Curso de Java

Unidad III

“Lenguaje Java”

Rogelio Ferreira Escutia

Contenido

1) *Programación Orientada a Objetos*
2) *Manejo de Objetos*
3) *Identificadores, Palabras Clave y Tipos de Datos*
4) *Expresiones y Control de Flujo*
5) *Arreglos*
6) *Diseño de Clases*
7) *Interfaz Gráfica*

1) Programación Orientada a Objetos

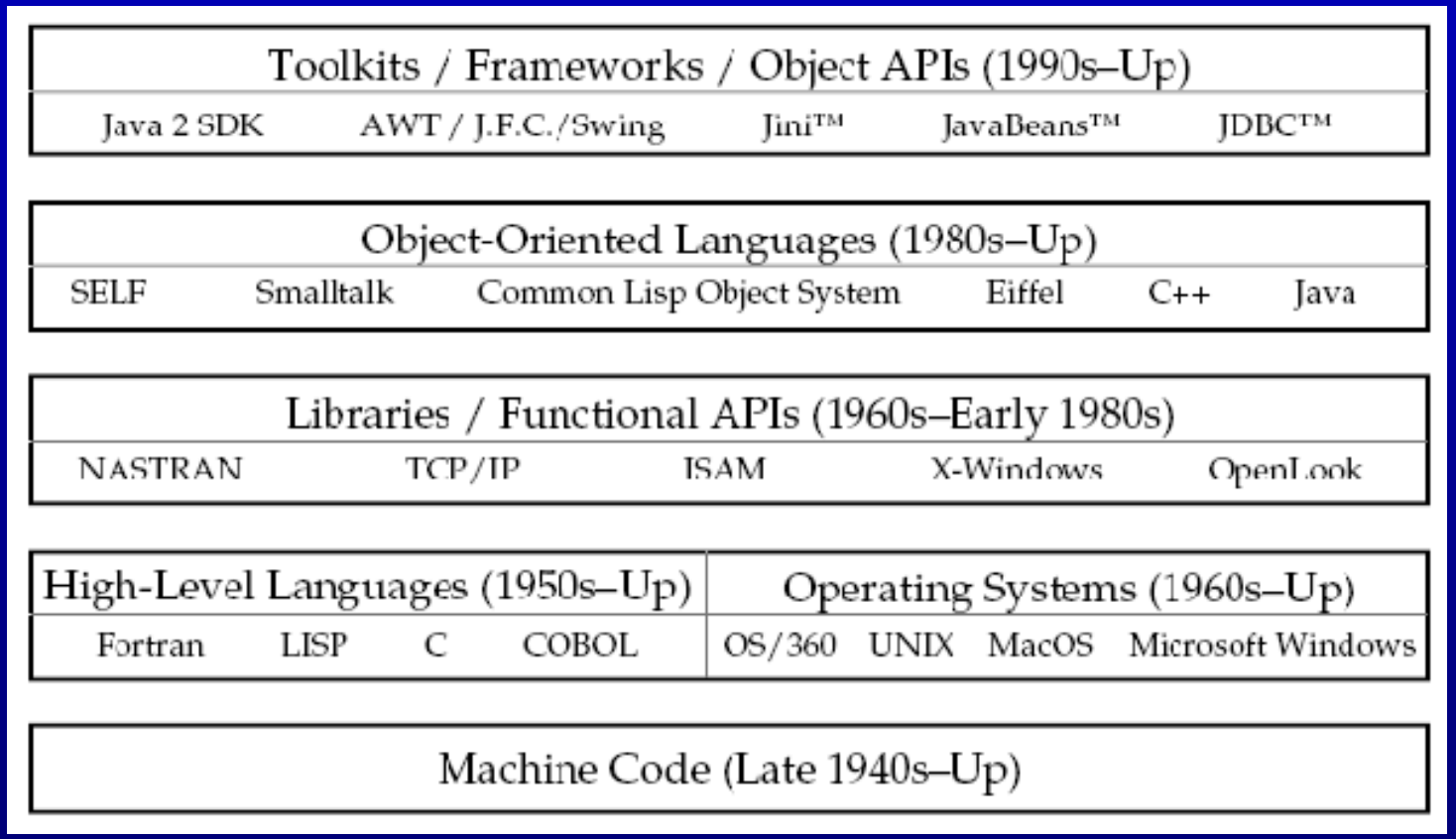
Abstracción

- Todos los lenguajes de programación están construidos a partir de abstracciones.
- El lenguaje ensamblador es un abstracción del lenguaje máquina.
- Algunos lenguajes imperativos (como Fortran, Basic y C) son abstracciones del lenguaje ensamblador.
- Estos lenguajes fueron grandes avances sobre el lenguaje máquina, pero su abstracción requiere que el programador piense de la manera en que la computadora estructura los datos, en vez de pensar en la estructura del problema a resolver.
- El programador debe establecer la asociación entre el modelo de la máquina y el modelo del problema que se piensa resolver

• "Thinking in Java", Bruce Eckel, Third Edition, Prentice Hall 2003

Lenguajes de Programación

- **Evolución de los Lenguajes**



OOP

- La abstracción basada en objetos provee herramientas al programador para representar elementos.
- Esta representación es por lo general suficiente para que el programador no se oriente hacia algún tipo de problema en particular.
- La Programación Orientada a Objetos (OOP, Object Oriented Programming) permite describir el problema en términos del problema, en vez de los términos de la computadora en la cual correrá.

• "Thinking in Java", Bruce Eckel, Third Edition, Prentice Hall 2003

Identificando objetos

Los objetos pueden ser *físicos o conceptuales*:

- ***Una cuenta de cliente es un ejemplo de un objeto conceptual porque no se puede tocar físicamente. Un cajero automático es algo que mucha gente toca todos los días y es un ejemplo de un objeto físico.***

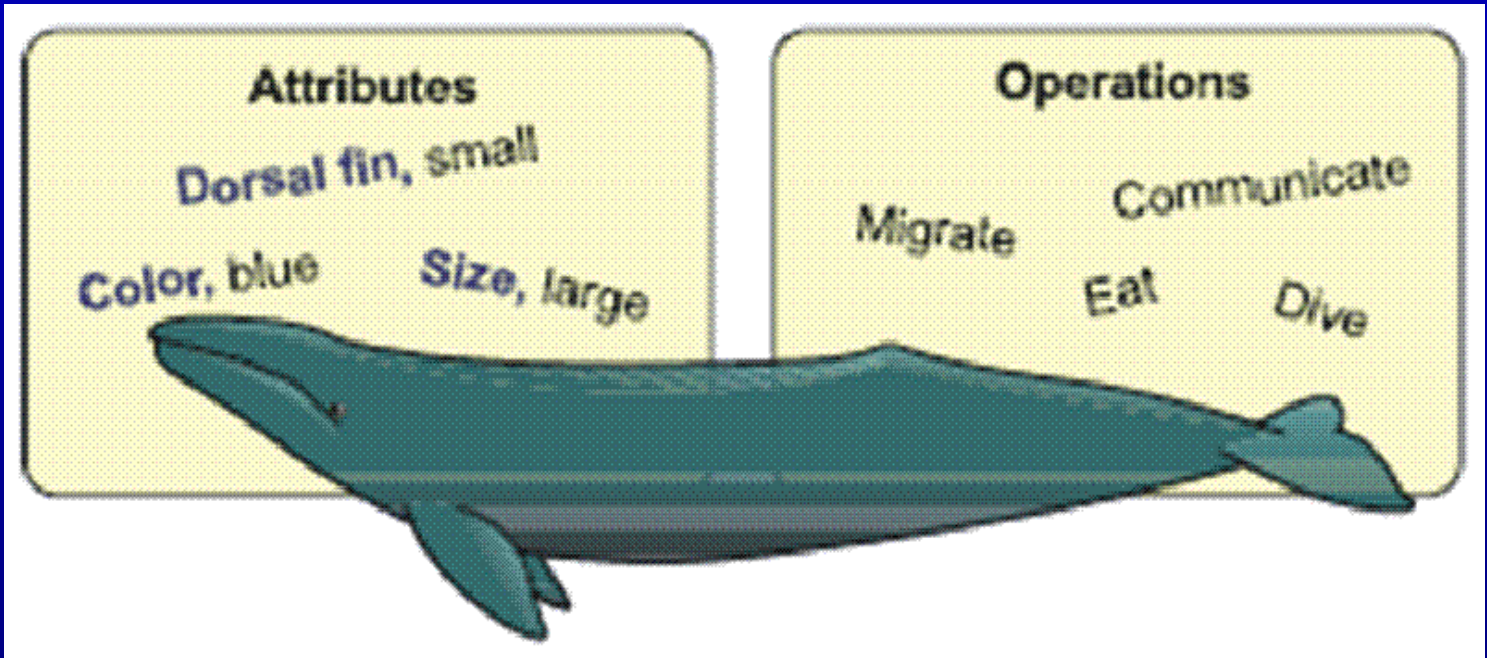
Los objetos tienen *atributos* (características):

- **Tal como tamaño, nombre, color, forma, etc. Los valores de los atributos son referidos como el *estado* actual del objeto. Por ejemplo, un objeto puede tener un atributo color con el valor de rojo.**

Los objetos tienen *operaciones* (las cosas que pueden hacer):

- **Tal como asignar un valor, desplegar una pantalla, o incrementar rapidez. Las operaciones usualmente afectan los atributos de los objetos.**

Identificando objetos

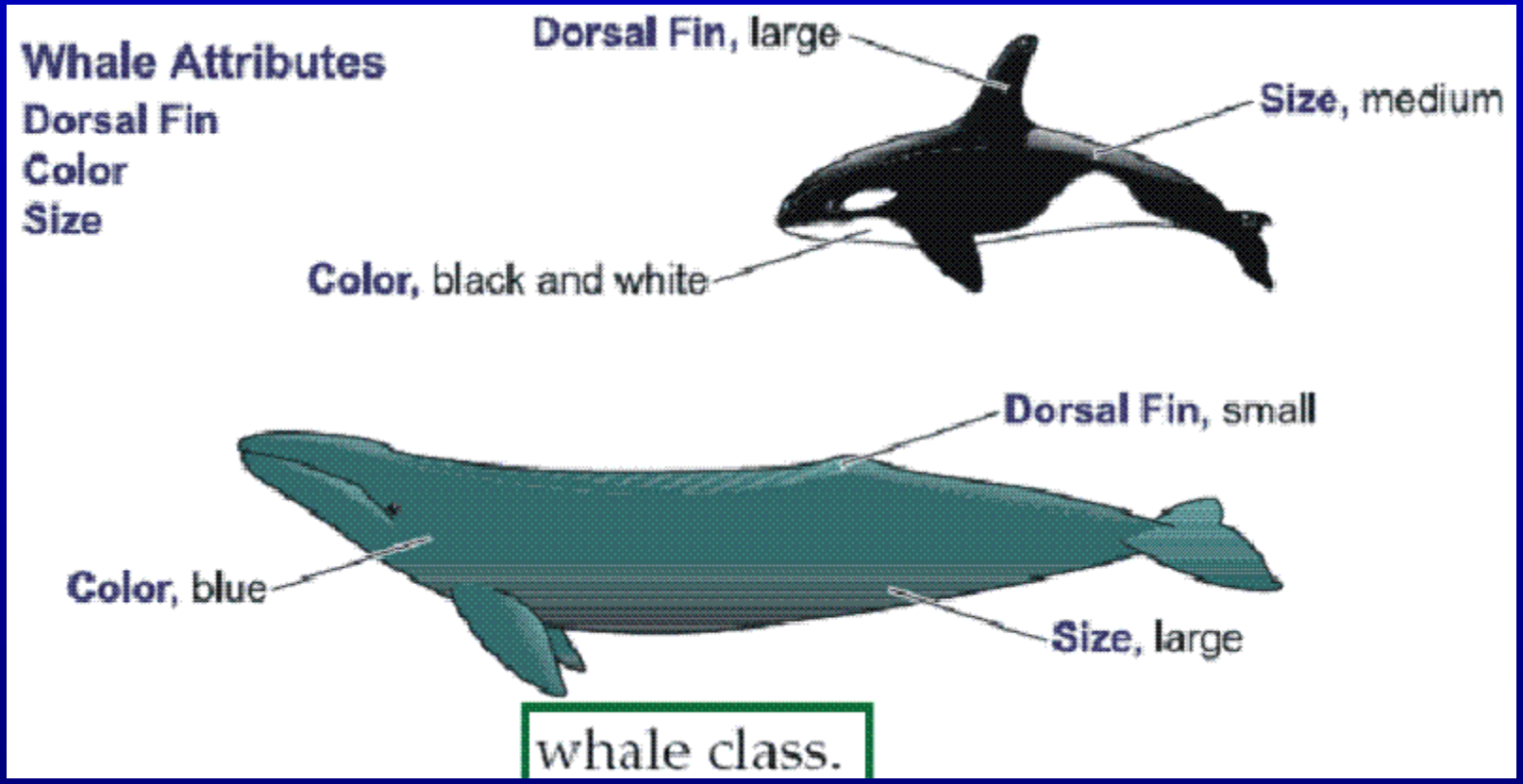


Clases

- Aristóteles probablemente fué de los primeros en utilizar el conceptos de “Clase", cuando hablaba acerca de "Las clases de pescados y las clases de pájaros".
- La idea de que todos los objetos son únicos y son parte de una clase de objetos (class) que tienen características y comportamientos, es utilizada en la construcción de Simula 67, el primer lenguaje orientado a objetos.
- Existen objetos idénticos, excepto por el estado en el que se encuentran durante la ejecución de un programa. Estos objetos se reúnen en grupos que se denominan "clases de objetos".
- La creación de tipos de datos abstractos (classes) es el concepto fundamental de la "Programación Orientada a Objetos".

• “Thinking in Java”, Bruce Eckel, Third Edition, Prentice Hall 2003

Clases


Whale Attributes
Dorsal Fin
Color
Size
Dorsal Fin, large
Size, medium
Color, black and white
Dorsal Fin, small
Color, blue
Size, large
whale class.

Instancias

- Se pueden crear variables de un tipo (“objetos” o “instancias de un objeto”) y manipular sus variables ("envío de mensajes").
- Los miembros ("elementos") de cada clase, comparten ciertas características comunes. Cada miembro tiene sus propio estado.
- Estas variables tienen una entidad única en el programa que corre la computadora. Esta entidad es un "objeto", y cada objeto pertenece a una "clase" particular que define sus "características" y "comportamientos".
- Una vez que una "clase" es establecida, se pueden crear n objetos de esa clase, asi como manipular el contenido de esos objetos.

• “Thinking in Java”, Bruce Eckel, Third Edition, Prentice Hall 2003

Java

- **Alan Kay definió 5 características básicas de Smalltalk (el lenguaje en el cual se basó Java):**

  1) Cada cosa es un objeto.

  2) Un programa es un conjunto de objetos que se envían mensajes entre sí para saber qué es lo que van a realizar.

  3) Cada objeto tiene su propia memoria construida a partir de otros objetos.

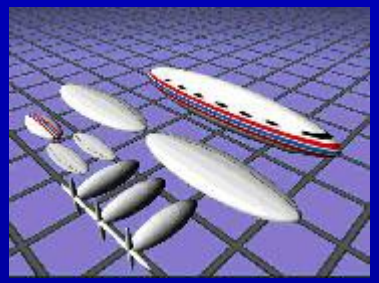
  4) Cada objeto tiene un tipo.

  5) Todos los objetos del mismo tipo pueden recibir los mismos mensajes.

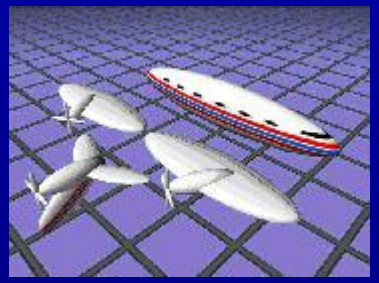
• "Thinking in Java", Bruce Eckel, Third Edition, Prentice Hall 2003

Java - Objetos

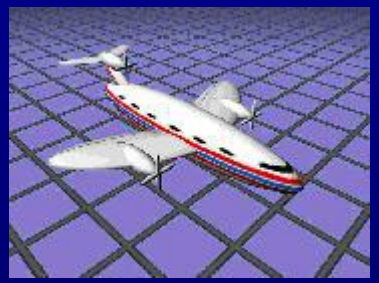
- **Construcción de un objeto:**



Definir componentes



Ensamble de componentes

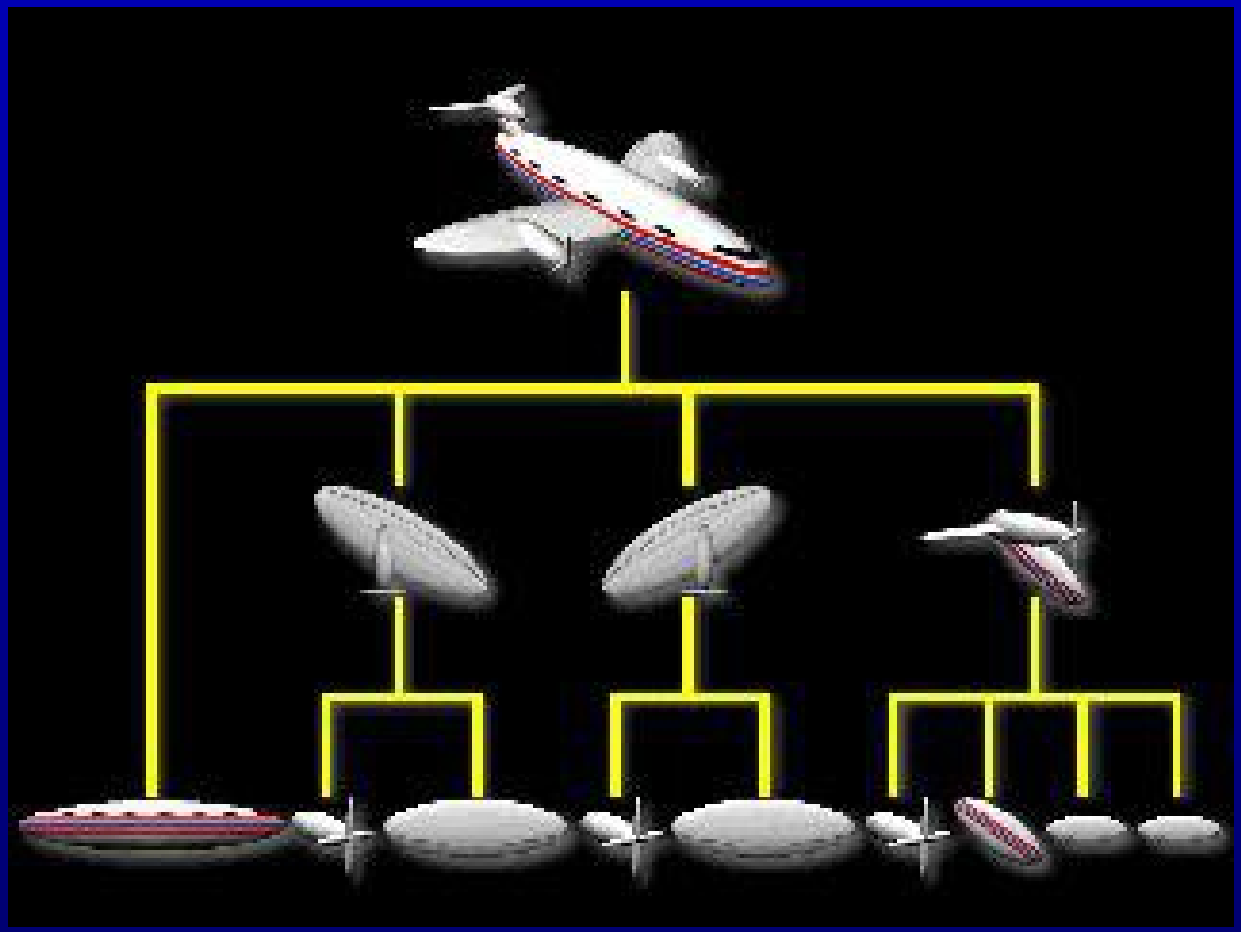


Objeto final

- Introduction to Programming with Java 3D, Sowizral y Nadeau, http://www.sdsc.edu/~nadeau, mayo 2005

