

UNIVERSIDAD AUTONOMA DEL ESTADO DE MÉXICO  
CENTRO UNIVERSITARIO UAEM ECATEPEC

# Curso Básico de JAVA

M.I.S.C. Cuauhtémoc Hidalgo C.

# Conceptos Básicos

Curso Básico de JAVA

# Conceptos básicos

- Los lenguajes *orientados a objetos* basados en clases están basados en dos tipos de entidades, las **clases** y los **objetos**.
- Una clase define las propiedades y comportamiento que comparten un conjunto de objetos.
- Un objeto, por otra parte es una instancia de la clase, representando un individuo en particular; esta instancia tiene las propiedades y comportamiento definidos por la clase de la cual es miembro.

# Conceptos básicos

Credito
Numero : Long
Tipo : Single
FechaExpiración : Date
Autorizar()

- **Abstracción:** se enfoca en las características esenciales de un objeto relativo a la perspectiva del observador
- **Modularidad:** agrupa abstracciones en unidades discretas. Es la propiedad de un sistema que se ha descompuesto en un conjunto de modulos coherentes y poco acoplados
- **Herencia:** es clasificar u ordenar abstracciones. Las abstracciones forman una jerarquía. Los objetos pueden heredar propiedades de otros objetos. La herencia puede ser simple o múltiple. (C++: Privado, Protegido, Público)
- **Objeto:** puede ser un objeto físico, un concepto, un evento, o lo que sea que queremos describir (e.g., un coche, un curso, un programa, etc). Un objeto tiene un estado, exhibe un comportamiento bien definido y tiene una identidad única

# Conceptos básicos

- **Encapsulación:** esconde los detalles de la implementación de un objeto
- **Reutilización:** objetos (bien hechos) pueden utilizarse en otros dominios
- **Ejecución:** por medio de propagación de mensajes
- **Mensajes:** el código privado que tiene el objeto puede ser accesado solo por medio de mensajes. El mensaje dice a que objeto se dirige, que procedimiento ejecutar y cuales son los *arguments*
- **Métodos:** es un procedimiento privado de un objeto que dice que hacer con un mensaje y como hacerlo. Como cada objeto tiene sus propios métodos, los objetos pueden responder diferente al mismo mensaje.
- **Respuestas:** una vez recibido un mensaje, el objeto manda su respuesta a otros objetos o al sistema.

# Tipos de Almacenamiento

Curso Básico de JAVA

# Diferentes Tipos de Almacenamiento

- Hay 6 diferentes lugares para almacenar datos:
  - 1 Registers (Registros)
  - 2 The stack (La pila)
  - 3 The heap (La cabeza)
  - 4 Static storage (Almacenamiento estático)
  - 5 Constant storage (Almacenamiento constante)
  - 6 Non-RAM storage (Almacenamiento No RAM)

# Diferentes Tipos de Almacenamiento

- Registers (Registros)

Es el más rápido de todos los almacenamientos. Está localizado en los registros del procesador. El usuario no tiene control directo. El compilador de Java los manipula de acuerdo a sus necesidades.

# Diferentes Tipos de Almacenamiento

- The stack ( La pila )

La pila se encuentra en RAM y tiene soporte directo por el microprocesador via stack pointer. El compilador debe saber mientras crea un programa el tamaño y tiempo de vida exacto de todos los datos para ser almacenados en el stack. Mientras algunos almacenamientos existen sobre la pila - En particular los handles - los objetos no son alojados en el stack. El almacenamiento en la pila es más lento que el almacenamiento en los registros.

# Diferentes Tipos de Almacenamiento

- The heap ( La cabeza )

La cabeza es un repositorio de memoria ( también en un área de RAM) de propósito general donde todos los objetos viven. El compilador no necesita saber como en el caso de la pila el tamaño y tiempo de vida exacto de los datos que se estén almacenado en la cabeza. Los objetos son almacenados en la cabeza con **new** cuando el programa es ejecutado. El precio a pagar por ésta flexibilidad es más tiempo para alojar memoria en la cabeza.

# Diferentes Tipos de Almacenamiento

- Static storage (Almacenamiento estático)

El almacenamiento estático ( se encuentra en RAM) contiene datos que están disponibles para el tiempo completo en que un programa se está ejecutando. Se usa la palabra llave static para especificar un elemento particular de un objeto como estático.

# Diferentes Tipos de Almacenamiento

- Constant storage (Almacenamiento constante)  
Los valores constantes son generalmente colocados directamente en el mismo código del programa.

# Diferentes Tipos de Almacenamiento

- Non-RAM storage (Almacenamiento No RAM)

El almacenamiento fuera de RAM los datos pueden existir completamente fuera de un programa cuando no se está ejecutando. Ejemplos:

**Stremead Objects:** Cuando los objetos son convertidos en flujos de bytes generalmente enviados a otra computadora.

**Persistent Objects:** Son objetos colocados en el disco y mantendrán su estado incluso cuando el programa ya haya terminado. El truco de estos tipos de almacenamientos es que se convierten en objetos también en otros medios y pueden ser resucitados en objetos de almacenamiento en RAM cuando sea necesario.

# Características de Java

Curso Básico de JAVA

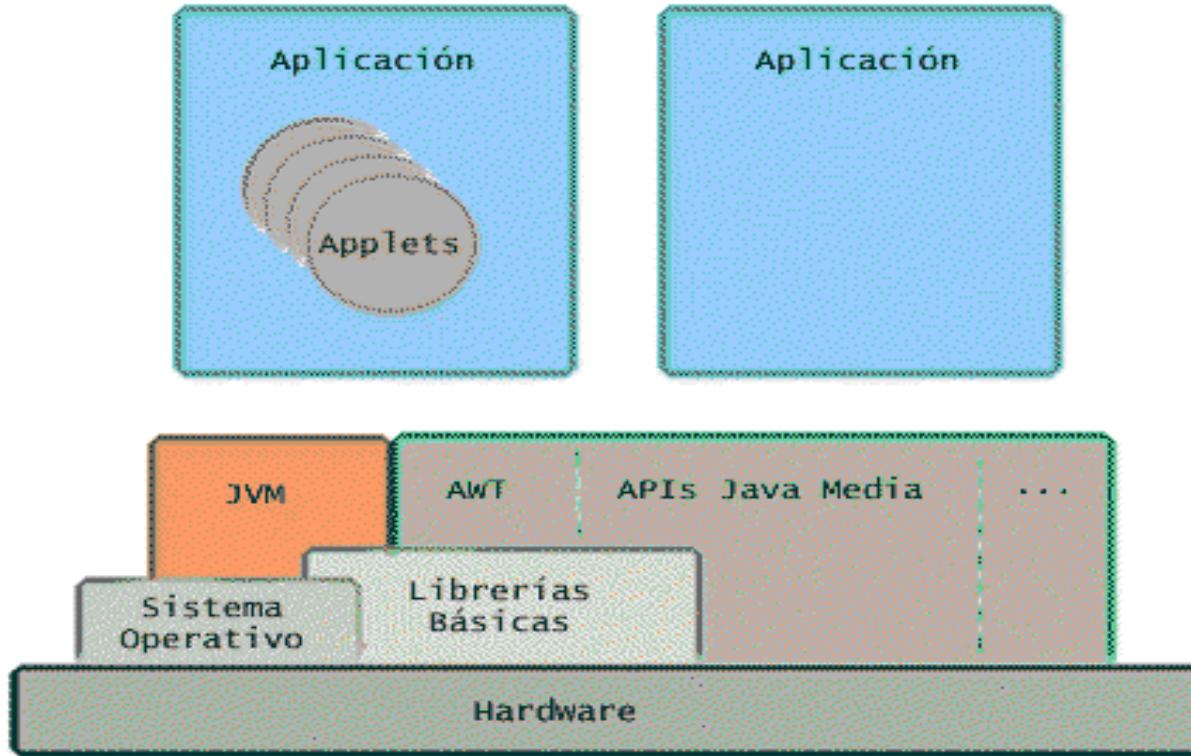
# El lenguaje Java - Características

- **Simple.** Elimina la complejidad de los lenguajes como "C".
- **Orientado a Objetos.**
- **Familiar.** Como la mayoría de los programadores están acostumbrados a programar en C o en C++, la sintaxis de Java es muy similar al de estos.
- **Robusto.** El sistema de Java maneja la memoria de la computadora por ti. No te tienes que preocupar por apuntadores, memoria que no se esté utilizando, etc..
- **Seguro.** El sistema de Java tiene ciertas políticas que evitan se puedan codificar virus con este lenguaje. Existen muchas restricciones, especialmente para los applets, que limitan lo que se puede y no puede hacer con los recursos críticos de una computadora.

