# UT3. Identificación de elementos de un programa informático

**“Programación”**

*Ciclo DAW- IES Clara del Rey*

Curso 2011 – 2012

# Índice

- Elementos de una POO vs Programación estruturada
- ¿Qué es Java?
  - JDK, JRE, Máquina Virtual
- Elementos de una clase
- Java
  - Comentarios del código
  - Palabras reservadas
  - Delimitadores
  - Tipos primitivos y sus literales
  - Definición de variables
  - Operadores y reglas de construcción de expresiones
  - Definición de constantes nominales

# Elementos de la POO vs Estructurada

| **POO** | **Estructurada** |
|---|---|
| *Clase* | *TAD (registro+subprogramas)*<br>*(Tipo de dato abstracto)* |
| *Objeto* | *Valor del TAD (registro concreto)* |
| *Método* | *Subprograma* |
| *Mensaje* | *Llamada a subprograma* |
| *Atributo* | *Campo de registro* |
| *Estado* | *Conjunto de valores de los campos de un registro concreto* |

# ¿Qué es Java?

- Un lenguaje Orientado a Objetos
- Es fuertemente tipado (buscar)
- Una plataforma
  - Lenguaje: un lenguaje de programación
  - *JVM (Java Virtual Machine):* software que independiza el programa Java del hardware
  - Biblioteca estándar (*API, Application Programming Interface*): interfaz para la programación de aplicaciones

- Conceptos
  - JDK
  - JRE
- Ejercicio
  - Instalar jdk
  - Programa “Hola Mundo”
  - Compilar un programa desde la línea de comandos
- Nota: En el material anexo hay que revisar la versión de java a descargar
- Era de SUN, ahora Oracle

# Plataforma Java

- Al compilar no se genera código nativo
  - Se genera código bytecode (código entendible por la máquina virtual)

# Plataforma Java

- Ediciones de la plataforma Java
  - *J2SE (Java2 Platform, Standard Edition)*
  - *J2EE (Java 2 Platform, Enterprise Edition)*
    - Define un estándar para el desarrollo de aplicaciones distribuidas multicapa, basada en componentes
  - *J2ME (Java 2 Platform, Micro Edition)*
    - Entorno para desarrollar aplicaciones en dispositivos con recursos limitados como dispositivos móviles, software empotrado en electrodomésticos, etc.
- Distribuciones de la plataforma:
  - *J2SE Runtime Environment* (JRE)
    - Proporciona APIs, la máquina virtual y otros componentes necesarios para ejecutar aplicaciones escritas en Java
    - No contiene compiladores o debuggers
  - *J2SE Development Kit* (JDK)
    - Incluye al JRE y, además, tiene herramientas como compiladores y debuggers para desarrollar aplicaciones

# Estructura básica de un programa java

- Esquema básico de un programa
  - nos acostumbraremos escribir cada clase en un fichero con el nombre de la clase y extensión java (nombreClase.java)
  - Cada clase una vez compilada generará un fichero nombreClase.class
  - Para que una clase se pueda ejectutar debe tener un método main
  - En cada clase se incorporán los atributos y métodos correspondientes
  - Ej. Lo más básico posible, sin atributos y sólo con el método main

```
class Programa {
   // Atributos
    public static void main (String args[]) {
            // sentencias
    }
```

# Java: Comentarios

- Comentarios del código
  - Una o varias líneas

    /* una o varias líneas de texto */

    Ejemplo: /* Esto es un comentario
    de varias líneas */
  - De línea

    // texto hasta el final de la línea

    Ejemplo: /*Esto es un comentario de una línea*/
    // Esto también es un comentario de una línea
- Comentarios especiales de Javadoc
  - Especial Javadoc comentarios se utilizan para la generación de una documentación HTML para sus programas de Java. Puede crear javadoc comentarios empezando por la línea con /** y termina con */.
  - 
```
/**   This is an example of special java doc
                comments used for \n  generating an
html
                documentation. It uses tags like:
                @author Florence Balagtas
                @version 1.2
*/
```

# Java: Palabras reservadas

- Palabras reservadas :no se pueden utilizar como nombres en sus programas Java en variables, clases o métodos. true, false, y null no son palabras clave, pero tienen el significado de palabras resevadas y tampoco se pueden utilizar como nombres en sus programas. Ejemplo:

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

# Java: Delimitadores

- Delimitadores
  - Delimitador de instrucción

    “.”