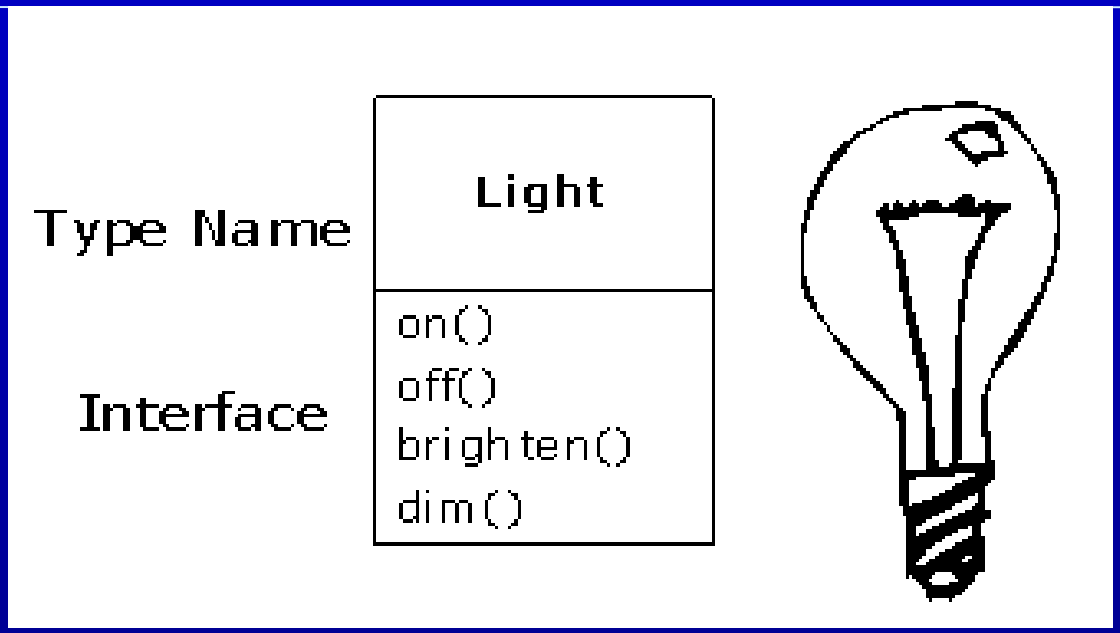
Java - Objetos

- **Objeto y sus componentes:**



- Introduction to Programming with Java 3D, Sowizral y Nadeau, http://www.sdsc.edu/~nadeau, mayo 2005

Java - objetos



Notación UML

Light lt = new Light();
lt.on();

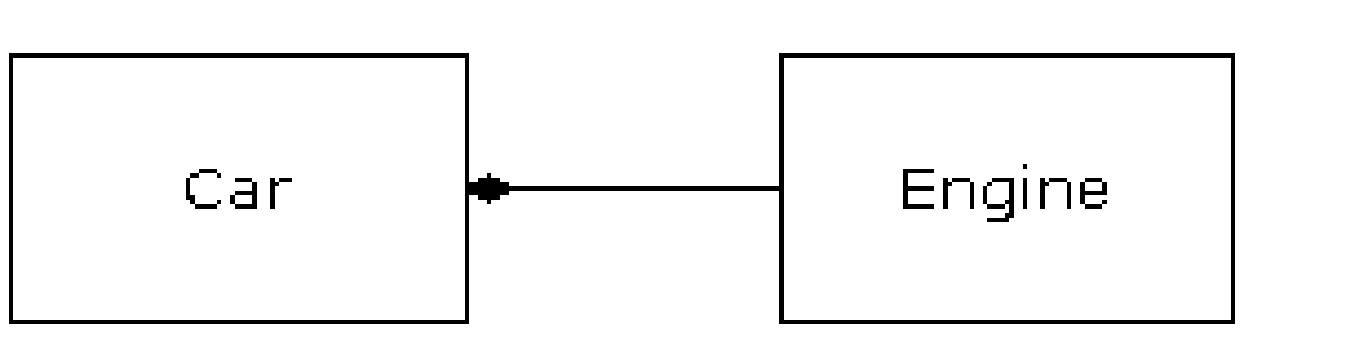
- **Clase = Light**
- **Nombre = lt**
- **Posibles comportamientos del objeto (métodos) = on, off, brighten, dim**

Java – Límites de una clase

- Existen 3 tipos de límites de un elemento que pertenece a una clase, los cuales nos indican quien puede acceder a ellas:
- public: significa que el elemento puede ser accesado por cualquiera.
- private: nadie, excepto uno mismo (el creador de la clase) puede accesar a ella.
- protected: funciona como privado, con excepción de que de que por medio de herencia se puede accesar a ella.
- Si no se selecciona ningun tipo de límite, por default Java lo declara de tipo private.

• "Thinking in Java", Bruce Eckel, Third Edition, Prentice Hall 2003

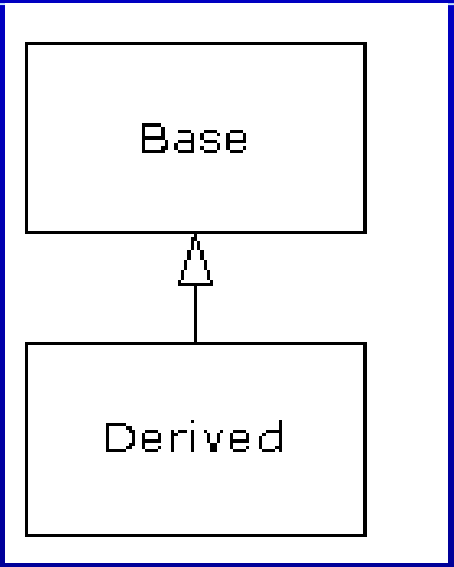
Java - Reutilización



Notación UML

- **Se denomina "creación de un miembro" cuando se coloca un objeto de una clase dentro de otra clase. La nueva clase puede estar formada por n cantidad de objetos, en cualquier combinación para lograr el funcionamiento deseado de la nueva clase. Con esto se logra la reutilización del código.**

Java - Herencia

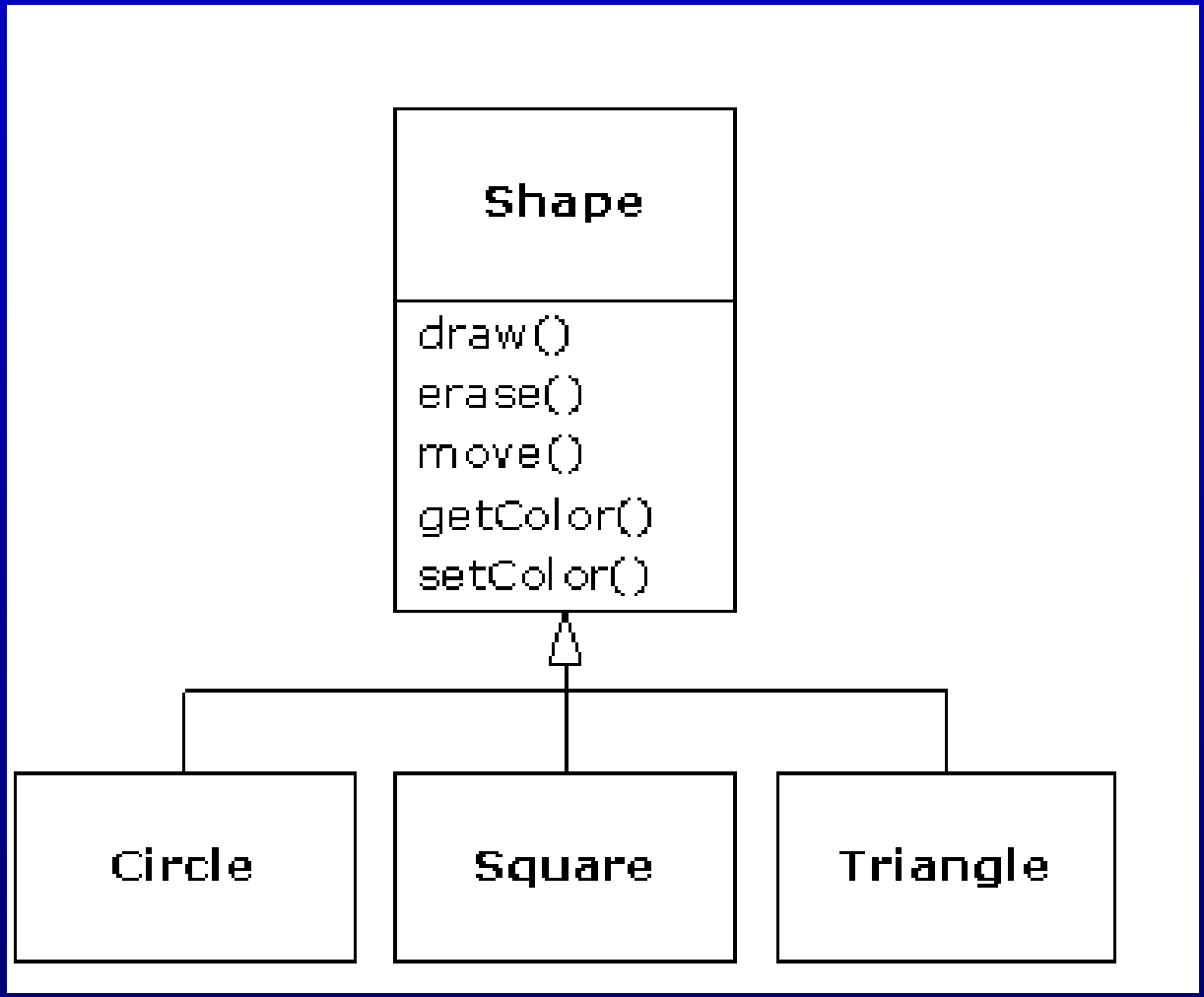


Notación UML

- Herencia es cuando, en vez de crear objetos nuevos cada vez que se hace un programa, se crea un clase a partir de una ya existente (clonar), y solamente se le hacen modificaciones al clon.

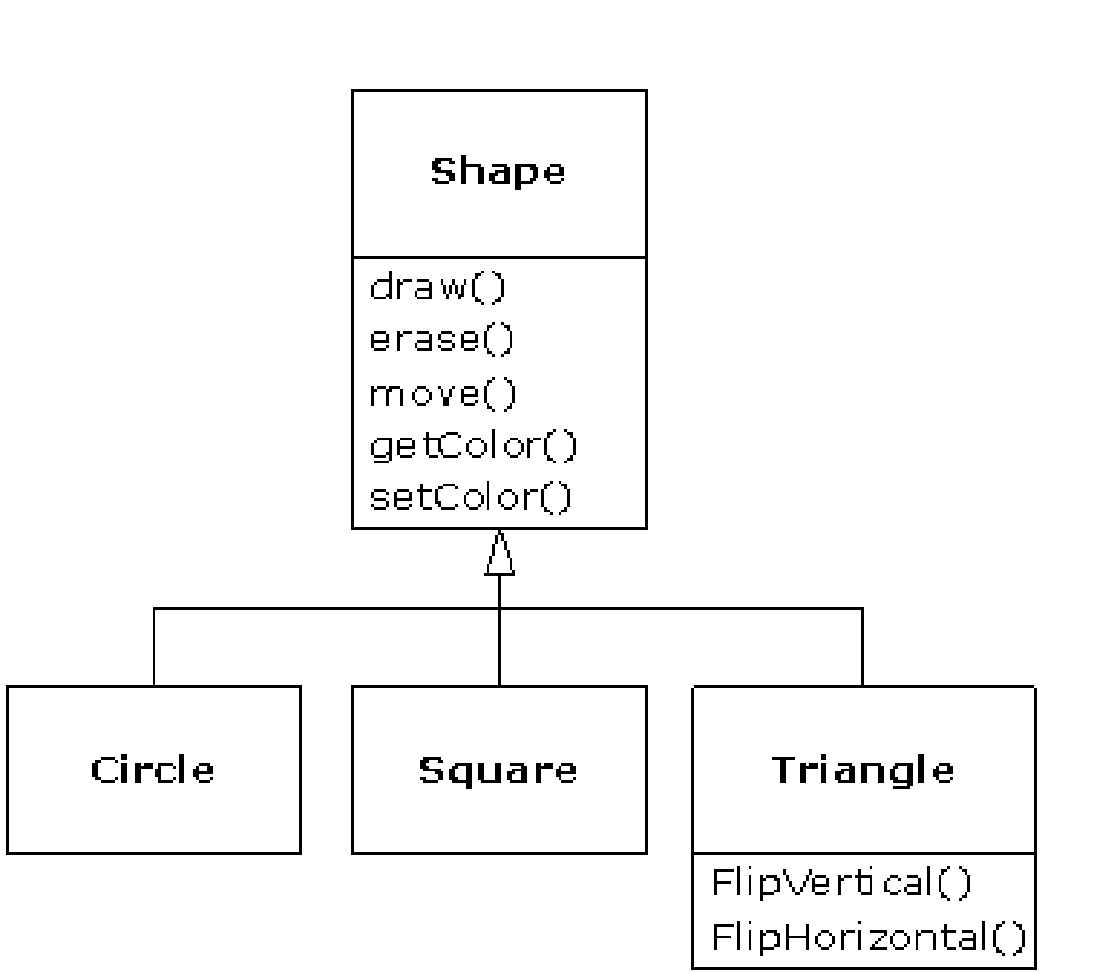
- Cuando se modifica la clase original (llamada a veces la "clase base", "superclase" o "clase padre") se modifican los clones que se formaron a partir de ella (llamados a veces "clases derivadas", "clases herenciadas", "subclases" o "clases hijos").

Java - Herencia



- "Thinking in Java", Bruce Eckel, Third Edition, Prentice Hall 2003

Java – Herencia y métodos



- "Thinking in Java", Bruce Eckel, Third Edition, Prentice Hall 2003

2) Manejo de Objetos

Creación de Objetos

new

- **String s = new String("asdf");**

Tipos Primitivos

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	—	—	—	**Boolean**
char	16-bit	Unicode 0	Unicode 2^{16}- 1	**Character**
byte	8-bit	-128	+127	**Byte**
short	16-bit	-2^{15}	$+2^{15}—1$	**Short**
int	32-bit	-2^{31}	$+2^{31}—1$	**Integer**
long	64-bit	-2^{63}	$+2^{63}—1$	**Long**
float	32-bit	IEEE754	IEEE754	**Float**
double	64-bit	IEEE754	IEEE754	**Double**
void	—	—	—	**Void**

- "Thinking in Java", Bruce Eckel, Third Edition, Prentice Hall 2003

Valores por default de tipos primitivos

Primitive type	Default
boolean	false
char	‘\u0000’ (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

- “Thinking in Java”, Bruce Eckel, Third Edition, Prentice Hall 2003

Ambito de los objetos

```
{
    int x = 12;

    // solo x esta disponible

    {
        int q = 96;
        // x y q están disponibles
    }

    // solo x esta disponible
}
```

- "Thinking in Java", Bruce Eckel, Third Edition, Prentice Hall 2003

Ambito ilegal

```
{
        int x = 12;

        {
                int x = 96;         // ilegal
        }

 }
```

Declaración de Clases

<modifier>* class <class_name> {
  <attribute_declaration>*
  <constructor_declaration>*
  <method_declaration>*
}

```
1    public class Vehicle {
2      private double maxLoad;
3      public void setMaxLoad(double value) {
4        maxLoad = value;
5      }
6    }
```

Declaración de Atributos

<modifier>* <type> <name> [=<initial_value>];

```
private int x;

private float y = 10000.0F;

private String name = “Juan Pérez”;
```

La palabra private indica que el atributo solo es accesible por los métodos dentro de la clase. El tipo puede ser primitivo o de cualquier clase.

Declaración de Métodos

<modifier>* <return_type><name> (<argument>*) {
<statement>*
}

Los modificadores pueden ser public, protected o private.

El modificador de acceso público indica que el método puede ser llamado desde otro código.

El método privado indica que el método puede ser llamado solo por otros métodos dentro de la clase.

Declaración de Métodos

```
1    public class Dog {
2      private int weight;
3      public int getWeight() {
4        return weight;
5      }
6      public void setWeight(int newWeight) {
7        if ( newWeight > 0 ) {
8          weight = newWeight;
9        }
10     }
11   }
```

Método “main”

- Es un método especial que Java reconoce como el punto inicial para que un programa se ejecute desde la línea de comando.
- Su sintaxis es

  public static void main(String args[])
- Siempre debe tener dos posibles modificadores de acceso:

  public y static
- El método main no returna ningún valor.
- Acepta cero o mas objetos de tipo String.

Accesando miembros del objeto

- El operador punto(.) permite accesar a atributos y métodos no privados de una clase.
- Dentro de un método no se necesita usar esta notación para accesar miembros locales del método.
- Generalmente se crean métodos set() y get() para accesar a miembros de un objeto que no es local.

