

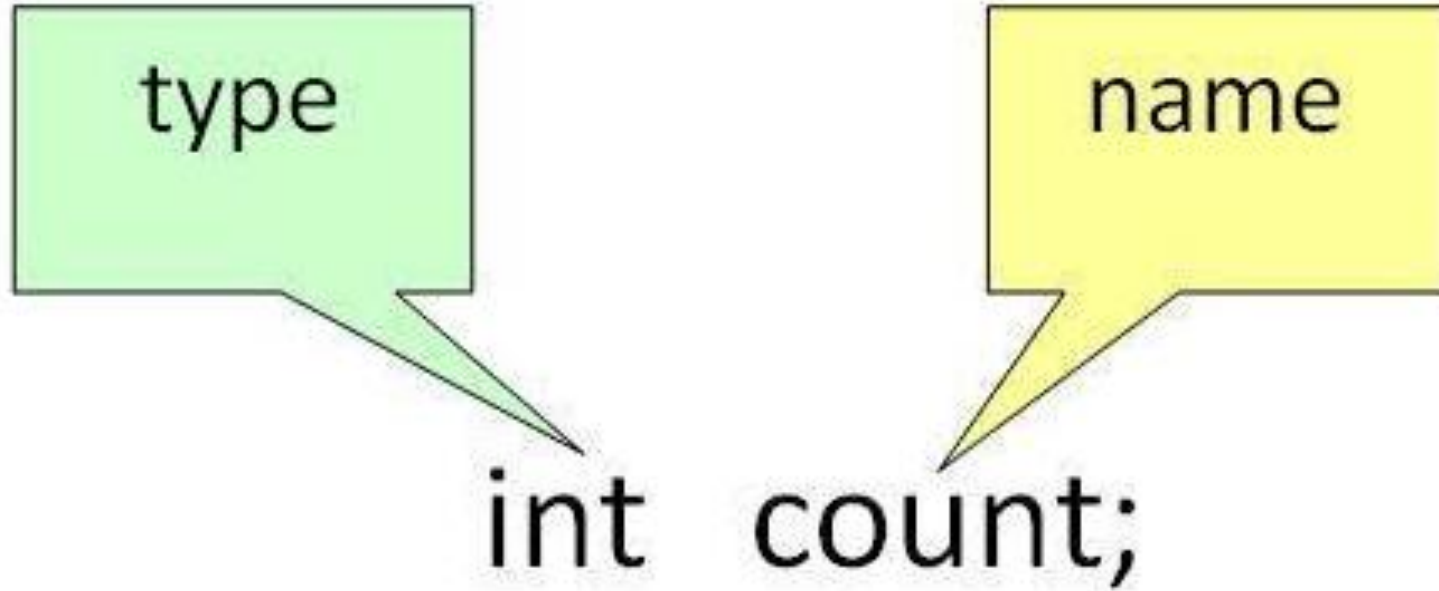
# Estructura básica de un programa en Java

Mikel San Vicente Maeztu

# Hola mundo

```
public class HolaMundo {  
    public static void main(String[] args) {  
        System.out.println("Hola mundo!");  
    }  
}
```

# Declaración de variables



The diagram illustrates the components of a variable declaration in C. It shows the code `int count;` with two callout boxes. A green callout box labeled 'type' points to the `int` keyword, indicating the data type of the variable. A yellow callout box labeled 'name' points to the `count` identifier, indicating the name of the variable.

```
int count;
```

# Hola mundo con variables

```
public class HolaMundo {  
    public static void main(String[] args) {  
        String greetings = "Hola mundo!";  
        System.out.println(greetings);  
    }  
}
```

# Nombres de variables (identificadores)

- Por convención en Java los nombres de las variables deberían empezar por minúscula y ser en inglés. Los programas no dejarán de funcionar por no seguir esta norma pero su legibilidad empeorará.
- Solo se permiten caracteres alfanuméricos y los símbolos '\$' y '\_'
- El primer carácter no puede ser un número
- No se puede usar como identificador palabra reservada (existen 53 palabras reservadas como public, class, static...)