# El lenguaje Java - Características

- **Portable.** Como el código compilado de Java (conocido como *byte code*) es interpretado, un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implementado el interprete de Java.
- **Independiente a la arquitectura.** Al compilar un programa en Java, el código resultante es un tipo de código binario conocido como *byte code*. Este códido es interpretado por diferentes computadoras de igual manera, solamente hay que implementar un intérprete para cada plataforma. De esa manera Java logra ser un lenguaje que no depende de una arquitectura computacional definida.
- **Multithreaded.** Un lenguaje que soporta multiples *threads* es un lenguaje que puede ejecutar diferentes líneas de código al mismo tiempo.
- **Interpretado.** Java corre en máquina virtual, por lo tanto es interpretado.
- **Dinámico.** Java no requiere que compiles todas las clases de un programa para que este funcione. Si realizas una modificación a una clase Java se encarga de realizar un *Dynamic Bynding* o un *Dynamic Loading* para encontrar las clases.

# Elementos que forman parte de la arquitectura de Java sobre una plataforma genérica



El código fuente Java se "compila" a un código de bytes de alto nivel independiente de la máquina. Este código (byte-codes) está diseñado para ejecutarse en una máquina hipotética que es implementada por un sistema run-time, que sí es dependiente de la máquina.

# JAVA(Introducción)

## ■ Ventajas de Java:

- Es un lenguaje relativamente sencillo y fácil de aprender
- - Es muy productivo, con poco código puedes hacer mucho más que la misma cantidad en código de C ó C++
- - Posee todas las ventajas de la programación orientada a objetos. (se explica más adelante)

## ■ Desventajas de Java:

- Java es lento comparado con C ó C++, pero en algunas tareas es casi igual de veloz.
- - Java no permite el acceso directo a todos los recursos de hardware de la PC. Aunque esto también podría ser una ventaja ya que así es más fácil.
- - Actualmente hay tantas bibliotecas de clases para java que es casi imposible conocer para qué sirven todas y cada una de ellas.

# El lenguaje Java - Características

- No hay variables globales, ni funciones Globales
- El archivo fuente debe tener el mismo nombre que la clase o interfaz publica, la extensión es .java
- Cada definición de clase o interfaz en un archivo Java es compilada en un archivo separado.
- Archivos ByteCode de java se conocen como archivos de clase.
- Únicamente variables locales y funciones locales.
- Sin preprocesador.
- Toda variable declarada final es constante.

# El lenguaje Java - Características

- Tipos de Datos Primitivos
- No hay punteros
- El valor por omisión de todos los tipos de referencia es null.

# El lenguaje Java - Características

- **Palabras Reservadas :** Java al igual que cualquier otro lenguaje de programación tiene su conjunto de palabras reservadas. Las palabras reservadas no puede ser empleadas como identificadores de variables, funciones, constantes, etc. ya que como su nombre lo dice, son palabras reservadas para el lenguaje de programación. La siguiente tabla es un resumen las palabras reservadas de Java.
- \* indica que la palabra reservada no es actualmente utilizada.

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

# Tipos de Datos y Variables

Curso Básico de JAVA

# Tipos primitivos.

Los tipos primitivos tienen un tratamiento especial. Son usados frecuentemente . La razón es que crear un objeto con new especialmente uno chico como una simple variable no es muy eficiente porque new aloja objetos en el heap.

Para éstos tipos Java hace lo mismo que C y C++ en lugar de crear la variable usando new, una variable “automática” es creada la cual no es un handle. La variable aloja el valor por si mismo y lo coloca en el stack que es mucho más eficiente.

Java determina el tamaño de cada tipo primitivo. Estos tamaños no cambian de una arquitectura de máquina a otra como en la mayoría de los lenguajes. Esto es la razón por la cual los programas en Java son muy portables.

# Tipos primitivos.

Tipo Primitivo	Tamaño	Mínimo	Maximo	Ejemplo
boolean	1-bit	false	true	true
char	16-bit	'\0000'	'\FFFF'	'c'
byte	8-bit	-128	+127	-15
short	16-bit	$-2^{15}$	$+2^{15} - 1$	1024
int	32-bit	$-2^{31}$	$+2^{31} - 1$	42325
long	64-bit	$-2^{63}$	$+2^{63} - 1$	262144
float	32-bit	$3.4 * (10^{-38})$	$3.4 * (10^{38})$	10.52
double	64-bit	$1.7 * (10^{-308})$	$1.7 * (10^{308})$	0.00045
void	-	Universidad Autónoma del Estado de México	-	Void

# Variables

- En java, una variable es una posición con nombre en memoria donde se almacena un valor de un cierto tipo de dato.
- Las variables de tipos de datos básicos, tipos primitivos, almacenan datos correspondientes con el tipo.
- Las variables de clases, variables de objetos, son referencias al objeto

# Variables

- Una variable típicamente tiene un nombre que describe su propósito
- Toda variable utilizada en un programa debe ser declarada previamente
- La definición en Java puede situarse en cualquier parte del programa
- La definición reserva un espacio de almacenamiento en memoria.

# Variables

- El procedimiento para definir una variable de tipo primitivo es escribir el tipo de dato, el identificador, y en ocasiones, el valor inicial que tomará.

```
char respuesta;  
int numero1;  
boolean bisiesto;  
bisiesto = true;
```

# Declaración de Variables

- Es preciso declarar las variables antes de utilizarlas.
- Se puede declarar una variable en tres lugares:
  - En una clase, como miembro de la clase
  - Al principio de un método o bloque de código
  - En el punto de utilización

# Declaración de Variables

- En una clase, como miembro de la clase:

```
class Suerte
{
    int miNumero;
    void entrada()
    {
        miNumero = 29;
    }
    void salida()
    {
        System.out.println("Mi numero de
la suerte es" + miNumero);
    }
}
```

En esta clase, la variable miNumero puede utilizarse en cualquier método.

# Declaración de Variables

## ■ Al principio de un bloque de código:

Este es el medio para declarar una variable en un método o en un bloque de código dentro de un método.

```
long factor(int n)
{
    int k;
    long f;
    f = 1L;
    for (k=1; k<n; k++)
    {
        long aux=10;
        .
        .
    }
    return f;
}
```

En este método las variables k y f están definidas en el método; son variables locales del método y se pueden utilizar en el ámbito del método; la variable aux está definida en el bloque del bucle for, y por tanto su ámbito es el bloque.

# Declaración de Variables

## ■ En el punto de utilización.

Java proporciona gran flexibilidad en la declaración de variables, ya que es posible declarar una variable en el punto donde se vaya a utilizar. Esta propiedad se utiliza mucho en el diseño de bucles, es posible declarar la variable en el momento de su utilización.

```
for (int j=0; j<10; j++)  
{  
    // ...  
}
```

En otros lenguajes, las declaraciones se han de situar siempre al principio del programa, mientras que en java, las declaraciones se pueden mezclar con sentencias ejecutables. Su ámbito es el bloque en el que están declaradas.

# Operadores

Curso Básico de JAVA

# Operadores básicos

- los operadores, usados para formar expresiones
- Sentencia: Expresión acabada en ;
- Asignación '=' Se pueden encadenar

$z = y = x = 0$

- Asignación tras operar

$+ = , \quad - = , \quad * = , \quad / =$

- Binarios: +, -, \*, /, %
- Unarios: -, ++, --

# Relacional y conversión

- Comparadores de igualdad: `==`, `!=`
- Relacionales: `<`, `<=`, `>`, `>=`
- Lógicos: `&&`, `||` (evaluación cortocircuitada)
- `&`, `|` para evaluación completa (poco usados)
- Conversión: (tipo) dato

# El Operador condicional

- Forma abreviada de sentencias `if-else`

`ExprTest ? ExprSí : ExprNo`

- En función del resultado al evaluar `ExprTest` se evaluará y devolverá la correspondiente expresión
- Asignación del menor valor de 2 variables

`valorMenor = x <= y ? x : y;`

Ejemplo

`x=1 ; y=10; z = (x<y)?x+3:y+8;`

asignarían a `z` el valor 4, es decir `x+3`.

# Estructuras de Control

Curso Básico de JAVA

# Sentencia condicional if

```
if (expresión) sentencia  
if (expresion)  
    sentencial1  
else  
    sentencia2
```

```
System.out.print("1/x es ");  
if (x != 0)  
    System.out.println(1 / x);  
else  
{  
    System.out.print("indeterminado.");  
    System.out.println(" x es cero");  
}
```

# Iteración: while y for

while (expresión) sentencia

- Sentencia debe afectar a expresión para evitar bucles infinitos

for (inicialización; test; actualización)  
sentencia

- Inicializa. Mientras test == true ejecuta sentencia. Tras ejecutar, actualiza

```
for (int i = 1; i <= 100; i++)  
    System.out.println(i);
```

# break y continue

- break permite interrumpir un bucle iterativo
- Interrumpe el bucle más interno (iteraciones anidadas)
- continue abandona la iteración actual.
- Pasa a la siguiente iteración

```
for (int i = 1; i <= 100; i++)  
{  
    if (i % 10 == 0)  
        continue;  
    System.out.println(i);  
}
```

# Iteración: do

```
do
    sentencia
while (expresión);
```

- Garantiza al menos una ejecución

```
do
{
    preguntar al usuario;
    leer valor
} while (valor es incorrecto);
```

# Selección múltiple: switch

```
switch (datoCaracter)
{
    case '(' :
    case '{ ' :
        // Código abrir
        break;
    case ')' :
    case '} ' :
        // Código cerrar
        break;
    case '\n' :
    case '{ ' :
        // Nueva línea
        break;
    default:
        // Otros casos
        break;
}
```

# Clases Envoltorio Wrappers

Curso Básico de JAVA

# Clases correspondientes a los tipos básicos (Wrappers)

- Java incorpora unos tipos básicos como `int`, `long`, `char`, ..., además proporciona la declaración de una clase para cada uno de los tipos básicos, denominadas clases envoltorio (wrappers).
- Estas clases están en el paquete `java.lang`, este se incorpora automáticamente en todos los programas de Java.
- Resultan útiles para la entrada de datos desde el teclado, ya que tienen métodos que convierten una cadena al tipo básico que se corresponde con la clase.