Accesando miembros del objeto

- **Clase ShirtTest que contiene el método “main”**

```
1
2   public class ShirtTest {
3
4      public static void main (String args[]) {
5
6      Shirt myShirt;
7      myShirt = new Shirt();
8
9      myShirt.displayShirtInformation();
10
11
12      }
13  }
```

Accesando miembros del objeto

- **Clase Shirt**

```
1
2   public class Shirt {
3
4     public int shirtID = 0; // Default ID for the shirt
5     public String description = "-description required-"; // default
6
7     // The color codes are R=Red, B=Blue, G=Green, U=Unset
8     public char colorCode = 'U';
9
10    public double price = 0.0; // Default price for all shirts
11
12    public int quantityInStock = 0; // Default quantity for all shirts
13
14    // This method displays the values for an item
15    public void displayShirtInformation() {
16
17      System.out.println("Shirt ID: " + shirtID);
18      System.out.println("Shirt description:" + description);
19      System.out.println("Color Code: " + colorCode);
20      System.out.println("Shirt price: " + price);
21      System.out.println("Quantity in stock: " + quantityInStock);
22
23    } // end of display method
24  } // end of class
```

Declaración de Constructores

- Un constructor es un conjunto de instrucciones diseñadas para inicializar una instancia.
- Los parámetros pueden ser pasados al constructor de la misma manera que un método.
- Su declaración básica es:

```
[<modifier> <class_name> (<argument>*) {

<statement>

}
```

- Los constructores no son métodos, ya que no pueden retornar valores y no tienen herencia.

Declaración de Constructores

```
public class Dog {

        private int weight;

        public Dog() {
                weight = 42;
        }
}
```

Constructor default

- Cada clase tiene al menos un constructor.
- Si no se escribe un constructor, Java proporciona uno por default. Este constructor no tiene argumentos y tiene un cuerpo vacío.
- Si se declara un constructor en una clase que no tenia constructores, se pierde el constructor por default.

Invocación de Constructores

```
import java.io.*;
class MyDate{
    public int day;
    public int month;
    public int year;
    public MyDate(int day, int month, int year){
       //day=10; month=05; year=2005;
       this.day=day; this.month=month; this.year=year;
    }
}
public class TestMyDate {
    public static void main(String args[]){
        MyDate fecha = new MyDate(18,05,72);
        System.out.println("El dia es: "+ fecha.day);
        System.out.println("El mes es: "+ fecha.month);
        System.out.println("el año es: "+ fecha.year);
    }
}
```

Uso de package

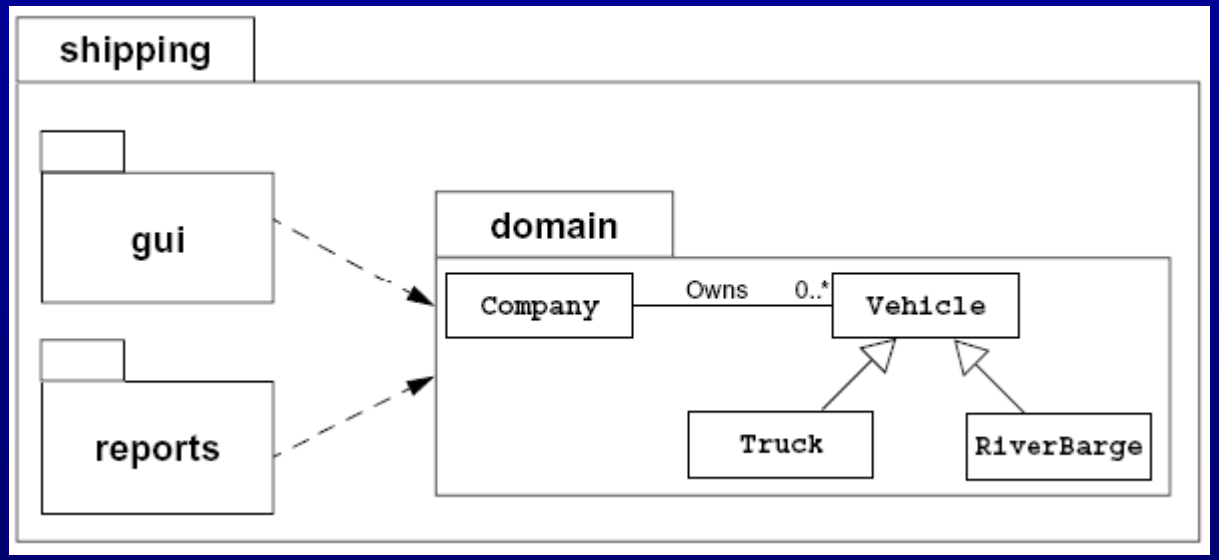
- **Sintaxis:**

  package <top_pkg_name>[.<sub_pkg_name>]*;

- **La declaración del paquete va al principio del programa Java.**
- **Solamente una se permite una declaración de paquete por programa.**
- **Si no se declara un paquete, entonces la clase es puesta en el paquete por default.**
- **Los nombre de paquetes deben ser jerárquicos y separados por puntos.**

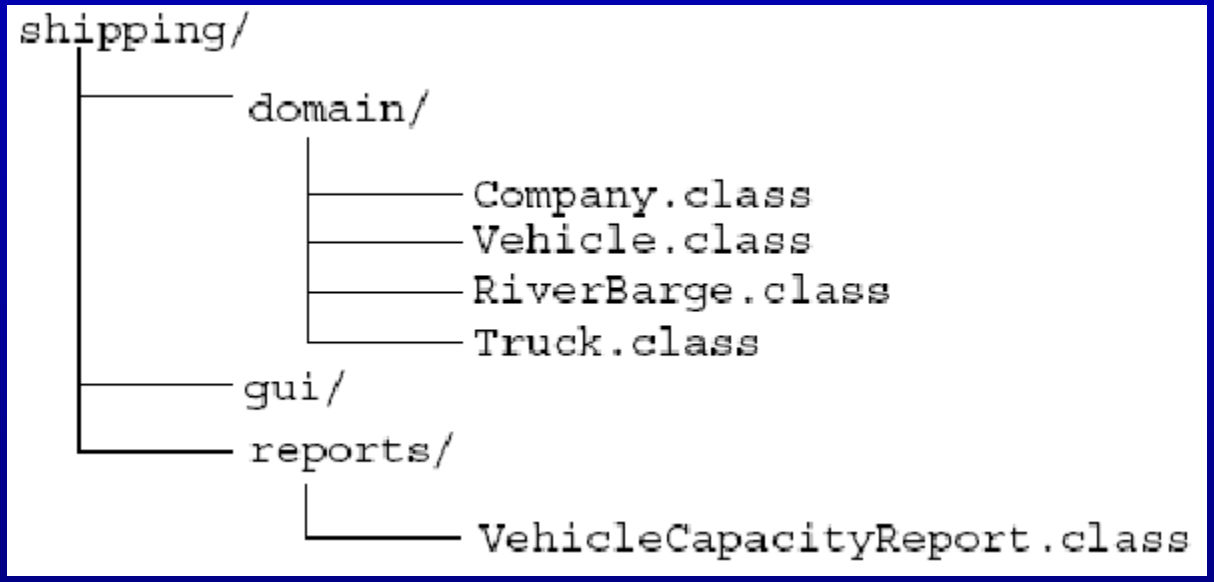
Uso de package

```
package shipping.reports;

import shipping.domain.*;
import java.util.List;
import java.io.*;

public class VehicleCapacityReport {
        private List vehicles;
        public void generateReport(Writer output) {...}
}
```



Uso de package

- **Estructura de directorios:**



Uso de import

- **Sintaxis:**

  import <pkg_name>[.<sub_pkg_name>]*.<class_name>;

  o

  import <pkg_name>[.<sub_pkg_name>]*.*;

- **Ejemplos:**

```
import java.util.List;
import java.io.*;
import shipping.gui.reportscreens.*;
```

- **La utilización de "import" va antes de la declaración de todas las clases y le indica al compilador donde se encuentran las clases.**

3) Identificadores, Palabras Clave y Tipos de Datos

Comentarios

- **Formas permitidas de introducir un comentario:**

```
// comentarios de una línea

/* comentario de una
* o mas líneas
*/
```

Instrucciones y Bloques

- **Una instrucción es una o mas líneas de código terminadas por ";"**

```
totals = a + b + c
+ d + e + f;
```

- **Un bloque es un conjunto de instrucciones encerradas por llaves "{ }**

```
{
x = y + 1;
y = x + 1;
}
```

Bloques Anidados

```
while ( i < largo ) {
        a = a + i;
        // bloque anidado
        if ( a == max ) {
                        b = b + a;
                        a = 0;
        }
        i = i + 1;
}
```

Espacios en blanco

- En Java es permitido cualquier cantidad de espacios en blanco:
- Ejemplo:

```
{int x;x=23*54;}
```

- Es equivalente a:

```
{
int x;
x = 23 * 54;
}
```

Variables

- Los nombres de las variables tienen las siguientes características:
- Los nombres de las variables, clases y métodos pueden empezar con letras de tipo Unicode, guión bajo o signo de pesos.
- Son sensibles a mayúsculas y minúsculas y no tienen una longitud máxima.
- Ejemplos:

```
nombreUsuario
Nombre_usuario
_usuario
$usuario
```

Palabras Clave

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

Palabras reservadas: null, true y false

Tipos primitivos

- Existen 8 tipos primitivos de datos:
  - Lógicos boolean
  - Texto char
  - Enteras byte, short, int y long
  - Flotantes double y float

Tipo de dato lógico

- **Se utilizan para almacenar el resultado de una decisión, por lo que solo tiene dos valores: true y false.**
- **El valor por default es false.**
- **Las variables se declaran de tipo boolean.**

Tipo de dato de texto

- Se declara de tipo char y tiene las siguientes características:
- Representa un carácter de Unicode de 16 bits.
- Debe tener los caracteres encerrados entre apóstrofes (’ ’).
- Ejemplos:
  - 'a‘ letra a
  - '\t‘ tabulador
  - '\u????‘ carácter Unicode ????, donde ???? son 4 dígitos en hexadecimal.
  - ’\u03A6’ es la letra griega phi.

Tipo String

- El tipo String no es un tipo primitivo, es una clase.
- Los valores de los tipos de dato String van encerrados entre comillas (" ").
- Ejemplos:
  - String greeting = “Buenos Días !! \n";
  - String errorMessage = "Record Not Found !";

Tipo de datos enteros

- Existen 4 tipos de datos enteros (byte, short, int y long).
- Este tipo de datos puede usar 3 formatos:
- Decimal, octal ó hexadecimal.
- Las letras tienen un tipo int por default.
- Letras con el sufijo “L” son del tipo “long”.
- Ejemplos:
  - 2 la forma decimal para el entero 2.
  - 077 El 0 inicial indica un valor en octal.
  - 0xBAAC El 0x inicial indica un valor en hexadecimal.

Rango de datos enteros

Longitud del entero	nombre	Rango
8 bits	byte	-2^7 to 2^7-1
16 bits	short	-2^{15} to $2^{15}-1$
32 bits	int	-2^{31} to $2^{31}-1$
64 bits	long	-2^{63} to $2^{63}-1$

Tipo de datos flotante

- Existen 2 tipos: float y double.
- Un tipo flotante incluye un punto decimal o alguno de los siguientes caracteres:
  - E ó e (valor exponencial)
  - F ó f (float)
  - D ó d (double)
- Ejemplos:
  - 3.14 un valor de punto flotante (double)
  - 6.02E23 un valor de tipo flotante con exponente
  - 2.718F un valor de punto flotante

Rango de datos flotante

- Los números con punto decimal por default son de tipo double y sus rangos son los siguientes:

Longitud	Tipo
32 bits	float
64 bits	double

Clase String

- La clase String es una de las muchas clases de las librerías de Java. Proporciona la habilidad de almacenar una secuencia de caracteres.
- Existen dos maneras de crear e inicializar objetos String
- 1) Una manera es utilizando la palabra new para crear un objeto y al mismo tiempo definir la cadena con la cuál se inicializará el objeto:

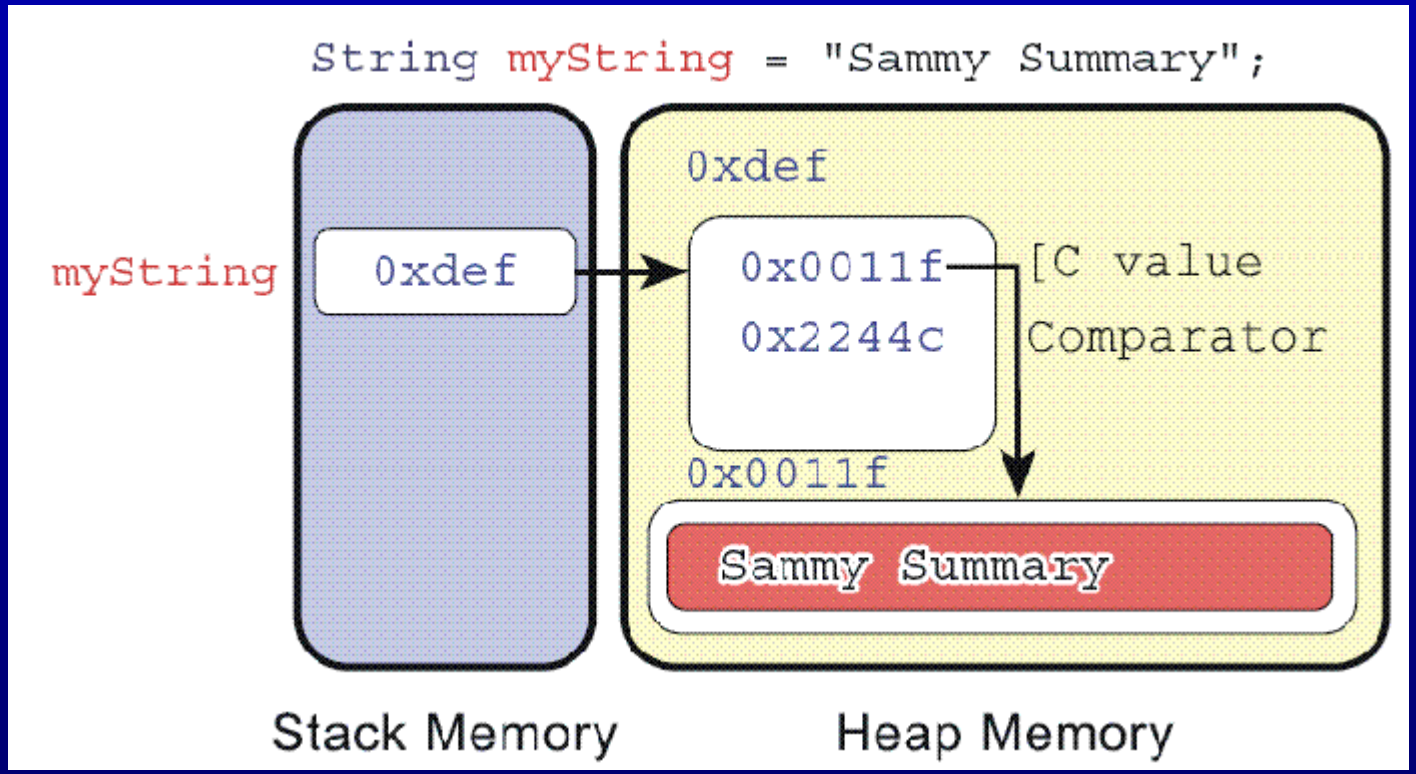
```
String myName = new String ("Hola como estas");
```

- 2) Crear un objeto String sin la palabra new, ya que el tipo String es la única clase que permite construir objetos sin usar la palabra new:

```
String myName = "Hola como estas";
```

Clase String

- La dirección contenida en la variable de referencia myString se refiere a un objeto en memoria. El objeto String tiene una variable atributo llamada C Value que contiene una dirección que referencia a la cadena de caracteres.