    ;

  - Delimitadores de bloques de instrucciones

    “{” y “}”

```
sentencia;
```

```
{
   sentencia1;
   sentencia2;
}
```

# Java: Tipos de datos

- Tipos primitivos y sus literales

  Los tipos primitivos son aquellos que permiten representar los valores básicos que se utilizan en un programa. Los valores de los tipos primitivos no son objetos.

  Clasificación (de tipos primitivos):

  - **Enteros:** (byte, shot, int, long) representan números enteros positivos y negativos con distintos rangos de valores.
  - **Reales** (float y double) : representan números reales.
    - Float: dato en coma flotante de 32 bits en el formato IEEE 754 (1 bit para el signo, 8 bits para el exponente y 23 para la mantisa)
    - Double: dato en coma flotante de 64 bits en el formato IEEE 754 (1 bit para el signo, 11 bits para el exponente y 52 para la mantisa
  - **Booleano** (boolean) : representa un valor lógico.
  - **Caracteres** (char): para representar cualquier carácter individual. En concreto en Java se trata de caracteres Unicode.

- Nota:
  - Más adelante se verá que la biblioteca Java también proporciona las clases Byte, Character, Short, Integer, Long, Float, Double y Boolean para encapsular cada uo de los tipos expuestos proporcionando así mayor funcionalidad para manipularlos

# Java: Tipo de datos Básicos

| **Tipo** | **Descripción** | **Valor min./máx.** |
|---|---|---|
| byte | Entero con signo | -128 a 127 |
| short | Entero con signo | -32768 a 32767 |
| int | Entero con signo | -2147483648 a 2147483647 |
| long | Entero con signo | -922117036854775808 a 922117036854775807 |
| float | Real de simple precisión | +-3.40282347e+38 a +-1.40239846e-45 |
| double | Real de doble precisión | +-1.79769313486231570e+308 a +-4.94065645841246544e-324 |
| char | Caracteres Unicode | \u0000 a \uFFFF |
| boolean | Verdadero o falso | true o false |

# Java: Tipos de datos . Enum

- Un tipo enumerado define un conjunto de valores.
- Es en sí una clase que se verá más adelante
- Ej.

```
publicstatic void main (String[] args){
        enum dia_semana {lunes, martes, miércoles jueves, viernes
          sabado, domingo};
        dia_semana dia = dia_semana.lunes;
        System.out.println (dia);
}
```

# Identificadores

- Los identificadores son nombres dados a tipos, lilterales, variables, cases, interfaces, métodos, paquetes y sentencias de un programa
- La sintaxis para formar un identificador es la siguiente:
  - {letra|_| $}[ { $letra | dígito | _ | $ } ] …
    - lo que significa que un identificador consta de uno o más caracteres y el el primer caárcter debe ser una letra, el subrayado o el carácter $.
    - No se puden comenzar por un dígito ni pueden contener caracteres especiales
    - Las letras pueden ser mayúsculas o minúsculas
    - Los identificadores pueden tener cualquier número de caracteres
- Recordar nomenclatura aconsejada

# Java: Delaración de una variable

## Definición de variables

**<tipo> <nombreVariable> [ “=” <expresion>] “;”**

¡CUIDADO! Java es *Case-sensitive* (sensible a mayúsculas y minúsculas)

<nombreVariable>: identificador que comienza con un carácter, “_” o “$”, seguido de 0 o más letras, dígitos, subrayados y “$”. No puede ser el identificador de una palabra reservada. No puede ser igual al nombre de otra variable dentro del mismo ámbito

# Java: Declaración de variables

## Definición de variables

### Ejemplos:

```
int contador;
double resultado;
boolean encontrado = false;
char simboloNota;
```

"Regla de estilo": el nombre de una variable debe ser en minúsculas salvo cuando se produce un cambio de palabra, donde la primera letra de la nueva palabra será en mayúscula.

Observación: se pueden declarar variables en cualquier punto del código siempre que se vayan a utilizar después de declaradas. Además, requieren inicialización antes de ser consultadas.

