

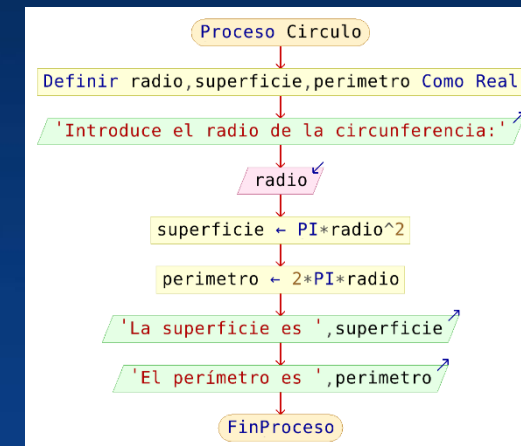
UD1. Elementos del lenguaje

0. Índice

1. Introducción
2. Java
3. Pseudocódigo
4. Variables
5. Constantes
6. Tipos de datos
7. Sentencias
8. Operadores de asignación
9. Entrada y salida de datos. Pseudocódigo
10. Estructura de una clase Java
11. Método main de una clase Java
12. Salida de datos en Java
13. Eclipse. Instalación
14. Eclipse. Personalizar editor
15. Salida de datos. Concatenación de valores
16. Salida de datos con formato
17. Entrada de datos en Java
18. Entrada de datos de tipo entero
19. Entrada de datos de tipo real
20. Entrada de datos de tipo carácter
21. Entrada de datos de tipo String (cadena)
22. Forzado de tipo de datos. Casting

1. Introducción

- La principal finalidad de la programación es la de **resolver un problema**, de cualquier índole, mediante un sistema informático.
- Para ello, lo primero que se debe realizar es un **estudio detallado** del problema.
- Este estudio tendrá como resultado unas especificaciones que se expresarán en **pseudocódigo**, que no es comprensible para un ordenador.
- Un **sistema informático** es un conjunto de dispositivos físicos que solo comprenden señales de tipo eléctrico (ausencia / presencia de tensión).



1. Introducción

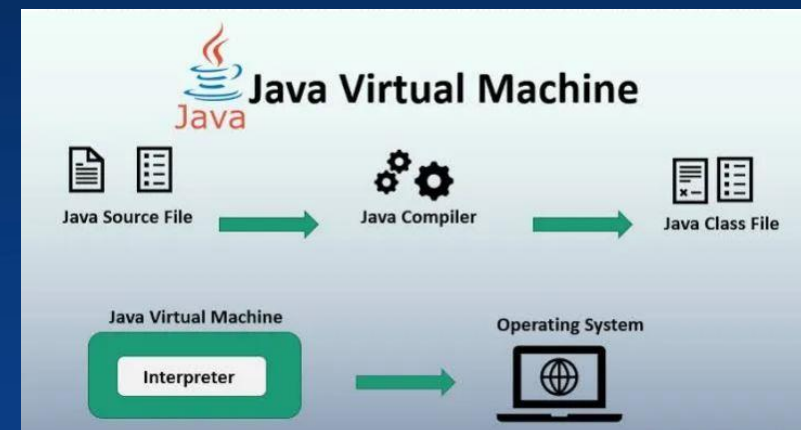
- Para que el sistema informático entienda la solución del pseudocódigo, debemos traducirlo a un **lenguaje de programación de alto nivel**, es decir, próximo a un lenguaje humano.
- Para que el sistema informático ejecute el programa en lenguaje de alto nivel, debemos usar **un compilador** o **intérprete**, que convierte todo el código en un código objeto.
- El código objeto para ser ejecutado necesita un **enlazador (linker)** que lo convierte en código comprensible para la máquina.



```
products: storeProducts
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title="product">
            <div className="row">
              <ProductConsumer>
                {(value) => {
                  console.log(value)
                }}
              </ProductConsumer>
            </div>
          </div>
        </div>
      </React.Fragment>
    )
}
```

2. Java

- Java es un lenguaje de alto nivel Orientado a Objetos que puede ser **interpretado o compilado** según las necesidades. Es independiente de la máquina por lo que un programa en Java puede ser ejecutado en varios dispositivos.
- Para ello, lo único que se necesita es tener instalado en el dispositivo la **Java Virtual Machine (JVM)** correspondiente, que ejecutará el código objeto para generar el código comprensible para la máquina.