Métodos de la clase String

- *char charAt(int index)* **Retorna el carácter indexado de una cadena, donde el índice del carácter inicial es 0.**

- *String concat (String addThis)* **Retorna una nueva cadena que consiste de la vieja cadena seguida por lo que se le añade.**

- *int compareTo(String otherString)* **Ejecuta una comparación léxica; retorna un int que es menor que 0 si la cadena actual es menor que la otra cadena, igual a 0 si las cadenas son idénticas, y mayor a 0 si la cadena actual es mayor que la otra cadena.**

Métodos de la clase String

- *boolean equalsIgnoreCase (String s)* **Retorna true si hace match con la actual cadena ignorando consideraciones de minúsculas o mayúsculas.**
- *int indexOf (int ch)* **Retorna el indice dentro de la cadena actual de la primera ocurrencia de ch.**
- *int lastIndexOf (int ch)* **Retorna el indice dentro de la cadena actual de la última ocurrencia de ch.**
- *int length()* **Retorna el número de caracteres en la cadena actual.**

Métodos de la clase String

- *String replace(char oldChar, char newChar)* **Retorna una nueva cadena, generada por medio de reemplazar cada ocurrencia de oldChar con newChar.**
- *boolean startsWith (String prefix)*
- *String toLowerCase()*
- *String toUpperCase()*

Declaración de Datos

```
1   public class Assign {
2     public static void main (String args []) {
3       // declare integer variables
4       int x, y;
5       // declare and assign floating point
6       float z = 3.414f;
7       // declare and assign double
8       double w = 3.1415;
9       // declare and assign boolean
10      boolean truth = true;
11      // declare character variable
12      char c;
13      // declare String variable
14      String str;
15      // declare and assign String variable
16      String str1 = "bye";
17      // assign value to char variable
18      c = 'A';
19      // assign value to String variable
20      str = "Hi out there!";
21      // assign values to int variables
22      x = 6;
23      y = 1000;
24    }
25  }
```

Tipos de Referencia

- **Existen 8 tipos de datos primitivos, todos los demás tipos se refieren a objetos que NO son primitivos.**
- **La variable que hace referencia a objetos, se les llama *variables de referencia*.**

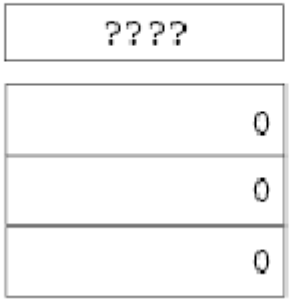
Construcción e inicialización de objetos

- Cuando se utiliza el operador new para la creación de objetos sucede lo siguiente:
  - Se asigna memoria al objeto.
  - Los atributos de inicialización explícita son ejecutados.
  - Se ejecuta el constructor.
  - La referencia al objeto se regresa por el operador new.
  - La referencia es asignada a una variable.
  - Ejemplo:
    - MyDate my_birth = new MyDate(22, 7, 1964);

Asignación de memoria a un objeto

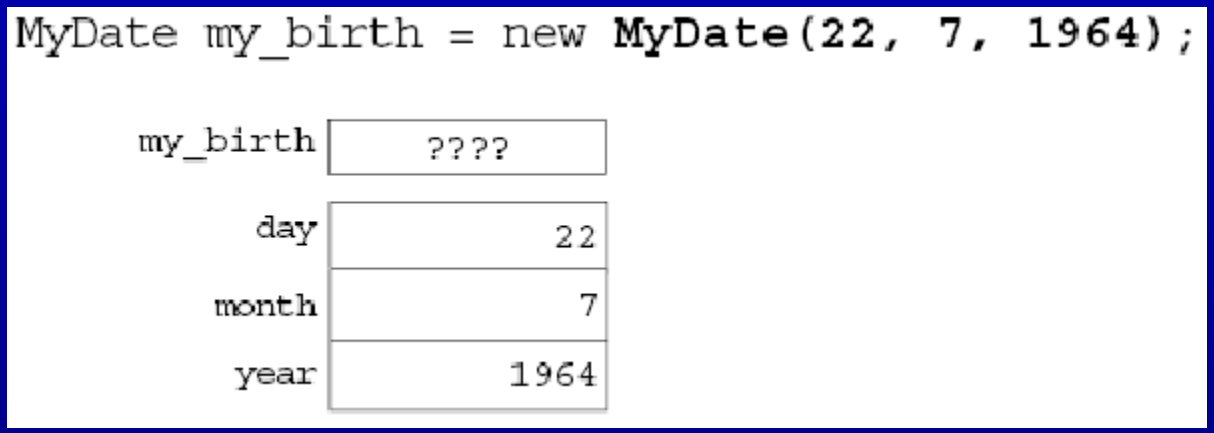
```
MyDate my_birth                                    ;
```


```
MyDate my_birth = new MyDate(                      );
```



Inicialización de atributos explícita

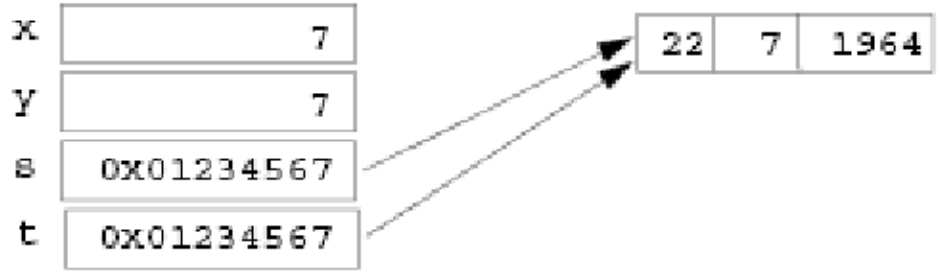
```
MyDate my_birth = new MyDate(22, 7, 1964);
```



Asignando Referencias

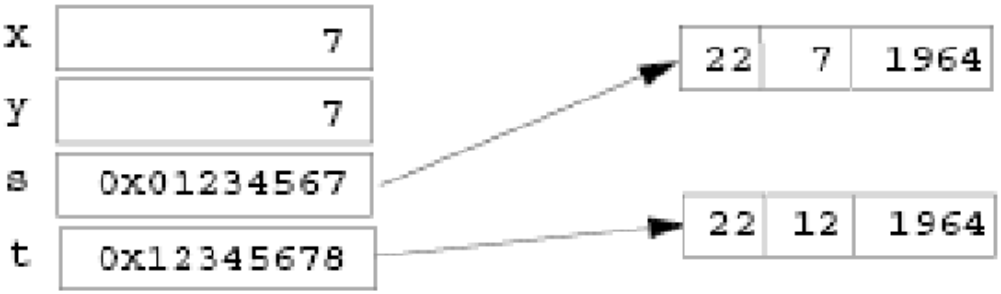
- **2 variables apuntando al mismo objeto**

```
1    int x = 7;
2    int y = x;
3    MyDate s = new MyDate(22, 7, 1964);
4    MyDate t = s;
```



- **2 variables apuntando a diferentes objetos**

```
5    t = new MyDate(22, 12, 1964);
```



Convenciones en la codificación de programas

- **Packages:**
  com.example.domain;
- **Classes, interfaces, and enum types:**
  SavingsAccount
- **Methods:**
  getAccount()
- **Variables:**
  currentCustomer
- **Constants:**
  HEAD_COUNT

Convenciones en la codificación de programas

- **Estructuras de Control**

```
if ( condición ) {
instrucción1;
} else {
instrucción2;
}
```

- **Espaciado:**

  Una instrucción por línea.
  Usar 2 ó 4 espacios por tabulador.

- **Comentarios:**

  Usar // para hacer comentarios dentro del código.
  Usar /* */ para los miembros de las clases.

4) Expresiones y Control de Flujo

Variables

- Las variables locales son aquellas que están definidas dentro de un método.
- Las variables son creadas cuando se ejecuta el método y destruidas cuando se termina la ejecución de dicho método.
- Las variables locales requieren inicialización explícita.
- Las variables instanciadas son iniciadas automáticamente.

Variables

- **El compilador verifica que las variables locales hayan sido inicializadas antes de ser usadas.**

```
3       public void doComputation() {
4         int x = (int)(Math.random() * 100);
5         int y;
6         int z;
7         if (x > 50) {
8           y = 9;
9         }
10        z = y + x;  // Possible use before initialization
11      }

javac TestInitBeforeUse.java
TestInitBeforeUse.java:10: variable y might not have been initialized
      z = y + x;  // Possible use before initialization
          ^
1 error
```

Operadores Aritméticos

/	División entera
*	Multiplicación
+	Suma
-	Resta
++	Incremento
--	Decremento
%	Residuo

Operadores de Relación

Condition	Operator	Example
Is equal to (or "is the same as")	==	`int i=1;` `(i == 1)`
Is not equal to (or "is not the same as")	!=	`int i=2;` `(i != 1)`
Is less than	<	`int i=0;` `(i < 1)`
Is less than or equal to	<=	`int i=1;` `(i <= 1)`
Is greater than	>	`int i=2;` `(i > 1)`
Is greater than or equal to	>=	`int i=1;` `(i >= 1)`

Precedencia de Operadores

Operators	Associative
++ -- + *unary* - *unary* ~ ! (<*data_type*>)	R to L
* / %	L to R
+ -	L to R
<< >> >>>	L to R
< > <= >= instanceof	L to R
== !=	L to R
&	L to R
^	L to R
|	L to R
&&	L to R
||	L to R
<*boolean_expr*> ? <*expr1*> : <*expr2*>	R to L
= *= /= %= += -= <<= >>= >>>= &= ^= |=	R to L

Operadores Lógicos

Operation	Operator	Example

If one condition AND another condition	&&	`int i = 2;`
		`int j = 8;`
		`((i < 1) && (j > 6))`

If either one condition OR another condition	||	`int i = 2;`
		`int j = 8;`
		`((i < 1) || (j > 10))`

NOT	!	`int i = 2;` `(!(i < 3))`

Operadores de Corrimiento

- **Operadores de corrimiento a la derecha:**

  128 >> 1 returns 128/21 = 64
  256 >> 4 returns 256/24 = 16
  -256 >> 4 returns -256/24 = -16

- **Operadores de Corrimiento a la Izquierda**

  128 << 1 returns 128 * 21 = 256
  16 << 2 returns 16 * 22 = 64

Concatenación de Cadenas

- Para concatenar cadenas se utiliza el operador “+”.
- Cuando se concatena una cadena, se produce una nueva cadena.

```
String titulo = "Dr.";
String nombre = “Juan" + " " + “Pérez";
String completo = titulo + " " + nombre;
```

- Los argumentos deben ser objetos de tipo cadena (String).
- Cuando se concatenan objetos que no son cadenas son convertidos a objetos de tipo cadena automáticamente.

Casting

- **Un casting se ocupa cuando es necesario convertir tipos de datos, aún sabiendo que existirá pérdida de información cuando se hace la conversión.**

- **Cuando se hace una cast entre "long" e "int" se requiere un cast explícito.**

```
long bigValue = 99L;
int squashed = bigValue;          // error, requiere un cast
int squashed = (int) bigValue;                // OK
int squashed = 99L;                           // error, requiere un cast
int squashed = (int) 99L;         // OK
```

Casting legal e ilegal

- **Las variables cambian automáticamente al nuevo formato y al nuevo tamaño cuando se hace un casting.**
- **Un casting es legal cuando al hacer el cambio, el tipo nuevo es tan grande como para contener al anterior tipo, y sería ilegal hacerlo al contrario.**

```
long bigval = 6;                    // 6 es int, OK
int smallval = 99L;     // 99L es long, ilegal
double z = 12.414F;     // 12.414F es float, OK
float z1 = 12.414;      // 12.414 es double, ilegal
```

Toma de decisiones (if)

- **La toma de decisiones se hace con el operador “if / else”. La sintaxis es:**

```
if ( <expresion_booleana> )
        <instrucción_1>
else
        <instrucción_2>
```

- **Cuando la expresión booleana se cumple, se ejecuta la instrucción o conjunto de instrucciones 1, en caso contrario, se ejecuta la instrucción ó el conjunto de instrucciones 2. Ejemplo:**

```
if ( x < 10 ) {
        System.out.println("Are you finished yet?");
        } else {
        System.out.println("Keep working...");
}
```

Toma de decisiones anidada (if)

- **Ejemplo:**

```
int count = getCount(); // a method defined in the class
if (count < 0) {
        System.out.println("Error: count value is negative.");
        } else if (count > getMaxCount()) {
                System.out.println("Error: count value is too big.");
                } else {
                System.out.println("There will be " + count +
                " people for lunch today.");
}
```

Toma de decisiones múltiple (switch)

- **La sintaxis es:**

```
switch ( <expression> ) {
        case <constant1>:
                <statement_or_block>*
                [break;]
        case <constant2>:
                <statement_or_block>*
                [break;]
        default:
                <statement_or_block>*
                [break;]
}
```

Toma de decisiones múltiple (switch)

- **Ejemplo:**

```
switch ( carModel ) {
        case DELUXE:
                addAirConditioning();
                addRadio();
                addWheels();
                addEngine();
                break;
        case STANDARD:
                addRadio();
                addWheels();
                addEngine();
                break;
        default:
                addWheels();
                addEngine();
}
```

Ciclos (for)

- **Sintaxis:**

  for (<init_expr>; <test_expr>; <alter_expr>)
  <statement_or_block>

- **Ejemplo:**

```
for ( int i = 0; i < 10; i++ )
System.out.println(i + " squared is " + (i*i));
```

Ciclos condicionales (while)

- **Sintaxis:**

```
while ( <test_expr> )
<statement_or_block>
```

- **Ejemplo:**

```
int i = 0;
while ( i < 10 ) {
System.out.println(i + " squared is " + (i*i));
i++;
}
```

Ciclos condicionales (do / while)

- **Sintaxis:**

```
do
        <statement_or_block>
while ( <test_expr> );
```

- **Ejemplo:**

```
int i = 0;
do {
        System.out.println(i + " squared is " + (i*i));
        i++;
} while ( i < 10 );
```

5) Arreglos

Arreglos

- **Un arreglo es un conjunto de datos del mismo tipo.**
- **La declaración de arreglos de tipos primitivos es:**

```
char s[];
Point p[];

char[] s;
Point[] p;
```

- **Un arreglo es un objeto y es creado con new.**
- **Cada arreglo empieza con una posición cero (0).**
- **Un arreglo no se puede redimensionar posteriormente.**

Arreglos de tipo char

```
1    public char[] createArray() {
2      char[] s;
3
4      s = new char[26];
5      for ( int i=0; i<26; i++ ) {
6        s[i] = (char) ('A' + i);
7      }
8
9      return s;
10   }
```

Inicialización de arreglos

```
String[] names;
names = new String[3];
names[0] = "Georgianna";
names[1] = "Jen";
names[2] = "Simon";
```

```
String[] names = {
    "Georgianna",
    "Jen",
    "Simon"
};
```

```
MyDate[] dates;
dates = new MyDate[3];
dates[0] = new MyDate(22, 7, 1964);
dates[1] = new MyDate(1, 1, 2000);
dates[2] = new MyDate(22, 12, 1964);
```

```
MyDate[] dates = {
    new MyDate(22, 7, 1964),
    new MyDate(1, 1, 2000),
    new MyDate(22, 12, 1964)
};
```

Arreglos multidimensionales

```
int[][] twoDim = new int[4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5];

int[][] twoDim = new int[][4]; // illegal
```

Arreglos multidimensionales

- **Arreglos no rectangulares:**

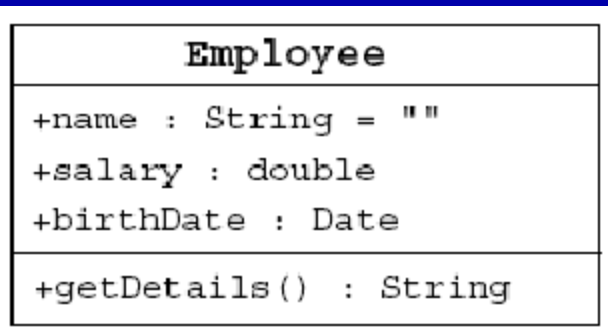
```
twoDim[0] = new int[2];
twoDim[1] = new int[4];
twoDim[2] = new int[6];
twoDim[3] = new int[8];
```

- **Arreglo de 4 filas conteniendo 5 enteros cada uno:**

```
int[][] twoDim = new int[4][5];
```

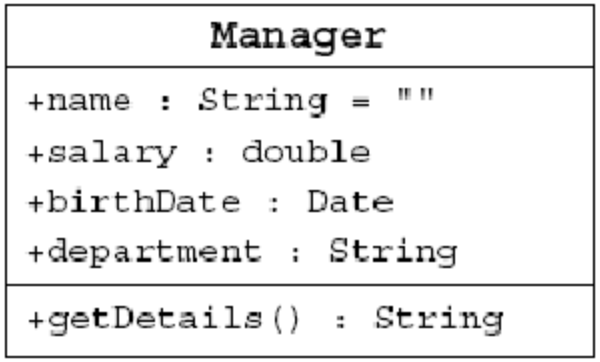
6) Diseño de Clases

Subclases



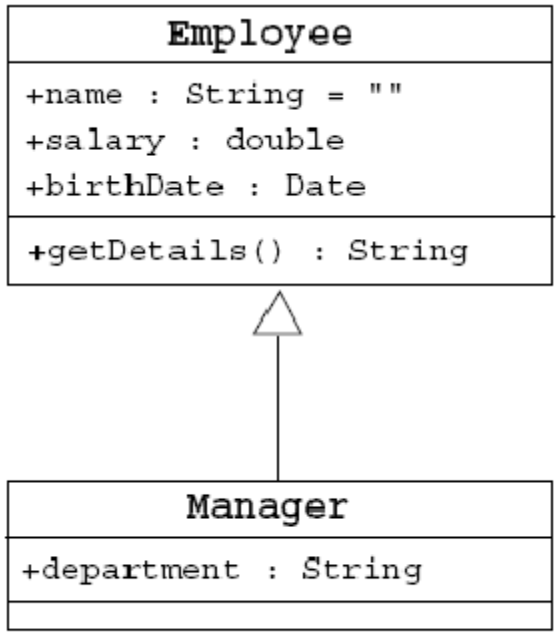

```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

Subclases




```
public class Manager {
   public String name = "";
   public double salary;
   public Date birthDate;
   public String department;

   public String getDetails() {...}
}
```

Creación de Subclases por Herencia




```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

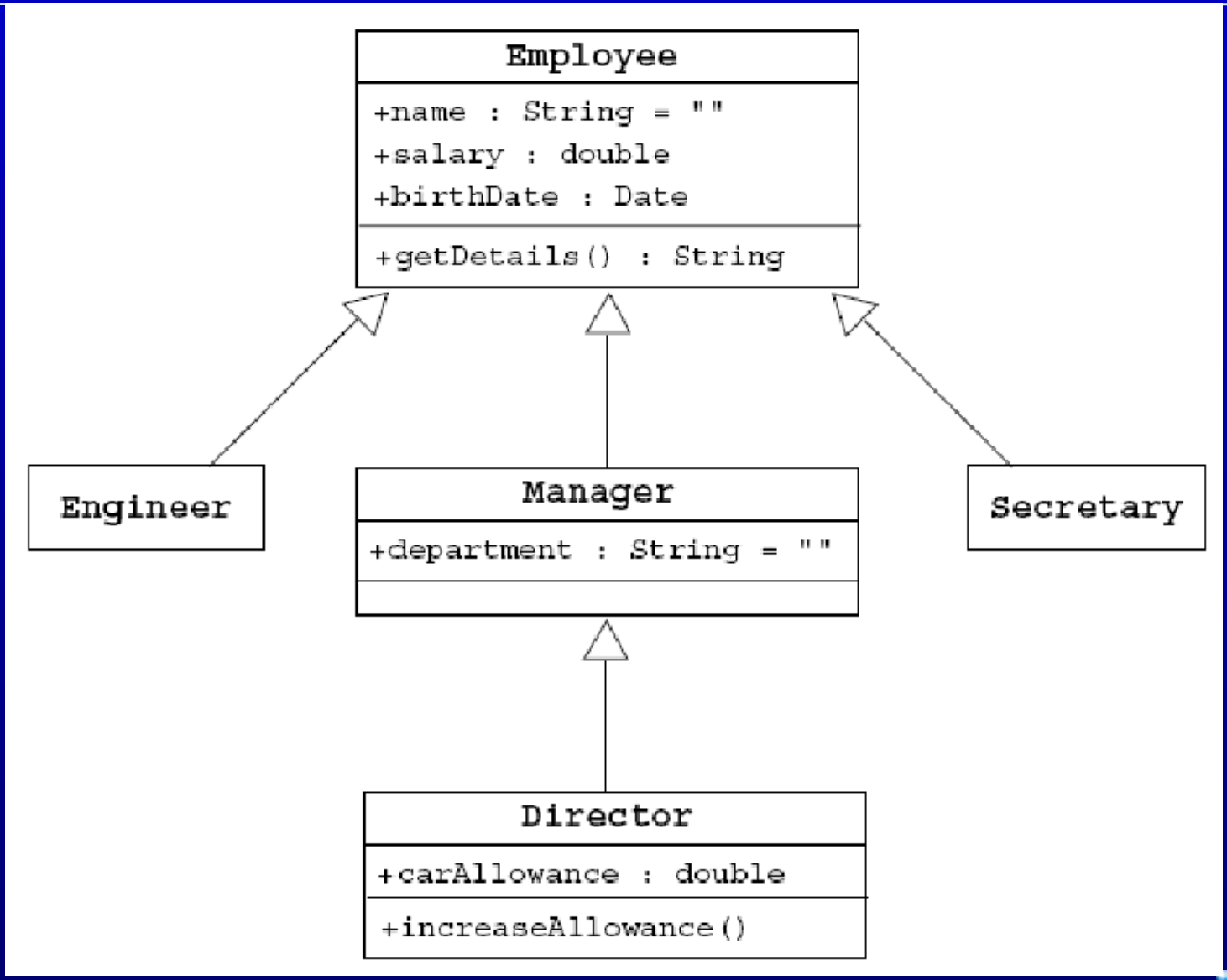
```
public class Manager extends Employee {
  public String department;
}
```