# Tipo de datos

- Según Wikipedia "En [ciencias de la computación](#), un **tipo de dato** informático o simplemente **tipo**, es un atributo de los datos que indica al ordenador (y/o al [programador/programadora](#)) sobre la clase de datos que se va a manejar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar."
- Como veremos más adelante existen tipos predefinidos (String, int, double...) y además podemos definir nuevos tipos del programa que estamos diseñando

# Tipos de datos primitivos

Type	Description	Default	Size	Example Literals
boolean	true or false	false	1 bit	true, false
byte	twos complement integer	0	8 bits	(none)
char	Unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\'', '\n', '\b'
short	twos complement integer	0	16 bits	(none)
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d

# Arithmetic Operators

+	Additive operator (also used for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator



# Unary Operators

- `+` Unary plus operator; indicates positive value (numbers are positive without this, however)
- `-` Unary minus operator; negates an expression
- `++` Increment operator; increments a value by 1
- `--` Decrement operator; decrements a value by 1
- `!` Logical complement operator; inverts the value of a boolean

# Equality and Relational Operators

<code>==</code>	Equal to
<code>!=</code>	Not equal to
<code>&gt;</code>	Greater than
<code>&gt;=</code>	Greater than or equal to
<code>&lt;</code>	Less than
<code>&lt;=</code>	Less than or equal to

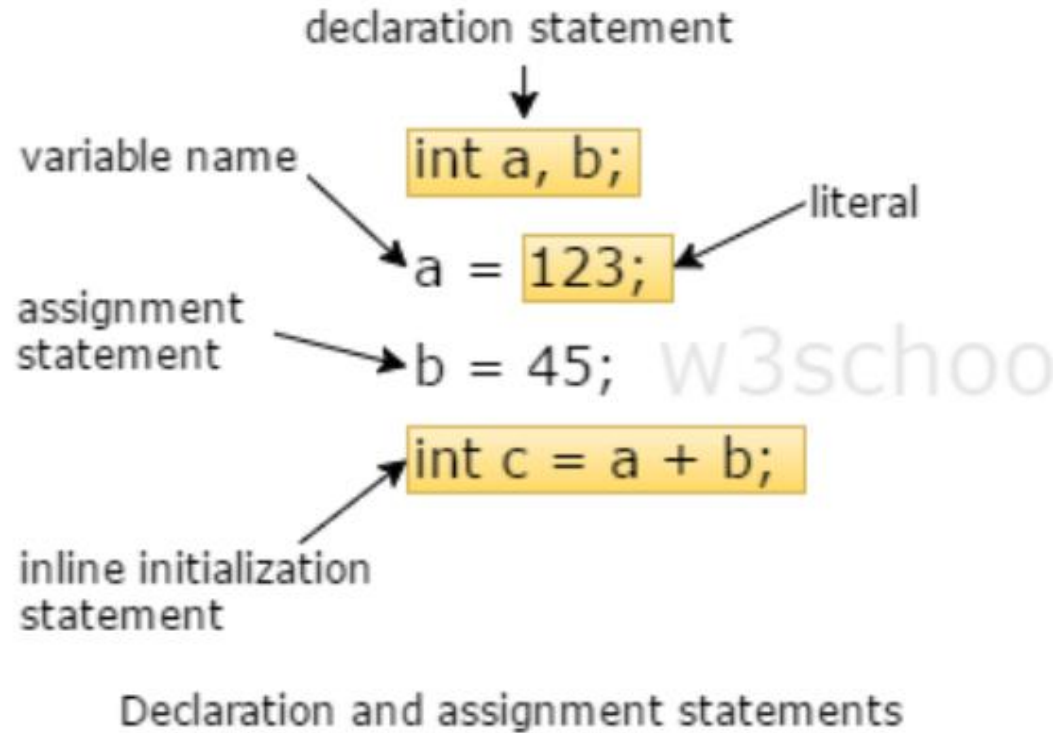
## Conditional Operators

<code>&amp;&amp;</code>	Conditional-AND
<code>  </code>	Conditional-OR
<code>?:</code>	Ternary (shorthand for <code>if-then-else statement</code> )

# Bitwise and Bit Shift Operators

<code>~</code>	Unary bitwise complement
<code>&lt;&lt;</code>	Signed left shift
<code>&gt;&gt;</code>	Signed right shift
<code>&gt;&gt;&gt;</code>	Unsigned right shift
<code>&amp;</code>	Bitwise AND
<code>^</code>	Bitwise exclusive OR
<code> </code>	Bitwise inclusive OR

# Expresiones, literales y sentencias



# Bloques de código

Un bloque de código es una sección de código delimitada por llaves

```
{  
    String hola = "Hola";  
    String mundo = " mundo!";  
    System.out.println(hola + mundo);  
}
```

# Hola mundo con expresiones

```
public class HolaMundo {  
    public static void main(String[] args) {  
        String hola = "Hola";  
        String mundo = " mundo!";  
        System.out.println(hola + mundo);  
    }  
}
```