# Clase Integer

- Esta clase corresponde al tipo básico entero, cada objeto de esta clase contiene un entero del tipo int.
- Las variables de tipo clase Integer son referencias a objetos que se crean con el operador new.
- El constructor que se aplica al crear un objeto de esta clase tiene como argumento un número entero:

```
Integer n;  
n = new Integer(15);
```

Donde n es una referencia al objeto creado que se ha inicializado con el valor 15.

## Clase Integer > parseInt( )

Además del constructor, la clase tiene diversos métodos, algunos de ellos son:

parseInt( )

- Método static que tiene como argumento una cadena y devuelve el valor entero correspondiente a la conversión de los dígitos de la cadena.
- Si la conversión no es posible, genera una excepción en tiempo de ejecución.

static int parseInt (String cad) ;

## Clase Integer > parseInt( )

El ser un método static implica que se llama desde la clase y no desde un objeto de la clase,

```
int k;  
k = Integer.parseInt("145");  
int h = Integer.parseInt("-86");
```

Los valores de k y h son 145 y -86 respectivamente

## Clase Integer > parseInt( )

Este método se utiliza con mucha frecuencia para entrada de datos enteros desde el teclado.

1. Primero se crea un objeto asociado con el teclado,
2. Se lee a continuación una línea con el número entero tecleado
3. Con parseInt ( ) se convierte a dato int.

```
BufferedReader entrada = new BufferedReader (
                           new InputStreamReader (System.in));
String cd;
int dato;
System.out.print("Introduzca un dato entero: ");
System.out.flush();
cd = entrada.readLine();
dato = Integer.parseInt(cd);
```

Se Puede simplificar concatenando llamadas:

```
Dato = Integer.parseInt(entrada.readLine());
```

## Clase Integer > intValue( )

Este método devuelve el valor entero que tiene almacenado el objeto.

```
int value();
```

Por ejemplo:

```
int k;  
Integer n = new Integer (27);  
k = n.intValue(); // el valor de k es 27
```

## Clase Integer > `toString( )`

- Este método devuelve el dato entero en forma de cadena de dígitos.
- Este método lo tienen todos los objetos y transforma el objeto en su representación en forma de cadena.

```
String toString( );
```

## Clase Long

- Esta clase corresponde con el tipo entero largo, un objeto de esta clase contiene (envuelve) un entero de tipo long.
- Una variable declarada de tipo Long es referencia a objetos que se crean con el operador new.

## Clase Long

El constructor que se llama al crear un objeto de esta clase tiene como argumento un número entero largo:

```
Long K;  
k = new Long(1988815);
```

k es una referencia al objeto creado que se ha inicializado con el argumento 1988815

Los métodos de la clase Long son similares a los de la clase Integer, permiten convertir una cadena a entero largo, devolver el entero almacenado, etc.

## Clases Float y Double

- Estas clases corresponden con los tipos float y double respectivamente;
- Resultan útiles para entrada de datos desde teclado y también para colecciones de elementos de tipo float y double, con la clase Vector.
- Como todas las clases envoltorio, los constructor de objetos de estas clases tienen como argumento un dato del tipo básico al que corresponden.

## Clases Float y Double

```
Float g;  
g = new Float(1.59);  
Double d = new Double (1.59E3);
```

g es una referencia al objeto creado que se ha inicializado a 1.59 y d es una referencia al objeto Double inicializado con el valor 1.59E3.

## Clases **Float** y **Double**

Estas clases no tienen un método similar a `parseInt( )` que convierte una cadena a entero, por lo que resulta un poco más complicado la entrada de datos de punto flotante desde el teclado.

## Clases **Float** y **Double**

Los pasos a seguir para entrada de un dato de tipo double:

1. Crear el objeto para entrada asociado con el dispositivo estándar de entrada, el teclado.  
Este objeto se utiliza para cualquier entrada.

```
BufferedReader entrada = new BufferedReader (   
        new InputStreamReader(System.in));
```

## Clases Float y Double

2. Crear un objeto Double con el método valueOf( ).

```
Double d;  
String cd;  
cd = entrada.readLine();  
d= Double.valueOf(cd);
```

Se pueden concatenar las llamadas:

```
d = Double.valueOf(entrada.readLine());
```

## Clases Float y Double

3. Obtener el dato double del objeto con el método doubleValue( ).

```
double x;  
x = d.doubleValue();
```

Para entrada de un número de tipo float se siguen los mismos pasos, utilizando un objeto Float y en vez de doubleValue( ) el método floatValue( ).

# Elementos del Lenguaje Java

Curso Básico de JAVA

# CLASES

- Cuando se define una clase se declara su forma y naturaleza exacta especificando los atributos y los métodos que operan sobre el mismo.
- Las clases más sencillas pueden contener solamente atributos o solamente métodos, pero en la práctica las clases contienen atributos y métodos.

# Atributo

- Característica fundamental de un objeto
- Todos los atributos pueden tener algún valor, puede ser una cantidad, una relación con otro objeto, etc.

# Método

- Es una acción que se realiza sobre una clase para consultar o modificar su estado.

# Tipos de métodos

- Modificador (setter/mutator ): altera el estado de un objeto
- Selector (getter/accesor ): accede al estado de un objeto sin alterarlo
- Constructor: crea un objeto e inicializa su estado
- Destructor: limpia el estado del objeto y lo destruye
- Propósito General: la lógica del programa

# MÉTODOS Y ATRIBUTOS

- Método Público: Un método público puede ser utilizado o accedido a través de cualquier clase no importando si la clase es heredada o instanciada.
- Método Privado: Este tipo de métodos únicamente pueden ser accedidos a través de la misma clase.
- Métodos Protegidos: Estos métodos pueden ser utilizados únicamente a través de la misma clase o por una subclase a través de herencia.

# MÉTODOS Y ATRIBUTOS

Nombre	
Atributos	class Hola {
Metodos	int a;
Hola	int b;
a: int	Saludo( ) {
b: double	}
Saludo(): void	Despedida( ) {
Despedida():void	}

# Tipo de Clases

- **Public:** Una clase publica puede ser accedida a través de otra clase del mismo paquete o a través de una clase que se encuentra fuera del mismo.
- **Final:** Una clase final es aquella que termina una cadena de herencia y por lo cual esa clase no puede ser heredada.

# Clases abstractas

- Es una clase con al menos un método abstracto en una clase abstracta
- La clase que hereda a una clase abstracta necesita implementar todas las operaciones abstractas o tendrá que ser declarada como abstracta
- Una clase abstracta no puede ser instanciada
- Solo definen funcionalidad

# Clases abstractas

- Dejando que cada subclase complete los detalles necesarios. Una clase de este tipo determina la naturaleza de los métodos que las subclases deben implementar.
- Se puede precisar que las subclases sobrescriban ciertos métodos especificando el modificador del tipo “**abstract**”.

# Clases abstractas

- Se suele hacer referencia a estos métodos de responsabilidad de la subclase, ya que no están implementados específicamente en la super clase.

# Métodos Finales

- Los métodos que se declaran como final no pueden ser sobrescritos y además pueden proporcionar a veces una mejora del rendimiento porque el compilador puede realizar llamadas en línea a dichos métodos ya que no pueden ser sobrescritos por una subclase.

# Atributos Finales

- Declarando un atributo como final se impide que su contenido sea modificado.
- Esto significa que un atributo que es declarado como final se debe inicializar cuando es definido.
- Un convenio muy utilizado en programación es el de utilizar nombres en mayúsculas para los atributos finales

# Excepciones

Curso Básico de JAVA

# EXCEPCIONES

- Una excepción es un objeto que describe una condición excepcional, es decir un error que ha ocurrido es una parte de un código.
- Cuando surge una condición excepcional, se crea un objeto que representa esa excepción y se envía al método que ha originado el error.

# EXCEPCIONES

- Este método puede decidir entre gestionar el mismo la excepción o pasarla.
- En cualquiera de los dos casos es capturada y procesada.
- Las excepciones pueden ser generadas por el interprete de Java o por el propio código.