Herencia Simple

- **Cuando una clase herede sus propiedades de solamente una clase, se denomina herencia simple.**
- **Por medio del manejo de interfaces es posible el uso de herencia múltiple.**
- **La sintaxis para la creación de una interface es:**

```
<modifier> class <name> [extends <superclass>] {
<declaration>*
}
```

Herencia Simple



Control de acceso a una clase

Modifier	Same Class	Same Package	Subclass	Universe
`private`	Yes			
`default`	Yes	Yes		
`protected`	Yes	Yes	Yes	
`public`	Yes	Yes	Yes	Yes

Sobreescritura de métodos

- **Una subclase puede modificar el comportamiento de la clase de cual está heredando.**
- **Una subclase puede crear un método con diferente funcionalidad que el método que contiene su clase padre y que tiene el mismo nombre.**

Sobreescritura de métodos

```
1   public class Employee {
2     protected String name;
3     protected double salary;
4     protected Date birthDate;
5
6     public String getDetails() {
7       return "Name: " + name + "\n" +
8              "Salary: " + salary;
9     }
10  }
```

```
1   public class Manager extends Employee {
2     protected String department;
3
4     public String getDetails() {
5       return "Name: " + name + "\n" +
6              "Salary: " + salary + "\n" +
7              "Manager of: " + department;
8     }
9   }
```

Sobreescritura de métodos ilegal

```
1   public class Parent {
2     public void doSomething() {}
3   }

1   public class Child extends Parent {
2     private void doSomething() {} // illegal
3   }

1   public class UseBoth {
2     public void doOtherThing() {
3       Parent p1 = new Parent();
4       Parent p2 = new Child();
5       p1.doSomething();
6       p2.doSomething();
7     }
8   }
```

Invocación de Métodos de la Superclase

- Una subclase puede invocar los métodos de la superclase usando la palabra reservada “super”.
- La palabra “super” es usada en una clase para referirse a su superclase.
- La palabra “super” es usada para hacer referencia a la superclase, incluyendo datos y atributos de los métodos.

Invocación de Métodos de la Superclase

```
1   public class Employee {
2     private String name;
3     private double salary;
4     private Date birthDate;
5
6     public String getDetails() {
7       return "Name: " + name + "\nSalary: " + salary;
8     }
9   }
```

```
1   public class Manager extends Employee {
2     private String department;
3
4     public String getDetails() {
5       // call parent method
6       return super.getDetails()
7              + "\nDepartment: " + department;
8     }
9   }
```

Sobrecarga de Métodos

- **Una sobrecarga de un método ocurre cuando:**

```
public void hola(int i)
public int hola(float f)
public float hola(String s)
```

- **Como se observa, la lista de argumentos es diferente en cada caso, así como también es diferente el tipo de dato que regresa cada método.**

Modificadores de visibilidad

- **Los atributos y métodos pueden tener modificadores, que indican los niveles de acceso que otros objetos pueden tener sobre ellos.**

- **Los modificadores más comúnmente usados son:** public**,** private ***y*** protected**.**

- **El resto de los modificadores no tienen una clasificación específica:**

  final
  abstract
  static
  native
  transient
  syncronized
  volatile

Modificadores de acceso

- **Los modificadores de acceso aplican para:**

  La clase misma (class file).
  Sus variables de instancia.
  Sus métodos y constructores.
  Sus clases anidadas.

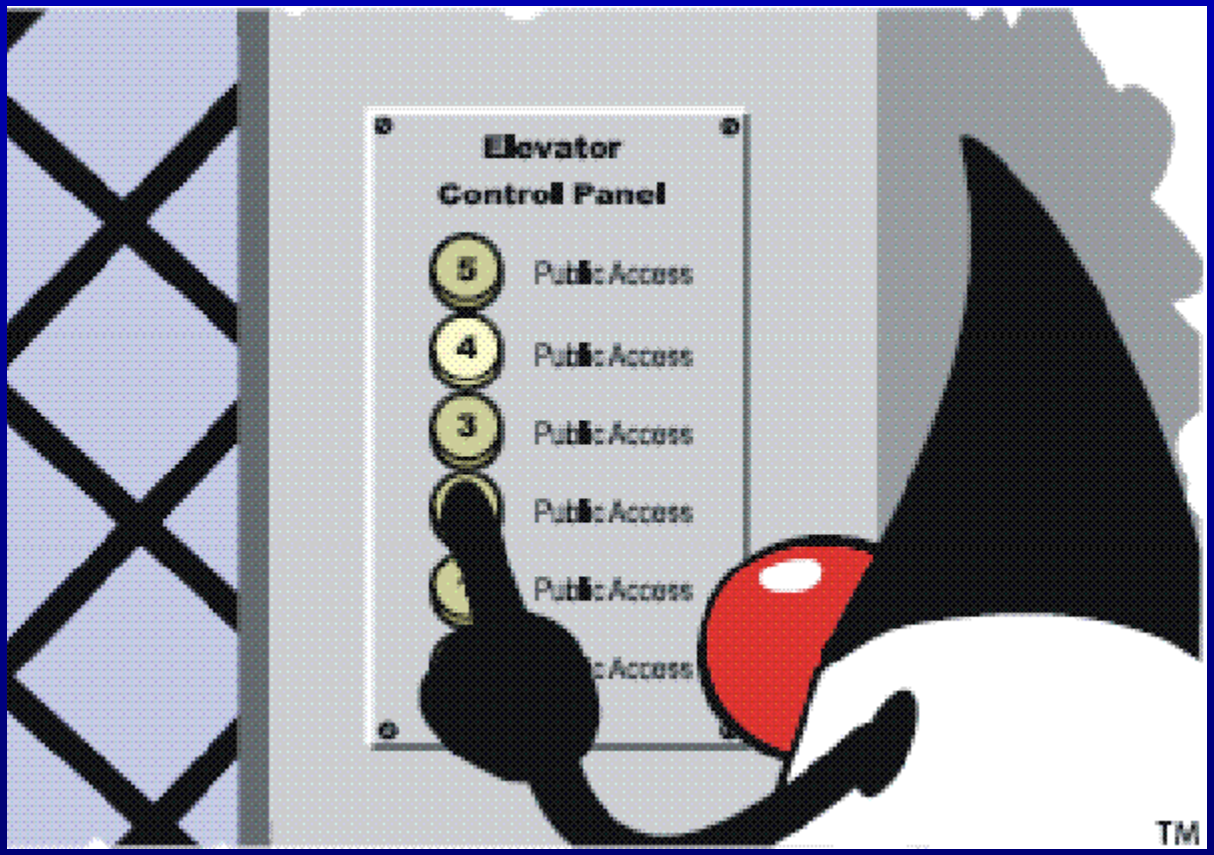
- **Con raras excepciones, las únicas variables que pueden ser controladas a través de modificadores de acceso son variables de instancia (de clase).**

- **Las variables que se declaran dentro de un método de una clase no pueden tener modificadores de acceso.**

- **Los modificadores de acceso son : public, private y protected.**

Modificador public

- Este modificador permite que la clase, sus atributos, y métodos sean visibles a cualquier objeto en el programa.

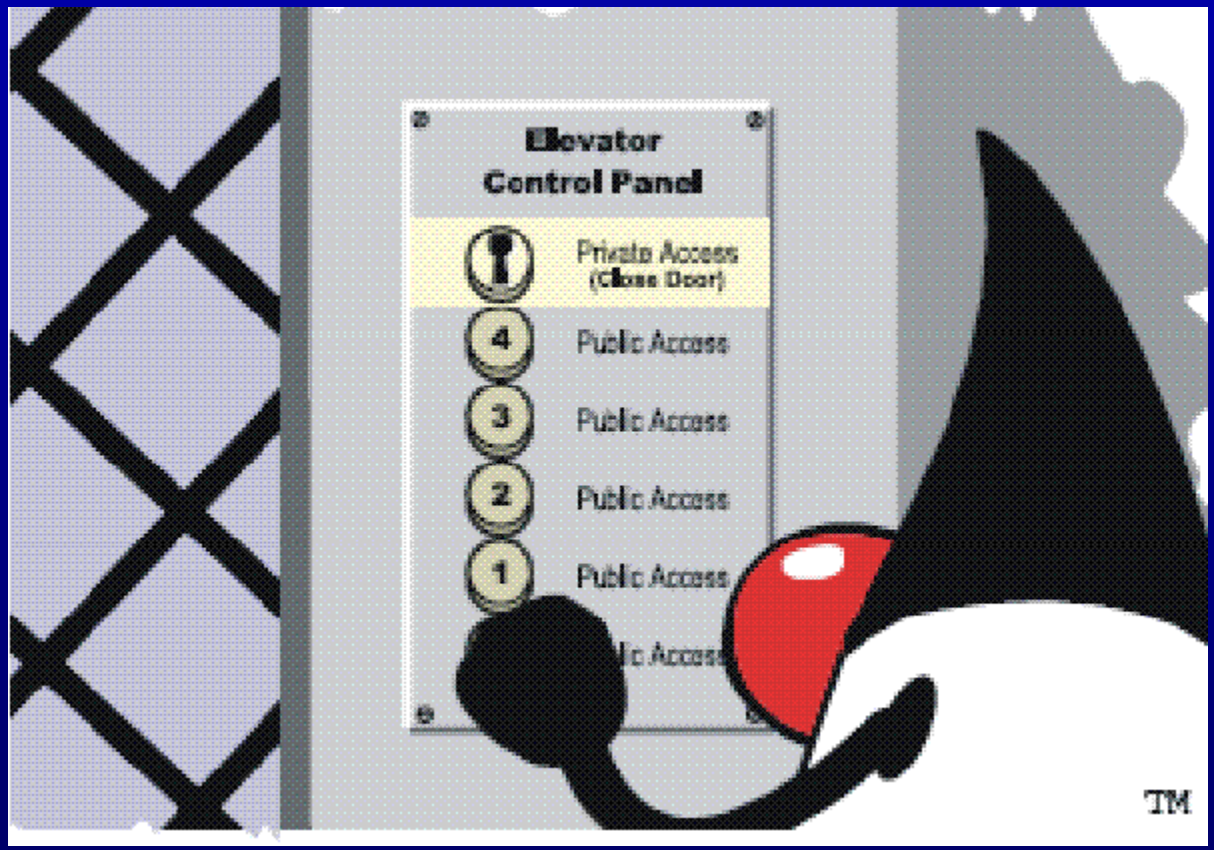


Convenciones del modificador public

- Es el único modificador de acceso permitido para clases no anidadas, no puede nunca existir una top-level class con private o protected.
- Una clase, variable o método público puede ser usado en cualquier programa de Java sin ninguna restricción.
- Cualquier método público puede ser sobreescrito por cualquier subclase.
- El main() es público porque puede ser invocado desde cualquier JRE.

Modificador private

- **Es el modificador de acceso menos generoso. Permite a los objetos de una clase, sus atributos y operaciones, ser inaccesibles por otros objetos. Los modificadores privados pueden ser accesados sólo por los miembros de la misma clase.**



Modificador default

- **Default es el nombre del acceso a clases, variables y métodos si no se ha especificado un modificador.**
- **Las características de una clase default son accesibles a cualquier clase en el mismo paquete de la clase en cuestión.**
- **Un método default puede ser sobreescrito por cualquier subclase que este en el mismo paquete que la superclase.**
- ***Default no es una palabra clave, es simplemente el nombre que se da al nivel de acceso que resulta de no especificar ningún modificador de acceso.***

Modificador protected

- El nombre de protected es un poco engañoso, ya que se piensa que es extremadamente restrictivo.
- Las características protected son aún mas accesibles que las características default.
- Solo variables y métodos pueden ser declarados protected.
- Una característica protected de una clase esta disponible para todas las clases en el mismo paquete (similar a default).
- Una característica protected de una clase puede estar disponible en forma limitada a TODAS las subclases de la clase que posee la característica protected; aún para las subclases que residan en diferentes paquetes que la clase.

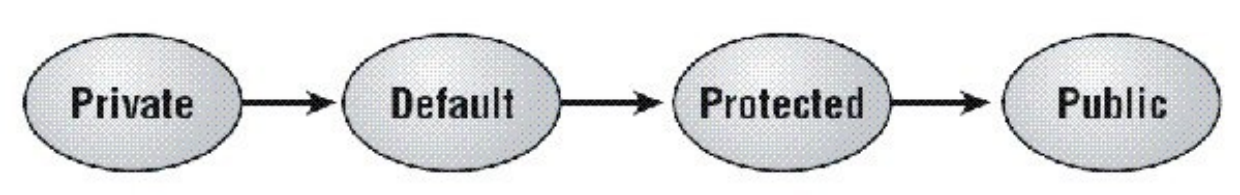
Límites para protected

- **Las subclases de diferentes paquetes tienen solo los siguientes privilegios:**
- **Ellas pueden sobreescribir métodos protected de la superclase.**
- **Una instancia puede leer y escribir campos protected que hereden de la superclase, sin embargo, la instancia no puede leer o escribir los campos protected heredados de otras instancias.**
- **Una instancia puede llamar métodos protected que herede de la superclase, sin embargo, la instancia no puede llamar métodos protected de otras instancias.**

Reglas de Sobreescritura

Las reglas para sobreescritura son:

- Un método private puede ser sobreescrito por un private, Default, Protected o Public.
- Un método Default puede ser sobreescrito por un default, protected o public.
- Un método protected puede ser sobreescrito por un protected o por un public
- Un método public solo puede ser sobreescrito por un public



Modificador final

- **Aplica a clases, variables y métodos, en general, significa que un elemento *final* no puede ser cambiado.**
- **Una clase *final* no puede tener subclases . Por ejemplo, el siguiente código no compila, porque la clase java.lang.Math es *final*.**

```
class SubMath extends java.lang.Math { }
```

- ***El compilador marca "Can't subclasses final classes"***
- **Una variable *final* no puede ser modificada una vez que le ha sido asignado un valor. En Java, las variables final son como const en C++ y #define en C.**

Modificador static

- Puede ser aplicado a variables y métodos. En ocasiones también a pequeños bloques de código que no pertenecen a ningún método.

```
1. class Ecstatic{
2.   static int x = 0;
3.   Ecstatic() { x++; }
4. }
```

- La variable x es static, esto significa que hay solamente una x, no importa cuantas instancias de la clase EcStatic existan, siempre habrá solo una x. Los 4 bytes de memoria ocupados por x son asignados cuando la clase EcStatic es cargada. La variable static es incrementada cada vez que el constructor es llamado, así será posible saber cuantas instancias han sido creadas.

Modificador native

- El modificador native puede referir sólo a métodos. Así como el modificador abstract, el native indica que el cuerpo de un método será encontrado en otro lugar.

- En el caso de métodos abstractos, el cuerpo está en una subclase; con métodos nativos, el cuerpo permanece totalmente fuera de la Java Virtual Machine, en un librería. El código native es escrito en un lenguaje fuera de Java, típicamente C y C++.

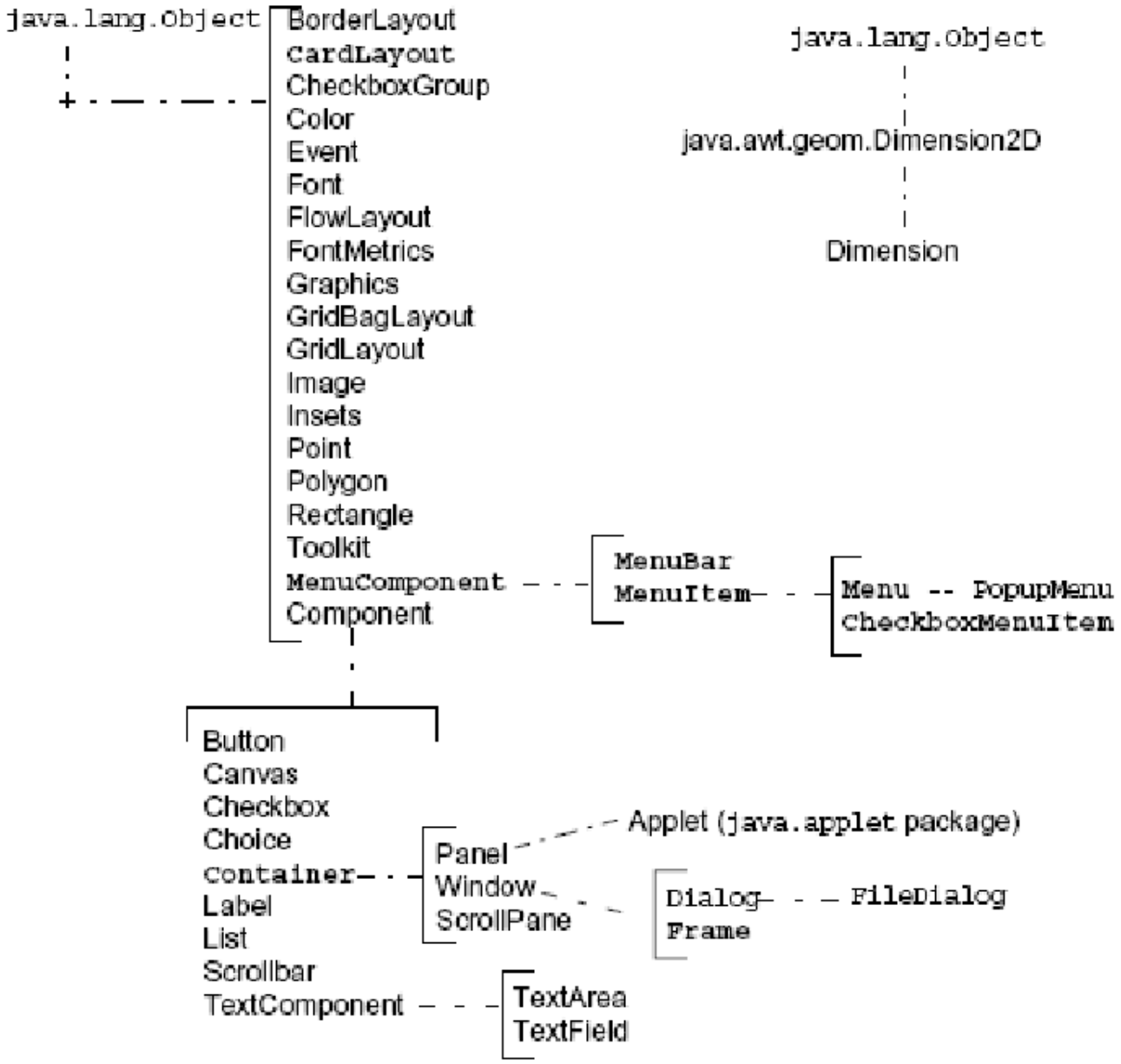
```
1. class NativeExample {
2.   native void doSomethingLocal(int i);
3.
4.   static {
5.     System.loadLibrary("MyNativeLib");
6.   }
7. }
```

7) Interfaz Gráfica

AWT

- AWT es el Abstract Window Toolkit package, el cual contiene contenedores, componentes y manejadores de capa, y define la forma en que ellos trabajan para construir una Interfaz Gráfica del Usuario (en inglés GUI – Graphical User Interface).