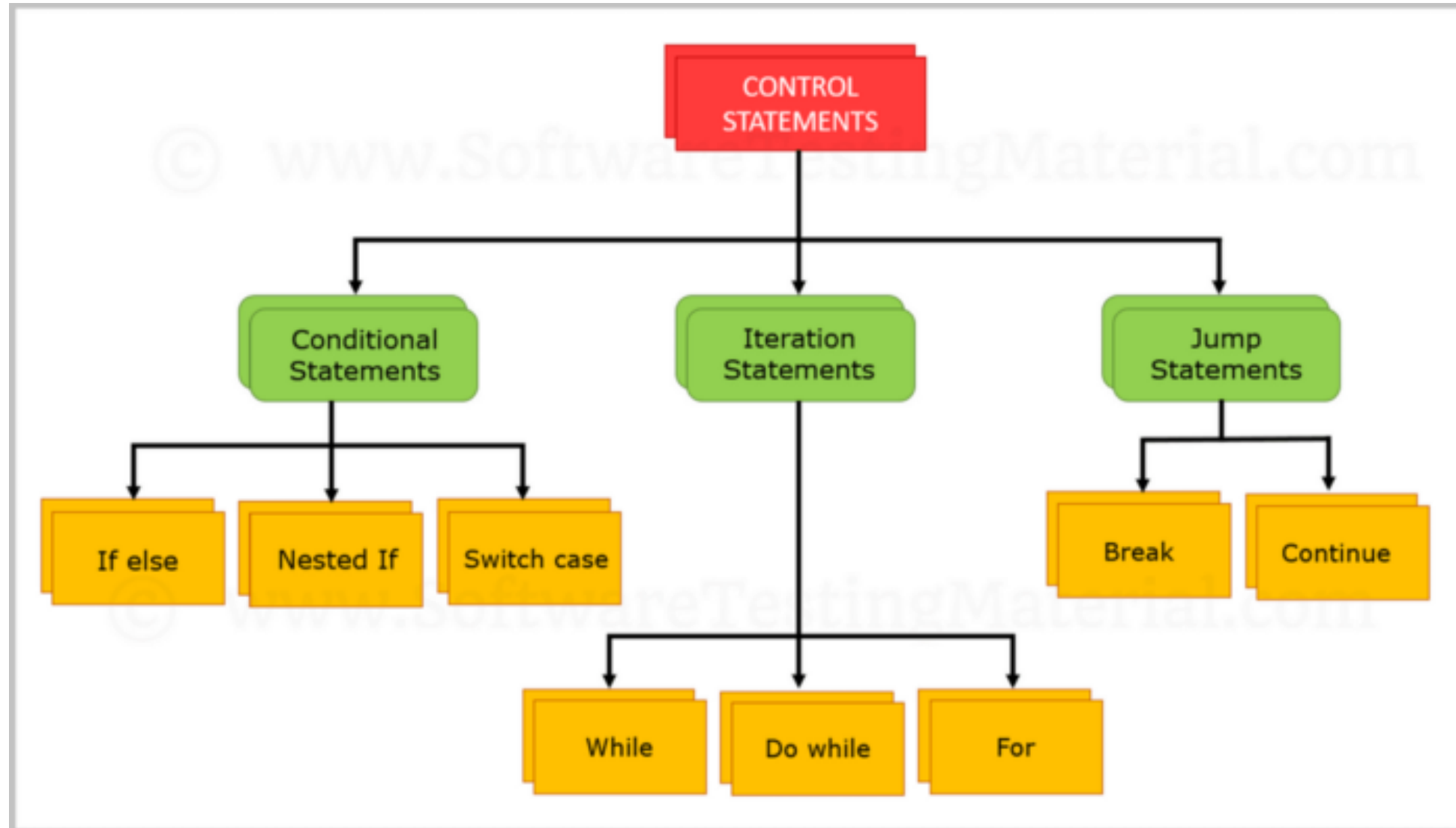
# Ejercicio

# Comentarios

```
// Comentario de una linea  
/*  
    Comentario  
    multilinea  
*/
```



# Estructuras de control



# Estructura if

## Syntax

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

# Estructura if

## Example

```
if (20 > 18) {  
    System.out.println("20 is greater than 18");  
}
```

# Estructura if ... else

## Syntax

```
if (condition) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```

# Estructura if ... else

## Example

```
int time = 20;  
if (time < 18) {  
    System.out.println("Good day.");  
} else {  
    System.out.println("Good evening.");  
}  
// Outputs "Good evening."
```

# Estructura if ... else if ... else

## Syntax

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false and  
} else {  
    // block of code to be executed if the condition1 is false and  
}
```

# Estructura if ... else if ... else

## Example

```
int time = 22;  
if (time < 10) {  
    System.out.println("Good morning.");  
} else if (time < 20) {  
    System.out.println("Good day.");  
} else {  
    System.out.println("Good evening.");  
}  
// Outputs "Good evening."
```