Las cadenas de caracteres son almacenadas en objetos de la clase String. P.ej:

```
String cadena;
cadena = "hola";
```

# Java: Asignación de valores a variables

- Operadores
  - De Asignación
    - Para asignar un valor a otro. Además del operador de asignación básico, Java proporciona varios operadores de asignación que permiten realizar operaciones aritméticas, lógicas o de bits y una operación de asignación al mismo tiempo
    - **<nombreVariable> “=” <expresión> “;”**

    Ejemplos:

    ```
    int contador = 4;
    int contador2;
    contador2 = 5 + contador;
    ```

# Java: Asignación de  Valores a variables

- Ejemplos
  - byte b =0
  - short i=0, j=0;
  - int ia= 2’’’
  - int ib = -20;
  - int ic = 0xF003;  /*valor en hexadecimal*/
  - log la = -1L; /* L indica que la constate -1 es long*/
  - long lb = 125;
  - long lc = 0x1F00230F; /*valor en hexadecimal*/
  - float a =3.14159F */la F indica que es float*/
  - float b= 2.2e-5F /* 2.2 por 10 elevado a -5*/
  - char c= ‘a’;
  - char c=97;  /*la ‘a’ es en decimal el código 97*/
  - char c=‘\u0061’; /la ‘a’ es en unicode 0061
  - double a =3.14159 /* Por defecto es double */
  - double b = 2.2e+5 /*2.2 por 10 elevado a 5*/

# Mostrar datos por pantalla

- Ejemplo

```
class MostrarDatos {
    public static void main (String[ ] args){
    x1 = 5;
    x2 = 3:
    res = x1+x2;
    System.out.println(“ La suma de ” + x1 + “+”, x2, “ es =“ + res);
    }
}
```

- Se están invocando al método println, del objeto out de la clase System de la biblioteca Java.
  - ver http://download.oracle.com/javase/1.5.0/docs/api/index.html
- Para unir elementos se utiliza el operador +

# Estructura de una clase

```
public class Circulo {
        //Atributos
        private Punto centro=null;
        private double radio=0.0;

        //Constructores
        public Circulo()
        {    centro=new Punto();
             radio=3.1;}

        public double calcularPerimetro()
        {
             return 2*Math.PI*radio;
        }
}
```

DEFINICIÓN DE LA CLASE

DEFINICIÓN DE ATRIBUTOS

CONSTRUCTOR

DEFINICIÓN DE MÉTODOS

* **Para que una clase sea ejecutable debe existir un método llamado main(..)**
* **El método main ( ) puede recibir parámetros**

# Java: Literales

- Literales:

  Un literal es la expresión de un valor de un tipo primitivo, de un tipo String o la expresión **null**
- **Literales enteros**
  - Java permite especificar un literal entero en base 10, 8 , 16
  - En general el signo `es opcional si el valor es positivo y el signo – estará siempre que el valor sea negativo
  - Un literal entero es de tipo int a no ser que su valor absoluto sea mayor que el de un int o se especificque el sufijo l o L en cuyo caso sería long
  - Ej
    - 4326 constante entera in
    - 4326L constante entera long
    - 342500342344 constante entera long
    - O326 constante entera int en base 8
    - 0x15 número decimal 21 expreado en base 16
- **Literales reales**
  - Reales: se escriben con un punto decimal o con una letra (e o E) que indica un exponente
    - Ej. 5., .76, 3.14, 56.34E-45,…
- **Literales booleanos:** únicos valores posibles `true` o `false`.
- **Literales de un solo carácter** :
  - Se escriben entre comillas simples.
  - También se puede utilizar su representación de la tabla Unicode en octal o hexadecimal.
    - Ej. ‘f’, ‘P’, ‘\u00A3’ (hexadecimal), ‘\102’ (octal),…

# Java: Literales

- Literales:

  Caracteres especiales en Java:

| **Carácter** | **Significado** |
|---|---|
| \b | Retroceso |
| \t | Tabulador |
| \n | Ir al principio de la siguiente línea |
| \r | Cambio de línea sin ir al principio |
| \” | Carácter comillas dobles |
| \’ | Carácter comillas simples |
| \\ | Carácter barra hacia atrás |

# Java: Literales

- Literales de cadenas de caracteres:
  - Es una secuencia de caracteres encerrados entre comillas dobles (incluidas las secuencias de escape como \”
  - Ej
    - “Esto es una constante de caracteres”
    - “ lenguaje \”Java\””  /*Produce Lenguaje “Java” */
- Las cadenas de caracteres son objetos de la clase String.
  - Ejercicio: Buscar qué metodos tiene la clase String
- Las cadenas de caracteres se pueden concatenar empleando el operador +.
  - Si alguna de las expresiones no se corresponde con una cadena, Java la convierte de forma automática en una cadena de caracteres
  - ¿Por qué?