The java.awt Package



Contenedores

- Para agregar componentes se utiliza el método add().
- Los 2 tipos principales de contenedores son Window y Panel.
- Un contenedor Window es una ventana flotando libremente en la pantalla.
- Un contenedor Panel esta formado por componentes GUI que deben existir en el contexto de algún otro contenedor, como Window o Applet.

Posicionando componentes

- La posición y tamaño de un componente dentro de un contenedor esta determinado por manejador de capas.
- Se puede controlar el tamaño o la posición de componentes deshabilitando el manejador de capas.
- Se deben usar los siguientes métodos para controlar los componentes:

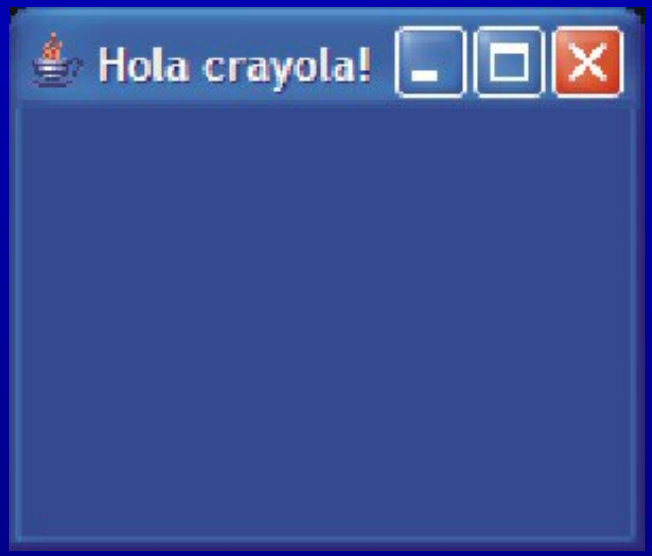
setLocation()
setSize()
setBounds()

Frames

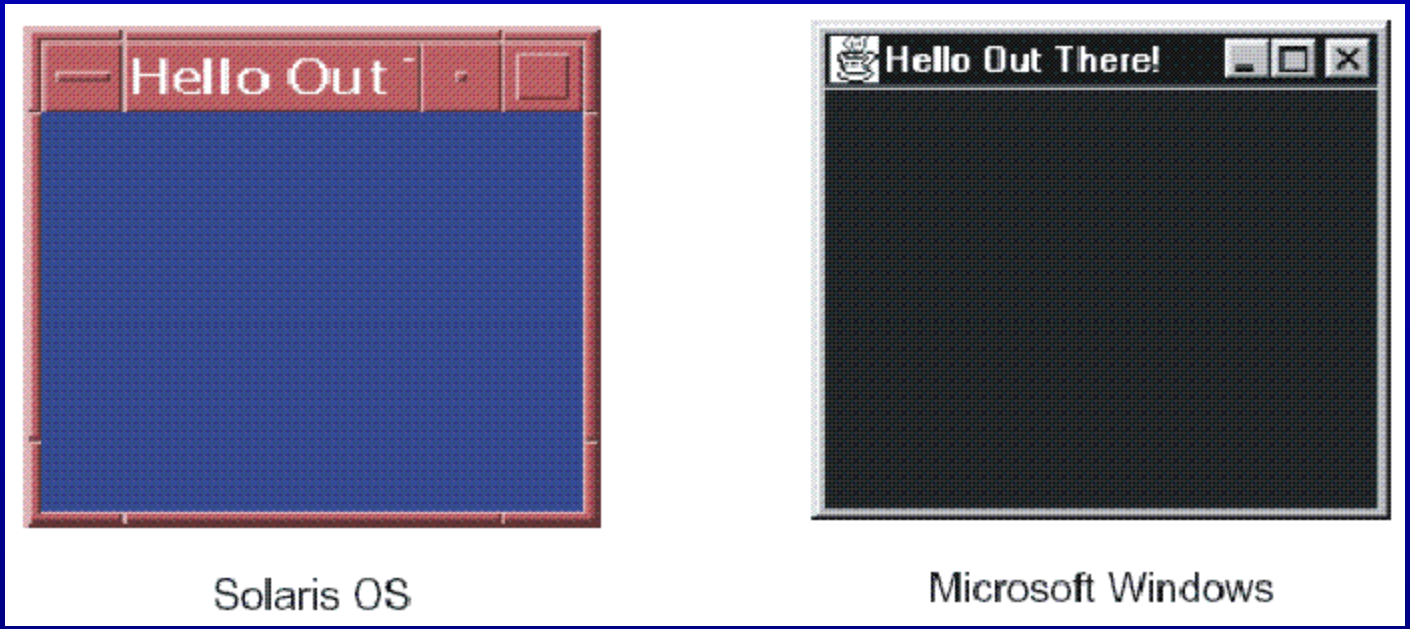
- Los Frames tienen las siguientes características:
  - Son subclases de Window.
  - Tienen un título y esquinas para redimensionarlo.
  - Inicialmente son invisibles, por lo que es necesario usar setVisible(true) para poder observarlo en pantalla.
  - Utiliza el BorderLayout como el manejador de capas por default.
  - Se usa el método setLayout para cambiar el manejador de capas.

Frames

```
import java.awt.*;

public class EjemploFrame {
private Frame f;

        public EjemploFrame() {
                f = new Frame("Hola crayola!");
        }

        public void lanzarFrame() {
                f.setSize(170,170);
                f.setBackground(Color.blue);
                f.setVisible(true);
        }

        public static void main(String args[]) {
                EjemploFrame ventana = new EjemploFrame();
                ventana.lanzarFrame();
        }
}
```



Frames



Panels

- Los Panels nos permiten nos permiten espacio para alojar componentes.
- Esto permite que los subpanels tengan su propio manejador de capas.

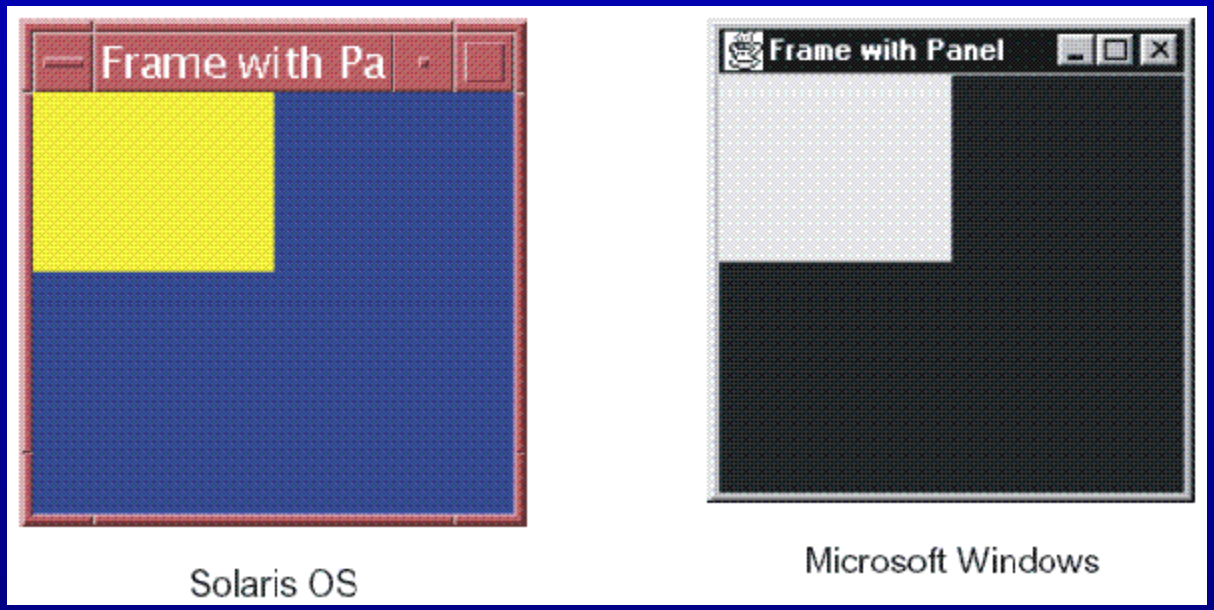
Panels

```
1    import java.awt.*;
2
3    public class FrameWithPanel {
4      private Frame f;
5      private Panel pan;
6
7      public FrameWithPanel(String title) {
8        f = new Frame(title);
9        pan = new Panel();
10     }
```

Panels

```
11
12      public void launchFrame() {
13        f.setSize(200,200);
14        f.setBackground(Color.blue);
15        f.setLayout(null);  // Use default layout
16
17        pan.setSize(100,100);
18        pan.setBackground(Color.yellow);
19        f.add(pan);
20        f.setVisible(true);
21      }
22
23      public static void main(String args[]) {
24        FrameWithPanel guiWindow =
25            new FrameWithPanel("Frame with Panel");
26        guiWindow.launchFrame();
27      }
28    }
```

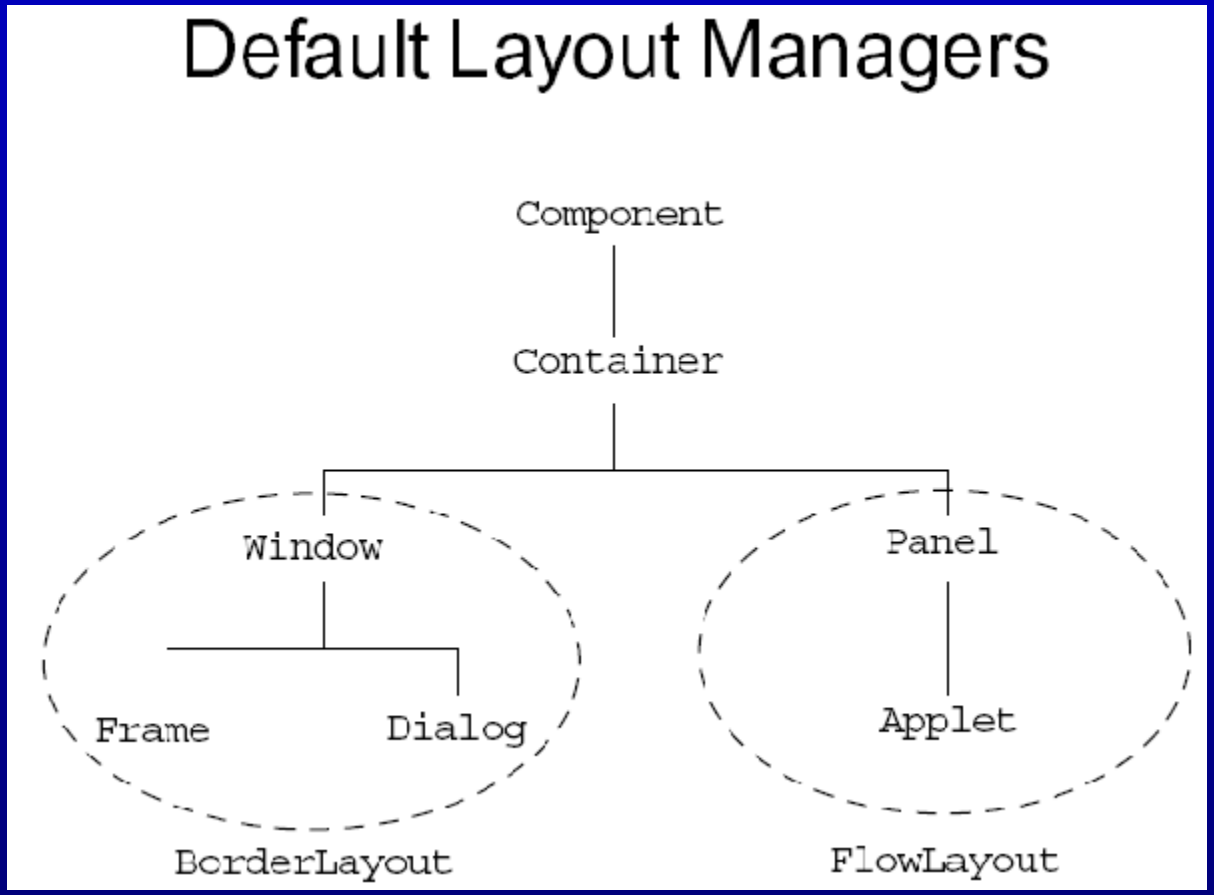
Panels


Frame with Pa
Solaris OS
Frame with Panel
Microsoft Windows

Layout Managers

- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- GridBagLayout

Layout Managers



Layout Managers

```
1    import java.awt.*;
2
3    public class LaycutExample {
4      private Frame f;
5      private Button b1;
6      private Button b2;
7
8      public LayoutExample() {
9        f = new Frame("GUI example");
10       b1 = new Button("Press Me");
11       b2 = new Button("Don't press Me");
12     }
```

Layout Managers

```
13
14    public void launchFrame() {
15      f.setLayout(new FlowLayout());
16      f.add(b1);
17      f.add(b2);
18      f.pack();
19      f.setVisible(true);
20    }
21
22    public static void main(String args[]) {
23      LayoutExample guiWindow = new LayoutExample();
24      guiWindow.launchFrame();
25    }
26
27  } // end of LayoutExample class
```

Layout Managers

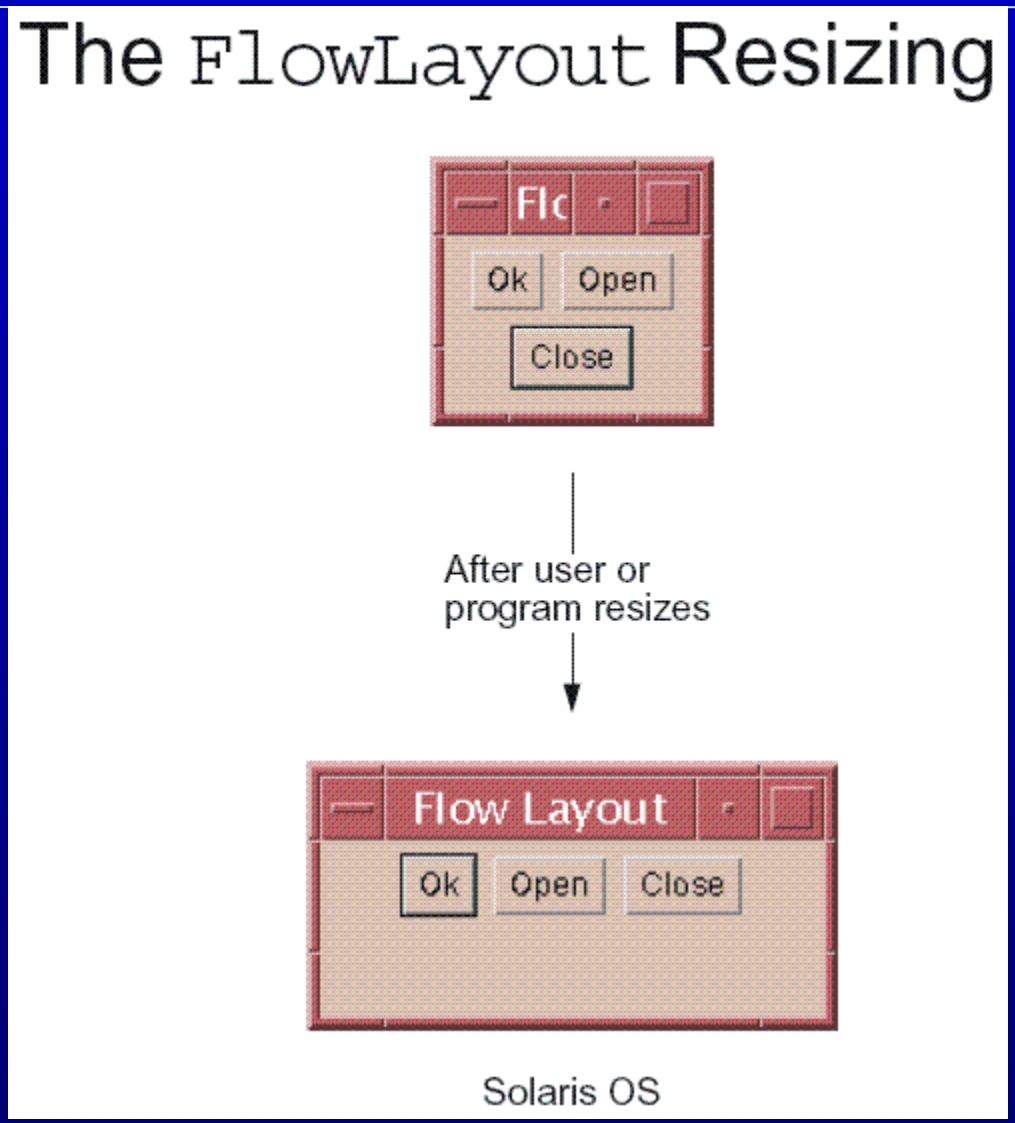
Solaris OS

Microsoft Windows

FlowLayout Manager

- El FlowLayout manager tiene las siguientes características:
  - Le da la forma al layout por default.
  - Agrega componentes de izquierda a derecha.
  - La alineación es centrada.
  - Utiliza el constructor para ajustar el comportamiento.