3. Pseudocódigo

- En la vida real, para resolver un problema seguimos numerosos pasos, a veces, debido a la repetición, de manera inconsciente. A la hora de preparar el desayuno:
 - 1. Comprobar si hay corriente eléctrica.
 - 2. Enchufar el microondas a la corriente eléctrica (si no lo está).
 - 3. Verter lo que vayamos a calentar en la taza de desayuno metálica.
 - 4. Abrir la puerta del microondas.
 - 5. Meter la taza de desayuno no metálica en el microondas.
 - 6. Cerrar la puerta del microondas.



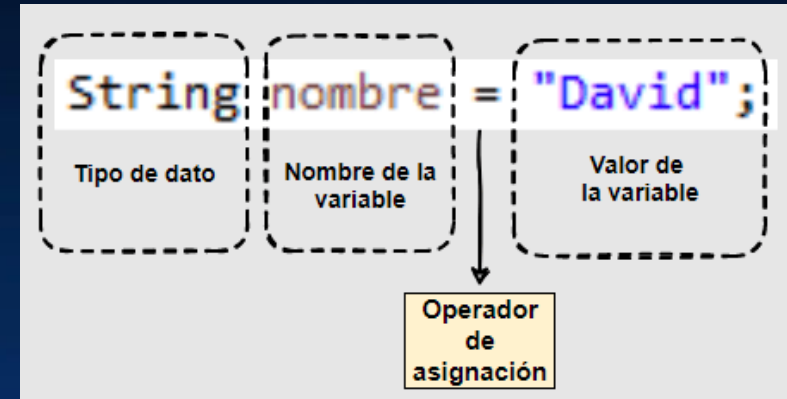
3. Pseudocódigo

- 7. Seleccionar la potencia deseada (por ejemplo 500 W).
- 8. Seleccionar el tiempo deseado (Por ejemplo 2 minutos).
- 9. Poner en marcha el microondas.
- 10. Esperar a que pase el tiempo deseado.
- 11. Abrir la puerta del microondas.
- 12. Sacar la taza de desayuno no metálica caliente.
- 13. Desayunar



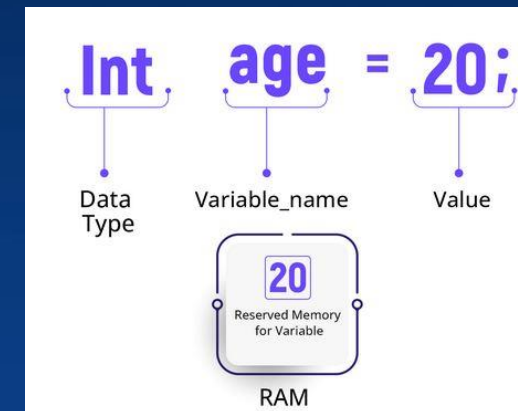
4. Variables

- En nuestro desayuno podemos encontrar diferentes variables:
 - Contenido del desayuno
 - Contenedor del desayuno
 - Potencia del microondas
 - Tiempo para calentar
- Una variable es la **unidad básica de almacenamiento de datos**. Su declaración se compone de un **identificador**, un **tipo**, un **ámbito** y un **valor**.
- `public int tiempoCalentamiento = 100;`



4. Variables

- Cuando se declara una variable el compilador reserva **espacio en memoria** para guardar un dato del tipo de la variable y asocia la dirección de memoria en que se encuentra ese espacio con el **identificador**, de manera que a partir de ese momento podemos utilizar el identificador para acceder a esa posición de memoria.
- Por lo que una variable consta de dos partes:
 - **Valor:** El contenido de la variable, al que se accede con el identificador de la variable. Se usa para asignar un valor a la variable, o para obtener el valor de la variable.
 - **Dirección:** Es la dirección de memoria donde se encuentra la variable.
- Las variables se pueden utilizar sólo dentro de su **ámbito** (**public**, **private** o **protected**).



