Maestría en Desarrollo de Software

Unidad 1

Semana 3

M.C.C. Uriel Edgardo Escobar Franco

uriel.escobar@upt.edu.mx

Ventajas de la Orientación a Objetos

- Suministra modelos similares a los del mundo real.
- Facilita el desarrollo de sistemas complejos.
- Facilita la reutilización.
- Permite el desarrollo iterativo de aplicaciones.
- Facilita la interoperabilidad de aplicaciones.

Facilita el desarrollo de Sistemas Complejos

- Elementos fundamentales del modelo de Objetos:
  - Abstracción.
  - Encapsulamiento.
  - Modularidad.
  - Herencia.

Facilita la reutilización

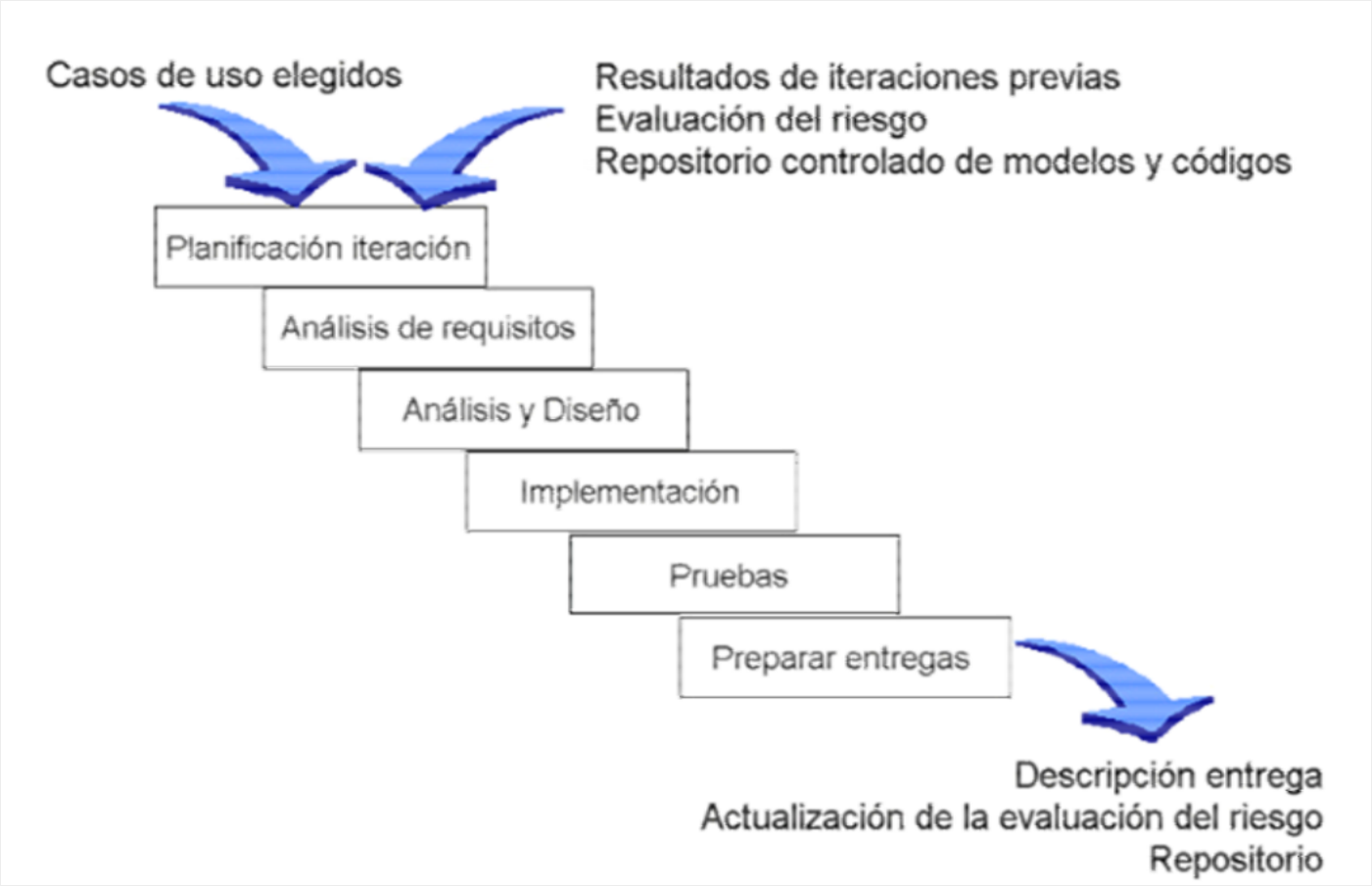
- La O.O. soporta la reutilización basada en la herencia, composición y parametrización.

- La O.O. soporta la reutilización basada en la utilización de librerias de componentes, patrones de diseño y arquitecturas (también conocidas con el nombre de framework).

