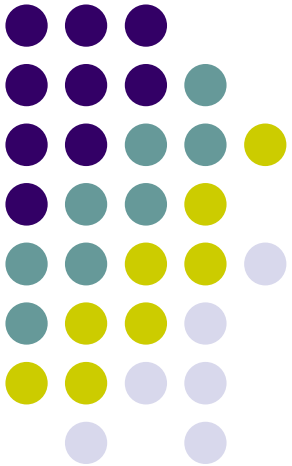
# INGENIERÍA EN SISTEMAS Y COMUNICACIONES

## UDA: PROGRAMACIÓN ORIENTADA A OBJETOS

## TEMA: APLICACIÓN DE LA POO EN EL LENGUAJE JAVA



ELABORÓ: DR. EN C. HÉCTOR RAFAEL OROZCO AGUIRRE
CU UAEM VM

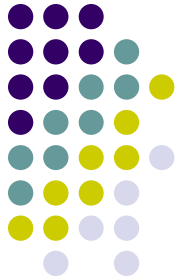
# Programa de Estudios Por Competencias
# Programación Orientada a Objetos

## IDENTIFICACIÓN DEL CURSO

**ISMO ACADÉMICO:**
Universitario UEAM Valle de México

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **ma Educativo**:<br>ía en Sistemas y Comunicaciones | | | **Área de docencia:**<br>Ciencias de la ingeniería | | | | | | |
| **ción por los H.H.<br>os Académico y de<br>no** | | **Fecha:**<br>Julio de 2006 | **Programa elaborado por:**<br>Morales Escobar Saturnino Job y Rodríguez Pérez Ivonne<br>**Actualizado por:**<br>Orozco Aguirre Héctor Rafael | | | | | **Fecha de elaboraci**<br>Julio de 2006<br>**Fecha de actualiza**<br>26/Junio/2012 | |
| **ve** | **Horas de teoría** | **Horas de práctica** | **Total de horas** | **Créditos** | **Tipo de Unidad de Aprendizaje** | **Carácter de la Unidad de Aprendizaje** | **Núcleo de formación** | **Mod** | |
| | 2 | 4 | 6 | 8 | Obligatoria | Curso | Básico | Preser | |
| **uisitos**<br>entos de Programación y Algoritmos y<br>ras de Datos | | | **Unidad de Aprendizaje Antecedente**<br>Algoritmos y Estructuras de Datos | | | **Unidad de Aprendizaje Consecue**<br>Programación Avanzada | | | |

**mas educativos en los que se imparte:** Ingeniería en Sistemas y Comunicaciones

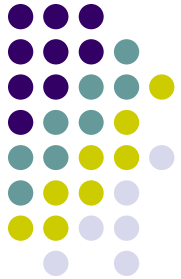


La estructura de un archivo .java debe ser como sigue:

- Una única sentencia de paquete (opcional), package nombrepaquete;
- Las sentencias de importación deseadas (opcional), import paquete;
- declaración e implementación de una sola clase pública o la declaración de una interfaz.
- Las clases privadas del paquete (opcional).

Se recomienda tener solo la declaración de una clase o interfaz por archivo y se haga el uso de los paquetes para un mejor manejo y entendimiento.

# Clases y objetos

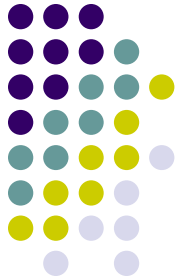


Una clase en Java se declara de la siguiente manera:

- [modificador] class NombreClase [extends ClasePadre] [implements Interfaz1, …, InterfazN] {

  cuerpo_de_la_clase

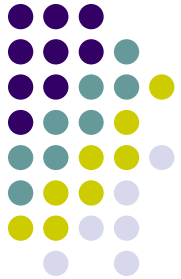
  }



Una clase contiene como mucho tres tipos de miembros:

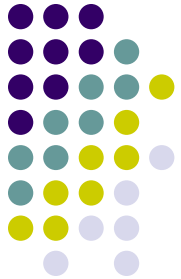
- instancias de objetos o de tipos simples (atributos)
- métodos (funciones y procedimientos)
- otras clases

No existen variables globales y el programa principal no es nada más que un método de una clase.