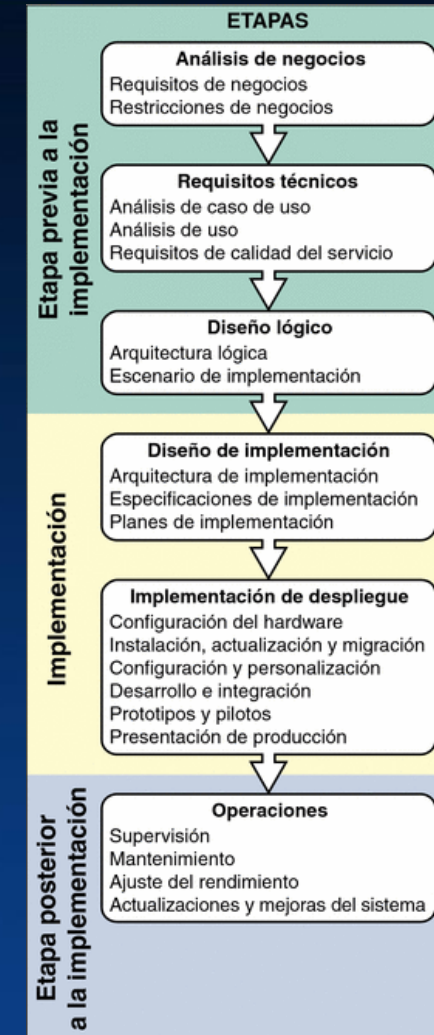
4. Variables

- El nombre de las variables se deben escribir en Java (difiere del lenguaje de programación sutilmente) en **minúsculas**. En caso de tener un nombre compuesto de varias palabras, se escribirán las **palabras seguidas sin espacios**, siendo la **primera inicial en minúscula** y las **siguientes en mayúsculas**.
 - `potenciaMicroondas = 500;`
- El nombre de la variable debe ser claro, para que no se pueda confundir con otra variable del programa:
 - `numeroPrimo = 3;`
 - Evitar **n**, **num** y si es posible evitar escribir **numero** en caso de que haya más de uno: **numeroPrimo1**, **numeroPrimo2**

```
5 public static void main(String[] args) {  
6  
7     //Identificadores válidos  
8     String nombre;  
9     int edad;  
10    String s;  
11    double _salario; //No recomendable, nombre debe ser intuitivo  
12    int $resultado; //No recomendable, comenzar por letra minúscula  
13    String Direccion; //No recomendable, comenzar por letra minúscula  
14    String numerotelefono; //No recomendable, debería empezar en minúscula  
15  
16    //Identificadores no válidos  
17    String 2apellidos;  
18    int :idUserio;  
19    char .sexo;  
20    int super;  
21 }
```

4. Variables

- Al programar, **no basta** con que el código funcione, cumpla con el diseño predefinido o supere todos los tests.
- Existe una **fase de mantenimiento** en la que los programadores deben hacer modificaciones en el código, a veces, meses o años después.
- Si el código no está bien escrito, esta fase se complica. Por lo que debemos acostumbrarnos a **seguir las recomendaciones** a la hora de escribir o nombrar los diferentes elementos que veremos a lo largo de la asignatura.



5. Constantes

- Son los **valores fijos** que no pueden ser modificados a lo largo del programa. Pueden ser de cualquier tipo.
- En java las constantes son variables con estado **Final**, que significa que no pueden cambiar.
- Su nombre se escribe en **mayúsculas** separando cada palabra con un **guion bajo “_”**:
 - `int IRPF_GENERAL = 21;`
- Las constantes se usan para **no escribir el mismo número** innumerables veces, para que si es necesario modificar el programa, solo haya que hacerlo una vez.

```
package com.clases;

/**
 * Created by ericka.montero .
 */
public class Constantes {

    public static Integer NUMERO_PUERTAS = 4;

    public static final Integer NUMERO_RUEDAS = 4;

    public static final String MARCA_CARRO = "TOYOTA";

}
```

6. Tipos de datos

- **Enteros:** Números sin decimales.
 - ENTERO NUMERO pseudocódigo
 - `int numero = 10;` java – entero simple
 - `long numero;` java – entero de mayor rango (números grandes)
- **Reales:** Números con decimales.
 - REAL DECIMAL pseudocódigo
 - `float decimal = 10.0;` java – precisión simple
 - `double decimal;` java – mayor precisión (más decimales)

```
package net.javaguides.corejava.variables;

public class LocalVariableExample {
    public int sum(int n) {
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum = sum + i;
        }
        return sum;
    }

    public static void main(String[] args) {
        LocalVariableExample localVariableExample = new LocalVariableExample();
        int sum = localVariableExample.sum(10);
        System.out.println("Sum of first 10 numbers -> " + sum);
    }
}
```