# EXCEPCIONES

- Las excepciones generadas por Java se refieren a errores fundamentalmente que violan las reglas del lenguaje o las restricciones del entorno de ejecución de Java.
- Las excepciones generadas por el código se usan normalmente para informar de alguna condición de error en la parte del código que llama al método.

# EXCEPCIONES

- La gestión de excepciones en Java se lleva a cabo mediante cinco palabras claves:
  - **try**
  - **catch**
  - **finally**
  - **throw**
  - **throws**

# EXCEPCIONES

- Las sentencias del programa que se quiera controlar se incluye en un bloque try, el código puede capturar esta excepción utilizando catch y gestionarla de forma racional.
- Se debe poner cualquier código que el programador desee que se ejecute siempre en el método final finally.

# EXCEPCIONES

- Las excepciones generadas por el sistema son automáticamente enviadas por el interprete Java.
- Para enviar manualmente una excepción se utiliza la palabra **throw**.
- Se debe especificar mediante la cláusula **throws** cualquier excepción que se envié desde un método.

# EXCEPCIONES

- Cualquier excepción que no sea capturada por el propio programa será finalmente procesada por el gestor por defecto, que presenta un mensaje que describe la excepción.
- Aunque el sistema de gestión de excepciones que proporciona Java es útil cuando se trata de depurar programas, normalmente el programador prefiere gestionar por si mismo una excepción.

# EXCEPCIONES

- Una vez que se lanza una excepción, el control del programa se transfiere del bloque try al bloque catch.
- La ejecución nunca vuelve al bloque try desde el catch.
- El objetivo de la mayor parte de las sentencias catch bien construidas debe ser resolver una condición excepcional y continuar como si el error nunca hubiera ocurrido.

# EXCEPCIONES

- En algunos casos una misma secuencia de código puede activar más de un tipo de excepción.
- Para gestionar esta situación se puede utilizar dos o más sentencias catch capturando cada una de ellas un tipo diferente de excepción.

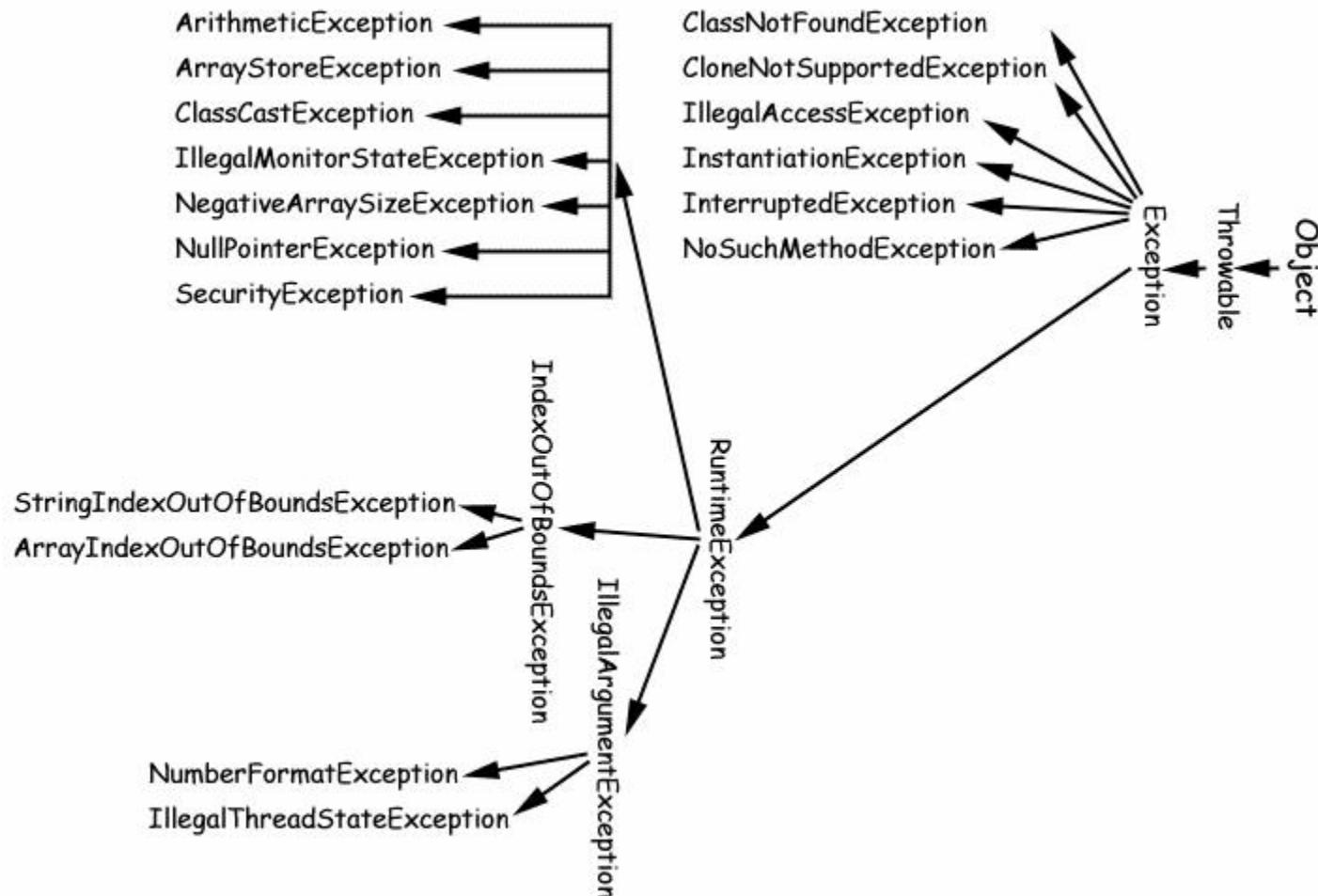
# EXCEPCIONES

- Cuando se lanza una excepción, se inspecciona por orden cada sentencia **catch**, y se ejecuta la primera cuyo tipo coincide con la excepción, después de la ejecución de una sentencia catch las demás no se ejecutan y la ejecución continua después del bloque **try catch**.

# EXCEPCIONES

- Una sentencia **try** puede estar dentro del bloque correspondiente a otra sentencia **try**.

# EXCEPCIONES



# Declaración de Objetos

Curso Básico de JAVA

# Declaración de objetos

- La obtención de objetos de una clase es un proceso que consta de dos etapas:
  1. Se debe declarar una variable del tipo de la clase, esta variable no define un objeto sino simplemente es una referencia a un objeto.
  2. Se debe obtener una copia física del objeto y asignarla a una variable, para ello se utiliza el operador **new**.

# Sobrecarga de métodos

Curso Básico de JAVA

# Declaración de objetos

- Son métodos que tienen el mismo nombre
  - public void println(int i)
  - public void println(float f)
  - public void println(String s)
- La lista de parámetros puede ser diferente
- El valor que devuelven puede ser diferente

# Paquetes (Packages)

Curso Básico de JAVA

# Paquetes (Packages)

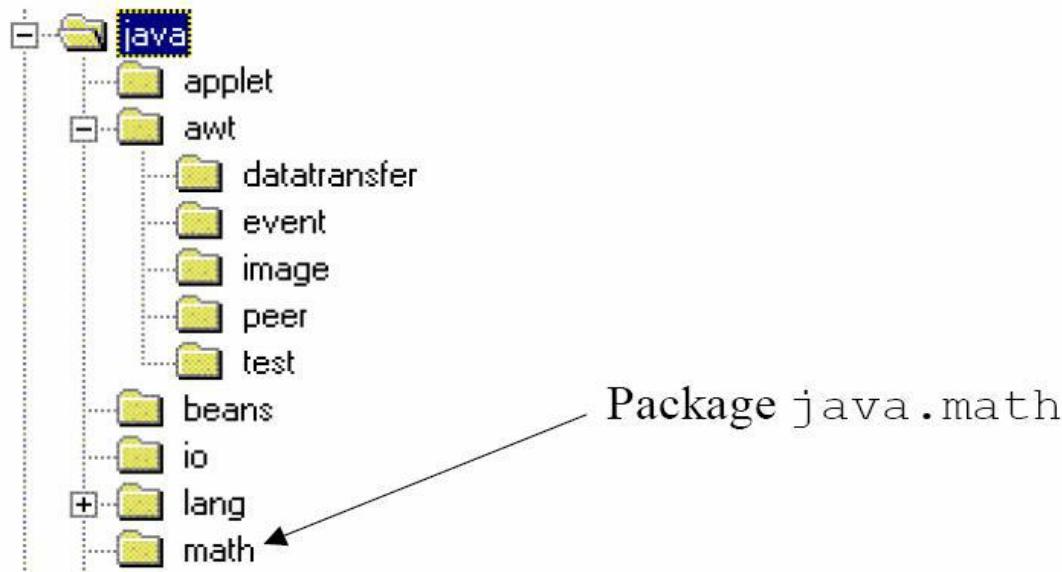
- Un paquete permite agrupar las clases en subdirectorios para evitar el conflicto con clases con el mismo nombre
- Para crear un paquete
  1. Crear un subdirectorio con el mismo nombre que tiene el paquete y colocar el archivo fuente en ese directorio
  2. Agregar la sentencia package a cada archivo

```
package nombreDelPaquete;
```
  3. Los archivos en el directorio principal que requieran utilizar el paquete deben incluir la línea

```
import nombreDelPaquete;
```

# Paquetes (Packages)

- La sentencia package debe ser la primera línea en el archivo
- Si package es omitido en un archivo, el código será parte del paquete principal que no tiene nombre



# Palabras clave this y super

Curso Básico de JAVA

# this

- Puede haber variables locales o parámetros formales de métodos que coincidan con el nombre de un atributo.
- En este caso la variable local esconde el atributo y para esto utilizamos, la palabra reservada **this**.
- En programación es más sencillo utilizar nombres diferentes
- **this** permite hacer referencia directamente al objeto y resolver de esta forma cualquier colisión(choque) entre nombres que pudiera darse entre las variables locales y los atributos.