Un archivo de Java debe tener el mismo nombre que la clase que contiene con la extensión *.java.* Por ejemplo: la clase *MiClase* se guardaría en un archivo que se llamase *MiClase.java*.

Hay que tener presente que en Java se diferencia entre mayúsculas y minúsculas; el nombre de la clase y el de archivo fuente han de ser exactamente iguales.



Los modificadores de clase son los siguientes:

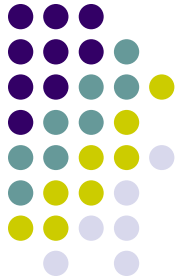
- **public:** la clase es accesibles desde otras clases.
- **abstract:** la clase no se instancia, sino que se utiliza como clase base para la herencia, debe tener al menos un método abstracto .
- **final:** no se puede extender la clase.

Los atributos de una clase pueden ser tipos simples u objetos. Estos se declaran de la siguiente manera:

- [modificador_acceso] [modificador_tipo_a] tipo nombre;

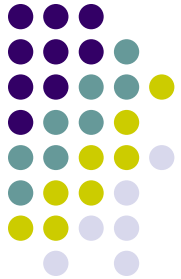
Los métodos de una clase se declaran como sigue:

- [modificador_acceso] [modificador_tipo_m] tipo_devuelto nombre ( [parámetros] ) [throws Excepción1, … , ExcepciónN]{
  cuerpo_del_método
  return [valor];
  }



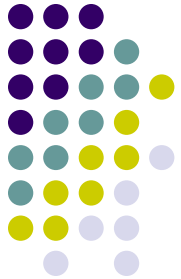
Los modificadores de acceso para atributos y métodos son:

- **private:** accesible solamente desde la propia clase.
- **protected:** accesible solamente desde la propia clase, dentro del mismo paquete, o desde clases derivadas.
- **public:** accesible donde la clase es visible.
- **ningún modificador:** accesible solamente desde la propia clase o dentro del mismo paquete.



Los modificadores de tipo para los atributos son los siguientes:

- **final:** declara constantes.
- **static:** declara miembros de clase que pertenecen a la clase y no a instancias de objetos, es decir, todos los objetos de la clase acceden a la misma cosa.
- **transient:** excluye un miembro del proceso de conversión en un flujo de bytes si el objeto se salva al disco o se transmite por una conexión.
- **volatile:** ordena a la máquina virtual de Java que no use ningún tipo de cache para el miembro, así es más probable (aunque no garantizado) que varios hilos vean el mismo valor de una variable; declarando variables del tipo primitivo aseguramos que las operaciones básicas sean atómicas.



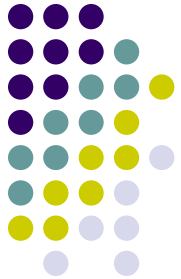
Los modificadores de tipo para métodos son los siguientes:

- **abstract:** el método todavía no está completo, es decir, no se puede instanciar objetos antes de que se haya implementado el método en una clase derivada.
- **static:** el método pertenece a la clase y no a un objeto de la clase, un método estático puede acceder solamente miembros estáticos.
- **final:** no se puede sobrescribir el método en una clase derivada.
- **synchronized:** el método pertenece a una región crítica.
- **native:** propone una interfaz para llamar a métodos escritos en otros lenguajes, su uso depende de la implementación de la máquina virtual de Java.

Toda clase tiene un constructor vacío por default. Un constructor es un método que inicia un objeto inmediatamente después de su creación y puede haber más de un constructor.