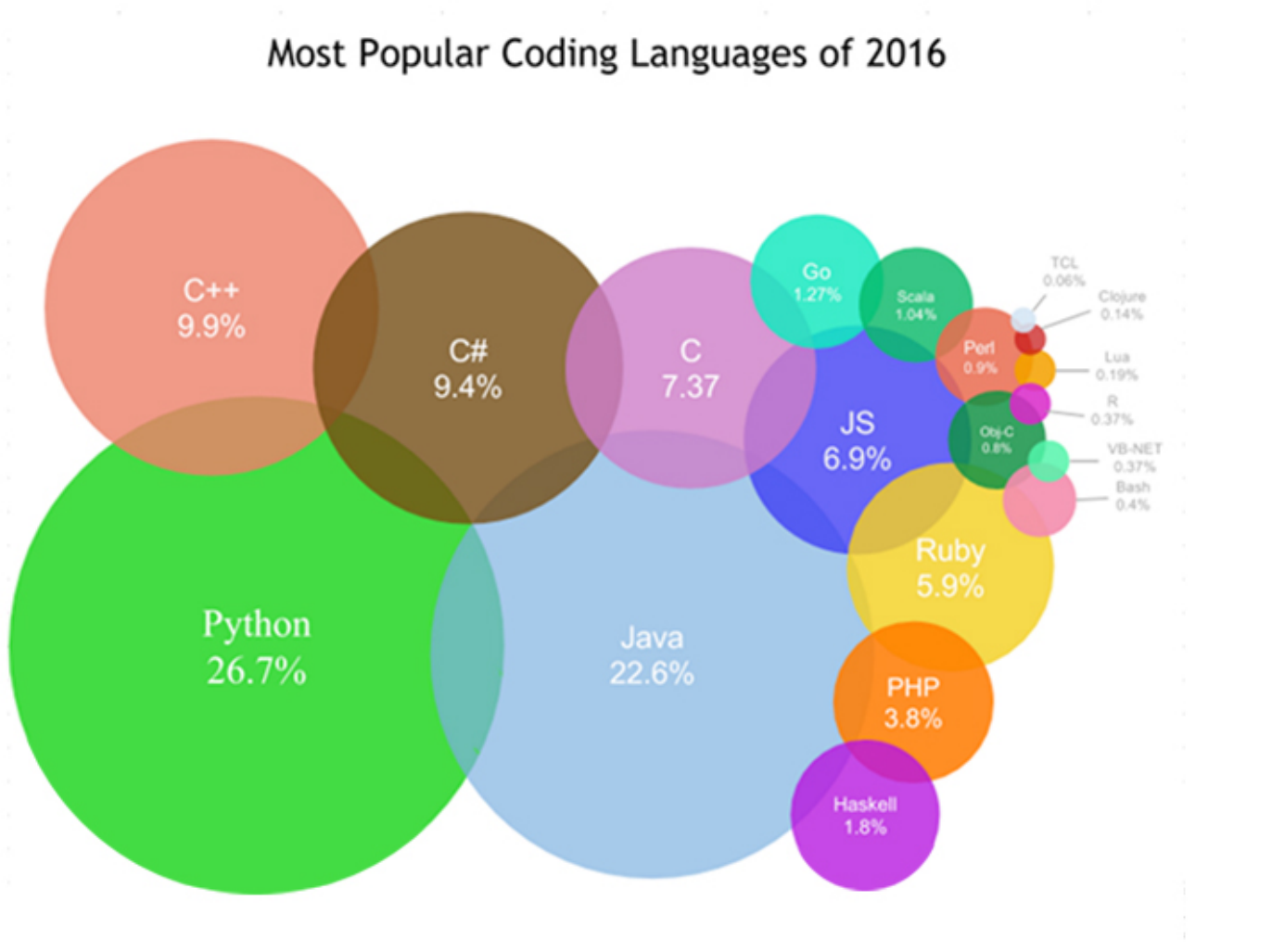
Permite el desarrollo iterativo

- De esta forma se consigue un prototipado controlado: se crea un prototipo al cual se le añaden capacidades de forma incremental.
- El cliente puede ir probando versiones mucho antes que en el desarrollo tradicional.
- Actualmente se basa en la utilización de ‘Casos de Uso’.

Ciclo de una iteración



El lenguaje de Programación Java



https://thelocalbrand.com/why-should-you-learn-java/

Java

- Creado por Sun Microsystems.
- Inicialmente orientado a la programación de microsistemas (proyecto OAK).
- Difundido en 1995 con una nueva orientación: Internet.
- Sintaxis muy similar a la de C++.