# super

- Siempre que una clase necesita referirse a su superclase inmediata se utiliza la palabra reservada **super**.
- La palabra **super** la podemos utilizar de dos formas:
- Se utiliza para acceder a un miembro (**método** y **atributo**) de la superclase que ha sido escondido por un miembro de la subclase.
- Llama al constructor de la superclase.

# Arreglos

Curso Básico de JAVA

# Arreglos

- Un arreglo, en Java, es un grupo de variables (llamadas elementos o componentes) que contienen valores del mismo tipo.
- Los arreglos, en Java, son objetos, por lo que se consideran como tipos de referencia.

# Arreglos

- Los elementos de un arreglo pueden ser tipos primitivos o de referencia.
- Para hacer referencia a un elemento específico en un arreglo, debemos especificar:
  - El nombre de la referencia al arreglo
  - El número de la posición del elemento en el arreglo (índice o subíndice)

# Representación lógica de un arreglo.

- El arreglo  $c$ , contiene 12 elementos
- Un programa puede hacer referencia a cualquiera de sus elementos mediante una expresión de acceso a un arreglo:
  - $\text{nombre\_del\_arreglo}[\text{índice\_del\_elemento}]$
- El primer elemento en cualquier arreglo tiene el índice cero  $c[0]$ , el  $i$ -ésimo elemento del arreglo de  $c$  es  $c[i-1]$

# Representación lógica de un arreglo.

- Un índice debe ser un entero positivo o una expresión entera que pueda promoverse a un int.
- Si un programa utiliza una expresión como índice, el programa evalúa la expresión para determinar el índice:

Si  $a=5$  y  $b=6$   
 $c[a + b] += 2;$

Se le suma 2 al elemento 11 del arreglo.

# Arreglo c de 12 elementos

Nombre del arreglo

(observe que todos los elementos  
de este arreglo tienen el mismo  
nombre)

Índice (o subíndice)  
del elemento en el arreglo c

c [ 0 ]	-45
c [ 1 ]	6
c [ 2 ]	0
c [ 3 ]	72
c [ 4 ]	1543
c [ 5 ]	-89
c [ 6 ]	0
c [ 7 ]	62
c [ 8 ]	-3
c [ 9 ]	1
c [ 10 ]	6453
c [ 11 ]	78

# Declaración y creación de un arreglo.

- Todos los objetos en Java, incluyendo los arreglos, deben crearse con la palabra clave new.
- El programador especifica *el tipo* de cada elemento y el *número de elementos* que se requieren para el arreglo

```
int c[ ] = new int[ 12 ];  
int c[ ];  
c = new int[ 12 ];
```

# Declaración y creación de un arreglo.

- Al crear un arreglo cada uno de sus elementos recibe un valor predeterminado:
  - `0` para los elementos numéricos de tipo primitivo
  - `false` para los elementos boolean
  - `null` para las referencias (cualquier tipo no primitivo)

# Declaración y creación de un arreglo.

- Se pueden crear varios arreglos en una sola declaración

```
String b[ ] = new String [100], x[ ] = new String [27];
```

- Al declarar un arreglo, su tipo y los corchetes pueden combinarse al principio para indicar que todos los identificadores en la declaración son referencias a arreglos.

```
double[ ] arreglo1, arreglo2;  
double[ ] arreglo1 = new double[10], arreglo2 = new double[20];
```

# Lenguaje Unificado de Modelado (UML)

Curso Básico de JAVA

# UML

- Diseñado para la traducción de modelos de aplicaciones a lenguajes de programación
- Los clases y los objetos se representan con rectángulos
- Para diferenciar entre clases y objetos, los nombres de los objetos se subrayan

Clase

Objeto

# Representación gráfica de Clases y Objetos

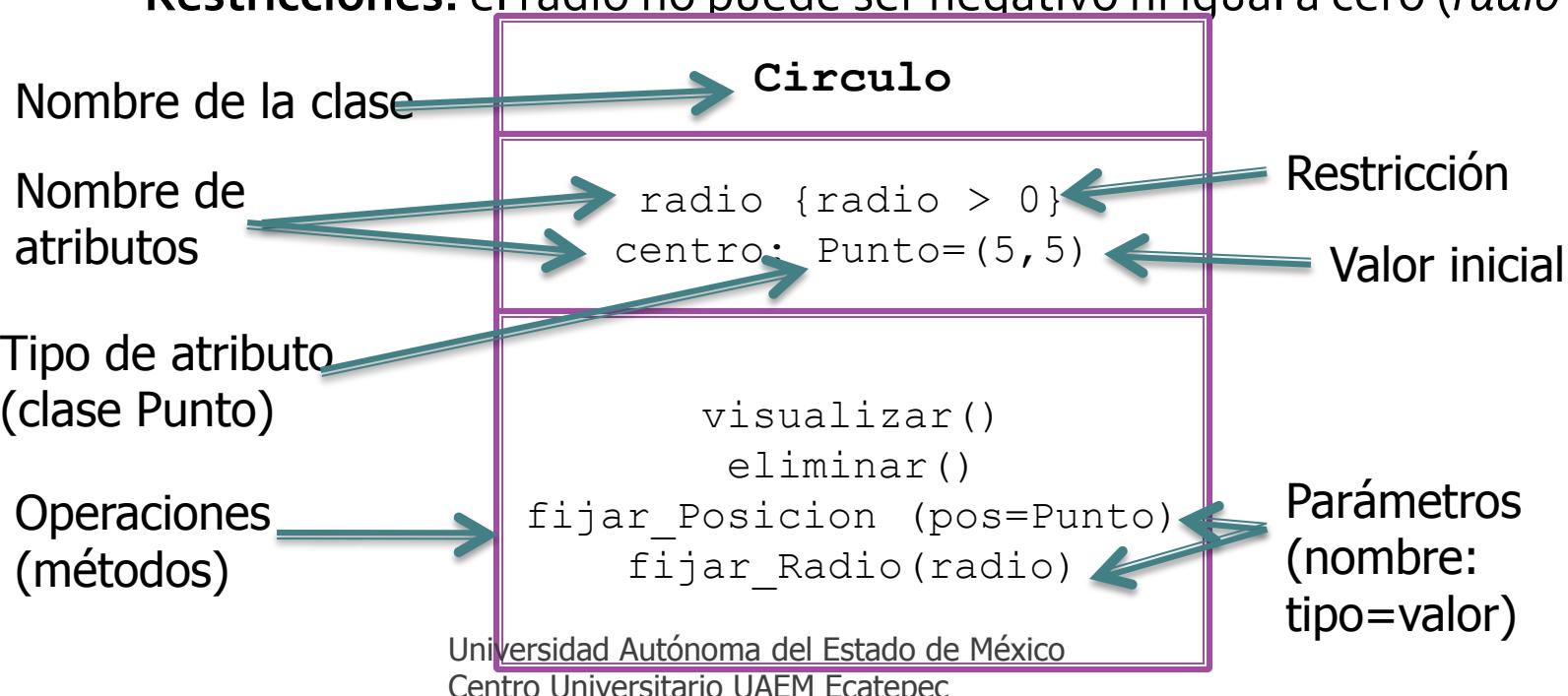
- El nombre de la Clase se representa en letra negrita comenzando con una letra mayúscula
- Los atributos se representan con su nombre seguido de las posibles restricciones que pueda tener
- Los nombres de los métodos vienen seguidos de una:
  - pareja de paréntesis: vacíos, si no tienen parámetros
  - El nombre y el tipo de dato del o de los parámetros

# Representación gráfica de Clases y Objetos

- Los objetos se representan de manera similar, con las siguientes diferencias:
  - El nombre del objeto esta subrayado
  - En algunos casos se puede agregar el nombre de la clase a la que pertenece después de :
  - Se pueden introducir valores iniciales a los atributos

# Ejemplo:

- Se desea representar un círculo en una pantalla con las siguientes propiedades:
  - **Atributos:** radio y coordenadas en la pantalla
  - **Operaciones:** Posibilidad de visualizar, mover, cambiar de posición y cambiar de tamaño
  - **Restricciones:** el radio no puede ser negativo ni igual a cero ( $\text{radio} > 0$ )