El constructor tiene exactamente el mismo nombre de la clase que lo implementa; no puede haber ningún otro método que comparta su nombre con el de su clase. Una vez definido, se llamará automáticamente al constructor al crear un objeto de esa clase (al utilizar el operador *new*). El constructor no devuelve ningún tipo, ni siquiera *void*. Su misión es iniciar todo estado interno de un objeto (sus atributos), haciendo que el objeto sea utilizable inmediatamente; reservando memoria para sus atributos, iniciando sus valores, etc. La forma de declararlo es como sigue:

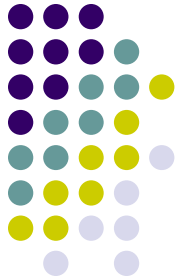
- [modificador_acceso] NombreClase([parámetros]) [throws Excepción1, …, ExcepciónN] {
  cuerpo_constructor
  }



Si una clase tiene un método principal este se declara de la siguiente manera:

```
public static void main(String args[]){
    cuerpo_del_metodo
}
```

En el cuerpo del método principal por lo general se declara una instancia de la clase y se inicializa, así se puede mandar ejecutar la clase.

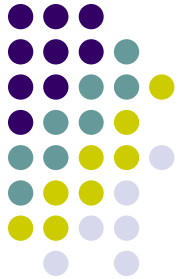


Cuando se declara una variable de una clase y esta se inicializa por constructor se crea un objeto. Para crear el objeto se usa el operador *new* de la siguiente manera:

- Clase variable = new Clase([parametros]);

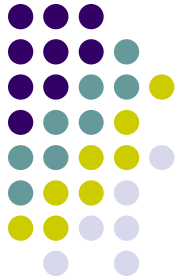
Para acceder a los métodos y atributos de la clase se usa el operador punto (.).

- variable.metodo([parametros]); o variable.atributo;



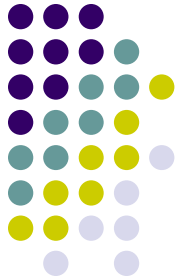
Cada objeto tiene por defecto una referencia llamada *this* que proporciona acceso al mismo. Obviamente, la referencia *this* no existe en métodos estáticos. Cada objeto (menos la clase object) tiene una referencia a su clase padre llamada *super*.

*this* y *super* se pueden usar especialmente para acceder a variables y métodos. Para facilitar las definiciones de constructores, un constructor puede llamar en su primer sentencia o bien a otro constructor con this([parámetros]) o bien a un constructor de su clase padre con super([parámetros]).

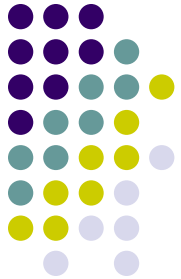


Cuando un objeto no va a ser utilizado, el espacio de memoria que utiliza ha de ser liberado, así como los recursos que poseía, permitiendo al programa disponer de todos los recursos posibles. A esta acción se la da el nombre de *destrucción del objeto*.

En Java la destrucción se puede realizar de forma automática o de forma personalizada, en función de las características del objeto.



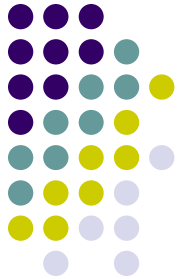
Java proporciona el recolector de basura, que será el encargado de liberar una zona de memoria dinámica que había sido reservada mediante el operador *new*, cuando el objeto ya no va a ser utilizado más durante el programa (por ejemplo: sale del ámbito de utilización, o no es referenciado nuevamente). El recolector de basura se ejecuta periódicamente, buscando objetos que ya no estén referenciados.



Cuando se libera memoria de manera personalizada se llama al método *finalize()*. Este método es de tipo *protected void* y por lo tanto deberá de sobrescribirse con este mismo tipo.

```
protected void finalize() {
    liberación_de_la_memoria
}
```

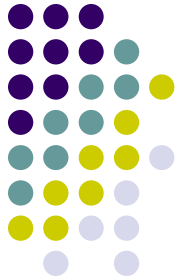
# Interfaces



Las interfaces Java son expresiones puras de diseño. Se trata de auténticas conceptualizaciones no implementadas que sirven de guía para definir un determinado concepto (clase) y lo que debe hacer, pero sin desarrollar un mecanismo de solución.

Se trata de declarar métodos abstractos y constantes que posteriormente puedan ser implementados de diferentes maneras según las necesidades de un programa.