Usage of int keyword

6. Tipos de datos

- **Carácter:** Una única letra.
 - CARÁCTER LETRA pseudocódigo
 - `char letra = 'A';` java
- **Lógicos o booleanos:** Verdadero o falso, sí o no, 0 o 1.
 - BOOLEANA ONOFF pseudocódigo
 - `boolean onOff = true;` java
- **String:** Cadena de caracteres. Es una clase (UD5).
 - CADENA FRASE pseudocódigo
 - `String palabra = "hola";` java

```
// Demonstrate char data type.
class CharDemo {
    public static void main(String args[]) {
        char ch1, ch2;
        ch1 = 88; // code for X
        ch2 = 'Y';
        System.out.print("ch1 and ch2: ");
        System.out.println(ch1 + " " + ch2);
    }
}
```

7. Sentencias

- En java todas las sentencias sencillas deben terminar con el carácter punto y coma (;);



```
78 ... trim(preg_replace('/\\\\\\\\/', '/', $image_src), '/');
79 $_SESSION['_CAPTCHA']['config'] = serialize($captcha_config);
80 return array(
81     'code' => $captcha_config['code'],
82     'image_src' => $image_src
83 );
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
```

The screenshot shows a PHP code editor with a file explorer on the left. The file explorer lists files like Empty.html, send_form_email.php, HTML, CSS, fonts, skins, custom.css, ie.css, theme.css, theme-animate.css, home-blog.css, home-elements.css, and home-shop.css. The main editor displays PHP code for a captcha system. The code includes a function to generate a captcha image and a function to verify it. The code is written in a dark-themed editor with syntax highlighting. The code is as follows:

8. Operadores de asignación

- **Operadores aritméticos:** Sirven para realizar operaciones aritméticas con los datos.

- `resta = 3 - 2;`
- `resta = - resta;`
- `resta--;`
- `resultado = numero1++/numero2-3;`
- `resultado = numero1++/(numero2-3);`

| OPERADOR | DESCRIPCIÓN | ORDEN |
|-------------------------|------------------------------|-------|
| - | Resta. | 3 |
| + | Suma | 3 |
| * | Multiplicación | 2 |
| / | División | 2 |
| % (MOD en pseudocódigo) | Resto de una división entera | 2 |
| - | signo (unitario). | 2 |
| -- | Decremento en 1. | 1 |
| ++ | Incrementa en 1. | 1 |

8. Operadores de asignación

- **Operadores relacionales:** Sirven para realizar comparaciones de expresiones.
 - `if (numero1 <= numero2)`
 - `if (numero1 != numero2)`

| OPERADORES RELACIONALES | | |
|-------------------------|----------------|-------|
| OPERADOR | DESCRIPCIÓN | ORDEN |
| < | Menor que. | 5 |
| > | Mayor que. | 5 |
| <= | Menor o igual. | 5 |
| >= | Mayor o igual | 5 |
| = = | Igual | 6 |
| != | Distinto | 6 |

8. Operadores de asignación

- **Operadores lógicos:** Sirven para realizar comparaciones de expresiones.
 - loginCorrecto && passwordCorrecto
 - loginCorrecto || passwordCorrecto
 - !loginCorrecto

| OPERADORES LÓGICOS | | |
|--------------------|-------------------------------|-------|
| OPERADOR | DESCRIPCIÓN PSEUDOCODIGO / | ORDEN |
| && | Y (AND) | 10 |
| | O (OR) | 11 |
| ! | NO (NOT) | 1 |

9. Entrada y salida de datos. Pseudocódigo

- LEER NUMERO
- ESCRIBIR NUMERO
- Ejercicio 1. Realiza el pseudocódigo para el programa LEERNUM que lee un número por teclado y lo muestra por pantalla.

| | |
|-----------------|--|
| LEERNUM | //nombre del programa |
| ENTERO NUMERO | //declaración de la variable NUMERO para su posterior uso |
| LEER NUMERO | //leer el valor que introduce el usuario del programa y lo guarda en la variable NUMERO |
| ESCRIBIR NUMERO | //Imprimir en la consola el valor contenido dentro de la variable NUMERO |

9. Entrada y salida de datos. Pseudocódigo

- Ejercicio 2. Realiza el pseudocódigo para el programa AREACIRC que lee el valor del radio por teclado y muestra el valor del área de un círculo de ese radio por pantalla con dos decimales.

