.out.println("Ha seleccionado
terminar el menu");break;
    default:  System.out.println("Opcion no valida");

  }
```

```
}while (op1!=3);
    }
     catch(IOException e){
      System.out.println(e.getMessage());
     }  }
}
```

Explicar el siguiente Código (Considere que el signo + significa concatener o unir)

```
1.  /**  @author Irene  */
2.  public class Circulo {
3.  float radio;
4.  public Circulo(){    radio=10;   }
5.  public Circulo(float radio){       this.radio=radio;   }
6.  public void imprime(){
7.        System.out.println ("El circulo tiene los siguientes datos:"
8.              + " \n Radio: "+ radio+           "\n diametro: "+ calcDiam()+
9.              "\n périmetro:  "+ calcPer()+           "\n área: " + calcArea());
10.             }
11.   public float calcArea(){     return (float) Math.PI*(radio*radio);  }
12.   public float calcDiam(){     return radio*2;  }
13.   public float calcPer(){     return (float)Math.PI*calcDiam();  }
14.  //Complete los métodos setters y getters
```

Resuelva los siguientes ejercicios con Java:

1. Una agencia automotriz está realizando descuentos a todos sus clientes que realicen una compra de un automóvil último modelo, estos descuentos dependen del modelo del automóvil. Calcule el costo de pago dependiendo de la modalidad de pago. Tome en cuenta la siguiente tabla:

Tipo de Automóvil	Precio sin descuento	Crédito	Contado
Chevy	110,000.00	6%	13%
Cutlas	130,000.00	8%	15%
Fiesta	109,000.00	9%	15%
Cavalier	150,000.00	10%	16%
Taurus	145,000.00	9%	15%

2. Haga una aplicación que muestre el salario anterior y la modificación del mismo cuando la empresa aplique la siguiente tabla de incremento salarial:

Ingreso Actual	Aumento
X ≤1000	12%
1000<X≤2500	10%
2500<X≤3500	9%
3500<X≤5000	8%
X≥5000	5%

3. Construya un programa, que dado la temperatura en grados Farenheit, determine el deporte que es apropiado practicar a esa temperatura, teniendo en cuenta la siguiente tabla:

Deporte	Temperatura
Natación	temp>85
Tenis	70<temp≤85
Golf	32<temp≤70
Esquí	10<temp≤32
Marcha	temp≤10

4. Haga una aplicación que imprima los números pares comprendidos entre el 1 y el 50
5. Haga una aplicación que muestre los números múltiplos de 5 entre el 20 y el 100
6. Genere la tabla de multiplicar del numero que el usuario indique por el teclado
7. Calcular el importe a pagar (pidiendo el precio unitario del artículo y la cantidad comprada) para 10 artículos
8. Calculo del sueldo neto de un trabajador (pidiendo por teclado el sueldo diario y los días trabajados)

Bibliografía

1. SZnajdleder Pablo Augusto, "Algoritmos a fondo con implementaciones en C y Java" AlfaOmega, Buenos Aires 2012

2. Cairó Osvaldo, "Fundamentos de Programación Piensa en C", Prentice Hall, México 2006

3. Cevallos Fco. Javier, "Enciclopedia de C", Editorial. Addison Wesley.

4. Deitel H. M. / Deitel P. J., "Como programar en C /C++". Editorial Prentice Hall Hispanoamericana.

5. Joyanes Aguilar Luís y Salonero Martínez Ignacio. "Programación en C", Editorial McGraw-Hill.

6. Tremblay Jean-Paul, Bunt Richard, "Introducción a la Ciencia de las computadoras enfoque", Mc Graw Hill, México 1988.

7. Vázquez Peña Mario "Introducción al lenguaje C", Universidad Autónoma Chapingo, México 1997