# Ejemplo:

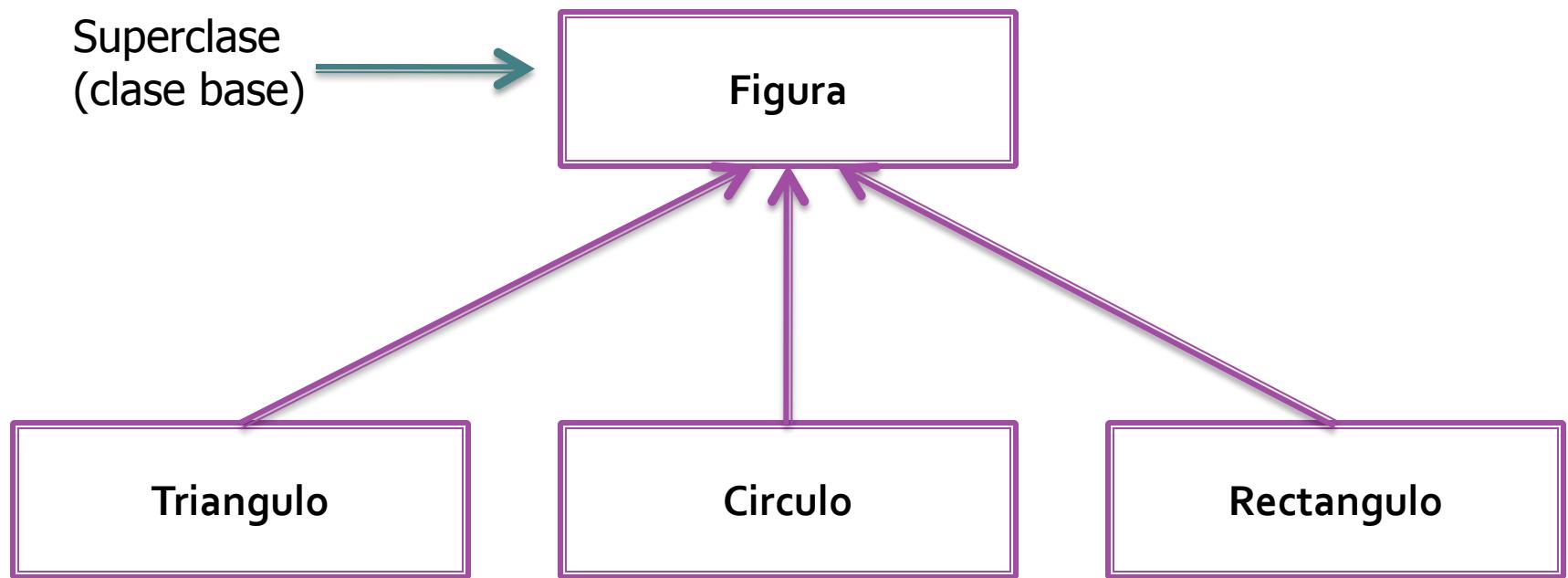
- Representación gráfica de un objeto en UML



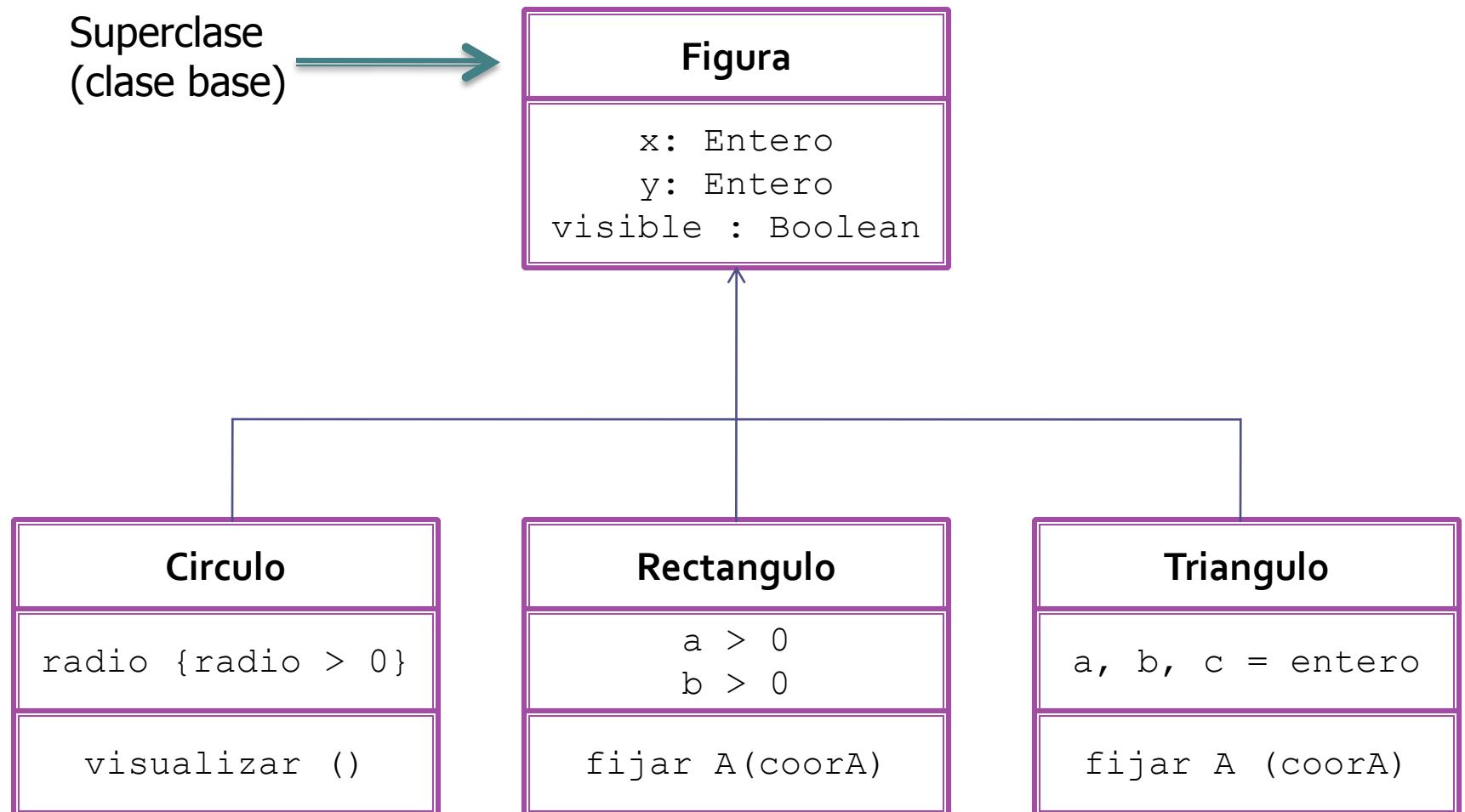
# Representación gráfica de la herencia

- La jerarquía de clases o herencia se representa mediante una flecha que une las clases derivadas o subclases con la clase base o superclase
- Pueden usarse
  - una flecha unidireccional de cada pareja de clases
  - Una única flecha a modo de tronco de árbol

# Representación gráfica de la herencia de clases



# Representación gráfica de la herencia de clases con indicación de sus propiedades



# Abstract Windows ToolKit (AWT)

Curso Básico de JAVA

# AWT

- El AWT es la parte de java que se ocupa de construir interfaces gráficas de usuario.
- AWT ha estado presente en java desde su versión 1.0, sin embargo es hasta la versión 1.1 donde se realiza un cambio notable en lo que se refiere al ***modelo de eventos***.
- En la versión 1.2 incorpora un modelo distinto de componentes llamado ***Swing***.

# Creación de una Interfaz Gráfica de Usuario (GUI)

1. Un contenedor o container, que es la ventana o parte de la ventana donde se situaran los componentes (botones, barras de desplazamiento, etc.) y donde se realizarán los dibujos.
2. Los componentes: menús, botones de comando, barras de desplazamiento, cajas y áreas de texto, botones de opción y selección, etc.
3. El modelo de eventos. Servirá para que el usuario controle la aplicación actuando sobre los componentes.

# Modelo de eventos.

- Cada vez que un usuario realiza una determinada acción, se produce el evento correspondiente, que el sistema operativo transmite al AWT.
- El AWT crea un **objeto** de una determinada clase de evento, derivada de **AWTEvent**. Este **evento** es transmitido a un determinado **método** para que lo gestione.
- El componente u objeto que recibe el evento debe “registrar” o indicar previamente que objeto se va a hacer cargo de gestionar ese evento.

# Objetos “event source” y “event listener”

- El modelo de eventos de **Java** está basado en que los objetos sobre los que se producen los eventos(**event sources**) “registran” los objetos que habrán de gestionarlos (**event listeners**), para lo cual los **event listeners** habrán de disponer de los **métodos** adecuados.
- Estos métodos se llamarán automáticamente cuando se produzca el evento. La forma de garantizar que los **event listeners** disponen de los métodos apropiados para gestionar los eventos es obligarles a implementar una determinada interface **Listener**.
- Las interfaces **Listener** se corresponden con los tipos de **eventos** que se pueden producir.