AREACIRC

REAL RADIO

REAL AREA

//declaramos tanto la variable que pedimos al usuario como la que queremos calcular y mostrar en la consola

LEER RADIO

AREA \leftarrow PI x radio²

//realizamos la operación matemática

ESCRIBIR AREA CON DOS DECIMALES

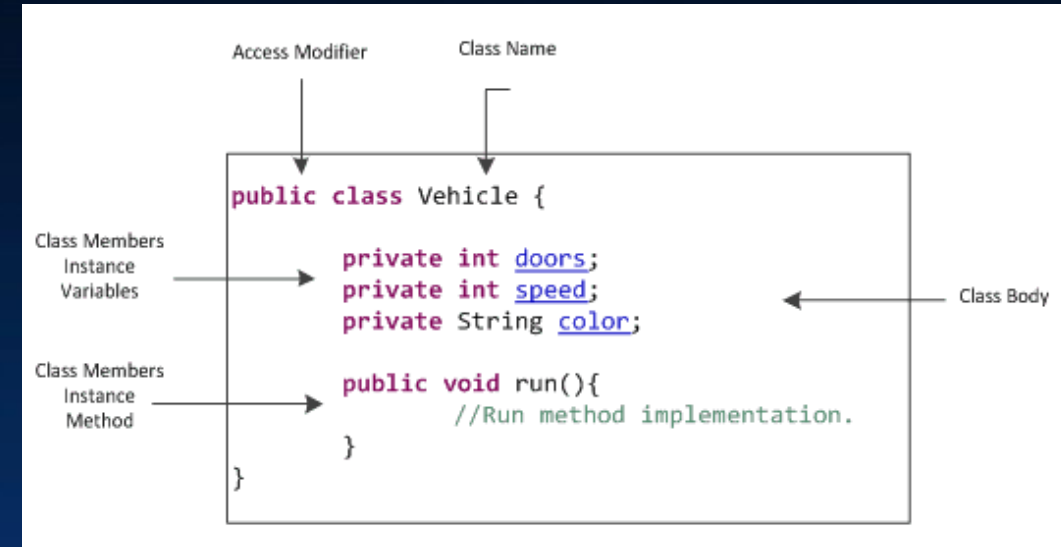
//imprimimos por consola pero especificando que solo ponga 2 decimales

10. Estructura de una clase Java

- Una clase java se define de la siguiente manera:

```
public class Hola {  
}
```

- Hola** es el nombre de la clase.
- Class** es la palabra Java para definir una clase.
- Public** es la palabra para permitir que la clase se pueda ejecutar desde el exterior (por ejemplo desde la línea de comandos).

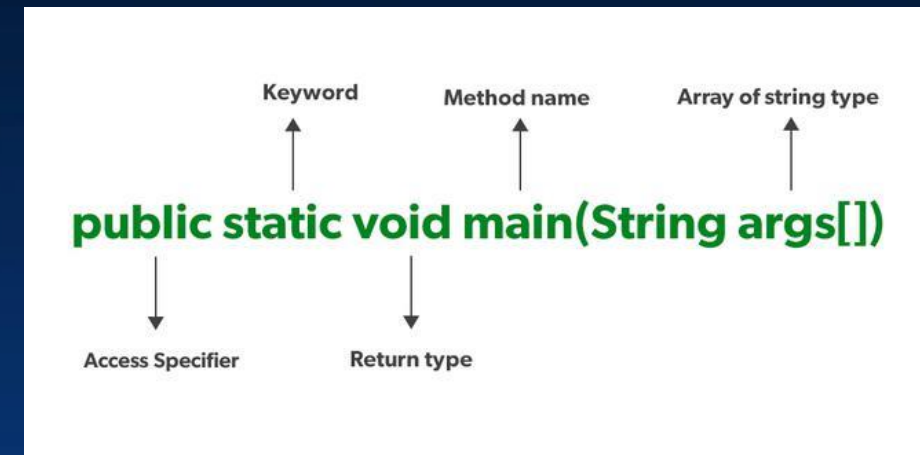


11. Método main de una clase Java

- El método **main** de una clase es el método por el que comienza la ejecución de dicha clase. Si una clase no dispone de método main no se ejecutará.

```
Public static void main (String[] args){  
}
```