# Escribiendo métodos

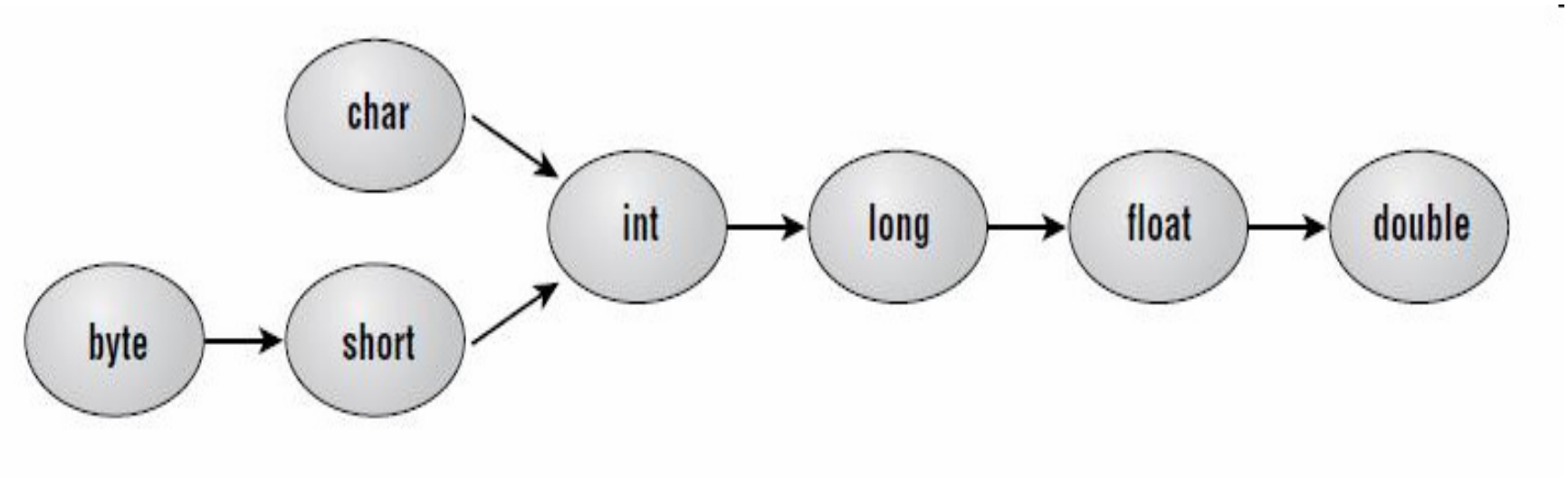
- En una clase existen atributos y métodos
- Al escribir métodos debemos básicamente pensar en:
  - Visibilidad
  - Tipo del valor que devuelve
  - Argumentos que recibe
  - Ej
    - …
    - public static int sumar(int x, int y){
      int resultado;
      resultado = x +y;
      return resultado;
      }
  - De momento no explicaremos el concepto static

# Java: Conversión de tipos

- Cuando java tiene que evaluar una expresión (ej 5+ 3.0), primero convierte los valores de los operandos al tipo de operando cuya precisión sea más alta
  - Ej conversión implícita

    double x = 2.

    double y = x – 30; /* 30 no es un double
- La siguiente figura resume los tipo colocados de menos a más precisos y las flechas indican las conversiones implícitas permitidas



# Java: Conversión de tipos

- No se pueden hacer conversiones entre los tipos enteros o reales y el tipo boolean
- Conversión explícita
  - Java permite una conversión explícita o forzada del tipo de una expresión mediante una construcción denominada **cast**
    - **(tipo) expresión**
    - Ej.

```
double x= 2.0;
int y = (int)( x – 30); /* x es un double y el resultado se convierte en int*/
```

  - Si la conversión no está permitida se obtendrá un error
  - Puede conllevar una périda de precisíon en el resultado
    - Ej.

```
float r
r =(float) Math.sqrt(10);
/* el resultado se redondea perdiendo precisión ya que sqrt devuelve
   double */
```