# Ejercicio



# Estructura switch

## Syntax

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

# Estructura switch

## Example

```
int day = 4;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    case 4:
        System.out.println("Thursday");
        break;
    case 5:
        System.out.println("Friday");
        break;
    case 6:
        System.out.println("Saturday");
        break;
    case 7:
        System.out.println("Sunday");
        break;
}
// Outputs "Thursday" (day 4)
```

# Estructura Switch

- La expresión usada en el switch debe ser del tipo int, byte, short, char o String.
- Los valores de los case deben ser del mismo tipo que la expresión del switch.
- Cuando el valor es igual que el case en el que se está se ejecutan todas las sentencias hay hasta que se llega a un break o acaba la estructura switch.
- No todos los casos deben contener un break.
- Al final puede haber un caso por defecto que se seleccionará si ninguno de los casos definidos coincide con la expresión del switch.

# Ejercicio

# Arrays

- Un array es un tipo de datos que contiene una lista de valores de un tipo.



- Para declarar una variable de tipo array hay que abrir y cerrar corchetes después del tipo de datos que va a contener ese array.

```
String[] cars;
```

# Arrays

Los arrays tienen un tamaño fijo que es definido en el momento de su inicialización. Es por esto que los arrays son una estructura de datos estática.

```
String[] array = new String[5];
```

# Arrays

También es posible instanciar un array asignando a la variable la lista de valores separados por comas y rodeado de llaves:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

# Arrays

Una vez definido el array es posible acceder a una posición utilizando su índice numérico. Para acceder a la primera posición se utiliza el índice 0 y así sucesivamente

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
System.out.println(cars[0]);  
// Outputs Volvo
```



# Arrays

Es posible modificar el valor en una posición del array utilizando el índice de esa posición en la sentencia de asignación

```
cars[0] = "Opel";
```

# Arrays

Otra operación que se puede realizar es consultar el tamaño del array, es decir el número de elementos que contiene.

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
System.out.println(cars.length);  
// Outputs 4
```

# Argumentos del programa

- El método main de un programa Java define un parámetro (args) que es un array de String que almacena los argumentos que se han pasado por línea de comandos en la ejecución de un programa.
- La forma de pasar argumentos de ejecución al programa es escribiéndolos separados por espacio después del nombre del programa

```
java NombrePrograma argumento1 argumento2 argumento3
```

- El array args contendrá será equivalente a uno que se hubiera instanciado de la siguiente manera

```
String[] args = { "argumento1", "argumento2", "argumento3" }
```

# Ejercicio

# Métodos en Java

```
public class MyClass {  
    static void myMethod() {  
        // code to be executed  
    }  
}
```

# Llamada al método

```
public class MyClass {  
    static void myMethod() {  
        System.out.println("I just got executed!");  
    }  
  
    public static void main(String[] args) {  
        myMethod();  
    }  
}  
  
// Outputs "I just got executed!"
```

# Llamada al método

```
public class MyClass {  
    static void myMethod() {  
        System.out.println("I just got executed!");  
    }  
  
    public static void main(String[] args) {  
        myMethod();  
    }  
}  
  
// Outputs "I just got executed!"
```

# Métodos con parámetros

```
public class MyClass {  
    static void myMethod(String fname, int age) {  
        System.out.println(fname + " is " + age);  
    }  
}
```

```
public static void main(String[] args) {  
    myMethod("Liam", 5);  
    myMethod("Jenny", 8);  
    myMethod("Anja", 31);  
}  
}
```

```
// Liam is 5  
// Jenny is 8  
// Anja is 31
```



# Métodos que devuelven valores

```
public class MyClass {  
    static int myMethod(int x) {  
        return 5 + x;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(myMethod(3));  
    }  
}  
// Outputs 8 (5 + 3)
```

# Ejercicio

# Loops

- Los loops o bucles ejecutan un bloque de código repetidas veces hasta que una condición se cumple
- Existen varios tipos de bucles que se pueden usar en Java
  - While loop
  - Do/while loop
  - For loop
  - For-each loop

# While loop

## Syntax

```
while (condition) {  
    // code block to be executed  
}
```

# While loop

## Example

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

# Ejercicio

- Escribe un programa que recibe dos argumentos de tipo entero
- El programa multiplicará por dos el valor del argumento más pequeño hasta que supere el valor del otro
- Imprime en pantalla el valor de los dos valores
- Utiliza un bucle while para este ejercicio