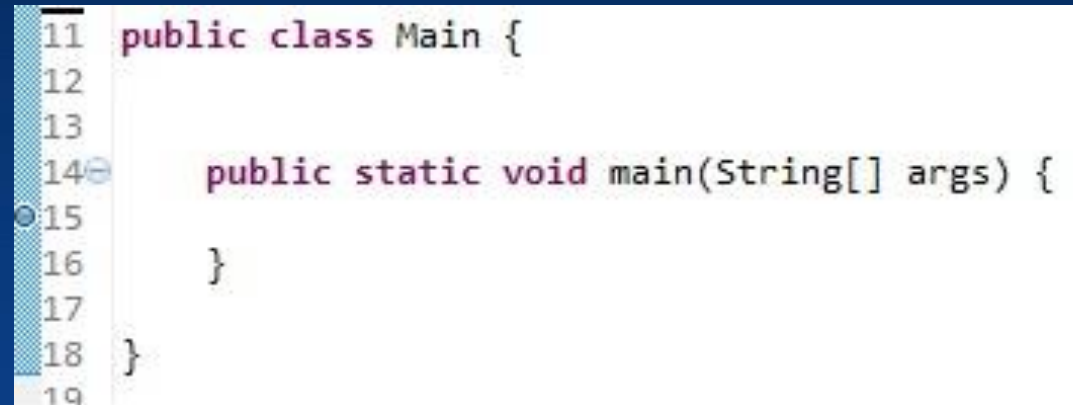
- Main** es el nombre del método.
- Void** indica que el método no devuelve ningún valor.
- Static** indica que sólo hay un método main para todos los objetos de la misma clase (comparten su contenido)



11. Método main de una clase Java

```
Public static void main (String[] args){  
}
```

- **Public** indica que se puede llamar al método para que se ejecute desde fuera de la clase.
- **String[] args** son datos que necesita el método para poder ejecutarse correctamente. Es un array de Strings cuyo nombre es args.
- Una clase no debe superar las 1000 líneas de código.
- Si esto ocurre, es indicativo de que se ha aplicado un mal diseño de clases.

A screenshot of a code editor with a light blue background. It shows a Java class named 'Main' with a 'main' method. The code is as follows:

```
11 public class Main {  
12  
13  
14     public static void main(String[] args) {  
15  
16     }  
17  
18 }  
19
```

Line numbers 11 through 19 are visible on the left side of the editor. The 'main' method is indented four spaces from the class opening brace.

12. Salida de datos en Java

- Se usa la clase predefinida `System.out`:
 - Para escribir “Hola” y saltar de línea:
`System.out.println("Hola");`
 - Para escribir “Hola” sin saltar de línea:
`System.out.print("Hola");`
- Cuando programamos, por cada línea, no debemos superar los 100 caracteres, para que cada instrucción sea fácil de leer, evitando así expresiones largas que pueden escribirse en dos o más líneas.

| Example | Result |
|---|---------------------|
| <code>System.out.print("one");</code> <code>System.out.print("two");</code> <code>System.out.println("three");</code> | onetwothree |
| <code>System.out.println("one");</code> <code>System.out.println("two");</code> <code>System.out.println("three");</code> | one two three |
| <code>System.out.println();</code> | [new line] |