Características del lenguaje

- Sencillo
- Orientado a Objetos
- Distribuido
- Interpretado
- Robusto
- Seguro
- Arquitectura Neutra
- Portable
- Altas prestaciones
- Multithread
- Dinámico

Sencillo

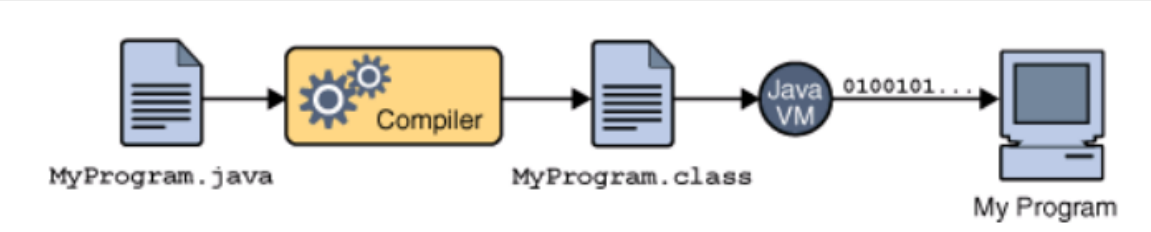
- Los creadores de Java se basaron en C++, pero eliminaron la mayoría de sus complejidades.
  - No soporta tipos de datos: struct, union, y puntero.
  - No soporta typedef ni #define.
  - No permite la sobrecarga de operadores.
  - No soporta la herencia múltiple.
  - Posee una clase String, en vez del array de tipo char[] finalizado con nulo.
  - Cuenta con un sistema automático para asignar y liberar memoria: el Garbage Collector.

Distribuido

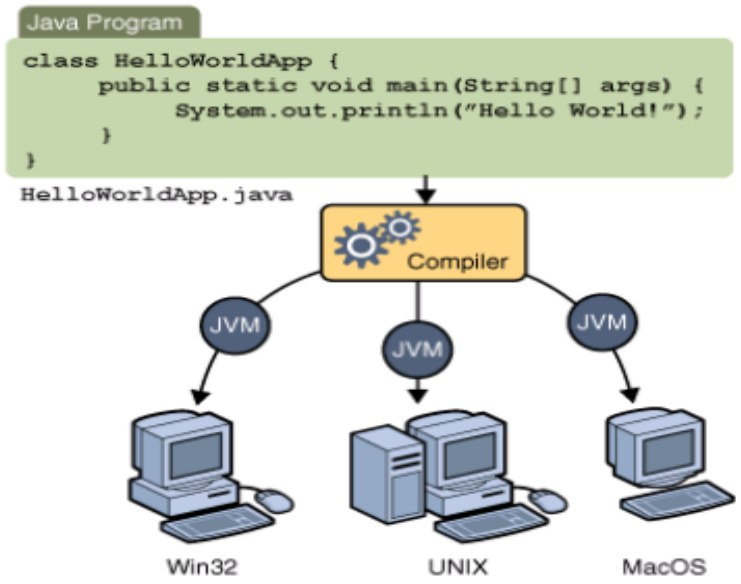
- Está concebido para trabajar en un entorno conectado en red.
- Cuenta con una amplia biblioteca de clases
- para comunicarse mediante TCP/IP: HTTP, FTP...
- Permite manipular con gran facilidad recursos vía URL.

Interpretado

- El compilador de Java traduce el código fuente a
- un código intermedio (bytecode).
- Los byetcodes son interpretados (ejecutados) en cualquier entorno donde exista un intérprete de Java.
- El intérprete de Java se llama Máquina Virtual Java o Java Virtual Machine (JVM).

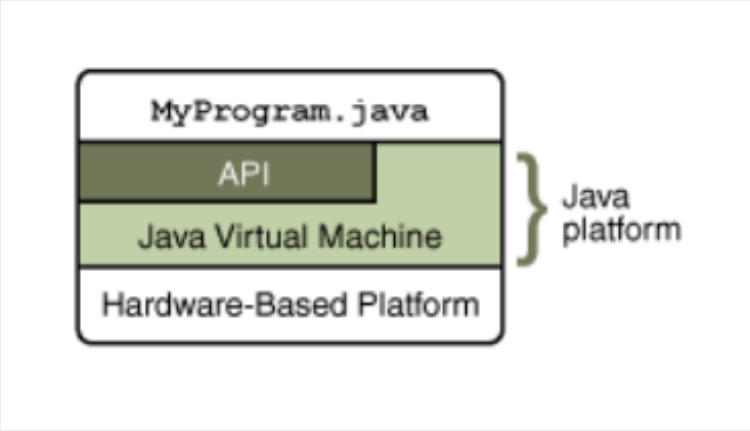


Write once, run everywhere!!!



La plataforma de Java

- La JVM es el intérprete Java.
- El API Java es un conjunto de clases ya desarrolladas que ofrecen un gran abanico de posibilidades al programador. (JEE, JSE,...)



Sintaxis

Comentarios

- Existen tres formas distintas de escribir los comentarios:
  - // Comentario en una línea.
  - /* Comentario de una o
    más líneas */
  - /** Comentario de documentación,
    utilizado por la herramienta javadoc.exe */

Puntos y comas, espacios en blanco y bloques

- Una sentencia es una línea simple de código terminada en un punto y coma:

```
System.out.println(“Hola”);
```

- Un bloque es un conjunto de sentencias agrupadas entre llaves ({ }):

```
while(true)
{
  x = y + 1;
  x = x + 1;
}
```

Puntos y comas, espacios en blanco y bloques

- Los bloques pueden estar anidados.

```
while(true)
{
  x = y + 1;
  if(x<0)
  {
    x = x + 1;
  }
}
```

- Java permite los espacios en blanco entre elementos de código fuente.