# Do -While loop

## Syntax

```
do {  
    // code block to be executed  
}  
while (condition);
```

- Es una variante del bucle while en la cual siempre se va a ejecutar el bloque de código al menos una vez
- La condición se comprueba por primera vez después de esta primera ejecución y a partir de ahí funciona como un while normal

# Do -While loop

## Example

```
int i = 0;  
do {  
    System.out.println(i);  
    i++;  
}  
while (i < 5);
```



# Ejercicio

- Escribe un programa que recibe un argumento de tipo entero
- El programa escribirá en pantalla el mensaje "Hola mundo " seguido de un número que irá incrementando hasta escribir tantas líneas como el argumento del programa
- El programa debe usar un bucle do-while

# For loop

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

- Statement 1: se ejecuta solo una vez, antes de comprobar la condición por primera vez. Suele utilizarse para inicializar la variable que se va a usar para iterar.
- Statement 2: expresa la condición que se va a comprobar en cada iteración del bucle
- Statement 3: Se ejecuta al finalizar cada iteración, normalmente se utiliza para modificar el valor de la variable que vamos a usar para iterar

# Ejemplo For loop

```
for (int i = 0; i < 5; i++) {  
    System.out.println(i);  
}
```

# Ejercicio

- Escribe un programa que recibe un argumento de tipo entero
- El programa escribirá en pantalla el mensaje "Hola mundo " seguido de un número que irá incrementando hasta escribir tantas líneas como el argumento del programa
- El programa debe usar un bucle for

# For each loop

```
for (type variableName : arrayName) {  
    // code block to be executed  
}
```

- Se utiliza cuando queremos recorrer todas las posiciones de una lista sin importarnos en qué posición está cada una de ellas
- A la derecha de los : tenemos la declaración de la variable en la que se va a almacenar cada uno de los valores del array
- A la derecha tenemos el array sobre el que vamos a iterar

# Ejemplo For each loop

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
for (String i : cars) {  
    System.out.println(i);  
}
```

# Ejercicio

- Declara un array que contenga los siguientes valores: 3, 5, 6, 22, 4 y -1
- Escribe un programa que use un bucle for-each para encontrar el valor mínimo de entre los elementos del array
- Imprime en pantalla el valor del mínimo

# Programas interactivos

- En ocasiones vamos a querer que el usuario interactúe con el programa durante su ejecución.
- Una forma de interacción sería que el programa solicite al usuario que este le proporcione datos con los realizar su ejecución.
- Para esto vamos a usar la utilidad Scanner
- Para utilizarla hay que importar el paquete `java.util.Scanner`



# Programas interactivos

```
import java.util.Scanner; // Import the Scanner class

class MyClass {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in); // Create a Scanner object
        System.out.println("Enter username");

        String userName = myObj.nextLine(); // Read user input
        System.out.println("Username is: " + userName); // Output user input
    }
}
```

# Break - Ruptura de un bucle

```
for (int i = 0; i < 10; i++) {  
    if (i == 4) {  
        break;  
    }  
    System.out.println(i);  
}
```

# Ejercicio

- Implementa un programa que busque un valor dentro de un array
- El array de valores (String) se definirá en el método main
- El elemento a buscar será el primer argumento del programa
- Para recorrer el array utiliza un bucle for, foreach o while
- Cuando se encuentre el valor se ejecutará una sentencia break para salir del bucle
- Se imprimira en pantalla un mensaje informando de si se ha encontrado el bucle o no

# Continue - Pasar a la siguiente iteración

```
for (int i = 0; i < 10; i++) {  
    if (i == 4) {  
        continue;  
    }  
    System.out.println(i);  
}
```

# Casting de tipos

El casting ocurre cuando asignamos un valor de un tipo primitivo a una variable de otro tipo primitivo

```
int myInt = 9;
```

```
double myDouble = myInt;
```

# Casting de tipos

Existen dos tipos de casting

- Widening casting (automático): se convierte un tipo más pequeño en uno más grande.

byte -> short -> char -> int -> long -> float -> double

- Narrowing casting (manual): se convierte un tipo más grande en uno más pequeño

double -> float -> long -> int -> char -> short -> byte

# Widening casting

```
public class MyClass {  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt; // Automatic casting: int to double  
  
        System.out.println(myInt);    // Outputs 9  
        System.out.println(myDouble); // Outputs 9.0  
    }  
}
```

# Narrowing casting

```
public class MyClass {  
    public static void main(String[] args) {  
        double myDouble = 9.78;  
        int myInt = (int) myDouble; // Manual casting: double to int  
  
        System.out.println(myDouble); // Outputs 9.78  
        System.out.println(myInt);    // Outputs 9  
    }  
}
```