# Proceso a seguir para crear una aplicación interactiva (orientada a eventos)

1. Determinar los **componentes** que van a constituir la interface de usuario (botones, cajas de texto, menús, etc.).
2. Crear una **clase** para la aplicación que contenga la función **main()**.
3. Crear una clase **Ventana**, sub-clase de **Frame**, que responda al evento **WindowClosing()**.
4. La función **main()** deberá crear un objeto de la clase **Ventana** (en el que se van a introducir las componentes seleccionadas) y mostrarla por pantalla con el tamaño y posición adecuados.
5. Añadir al objeto **Ventana** todos los **componentes** y **menús** que deba contener.

# Proceso a seguir para crear una aplicación interactiva (orientada a eventos)

6. Definir los objetos ***Listener*** (objetos que se ocuparán de responder a los eventos, cuyas clases implementan las distintas interfaces ***Listener***) para cada uno de los eventos que deban estar soportados. En aplicaciones pequeñas, el propio objeto ***Ventana*** se puede ocupar de responder a los eventos de sus componentes. En programas más grandes se puede crear uno o más objetos de clases especiales para ocuparse de los eventos.

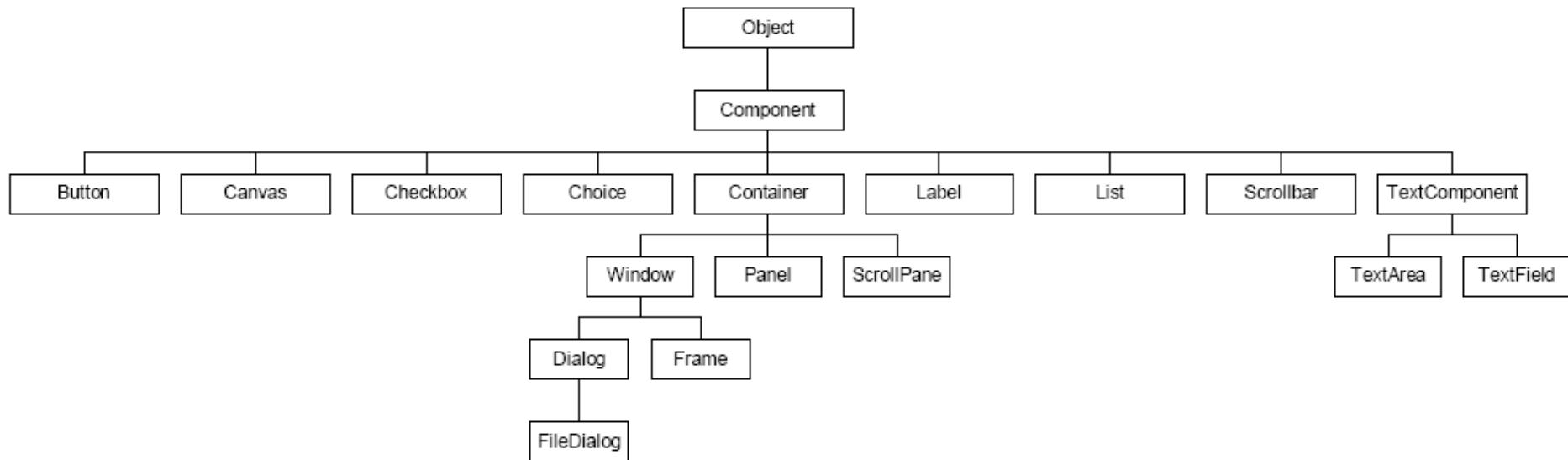
7. Finalmente, se deben implementar los métodos de las interfaces ***Listener*** que se vayan a hacer cargo de la gestión de los eventos.

# Componentes y eventos soportados por el AWT de Java

## *Jerarquía de Componentes*

- Como todas las clases de *Java*, los componentes utilizados en el AWT pertenecen a una determinada jerarquía de clases, que es muy importante conocer.
- Todos los componentes descienden de la clase ***Component***, de la que pueden ya heredar algunos métodos interesantes.
- El *package* al que pertenecen estas clases se llama *java.awt*.

# *Jerarquía de Componentes*



# *Jerarquía de Componentes*

## *Características importantes:*

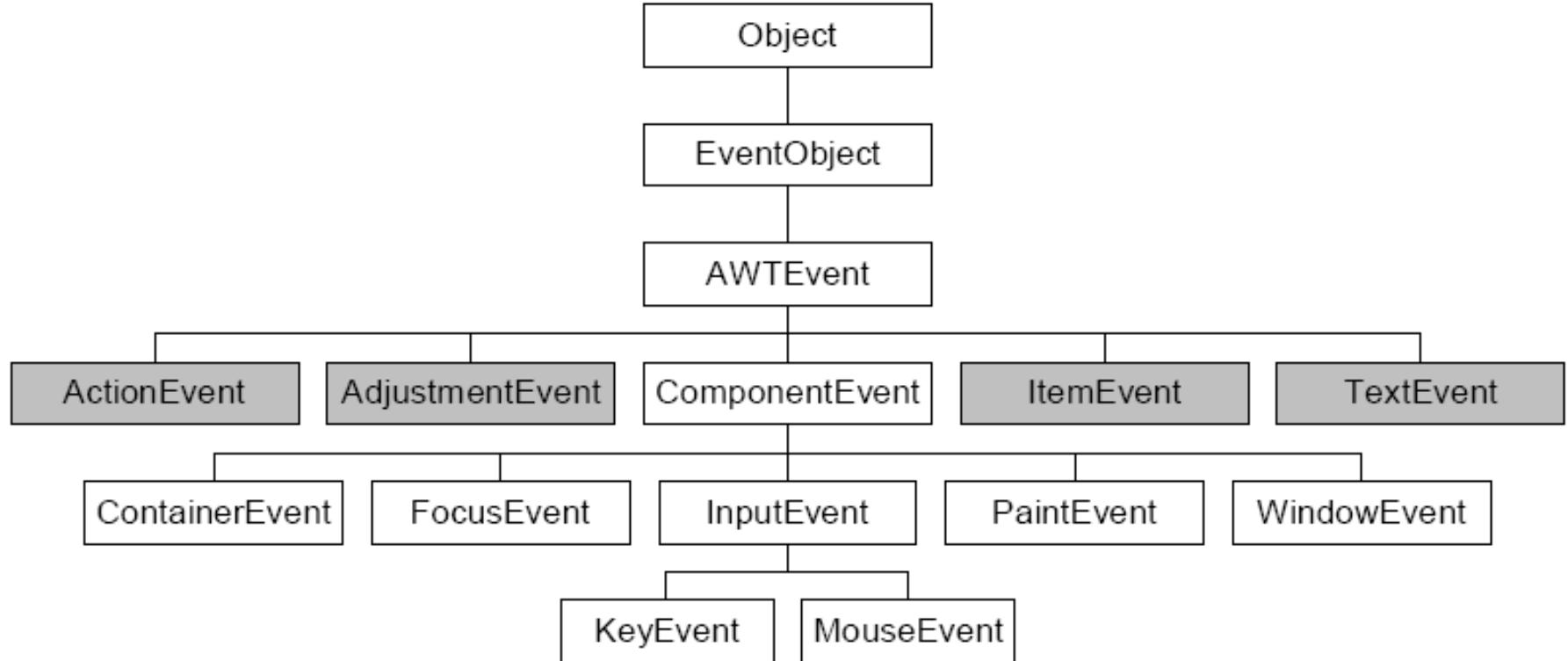
1. *Todos los Components (excepto Window y los que derivan de ella) deben ser añadidos a un Container. También un Container puede ser añadido a otro Container.*
2. *Para añadir un Component a un Container se utiliza el método add() de la clase Container:*  
`containerName.add(componentName);`
3. *Los Containers de máximo nivel son las Windows (Frames y Dialogs). Los Panels y ScrollPanes deben estar siempre dentro de otro Container.*
4. *Un Component sólo puede estar en un Container. Si está en un Container y se añade a otro, deja de estar en el primero.*
5. *La clase Component tiene una serie de funcionalidades básicas comunes (variables y métodos) que son heredadas por todas sus sub-clases.*

# Componentes y eventos soportados por el AWT de Java

## *Jerarquía de eventos*

- Todos los eventos de **Java 1.1** y **Java 1.2** son objetos de clases que pertenecen a una determinada jerarquía de clases.
- La super-clase ***EventObject*** pertenece al package ***java.util***.
- De ***EventObject*** deriva la clase ***AWTEvent***, de la que dependen todos los eventos de AWT.
- Por conveniencia, estas clases están agrupadas en el package ***java.awt.event***.