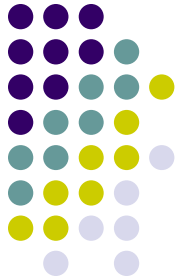
Todas las declaraciones de objetos en una interfaz automáticamente son finales y estáticos y los métodos de una interfaz son implícitamente públicos y abstractos, aunque no se haya descrito explícitamente.



Es posible extender una interfaz a partir de más de una interfaz.

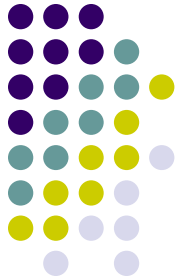
- interface NombreInterfaz [extends Interfaz1, …, InterfazN] {
  método1;
  …
  métodoN;
  }

Una clase puede implementar varias interfaces al mismo tiempo (aunque una clase puede extender como mucho una clase). Se identifican las interfaces implementadas con *implements* después de una posible extensión (*extends*) de la clase.



Las interfaces carecen de funcionalidad por no estar implementados sus métodos, por lo que se necesita algún mecanismo para dar cuerpo a sus métodos.

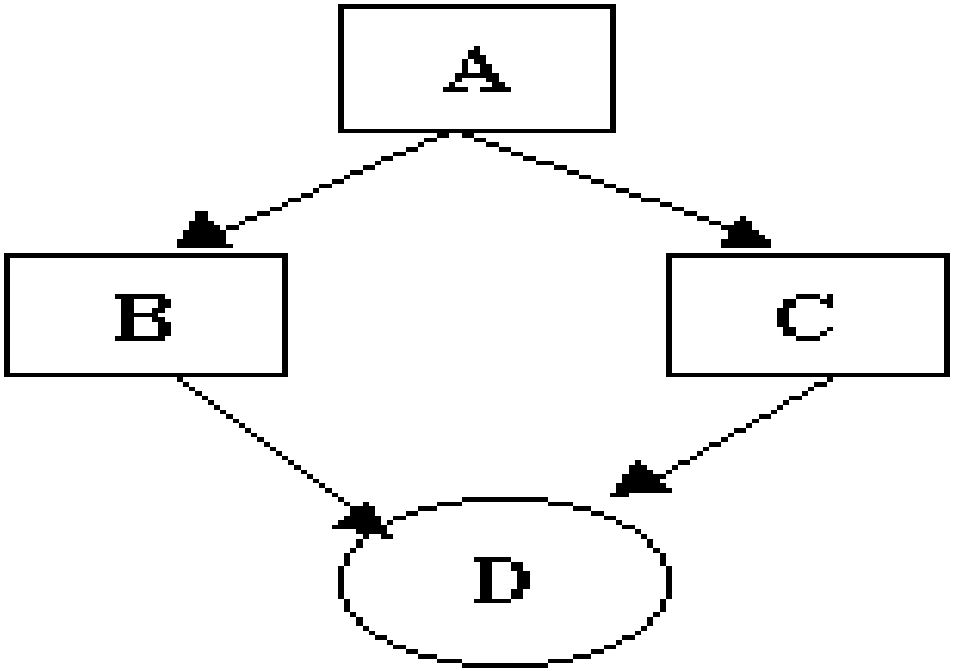
La palabra reservada *implements* utilizada en la declaración de una clase indica que la clase implementa la interfaz, es decir, que asume las constantes de la interfaz, y codifica sus métodos. Una interfaz no puede implementar otra interfaz, aunque sí extenderla (ampliándola) con *extends*.



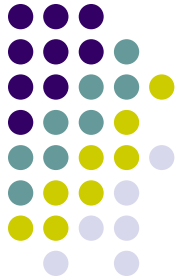
Java es un lenguaje que incorpora herencia simple de implementación pero que puede aportar herencia múltiple de interfaz. Esto posibilita la herencia múltiple en el diseño de los programas Java.

Una interfaz puede heredar de más de una interfaz antecesora.

Una clase no puede tener más que una clase antecesora, pero puede implementar más de una interfaz.