Identificadores

- Son los nombres unívocos que se le dan a las clases, métodos y variables.
- Hay que tener presente las siguientes reglas:
  - Deben empezar por una letra, subrayado (_) o dólar ($).
  - Después del primer carácter pueden usar números.
  - Distinguen las mayúsculas y minúsculas.
  - Nunca pueden coincidir con una ‘keyword’.

Keyword

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

★ Tipos de datos primitivos

Variables

- Una variable es un contenedor de datos identificado mediante un nombre (identificador).
- Dicho identificador se utilizará para referenciar el dato que contiene.
- Toda variable debe llevar asociado un tipo que describe el tipo de dato que guarda.
- Por tanto, una variable tiene:
  - Un tipo.
  - Un identificador.
  - Un dato (o valor).

Tipos de dato

- En Java existen dos tipos de datos genéricos:
  - Tipos primitivos.
  - Tipos complejos: clases.
- Existen ocho tipos de datos primitivos clasificados en cuatro grupos diferentes:
  - Lógico: boolean.
  - Carácter: char.
  - Números enteros: byte, short, int y long.
  - Números reales: double y float.

Variables tipo de dato primitivas

```
boolean switch1 = true;
char letra1 = ‘a’;
char letra2 = ‘\n’;
byte unByte = 12;
short unShort;
int unInt = -199;
int otroInt = 065;
long unLong = 2; (o long unLong = 2L;)
long otroLong = 0xABCD;
float unFloat = 0.17f;
double unDouble;
double otroDouble = -12.01E30;
```

Variables tipo de dato complejo (referenciables)

```
String unString = new String(“Hola”);
String otroString;
```

Su valor por defecto es null

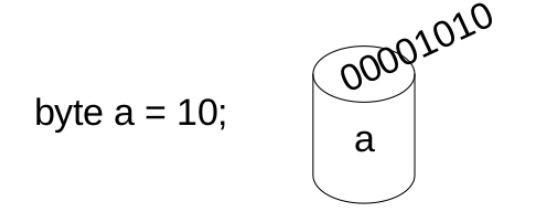
Tipo de dato enumeración

- La ‘keyword’ es: enum.
- Se trata de un tipo de dato complejo algo especial.
- Implementa una clase que tiene un atributo que puede tomar varios valores y solo esos.
- Ejemplo:

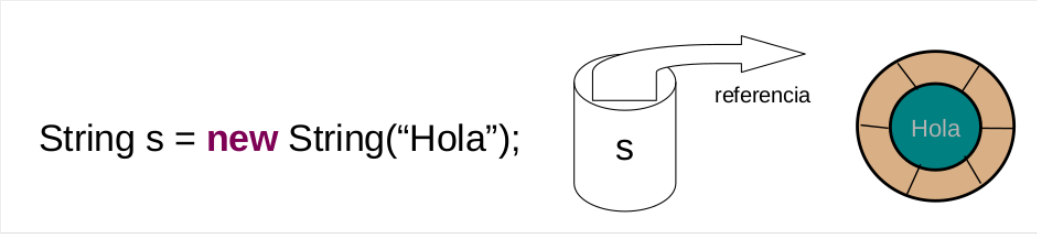
```
enum Semaforo { VERDE, AMBAR, ROJO }
```

Variables primitivas vs. referenciables

- Una variable de tipo primitivo contiene el dato directamente:

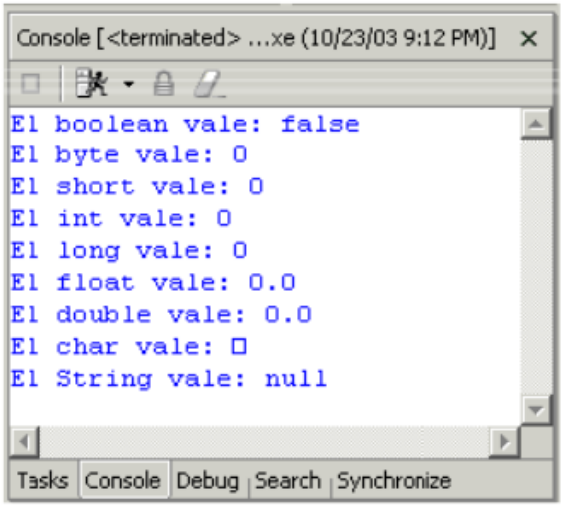


- Una variable de tipo complejo contiene una referencia (puntero) a la zona de memoria donde está el objeto:



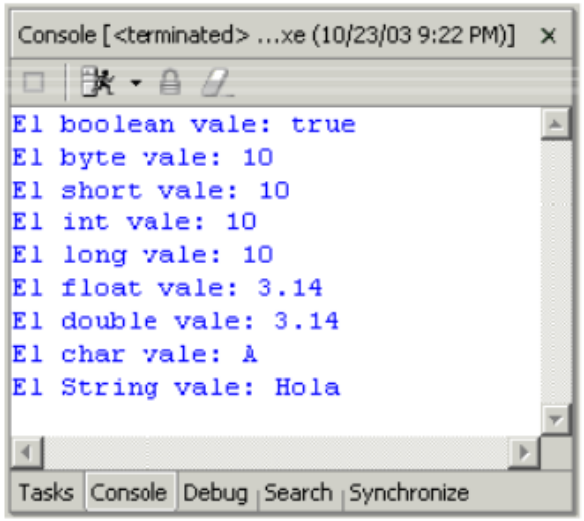
Ejemplo

```
public class VariablesTest1
{
   static boolean unBoolean;
   static byte    unByte;
   static short   unShort;
   static int     unInt;
   static long    unLong;
   static float   unFloat;
   static double  unDouble;
   static char    unChar;
   static String  unString;

   public static void main(String[] args)
   {
      System.out.println("El boolean vale: " + unBoolean);
      System.out.println("El byte vale: " + unByte);
      System.out.println("El short vale: " + unShort);
      System.out.println("El int vale: " + unInt);
      System.out.println("El long vale: " + unLong);
      System.out.println("El float vale: " + unFloat);
      System.out.println("El double vale: " + unDouble);
      System.out.println("El char vale: " + unChar);
      System.out.println("El String vale: " + unString);
   }
}
```



Ejemplo

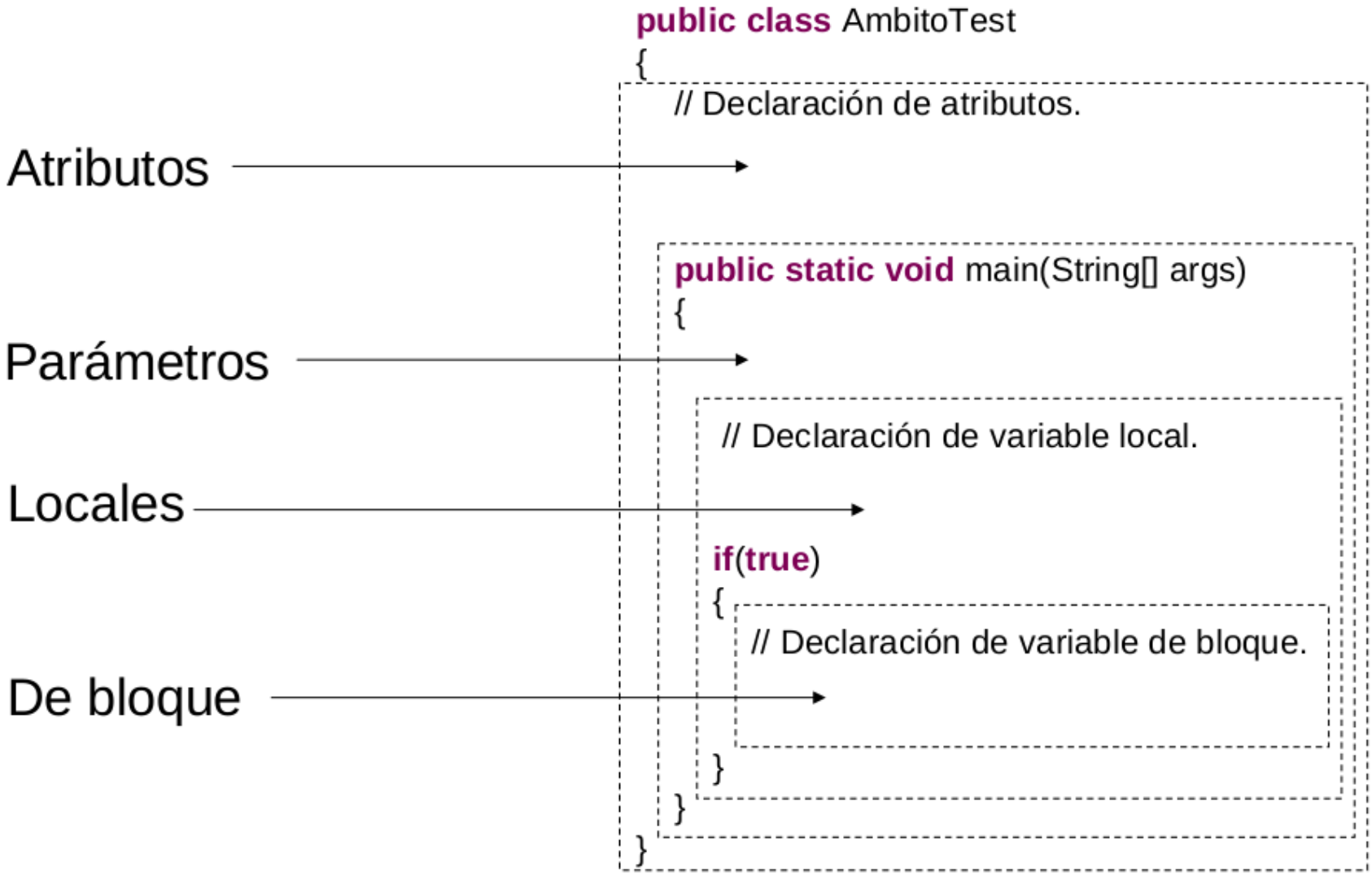
```
public class VariablesTest2
{
   public static void main(String[] args)
   {
      boolean unBoolean = true;
      byte    unByte    = 10;
      short   unShort   = 10;
      int     unInt     = 10;
      long    unLong    = 10;
      float   unFloat   = 3.14F;
      double  unDouble  = 3.14;
      char    unChar    = 'A';
      String  unString  = new String("Hola");

      System.out.println("El boolean vale: " + unBoolean);
      System.out.println("El byte vale: " + unByte);
      System.out.println("El short vale: " + unShort);
      System.out.println("El int vale: " + unInt);
      System.out.println("El long vale: " + unLong);
      System.out.println("El float vale: " + unFloat);
      System.out.println("El double vale: " + unDouble);
      System.out.println("El char vale: " + unChar);
      System.out.println("El String vale: " + unString);
   }
}
```