# *Jerarquía de Eventos*



# *Jerarquía de eventos*

- Los eventos de *Java* pueden ser de alto y bajo nivel.
- Los *eventos de alto nivel* se llaman también *eventos semánticos*, porque la acción de la que derivan tiene un significado en sí misma, en el contexto de las interfaces gráficas de usuario.
- Los *eventos de bajo nivel* son las acciones elementales que hacen posible los eventos de alto nivel.

# *Jerarquía de eventos*

## *Eventos de Alto Nivel*

- Son ***eventos de alto nivel*** los cuatro que tienen que ver con:
  - clicar sobre botones o elegir comandos en menús (***ActionEvent***),
  - cambiar valores en barras de desplazamiento (***AdjustmentEvent***),
  - elegir valores (***ItemEvents***) y
  - cambiar el texto (***TextEvent***).

# *Jerarquía de eventos*

## *Eventos de Bajo Nivel*

- Los ***eventos de bajo nivel*** son los que se producen con las operaciones elementales con el ratón, teclado, containers y windows.
- Las seis clases de eventos de bajo nivel son los eventos relacionados con:
  - componentes (***ComponentEvent***),
  - los containers (***ContainerEvent***),
  - pulsar teclas (***KeyEvent***),
  - mover, arrastrar, pulsar y soltar con el ratón (***MouseEvent***),
  - obtener o perder el focus (***FocusEvent***) y
  - las operaciones con ventanas (***WindowEvent***).

# Componentes y eventos soportados por el AWT de Java

El modelo de eventos se complica cuando se quiere construir un tipo de componente propio, no estándar del AWT. En este caso hay que interceptar los eventos de bajo nivel de *Java* y adecuarlos al problema que se trata de resolver.

# Componentes y eventos soportados por el AWT de Java

## *Relación entre Componentes y Eventos*

La siguiente tabla muestra los componentes del AWT y los eventos específicos de cada uno de ellos, así como una breve explicación de en qué consiste cada tipo de evento.

Component	Eventos generados	Significado
Button	ActionEvent	Clicar en el botón
Checkbox	ItemEvent	Seleccionar o deseleccionar un ítem
CheckboxMenuItem	ItemEvent	Seleccionar o deseleccionar un ítem
Choice	ItemEvent	Seleccionar o deseleccionar un ítem
Component	ComponentEvent	Mover, cambiar tamaño, mostrar u ocultar un componente
	FocusEvent	Obtener o perder el focus
	KeyEvent	Pulsar o soltar una tecla
	MouseEvent	Pulsar o soltar un botón del ratón; entrar o salir de un componente; mover o arrastrar el ratón (recordar que este evento tiene dos Listener)
Container	ContainerEvent	Añadir o eliminar un componente de un container
List	ActionEvent	Hacer doble click sobre un ítem de la lista
	ItemEvent	Seleccionar o deseleccionar un ítem de la lista
MenuItem	ActionEvent	Seleccionar un ítem de un menú
Scrollbar	AdjustementEvent	Cambiar el valor de la scrollbar
TextComponent	TextEvent	Cambiar el texto
TextField	ActionEvent	Terminar de editar un texto pulsando Intro
Window	WindowEvent	Acciones sobre una ventana: abrir, cerrar, iconizar, restablecer e iniciar el cierre

## Componentes y eventos soportados por el AWT de Java

- La relación entre componentes y eventos indicada en la tabla anterior pueden confundir al programador si no se tiene en cuenta que los eventos propios de una *super-clase* de componentes pueden afectar también a los componentes de sus *sub-clases*.
- Por ejemplo, la clase *TextArea* no tiene ningún evento específico o propio, pero puede recibir los de su *super-clase* *TextComponent*.

# Componentes y eventos soportados por el AWT de Java

- La siguiente tabla muestra los ***componentes*** del AWT y ***todos los tipos de eventos*** que se pueden producir sobre cada uno de ellos, teniendo en cuenta también los que son específicos de sus ***superclases***.
- Entre ambas tablas se puede sacar una idea bastante precisa de qué tipos de eventos están soportados en ***Java*** y qué eventos concretos puede recibir cada componente del AWT.
- En la práctica, no todos los tipos de evento tienen el mismo interés.

# Componentes y eventos soportados por el AWT de Java

Eventos que generan los distintos componentes del AWT.

AWT Components	Eventos que se pueden generar										
	ActionEvent	AdjustmentEvent	ComponentEvent	ContainerEvent	FocusEvent	ItemEvent	KeyEvent	MouseEvent	MouseMotionEvent	TextEvent	WindowEvent
Button	✓				✓			✓	✓		
Canvas		✓		✓	✓		✓	✓	✓		
Checkbox		✓		✓	✓		✓	✓	✓		
Checkbox-MenuItem					✓		✓				
Choice		✓		✓	✓		✓	✓	✓		
Component		✓		✓			✓	✓	✓		
Container		✓	✓	✓			✓	✓	✓		
Dialog		✓	✓	✓			✓	✓	✓		✓
Frame		✓	✓	✓			✓	✓	✓		✓
Label		✓		✓			✓	✓	✓		
List	✓		✓	✓	✓	✓	✓	✓	✓		
MenuItem	✓										
Panel			✓	✓	✓		✓	✓	✓		
Scrollbar		✓	✓		✓		✓	✓	✓		
TextArea			✓		✓		✓	✓	✓	✓	
TextField	✓	✓	✓	✓	✓		✓	✓	✓	✓	
Window	✓	✓	✓	✓	✓		✓	✓	✓		✓

# Conectividad de bases de datos con JDBC

Curso Básico de JAVA

# JDBC

- Los programas en Java se comunican con las bases de datos y manipulan sus datos utilizando la API JDBC.
- Un controlador de JDBC implementa la interfaz para una base de datos específica.
- Al separar la API de los controladores específicos, los desarrolladores pueden cambiar la BD subyacente sin necesidad de modificar el código de Java.

# JDBC

- La mayoría de los SGBD incluyen ahora controladores de JDBC.
- También existen muchos controladores de JDBC desarrollados por terceros.

# Controladores JDBC

- JDBC soporta cuatro categorías de controladores:
  1. Tipo 1 – ***Controlador Puente JDBC-ODBC***
  2. Tipo 2 – ***Controlador Nativo de API*** desarrollado parcialmente en JAVA
  3. Tipo 3 – ***Controlador JDBC-Net*** desarrollado puramente en JAVA
  4. Tipo 4 – ***Controlador de Protocolo Nativo*** desarrollado puramente en JAVA

# *Controlador Puente JDBC-ODBC*

- Conecta los programas de Java con orígenes de datos de Microsoft ODBC.
- Esta incluido en el J2SDK
- El nombre del controlador es:  
**`sun.jdbc.odbc.JdbcOdbcDriver`**

# *Controlador Puente JDBC-ODBC*

- Por lo general, requiere que el controlador ODBC este instalado en el equipo cliente
- Es necesario configurar el origen de datos ODBC
- Se introdujo para fines de desarrollo, por lo que no es recomendable usarse en aplicaciones de producción

# *Controlador Nativo de API*

- Permite a los programas de JDBC utilizar APIs para bases de datos específicas (generalmente escritas en C o C++)
- Dichas APIs permiten a los programas clientes utilizar BD mediante la Interfaz Nativa de Java (JNI)
- Traduce el código JDBC en código de una base de datos específica.

# *Controlador JDBC-Net*

- Toman las peticiones de JDBC y las traducen en un protocolo de red que no es para una BD específica.
- Estas peticiones se envían a un servidor, el cual traduce las peticiones de la BD en un protocolo específico para esa base de datos.

# *Controlador de Protocolo Nativo*

- Convierten las peticiones de JDBC en protocolos de red para bases de datos específicas
- Los programas de Java pueden conectarse directamente con una base de datos.

# Compatibilidad entre Tipos de Datos JDBC-SQL

Tipo de dato SQL	Tipo de dato Java devuelto por getObject()	Método get apropiado
BIGINT	Long	long getLong()
BINARY	byte[]	byte[] getBytes()
BIT	Boolean	long getLong()
CHAR	String	String getString()
DATE	java.sql.Date	java.sql.Date getDate()
DECIMAL	java.math.BigDecimal	java.math.BigDecimal getBigDecimal
DOUBLE	Double	double getDouble()
FLOAT	Double	double getDouble()
INTEGER	Integer	int getInt()
LONGVARBINARY	byte[]	InputStream getBinaryStream()

# Compatibilidad entre Tipos de Datos JDBC-SQL

LONGVARCHAR	String	InputStream getAsciiStream() InputStream getUnicodeStream()
NUMERIC	java.math.BigDecimal	java.math.BigDecimal getBigDecimal()
REAL	Float	float getFloat()
SMALLINT	Integer	short getShort()
TIME	java.sql.Time	java.sql.Time getTime()
TIMESTAMP	java.sql.Timestamp	java.sql.Timestamp getTimestamp()
TINYINT	Integer	byte getByte()
VARBINARY	byte[]	byte[] getBytes()
VARCHAR	String	String getString()
VARCHAR2	String	String getString()