Ejemplo de herencia múltiple



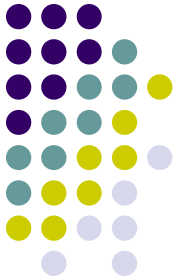
Para realizar lo que se expresa en el diagrama anterior, se debe hacer:

- interface A{ }
- interface B extends A{ }
- interface C extends A{ }
- class D implements B,C{ }

# Paquetes

Los paquetes son el mecanismo por el que Java permite agrupar clases, interfaces, excepciones y constantes. De esta forma, se agrupan conjuntos de estructuras de datos y de clases con algún tipo de relación en común.

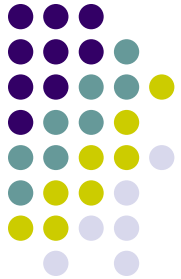
Con la idea de mantener la reutilización y facilidad de uso de los paquetes desarrollados es conveniente que las clases e interfaces contenidas en los mismos tengan cierta relación funcional. De esta manera los desarrolladores ya tendrán una idea de lo que están buscando y fácilmente sabrán qué pueden encontrar dentro de un paquete.



Para declarar un paquete se utiliza la sentencia *package* seguida del nombre del paquete que se este creando:

- package paquete;

```
package animales.aves;
public class Paloma {
    cuerpo_de_la_clase
}
```



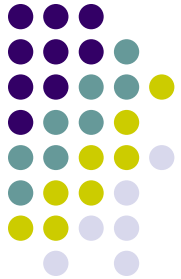
Con el fin de importar paquetes ya desarrollados se utiliza la sentencia *import* seguida del nombre del paquete o paquetes a importar. Se pueden importar todos los elementos de un paquete o sólo algunos.

Para importar todas las clases e interfaces de un paquete se utiliza el metacaracter *:

- import paquete.*;

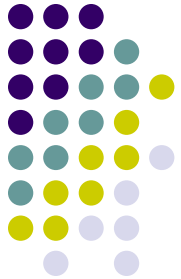
También existe la posibilidad de que se deseen importar sólo algunas de las clases de un cierto paquete o subpaquete:

- import paquete.subpaquete1.subpaquete2.Clase1;



Para acceder a los elementos de un paquete, no es necesario importar explícitamente el paquete en que aparecen, sino que basta con referenciar el elemento tras una especificación completa de la ruta de paquetes y subpaquetes en que se encuentra.

- paquete.subpaq1.subpaq2.Clase_o_Interfaz.elemento



| elemento | private | sin modificador | protected | public |
|---|---|---|---|---|
| **En la misma clase** | Sí | Sí | Sí | Sí |
| **En una clase en el mismo paquete** | No | Sí | Sí | Sí |
| **En una clase hija en otro paquete** | No | No | Sí | Sí |
| **En una clase no hija en otro paquete** | No | No | No | Sí |

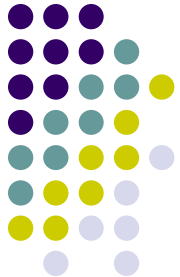
Visibilidad dentro de un paquete

El lenguaje Java proporciona una serie de paquetes que incluyen ventanas, utilidades, un sistema de entrada/salida general, herramientas y comunicaciones. Algunos paquetes que se incluyen son:

- **java.applet:** Este paquete contiene clases diseñadas para usar con applets. Hay una clase Applet y tres interfaces: AppletContext, AppletStub y AudioClip.

- **java.awt:** El paquete Abstract Windowing Toolkit (awt) contiene clases para generar widgets y componentes GUI (Interfaz Gráfico de Usuario). Incluye las clases Button, Checkbox, Choice, Component, Graphics, Menu, Panel, TextArea y TextField.

- **java.io:** El paquete de entrada/salida contiene las clases de acceso a archivos: FileInputStream y FileOutputStream.

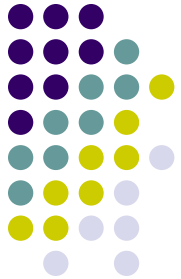


- **java.lang:** Este paquete incluye las clases del lenguaje Java propiamente dicho: Object, Thread, Exception, System, Integer, Float, Math, String, etc.

- **java.net:** Este paquete da soporte a las conexiones del protocolo TCP/IP y, además, incluye las clases Socket, URL y URLConnection.