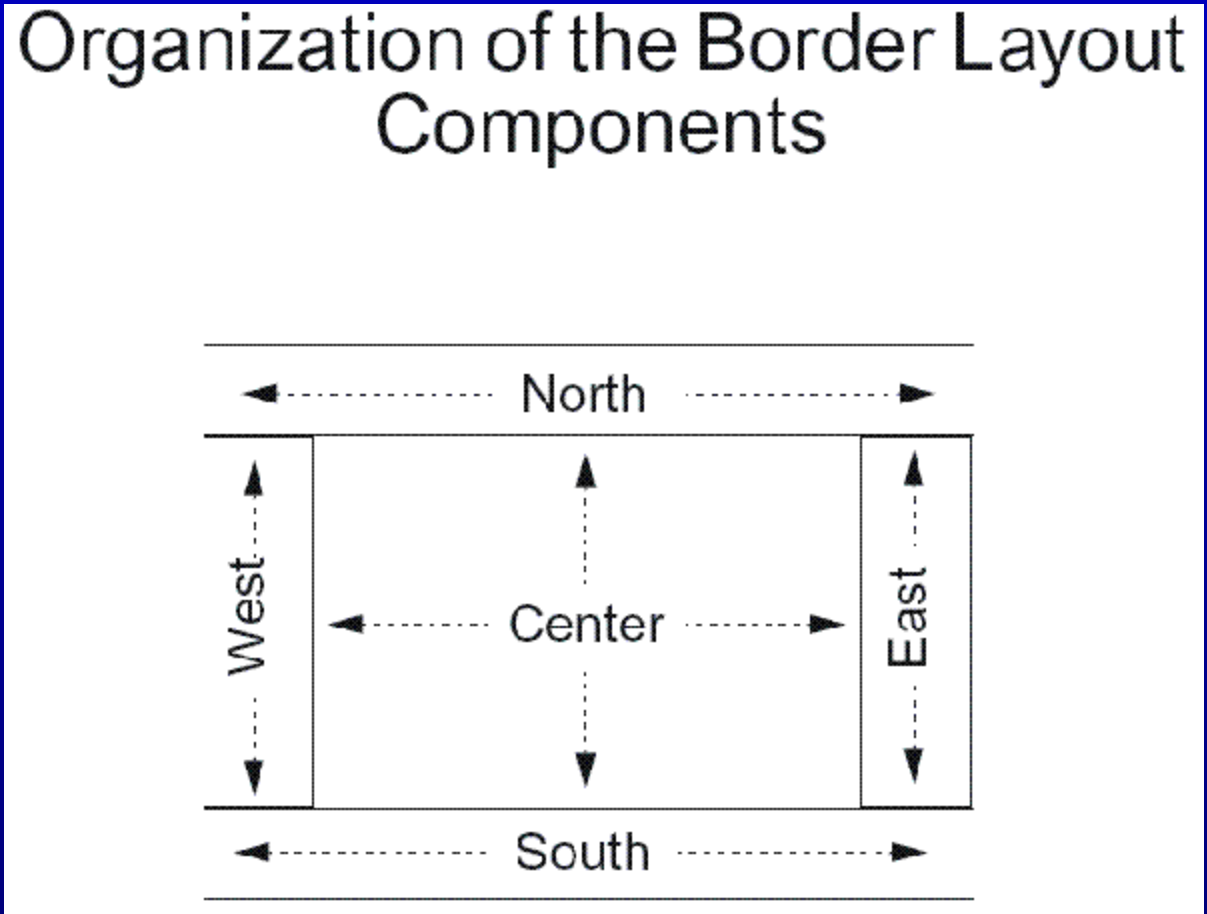
FlowLayout Manager



BorderLayout Manager

- El BorderLayout Manager es el layout por default.
- Los componentes se agregan a regiones específicas.
- Para el ajuste del tamaño de manera horizontal se utilizan las regiones North, South y Center.
- Para ajustar verticalmente se utilizan las regiones East, West y Center.

BorderLayout Manager



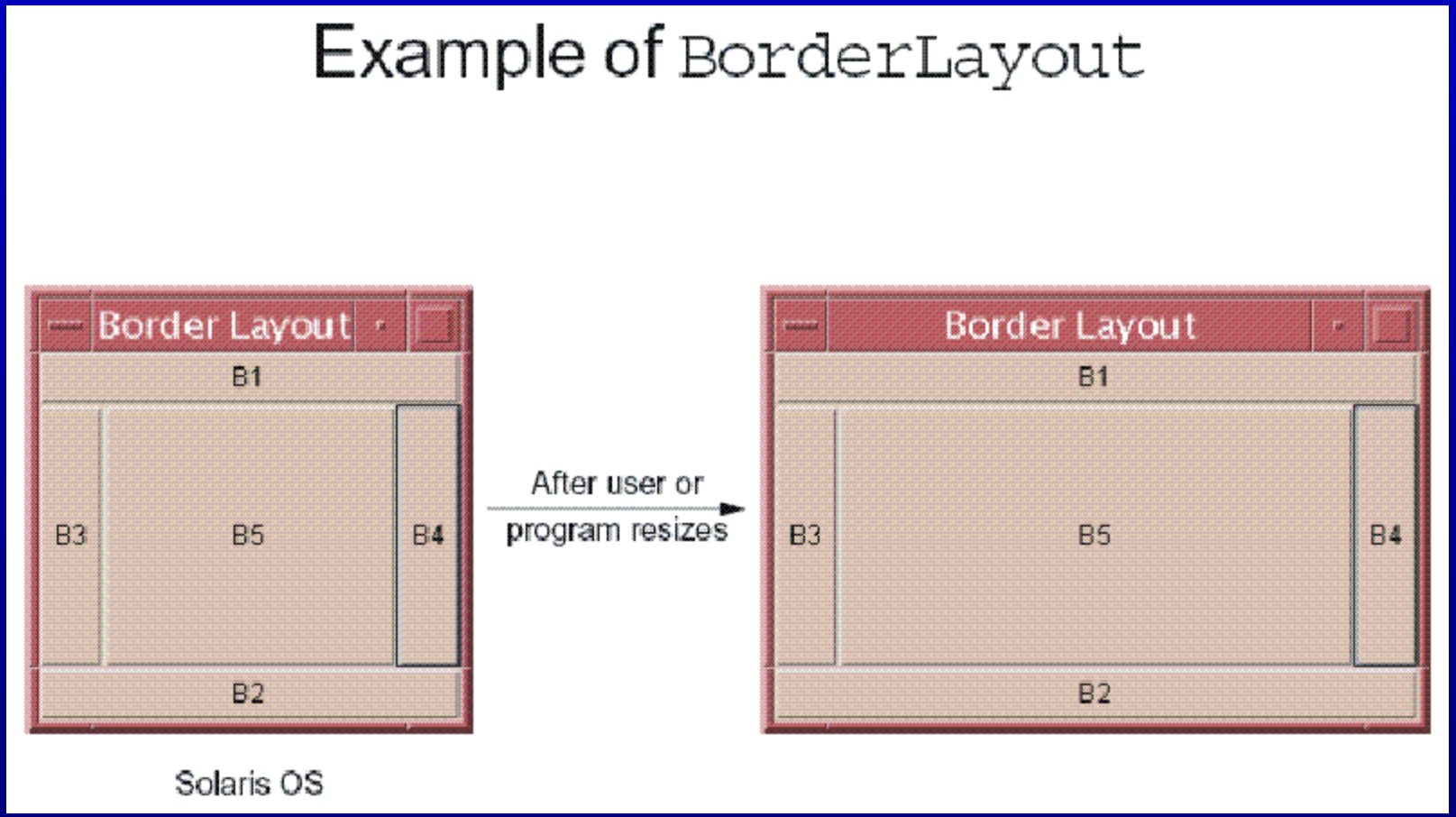
BorderLayout Manager

```
1    import java.awt.*;
2
3    public class BorderExample {
4      private Frame f;
5      private Button bn, bs, bw, be, bc;
6
7      public BorderExample() {
8        f = new Frame("Border Layout");
9        bn = new Button("B1");
10       bs = new Button("B2");
11       bw = new Button("B3");
12       be = new Button("B4");
13       bc = new Button("B5");
14     }
```

BorderLayout Manager

```
15
16    public void launchFrame() {
17      f.add(bn, BorderLayout.NORTH);
18      f.add(bs, BorderLayout.SOUTH);
19      f.add(bw, BorderLayout.WEST);
20      f.add(be, BorderLayout.EAST);
21      f.add(bc, BorderLayout.CENTER);
22      f.setSize(200,200);
23      f.setVisible(true);
24    }
25
26    public static void main(String args[]) {
27      BorderExample guiWindow2 = new BorderExample();
28      guiWindow2.launchFrame();
29    }
30  }
```

BorderLayout Manager



GridLayout Manager

- Los componentes se agregan de izquierda a derecha, de arriba hacia abajo.
- Todas las regiones son de igual tamaño.
- El constructor especifica los renglones y las columnas.

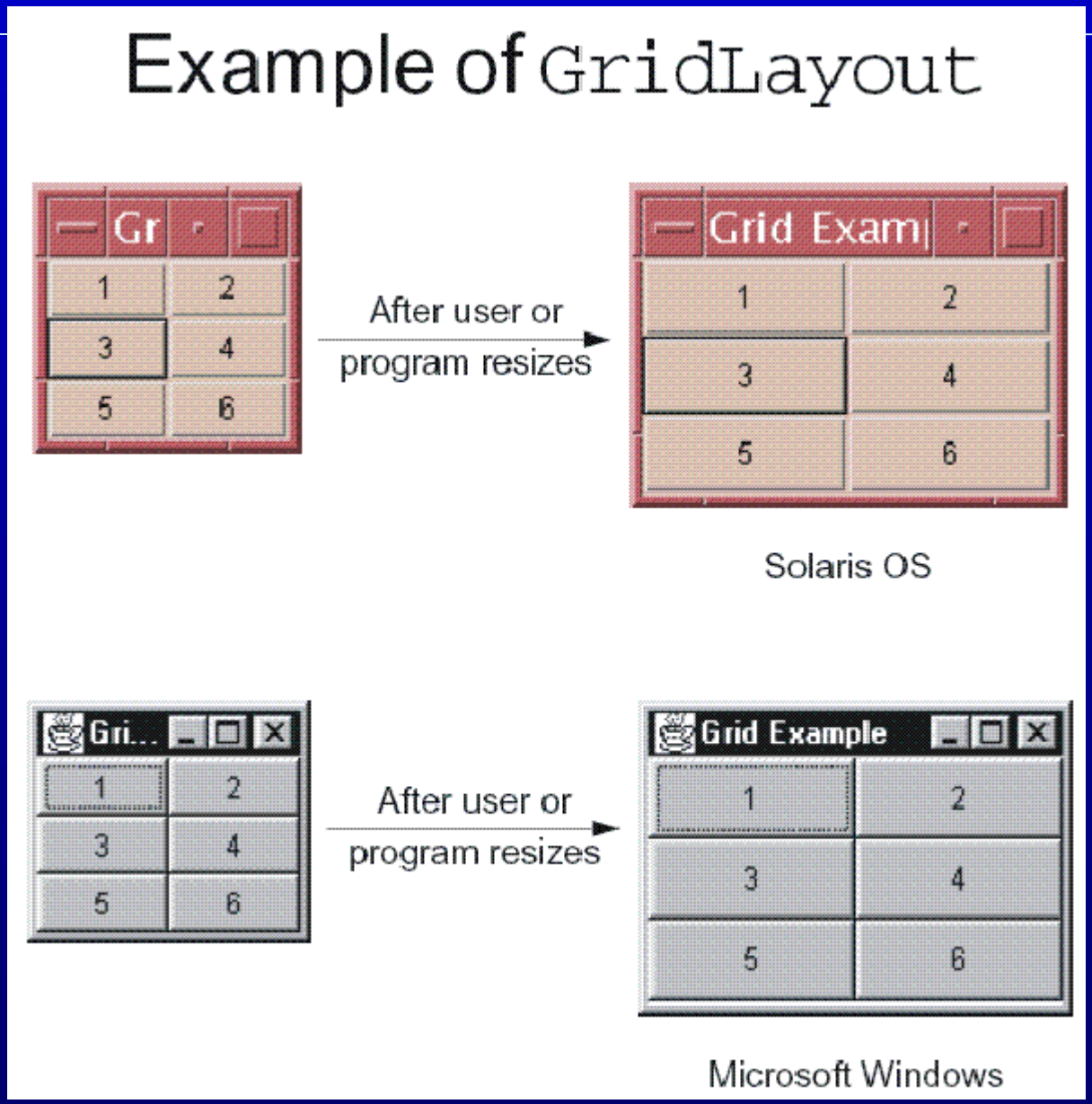
GridLayout Manager

```
1    import java.awt.*;
2
3    public class GridExample {
4      private Frame f;
5      private Button b1, b2, b3, b4, b5, b6;
6
7      public GridExample() {
8        f = new Frame("Grid Example");
9        b1 = new Button("1");
10       b2 = new Button("2");
11       b3 = new Button("3");
12       b4 = new Button("4");
13       b5 = new Button("5");
14       b6 = new Button("6");
15     }
```

GridLayout Manager

```
16
17    public void launchFrame() {
18      f.setLayout (new GridLayout(3,2));
19      f.add(b1);
20      f.add(b2);
21      f.add(b3);
22      f.add(b4);
23      f.add(b5);
24      f.add(b6);
25      f.pack();
26      f.setVisible(true);
27    }
28
29    public static void main(String args[]) {
30      GridExample grid = new GridExample();
31      grid.launchFrame();
32    }
33  }
```

GridLayout Manager



Hola Mundo con AWT y Swing

```
import java.awt.*;
import javax.swing.*;

public class AwtSwing extends JFrame {

        public AwtSwing()
        {
                super("Hola Mundo con AWT y Swing");
                setSize( 300, 100 );
                setVisible( true );
        }

        public void paint ( Graphics g )
        {
                super.paint( g );
                g.drawString("Hola Mundo", 50, 50 );
        }

        public static void main (String args[])
        {
                AwtSwing aplicacion = new AwtSwing();
                aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Uso de JColorChooser

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class ShowColors2 extends JFrame {
  private JButton changeColorButton;   private Color color = Color.LIGHT_GRAY;   private Container container;
  public ShowColors2()
  {
    super( "Uso de JColorChooser" );
    container = getContentPane();
    container.setLayout( new FlowLayout() );
    changeColorButton = new JButton( "Cambiar color" );
    changeColorButton.addActionListener(
      new ActionListener() {
        public void actionPerformed( ActionEvent event )
        {
          color = JColorChooser.showDialog(ShowColors2.this, "Escoge un color", color );
          if ( color == null )
            color = Color.LIGHT_GRAY;
          container.setBackground( color );
        }
      }
    );
    container.add( changeColorButton );
    setSize( 400, 130 ); setVisible( true );
  }
  public static void main( String args[] )
  {
    ShowColors2 application = new ShowColors2();      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }
}
```

Cambiando Tipos de Letra

```
import java.awt.*;import javax.swing.*;

public class Fonts extends JFrame {

   public Fonts()
   {
      super( "Cambiando tipos de letra" );      setSize( 420, 125 );      setVisible( true );
   }

   public void paint( Graphics g )
   {
      super.paint( g );
      g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
      g.drawString( "Serif 12 point bold.", 20, 50 );
      g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
      g.drawString( "Monospaced 24 point italic.", 20, 70 );
      g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
      g.drawString( "SansSerif 14 point plain.", 20, 90 );
      g.setColor( Color.RED );
      g.setFont( new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
      g.drawString( g.getFont().getName() + " " + g.getFont().getSize() +
         " point bold italic.", 20, 110 );
   }

   public static void main( String args[] )
   {
      Fonts application = new Fonts();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }
}
```

Dibujando Arcos

```
import java.awt.*;import javax.swing.*;
public class DrawArcs extends JFrame {
  public DrawArcs()
  {
    super( "Dibujando Arcos" );      setSize( 300, 170 );      setVisible( true );
  }
  public void paint( Graphics g )
  {
    super.paint( g );
    g.setColor( Color.YELLOW );
    g.drawRect( 15, 35, 80, 80 );
    g.setColor( Color.BLACK );
    g.drawArc( 15, 35, 80, 80, 0, 360 );
    g.setColor( Color.YELLOW );
    g.drawRect( 100, 35, 80, 80 );
    g.setColor( Color.BLACK );
    g.drawArc( 100, 35, 80, 80, 0, 110 );
    g.setColor( Color.YELLOW );
    g.drawRect( 185, 35, 80, 80 );
    g.setColor( Color.BLACK );
    g.drawArc( 185, 35, 80, 80, 0, -270 );
    g.fillArc( 15, 120, 80, 40, 0, 360 );
    g.fillArc( 100, 120, 80, 40, 270, -90 );
    g.fillArc( 185, 120, 80, 40, 0, -270 );
  }
  public static void main( String args[] )
  {
    DrawArcs application = new DrawArcs();      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }
}
```

Líneas Rectángulos y Ovalos

```
import java.awt.*;import javax.swing.*;
public class LinesRectsOvals extends JFrame {
  public LinesRectsOvals()
  {
    super( "Drawing lines, rectangles and ovals" );     setSize( 400, 165 );     setVisible( true );
  }
  public void paint( Graphics g )
  {
    super.paint( g );
    g.setColor( Color.RED );
    g.drawLine( 5, 30, 350, 30 );
    g.setColor( Color.BLUE );
    g.drawRect( 5, 40, 90, 55 );
    g.fillRect( 100, 40, 90, 55 );
    g.setColor( Color.CYAN );
    g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
    g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
    g.setColor( Color.YELLOW );
    g.draw3DRect( 5, 100, 90, 55, true );
    g.fill3DRect( 100, 100, 90, 55, false );
    g.setColor( Color.MAGENTA );
    g.drawOval( 195, 100, 90, 55 );
    g.fillOval( 290, 100, 90, 55 );
  }
  public static void main( String args[] )
  {
    LinesRectsOvals application = new LinesRectsOvals();
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }
}
```