Ámbito de las variables

- El ámbito de una variable es la zona de código donde se puede referenciar dicha variable a través de su identificador.
- El lugar de definición de una variable establece su ámbito.
- Ámbitos:
  - Atributos (o variables miembro). Si no se inicializan, toman valores por defecto.
  - Parámetros de método.
  - Variables locales: siempre hay que inicializarlas.
  - Variables de bloque: siempre hay que inicializarlas.

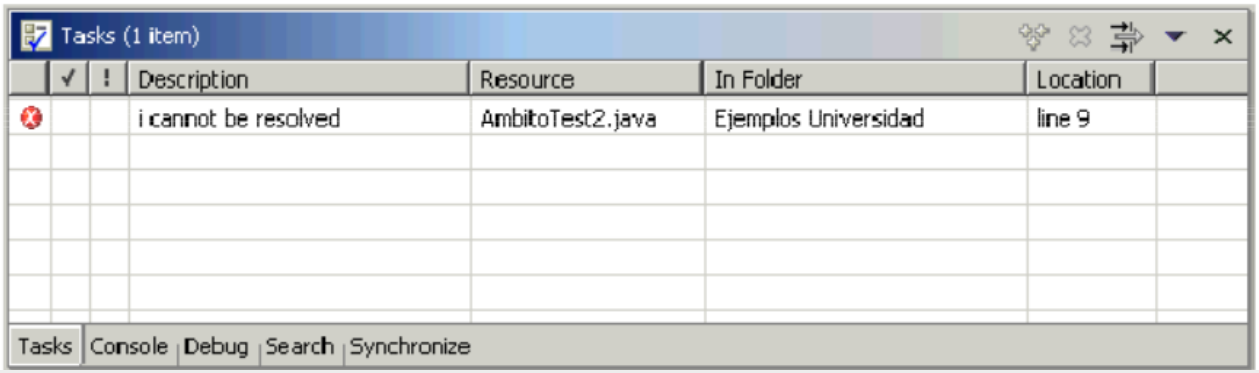
Ámbito de las variables



Ámbito de las variables

Ejemplo

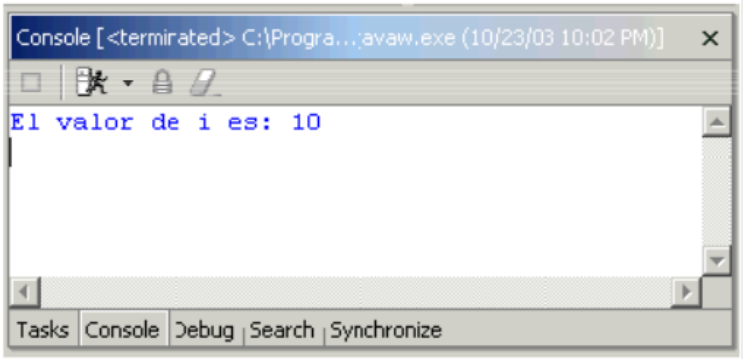
```
public class AmbitoTest2
{
   public static void main(String[] args)
   {
      if(true)
      {
         int i = 12;
      }
      System.out.println("El valor de i es: " + i);
   }
}
```