# Java: Operadores y expresiones

## Operadores y reglas de construcción de expresiones

Operador: símbolo utilizado para desempeñar una función específica en una o más variables

Expresión: combinación de operadores y operandos

(Java tiene una amplia variedad de operadores)

# Java: Operadores Aritméticos

| Operador | Uso | Descripción |
|---|---|---|
| **+** | **op1 + op2** | **suma op1 y op2** |
| **-** | **op1 – op2** | **resta op2 de op1** |
| ***** | **op1 * op2** | **multiplica op1 y op2** |
| / | **op1 / op2** | **divide op1 por op2** |
| **%** | **op1 % op2** | **obtiene el resto de dividir op1 por op2** |

# Ejercicio

- UT3E01: Programa que escriba “hola Mundo”
- UT3E02: Programa que permita sumar y  multiplicar 2 números cuyo valor se inicializa en el programa y muestre el resultado por pantalla:
  - LA suma de x + y es = res
  - El producto de x * y es = res

  (los valores de x,y, res mostrará el valor numérico)
- UT3E03: Programa que haga lo mismo que el UT3E02 pero definiendo los métodos adecuados
- UT3E04: Programa que resuelva ecuaciones de 2º grado (b2-4ac)/2a
  - ¿Has utilizado algún método además del main?

# Java : Operadores de relación

- Ej

  int x=10.0;

  int y = 0;

  boolean r=false

  r = x==y; // r vale false

  r = x>y;  // r vale true

  r = x !=y; // r vale true

- Consejo
  - Utilizar  () aunque no es necesario en este caso ya que la prioridadde ==, > y != es mayor que la de la asignación =

| Operador | Uso | Retorna true si |
|---|---|---|
| > | op1 > op2 | op1 mayor que op2 |
| >= | op1 >= op2 | op1 mayor o igual que op2 |
| < | op1 < op2 | op1 menor que op2 |
| <= | op1 <= op2 | op1 menor o igual que op2 |
| == | op1 == op2 | op1 es igual que op2 |
| != | op1 != op2 | op1 distinto de op2 |

# Java: Operadores lógicos

| Operador | Uso | Retorna true si |
|---|---|---|
| && | op1 && op2 | AND: op1 y op2 son verdaderos |
| \|\| | op1 \|\| op2 | OR: uno de los dos es verdadero |
| ! | !op | NOT: op es falso |
| ^ | op1 ^op2 | XOR: op1 es true y op2 es false o viceversa |

- Ej

```
int p = 10, q=0;
boolean r=false
r = (p!=0) && (q>0);  // r vale false
```

# Java : Operadores Unitarios

- Los poreadores unitarios se aplican a un solo operando y son los siguientes:
  - !: NOT
  - ~ : Complemento 1 (cambiar ceros por unos y unos por ceros a nivel de bit)
  - - : Cambia el signo del operando

# Java: Operadores a nivel de bit

- Permiten realizar con sus operandos las operaciones AND, OR, XOR y desplazamientos bit por bit

| Operador | Uso | Descripción |
|---|---|---|
| & | op1 & op2 | Operación And a nivel de bits |
| \| | op1 \| op2 | Operación Or a nivel de bits |
| ^ | op1 ^ op2 | Operación XOR a nivel de bits |
| << | op1 << op2 | Desplazamiento a la izquierda rellenando con 0's por la derecha |
| >> | op1 >> op2 | Desplazamiento a la derecha rellenando con el bit de signo por la izquierda |
| >>> | Op1 >>> op2 | Desplazamiento a la derecha rellenando con ceros por la izquierda |

# Java: Operadores a nivel de bit

- Ejemplos para comprobar (vale el valor de la línea anterior)

```
int a=255, r = 0, m=32;
r = a & O17;  // r = 15. Pone a cero todos los bits
                      // excepto los 4 de menor peso   (17
                      // está en Octal)
r = r | m;   // r = 47. Pone a 1 todos los bits de r
                    // que están a 1 en m
r = a & ~07;  // r = 248
r = a >> 7;    // r = 1
r = m << 1;   // r = 64
r = m >> 1;   // r = 16
```