Dibujando Polígonos

```
import java.awt.*;import javax.swing.*;
public class DrawPolygons extends JFrame {
  public DrawPolygons()
  {
    super( "Dibujando Polígonos" );      setSize( 275, 230 );      setVisible( true );
  }
  public void paint( Graphics g )
  {
    super.paint( g );
    int xValues[] = { 20, 40, 50, 30, 20, 15 };      int yValues[] = { 50, 50, 60, 80, 80, 60 };
    Polygon polygon1 = new Polygon( xValues, yValues, 6 );
    g.drawPolygon( polygon1 );
    int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };      int yValues2[] = { 100, 100, 110, 110, 130, 110, 90 };
    g.drawPolyline( xValues2, yValues2, 7 );
    int xValues3[] = { 120, 140, 150, 190 };      int yValues3[] = { 40, 70, 80, 60 };
    g.fillPolygon( xValues3, yValues3, 4 );
    Polygon polygon2 = new Polygon();
    polygon2.addPoint( 165, 135 );
    polygon2.addPoint( 175, 150 );
    polygon2.addPoint( 270, 200 );
    polygon2.addPoint( 200, 220 );
    polygon2.addPoint( 130, 180 );
    g.fillPolygon( polygon2 );
  }
  public static void main( String args[] )
  {
    DrawPolygons application = new DrawPolygons();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }
}
```

JLabel

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class LabelTest extends JFrame {
   private JLabel label1, label2, label3;
   public LabelTest()
   {
      super( "Probando JLabel" );
      Container container = getContentPane();
      container.setLayout( new FlowLayout() );
      label1 = new JLabel( "Etiquetas con texto" );
      label1.setToolTipText( "Esta es la etiqueta 1" );
      container.add( label1 );
      Icon bug = new ImageIcon( "bug1.gif" );
      label2 = new JLabel( "Etiqueta con texto e icono", bug, SwingConstants.LEFT );
      label2.setToolTipText( "Esta es la etiqueta 2" );
      container.add( label2 );
      label3 = new JLabel();
      label3.setText( "Etiqueta con icono y texto en la parte inferior" );
      label3.setIcon( bug );
      label3.setHorizontalTextPosition( SwingConstants.CENTER );
      label3.setVerticalTextPosition( SwingConstants.BOTTOM );
      label3.setToolTipText( "Esta es la etiqueta 3" );
      container.add( label3 );
      setSize( 275, 170 );
      setVisible( true );
   }
   public static void main( String args[] )
   {
      LabelTest application = new LabelTest();      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }
}
```

Campos de texto y manejo de eventos (1)

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;

public class TextFieldTest extends JFrame {
   private JTextField textField1, textField2, textField3;
   private JPasswordField passwordField;

   public TextFieldTest()
   {
      super( "Probando JTextField y JPasswordField" );
      Container container = getContentPane();
      container.setLayout( new FlowLayout() );
      textField1 = new JTextField( 10 );
      container.add( textField1 );
      textField2 = new JTextField( "Introducir texto aquí" );
      container.add( textField2 );
      textField3 = new JTextField( "Texto no editable", 20 );
      textField3.setEditable( false );
      container.add( textField3 );
      passwordField = new JPasswordField( "Texto Oculto" );
      container.add( passwordField );
      TextFieldHandler handler = new TextFieldHandler();
      textField1.addActionListener( handler );
      textField2.addActionListener( handler );
      textField3.addActionListener( handler );
      passwordField.addActionListener( handler );
      setSize( 325, 100 );
      setVisible( true );
   }
```

Campos de texto y manejo de eventos (2)

```
public static void main( String args[] )
  {
    TextFieldTest application = new TextFieldTest();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

  private class TextFieldHandler implements ActionListener {

    public void actionPerformed( ActionEvent event )
    {
      String string = "";

      if ( event.getSource() == textField1 )
        string = "textField1: " + event.getActionCommand();
      else if ( event.getSource() == textField2 )
        string = "textField2: " + event.getActionCommand();
      else if ( event.getSource() == textField3 )
        string = "textField3: " + event.getActionCommand();
      else if ( event.getSource() == passwordField ) {
        string = "passwordField: " +
          new String( passwordField.getPassword() );
      }
      JOptionPane.showMessageDialog( null, string );
    }
  }
}
```

Botones (1)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ButtonTest extends JFrame {
   private JButton plainButton, fancyButton;

   public ButtonTest()
   {
      super( "Prueba de Botones" );

      Container container = getContentPane();
      container.setLayout( new FlowLayout() );

      plainButton = new JButton( "Botón simple" );
      container.add( plainButton );

      Icon bug1 = new ImageIcon( "bug1.gif" );
      Icon bug2 = new ImageIcon( "bug2.gif" );
      fancyButton = new JButton( "Botón Elegante", bug1 );
      fancyButton.setRolloverIcon( bug2 );
      container.add( fancyButton );

      ButtonHandler handler = new ButtonHandler();
      fancyButton.addActionListener( handler );
      plainButton.addActionListener( handler );

      setSize( 275, 100 );
      setVisible( true );
   }
```

Botones (2)

```
public static void main( String args[] )
  {
    ButtonTest application = new ButtonTest();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

  private class ButtonHandler implements ActionListener {

    public void actionPerformed( ActionEvent event )
    {
      JOptionPane.showMessageDialog( ButtonTest.this,
        "Tu presionaste: " + event.getActionCommand() );
    }

  }

}
```

JCheckBox (1)

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;

public class CheckBoxTest extends JFrame {
   private JTextField field;
   private JCheckBox bold, italic;

   public CheckBoxTest()
   {
      super( "Probando JCheckBox" );

      Container container = getContentPane();
      container.setLayout( new FlowLayout() );
      field = new JTextField( "Observa el cambio del tipo de letra", 20 );
      field.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
      container.add( field );

      bold = new JCheckBox( "Bold" );
      container.add( bold );

      italic = new JCheckBox( "Italic" );
      container.add( italic );

      CheckBoxHandler handler = new CheckBoxHandler();
      bold.addItemListener( handler );
      italic.addItemListener( handler );

      setSize( 275, 100 );
      setVisible( true );
   }
```

JCheckBox (2)

```
public static void main( String args[] )
   {
      CheckBoxTest application = new CheckBoxTest();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

   private class CheckBoxHandler implements ItemListener {
      private int valBold = Font.PLAIN;
      private int valItalic = Font.PLAIN;

      public void itemStateChanged( ItemEvent event )
      {
         if ( event.getSource() == bold )
            valBold = bold.isSelected() ? Font.BOLD : Font.PLAIN;
         if ( event.getSource() == italic )
            valItalic = italic.isSelected() ? Font.ITALIC : Font.PLAIN;
         field.setFont( new Font( "Serif", valBold + valItalic, 14 ) );
      }
   }
}
```

RadioButton (1)

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;

public class RadioButtonTest extends JFrame {
   private JTextField field;
   private Font plainFont, boldFont, italicFont, boldItalicFont;
   private JRadioButton plainButton, boldButton, italicButton,
      boldItalicButton;
   private ButtonGroup radioGroup;
   public RadioButtonTest()
   {
      super( "Prueba de RadioButton" );
      Container container = getContentPane();
      container.setLayout( new FlowLayout() );

      field = new JTextField( "Oberva el cambio del estilo", 25 );      container.add( field );

      plainButton = new JRadioButton( "Plain", true );      container.add( plainButton );

      boldButton = new JRadioButton( "Bold", false );      container.add( boldButton );

      italicButton = new JRadioButton( "Italic", false );      container.add( italicButton );

      boldItalicButton = new JRadioButton( "Bold/Italic", false );      container.add( boldItalicButton );

      radioGroup = new ButtonGroup();
      radioGroup.add( plainButton );
      radioGroup.add( boldButton );
      radioGroup.add( italicButton );
      radioGroup.add( boldItalicButton );
```

RadioButton (2)

```
plainFont = new Font( "Serif", Font.PLAIN, 14 );
    boldFont = new Font( "Serif", Font.BOLD, 14 );
    italicFont = new Font( "Serif", Font.ITALIC, 14 );
    boldItalicFont = new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
    field.setFont( plainFont );

    plainButton.addItemListener( new RadioButtonHandler( plainFont ) );
    boldButton.addItemListener( new RadioButtonHandler( boldFont ) );
    italicButton.addItemListener(
      new RadioButtonHandler( italicFont ) );
    boldItalicButton.addItemListener(
      new RadioButtonHandler( boldItalicFont ) );      setSize( 300, 100 );      setVisible( true );
  }
  public static void main( String args[] )
  {
    RadioButtonTest application = new RadioButtonTest();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }
  private class RadioButtonHandler implements ItemListener {
    private Font font;
    public RadioButtonHandler( Font f )
    {
      font = f;
    }
    public void itemStateChanged( ItemEvent event )
    {
      field.setFont( font );
    }
  }
}
```

JComboBox (1)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ComboBoxTest extends JFrame {
   private JComboBox imagesComboBox;
   private JLabel label;

   private String names[] =
      { "bug1.gif", "bug2.gif",  "travelbug.gif", "buganim.gif" };
   private Icon icons[] = { new ImageIcon( names[ 0 ] ),
      new ImageIcon( names[ 1 ] ), new ImageIcon( names[ 2 ] ),
      new ImageIcon( names[ 3 ] ) };

   public ComboBoxTest()
   {
      super( "Probando JComboBox" );

      Container container = getContentPane();
      container.setLayout( new FlowLayout() );

      imagesComboBox = new JComboBox( names );
      imagesComboBox.setMaximumRowCount( 3 );
```

JComboBox (2)

```
imagesComboBox.addItemListener(

        new ItemListener() {

          public void itemStateChanged( ItemEvent event )
          {
                                              if ( event.getStateChange() == ItemEvent.SELECTED )
             label.setIcon( icons[
               imagesComboBox.getSelectedIndex() ] );
          }
        }
     );

     container.add( imagesComboBox );

     label = new JLabel( icons[ 0 ] );
     container.add( label );

     setSize( 350, 100 );
     setVisible( true );
   }

   public static void main( String args[] )
   {
     ComboBoxTest application = new ComboBoxTest();
     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }
}
```

JList (1)

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class ListTest extends JFrame {
   private JList colorList;
   private Container container;

   private final String colorNames[] = { "Black", "Blue", "Cyan",
      "Dark Gray", "Gray", "Green", "Light Gray", "Magenta",
      "Orange", "Pink", "Red", "White", "Yellow" };

   private final Color colors[] = { Color.BLACK, Color.BLUE, Color.CYAN,
      Color.DARK_GRAY, Color.GRAY, Color.GREEN, Color.LIGHT_GRAY,
      Color.MAGENTA, Color.ORANGE, Color.PINK, Color.RED, Color.WHITE,
      Color.YELLOW };

   public ListTest()
   {
      super( "Probando List" );

      container = getContentPane();
      container.setLayout( new FlowLayout() );
```

JList (2)

```
colorList = new JList( colorNames );
    colorList.setVisibleRowCount( 5 );

    colorList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );

    container.add( new JScrollPane( colorList ) );
    colorList.addListSelectionListener(

       new ListSelectionListener() {
         public void valueChanged( ListSelectionEvent event )
         {
           container.setBackground(
             colors[ colorList.getSelectedIndex() ] );
         }
       }
    );
    setSize( 350, 150 );
    setVisible( true );
  }

  public static void main( String args[] )
  {
    ListTest application = new ListTest();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

}
```

JMenu (1)

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class Menus extends JFrame
{
        private JLabel mostrarEtiqueta;
        public Menus()
        {
        super( "Ejemplo de Menus" );JMenu Menu = new JMenu( "Opciones" );    Menu.setMnemonic( 'o' );
        JMenuItem acercaItem = new JMenuItem( "Acerca de..." );
        acercaItem.setMnemonic( 'd' );                  Menu.add( acercaItem );        acercaItem.addActionListener
        (
                                        new ActionListener()
                                        {
                                                        public void actionPerformed( ActionEvent event )
                                                        {
JOptionPane.showMessageDialog( Menus.this, "Aqui va el evento del Menu < Acerca de...>", "Acerca de",
        JOptionPane.PLAIN_MESSAGE );
                                                        }
                                        }
                        );
                        JMenuItem salirItem = new JMenuItem( "Salir" );
                        salirItem.setMnemonic( 's' );  Menu.add( salirItem );          salirItem.addActionListener
                        (
                                        new ActionListener()
                                        {
                                                        public void actionPerformed( ActionEvent event )
                                                        {
                                                                        System.exit( 0 );
                                                        }
                                        }
                        );
```

JMenu (2)

```
        JMenuBar barra = new JMenuBar();setJMenuBar( barra );barra.add( Menu );

        JMenu Aplicaciones = new JMenu( "Aplicaciones" );
        Aplicaciones.setMnemonic( 'a' );

        JMenuItem examenItem = new JMenuItem( "Examen" );
examenItem.setMnemonic( 'e' );Aplicaciones.add( examenItem );examenItem.addActionListener
(
                        new ActionListener()
                        {
                                public void actionPerformed( ActionEvent event )
                                {
JOptionPane.showMessageDialog( Menus.this, "Aqui va el evento del Menu < Examen >", "Examen",
JOptionPane.PLAIN_MESSAGE );
                                }
                        }
        );

barra.add( Aplicaciones );

mostrarEtiqueta = new JLabel( "Hola", SwingConstants.CENTER );mostrarEtiqueta.setForeground( Color.black );
mostrarEtiqueta.setFont( new Font( "Serif", Font.PLAIN, 72 ) );getContentPane().setBackground( Color.CYAN );
getContentPane().add( mostrarEtiqueta, BorderLayout.CENTER );setSize( 500, 200 );setVisible( true );
}

public static void main( String args[] )
{
        Menus aplicacion = new Menus();aplicacion.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}
}
```

JTabbedPane (1)

```
import java.awt.*;
import javax.swing.*;

public class JTabbedPaneDemo extends JFrame  {

   public JTabbedPaneDemo()
   {
      super( "Demostración de JTabbedPane" );

      JTabbedPane tabbedPane = new JTabbedPane();

      JLabel label1 = new JLabel( "Panel uno", SwingConstants.CENTER );
      JPanel panel1 = new JPanel();
      panel1.add( label1 );
      tabbedPane.addTab( "Tab uno", null, panel1, "Primer Panel" );

      JLabel label2 = new JLabel( "Panel dos", SwingConstants.CENTER );
      JPanel panel2 = new JPanel();
      panel2.setBackground( Color.YELLOW );
      panel2.add( label2 );
      tabbedPane.addTab( "Tab dos", null, panel2, "Segundo Panel" );
```

JTabbedPane (2)

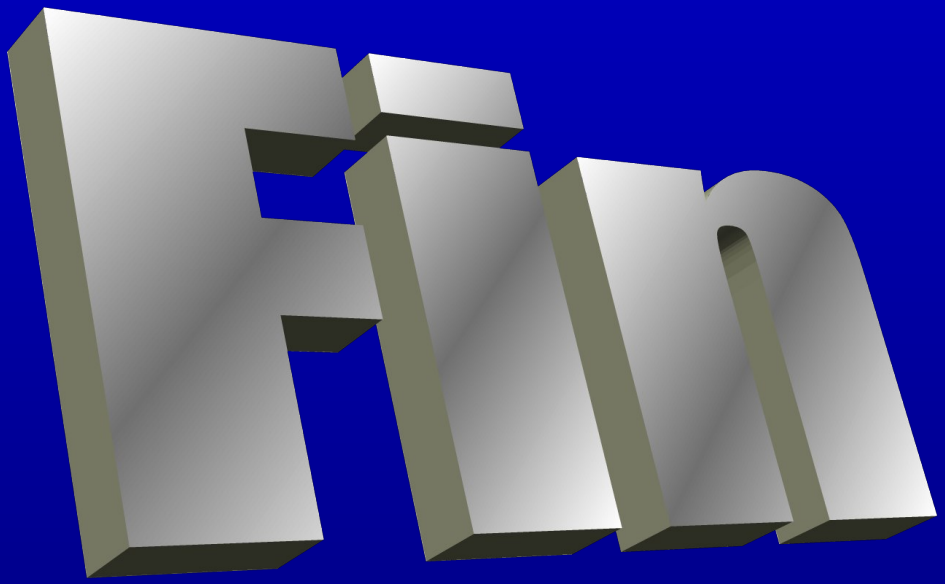
```
JLabel label3 = new JLabel( "Panel tres" );
    JPanel panel3 = new JPanel();
    panel3.setLayout( new BorderLayout() );
    panel3.add( new JButton( "North" ), BorderLayout.NORTH );
    panel3.add( new JButton( "West" ), BorderLayout.WEST );
    panel3.add( new JButton( "East" ), BorderLayout.EAST );
    panel3.add( new JButton( "South" ), BorderLayout.SOUTH );
    panel3.add( label3, BorderLayout.CENTER );
    tabbedPane.addTab( "Tab tres", null, panel3, "Tercer Panel" );

    getContentPane().add( tabbedPane );

    setSize( 250, 200 );
    setVisible( true );
  }

  public static void main( String args[] )
  {
    JTabbedPaneDemo tabbedPaneDemo = new JTabbedPaneDemo();
    tabbedPaneDemo.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }
}
```

Leer Archivos

```
import java.io.*;

public class ArchivosLeer {

        public static void main(String args[]) throws IOException{

            //Creación del flujo para leer datos
            InputStreamReader lect = new InputStreamReader(System.in);

            //Creación del filtro para optimizar la lectura de datos
            BufferedReader br=new BufferedReader(lect);

            //Lectura de datos mediante el método readLine()
            System.out.println("Que archivo quieres leer?");
            String nom = br.readLine();

            File fichero = new File (nom);
            FileInputStream canalEntrada = new FileInputStream(fichero);
            byte bt[] = new byte[(int)fichero.length()];
            int numBytes = canalEntrada.read(bt);
            String cadena = new String(bt);
            System.out.println("Archivo leido\n" + cadena);
            canalEntrada.close();
            }
}
```

Escribir Archivos

```
import java.io.*;

public class ArchivosEscribir {

        public static void main(String args[]) throws IOException{
           System.out.println("Escribe lo que quieras grabar en el archivo:");

           //Creación del flujo para leer datos
           InputStreamReader lect = new InputStreamReader(System.in);

           //Creación del filtro para optimizar la lectura de datos
           BufferedReader br=new BufferedReader(lect);

           //Lectura de datos mediante el método readLine()
            String txt = br.readLine();
            byte b[]=txt.getBytes();
            System.out.println("En que archivo lo quieres grabar?");
            String nom=br.readLine();
            File fichero = new File (nom);
            FileOutputStream canalSalida = new FileOutputStream(fichero);
            canalSalida.write(b);
            canalSalida.close();
        }
}
```



Unidad III – Lenguaje Java