Ámbito de las variables

Ejemplo

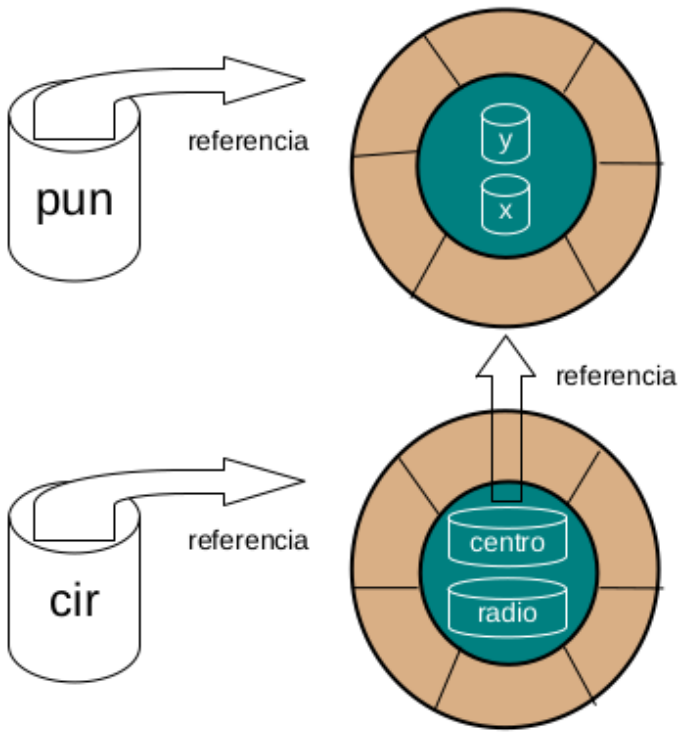
```
public class AmbitoTest3
{
   static int i = 5;

   public static void main(String[] args)
   {
      int i = 10;
      System.out.println("El valor de i es: " + i);
   }
}
```



Variables referenciables

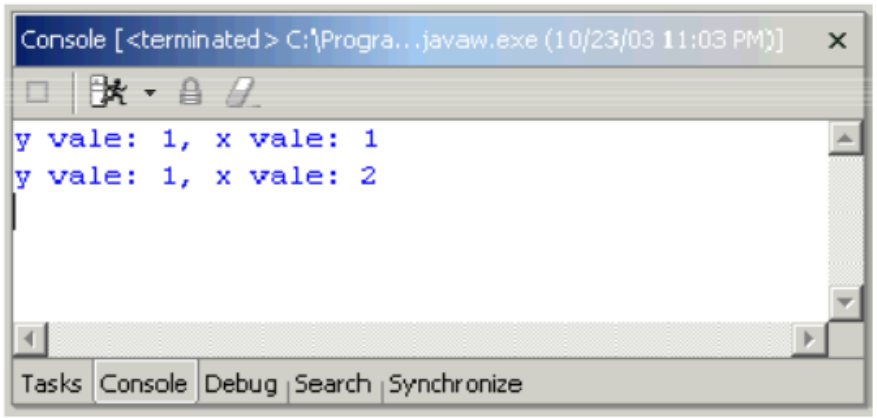
Ejemplo

```
public class Punto
{
   private int x = 0;
   private int y = 0;

   public Punto(int param1, int param2)
   {
      x = param1;
      y = param2;
   }
}
public class Circulo
{
   private Punto centro = null;
   private int radio = 0;

   public Circulo(Punto param1, int param2)
   {
      centro = param1;
      radio = param2;
   }
}
public class Test
{
   public static void main(String[] args)
   {
      Punto pun = new Punto(2,2);
      Circulo cir = new Circulo(pun,3);
   }
}
```



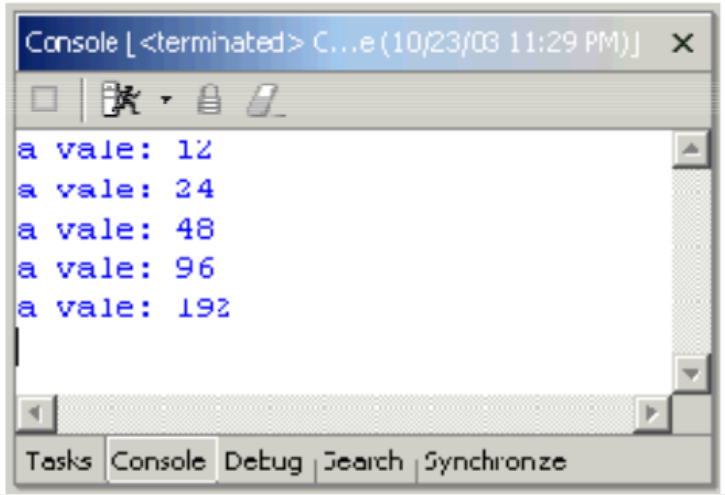
Ejemplo

```
public class OperadoresUnariosTest
{
   public static void main(String[] args)
   {
      int x = 0;
      int y = 0;
      y = ++x;
      System.out.println("y vale: " + y + ", x vale: " + x);
      y = x++;
      System.out.println("y vale: " + y + ", x vale: " + x);
    }
}
```