# Java: Operadores de asignación

| Operador | Uso | Descripción |
|---|---|---|
| ++ | op1++ ó ++op1 | suma en 1 el valor de op (predecremento o postdecremento) |
| -- | op1- - ó - -op1 | Resta en 1 el valor de op (pre o post) |
| = | op1 = expresión | Asignación simple |
| *=<br>/=<br>%=<br>+=<br>-= | op1*=op2<br>op1 /=op2<br>op1%=op2<br>op1+=op2<br>op1-= op2 | Multiplicación más asignación<br>División más asignación<br>Módulo más asignación<br>Suma más asignación<br>Resta más asignación |
| <<=<br>>>=<br>>>>= | op1<<=op2<br>op1 >>= op2<br>op1 >>>=op2 | Desplazamiento a la izquierda más asingación<br>Desplazamiento a la derecha más asignación<br>Desplazamiento a la derecha más asignación rellenando con 0's |
| &=<br>\|=<br>^= | op1 &=op2<br>op1 \|=op2<br>op1 ^ op2 | And sobre bits más asignación<br>Or sobre bits más asignación<br>XOR sobre bits más asignación |

# Java: Operadores de asignación

- Ejemplos

```
int x=0, n = 10, i =1;
n++;   // incrementa el valor de n en 1
++n;  // incrementa el valor de n en 1
x = ++n; // (PRE) Incrementa en 1 y asigna el resultado a x
x = n++; // (POST) realiza la asignación y luego
                // incrementa el valor de n.
                //Equivale a x=n; n++;
i+=2; //Realiza la operación i=i+2;
x*= n -3; // realiza la operación x = x*(n-3) y no x= x*n – 3
n>>1;  // realiza la operación n= n >> 1 la cual desplaza el
          //contenido de n 1 bit a la derecha
```

- Ejercicio
  - x= (a – b++) * (--c –d) / 2

# Java: Operadores de asignación

- x= (a – b++) * (--c –d)/2
- Escrito en java para comprobar

  ***float x=0, a=15, b=5, c=11, d=4;***

  ***x = (a – b++) * (--c –d)/2;***

  – Resultado: a = -30.0, b=6.0; c= 10.0

  – Es como hacer los predecrementos antes de empezar y los postdecrementos una vez acabados los cálculos

  – Equivale a

  ***float x=0, a=15, b=5, c=11, d=4;***

  ***c= c-1; // o bien - -c; ó c- -;***

  ***x = (a-b) *(c – d) /2;***

  ***b++;***

# Java: Operador condicional

- El operador condicional (?:) llamado también operador ternario se utiliza para expresiones condicionales de la forma siguiente
  - operando1 ? operando 2: operando 3
  - La expresión operando1 debe ser una expresión booleana.
    - Si el resultado de la evaluación de operando1 es true, el resultado de la expresión condicional es operando2
    - Si el resultado de la evaluación de operando1 es false, el resultado de la expresión condicional es operando3
  - Ej
    - double a=10.2, b=20.5, mayor =0
    - mayor = (a>b) ? a: b;

# Java: Precedencia de operadores

- Determinan el orden de evaluación
- Los operadores se evalúan en orden de prioridad
- Si dos operadores tienen la misma prioridad, se evalúan según asociatividad
- Asociatividad izquierda-derecha significa que el operador a la izquierda se evalúa primero
- Siempre se puede utilizar paréntesis para cambiar el orden de evaluación
- Todos los operadores binarios que no son de asignación asocian por la izquierda. Los de asignación asocian por la derecha. Lo que hace que
  - a=b=c; sea equivalente a
  - a=(b=c);

# Java: Precedencia de operadores

| | | | |
|---|---|---|---|
| De mayor a menor prioridad. En la misma línea igual prioridad | `()` | paréntesis | |
| | `- ~ ! ++ -- (tipo)` | sgn/inc/dec/NO/casting | der-izq |
| | `* / %` | mult/div/resto | izq-der |
| | `+ - +` | ad/sust/concatenación | izq-der |
| | `> >= < <=` | mayor/menor | izq-der |
| | `== !=` | igual/distinto | izq-der |
| | `&` | And bit | Izq-der |
| | `^` | Xor bit | Izq-der |
| | `\|` | Or bit | Izq-der |
| | `^` | O exclusiva | izq-der |
| | `&&` | Y lógica | izq-der |
| | `\|\|` | O lógica | izq-der |
| | `?:` | Operador condicional | Der-izq |
| | `= += -= *= /= %=` | asignación | der-izq |