- **java.util:** Este paquete es una miscelánea de clases útiles para muchas cosas en programación. Se incluyen, entre otras, Date (fecha), Dictionary (diccionario), Stack (pila LIFO), etc.

# Compilación y ejecución



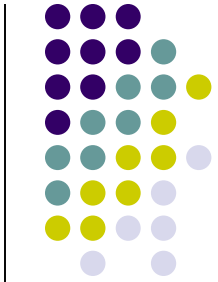
**javac:** Compilador de programas Java.

- javac [opciones] archivo.java

**java:** Interpreta (ejecuta) byte-code Java.

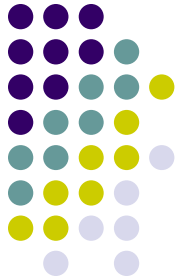
- java [opciones] nombre_de_clase <argumentos>
- java [opciones] -jar archivo_jar <argumentos>

# Tipos de programas Java



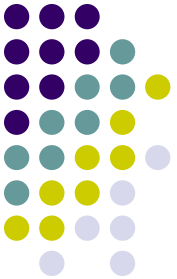
Con Java se pueden construir varios tipos de programas, cada uno con unas características específicas, y que se ejecutan de distintas maneras.

Los principales tipos de programas son los siguientes:

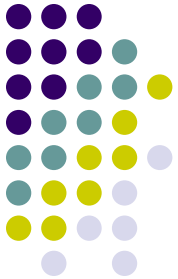


- ***Aplicaciones:*** programas básicos de Java. Se ejecutan en una determinada máquina, por el Java Runtime Enviroment (JRE).
- ***Applets:*** deben incluirse en páginas Web para ser observadas por otra aplicación y se ejecutan cuando el usuario intenta visualizarlas.
- ***JavaBeans:*** son componentes gráficos de Java, que se pueden incorporar a otros componentes gráficos. Se incluyen en la API de Java (paquete java.beans).
- ***JavaScript:*** JavaScript es un subconjunto del lenguaje Java que puede codificarse directamente sobre cualquier documento HTML.
- ***Servlets:*** módulos que permiten sustituir o utilizar el lenguaje Java en lugar de programas CGI (Common Gateway Interface) a la hora de dotar de interactividad a las páginas Web.

# Referencias

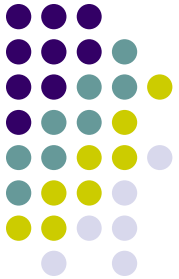



- Deitel Harvey & Deitel Paul, Como Programar en JAVA, 5a edición. Pearson/Prentice Hall, 2008.
- Booch Grady, Rumbaugh James & Jacobson Ivar, UML El Lenguaje Unificado de Modelado (Guía de usuario), Pearson, 2006.
- Luis Joyanes Aguilar, Programación en JAVA 2, 1ª edición. Mc Graw Hill.
- Wang Paul S. Java con Programación Orientada a Objetos y aplicaciones en la WWW.1ª Edición. Thomson Editores.
- Froufe Agustín. JAVA 2 Manual de Usuario y Tutorial. 2ª edición. Alfaomega RA-MA.
- Larman Craig, UML y Patrones (Introducción al análisis y diseño orientado a objetos), Pearson, 2003.
- Herbert Schildt, Java: The Complete Reference, McGraw-Hill Osborne Media, 8 edition, June 22, 2011.
- Wu C. Thomas, Introducción a la Programación Orientada a Objetos. Programación en Java, McGraw-Hill, 2008.



# Guión explicativo

- Esta presentación tiene como fin lo siguiente:
  - Estructura de un archivo .java
  - Clases y objetos
  - Interfaces
  - Paquetes
  - Compilación y ejecución de programas
  - Tipos de programas



# Guión explicativo

- El contenido de esta presentación contiene temas de interés contenidos en la Unidad de Aprendizaje Inteligencia Artificial.

- Las diapositivas deben explicarse en orden, y deben revisarse aproximadamente en 24 horas, además de realizar preguntas a la clase sobre el contenido mostrado.