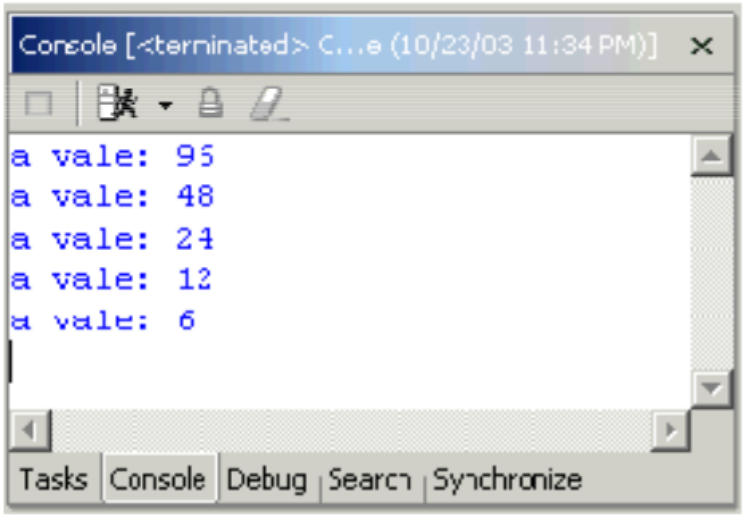
Ejemplo

```
public class Multiplicador
{
    public static void main(String[] args)
    {
        int a = 6;
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
    }
}
```



Ejemplo

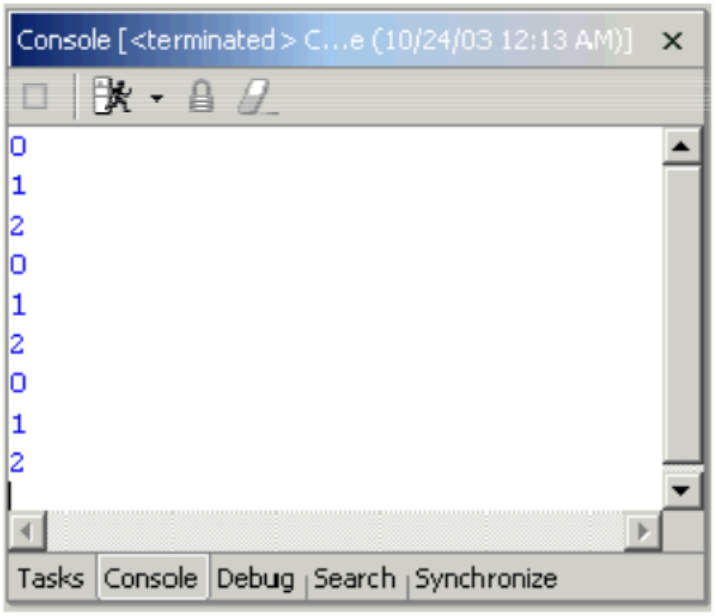
```
public class Divididor
{
   public static void main(String[] args)
   {
      int a = 192;
      a = a >> 1;
      System.out.println("a vale: " + a);
      a = a >> 1;
      System.out.println("a vale: " + a);
      a = a >> 1;
      System.out.println("a vale: " + a);
      a = a >> 1;
      System.out.println("a vale: " + a);
      a = a >> 1;
      System.out.println("a vale: " + a);
   }
}
```



Bucles

Ejemplo

```
public class Bucles
{
    public static void main(String[] args)
    {
        int cont1 = 0;
        while(cont1 < 3)
        {
            System.out.println(cont1);
            cont1++;
        }

        int cont2 = 0;
        do
        {
            System.out.println(cont2);
            cont2++;
        }
        while(cont2 < 3);

        for(int cont3 = 0; cont3 < 3; cont3++)
        {
            System.out.println(cont3);
        }
    }
}
```



Ejercicio 1

- Identifica qué sentencias son correctas y cuales no

```
1.  int x = 34.5;
2.  boolean boo = x;
3.  int g = 17;
4.  int y = g;
5.  y = y + 10;
6.  short s;
7.  s = y;
8.  byte b = 3;
9.  byte v = b;
10. short n = 12;
11. v = n;
12. byte k = 128;
13. int p = 3 * g + y;
```

Ejercicio 2

- Identificar si este código compila bien. Si no compila solucionarlo. Si compila decir cuál sería la salida.

```
public class Temp
{
    public static void main(String[] args)
    {
        int x = 1;
        while(x<10)
        {
            if(x>3)
            {
                System.out.println("Hola");
            }
        }
    }
}
```

Ejercicio 3

- Identificar si este código compila bien. Si no compila solucionarlo. Si compila decir cuál sería la salida.

```
public class Temp
{
    public static void main(String[] args)
    {
        int x = 5;
        while(x>1)
        {
            x = x - 1;
            if(x<3)
            {
                System.out.println("Hola");
            }
        }
    }
}
```

Ejercicio 4 (parte 1)

- Al siguiente programa Java le falta un trozo de código.

```
public class Temp
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 0;
        while(x<5)
        {
            ????
            System.out.print(x + " " + y + " ");
            x = x + 1;
        }
    }
}
```

Ejercicio 4 (parte 2)

- Seleccionar para cada trozo de código de la izquierda, la salida por pantalla al ejecutar el programa anterior con ese trozo de código.