# Java: Variables de referencia

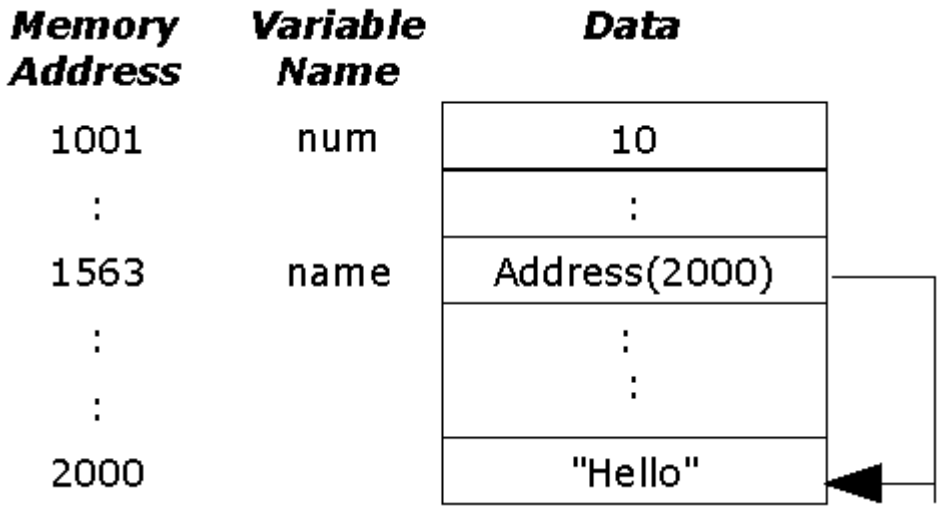
- Dos tipos de variables en Java:
  - Variables primitivas
  - Variables de referencia
- Variables primitivas
  - variables con tipos de datos primitivos como int o largo.
  - almacena los datos en la memoria real de la ubicación donde la variable está
- Variables de referencia
  - las variables que almacena la dirección en la ubicación de la memoria
  - apunta a otra ubicación de memoria donde los datos reales están
  - Cuando declara una variable de una determinada clase, en realidad está declarando una variable de referencia con el objeto de cierta clase.

# Java: Variables de referencia

- Ejemplo
  - Supongamos que tenemos dos variables con tipos de datos int y string.

```
int num = 10; // primitive type
String name = "Hello"; // reference type
```

  - La imagen se muestra a continuación es la memoria de su ordenador, que tiene la dirección de la memoria de las células, el nombre de la variable y los datos que posean.



# Java: Constantes: Final / Static

- Por qué utilizar constantes
  - Porque es más facil modificar un programa. P.ej Valor de PI
- Definición de constantes nominales

  **[public] "final" [static] <tipo> <NOMBRE_CONSTANTE> ["="<expresion>] ";"**

  Una vez inicializada una constante no cambiará de valor.

  *Convención de denominación*: todo en mayúsculas. Si hay más de una palabra separadas por *underscore*.

  Ejemplos:

```
final int MAXIMO = 100;
final double PONDERACION = 0.8;
final char SEPARADOR_FECHAS = '/';
```

# Java Constantes: Final / Static

- La constante puede estar definida como atributo de la clase
  - Todos los objetos que se instancien tendrán un espacio de memoria reservado par almacenar el valor de la constante.
  - En este caso se puede usar static para que todos los objetos que se definan de esa clase compartan el mismo valor de la constante estando definida sólo 1 vez en memoria
  - Static también permite el usar un atributo o método sin tener que instanciar un objeto de la clase
    - Ej. Clase Math
- La constante puede estar definida dentro de un método. Sólo será conocida por tanto dentro de ese método

# Ejercicios

- Codificar en java los ejercicios de algoritmos realizados

# Lenguaje de programación Java

- Lo visto hasta ahora del lenguaje es una breve introducción al mismo
  - La especificación completa se puede consultar en
    - http://java.sun.com/docs/books/jls/

# Referencias

- Wikipedia
- Bluej
  - http://www.bluej.org/
- Java
  - Descarga:
    - http://www.oracle.com/technetwork/java/javase/downloads/index.html
  - Documentación
    - http://download.oracle.com/javase/
  - API (Aplication Programming Interface)
    - Java 6: http://download.oracle.com/javase/6/docs/api/
    - Java 7: http://download.oracle.com/javase/7/docs/api/