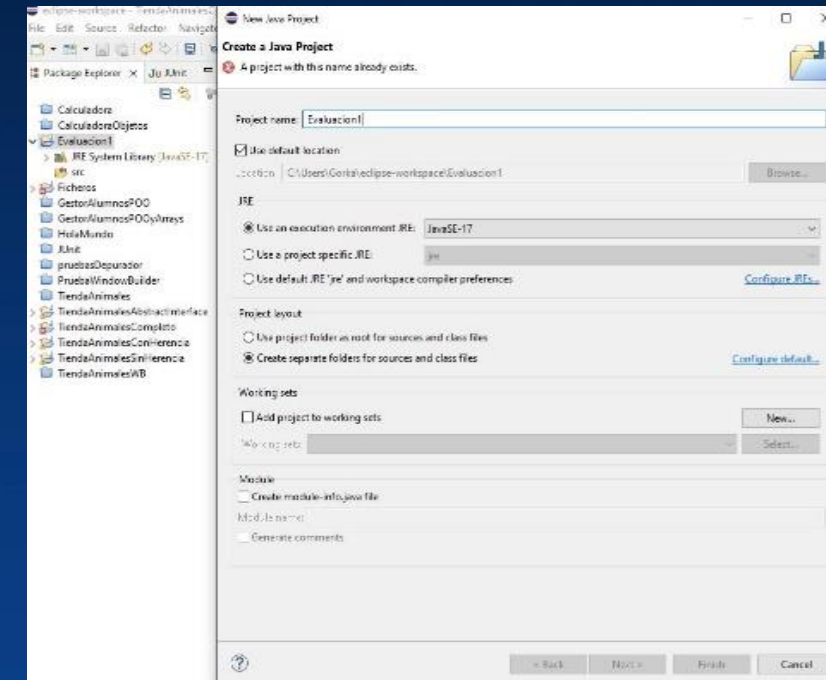
13. Eclipse. Instalación

- Eclipse es el **IDE (Entorno de Desarrollo Integrado)** que vamos a utilizar para desarrollar en Java.
- Si existe una versión previa de Eclipse en nuestro ordenador, la desinstalamos.
- <https://www.eclipse.org/downloads/>
- Una vez descargada, descomprimos el contenido del fichero zip en la carpeta **C:\eclipse**.
- Al ejecutar, nos pedirá una carpeta donde crearemos el **Workspace**, que es donde guardará los proyectos que realicemos.



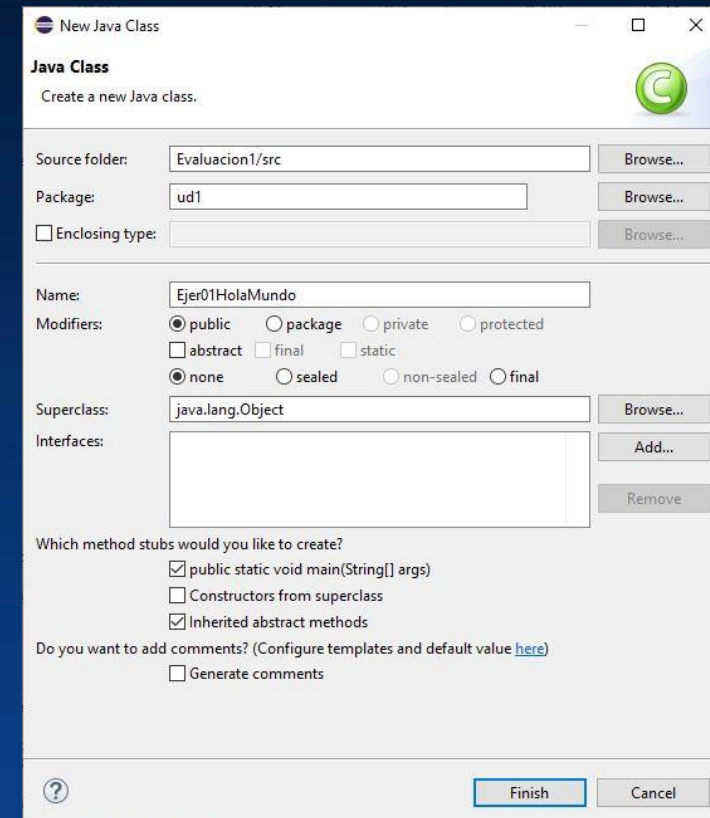
13. Eclipse. Instalación

- Cerramos la pestaña de Bienvenida y vamos a **File → New → Java Project** para crear un proyecto java nuevo.
- Ponemos como nombre del proyecto **Evaluacion1**. El nombre de un proyecto siempre **debe empezar por Mayúscula**. Se guardará en nuestro Workspace.
- En la Pestaña Package Explorer, desplegamos el contenido del proyecto Java y hacemos click con el botón derecho sobre la carpeta src (Source) y seleccionamos **New → Package** para crear un nuevo paquete donde guardar los ejercicios de la primera unidad, que tendrá el nombre **ud1**. Los nombres de los paquetes empiezan siempre por minúscula.



13. Eclipse. Instalación

- Después hacemos clic con el botón derecho sobre el paquete ud1 y seleccionamos **New** → **Class** para crear una nueva clase Java.
- El nombre de la clase, que siempre debe empezar por mayúscula, será **Ejer01HolaMundo**.
- Marcamos la casilla **public static void main(String[] args)** para que nos cree ese método automáticamente ya que es el método en el que tenemos que escribir el código que queremos ejecutar.
- Pulsamos **Finish** para terminar de crear el proyecto.



13. Eclipse. Instalación

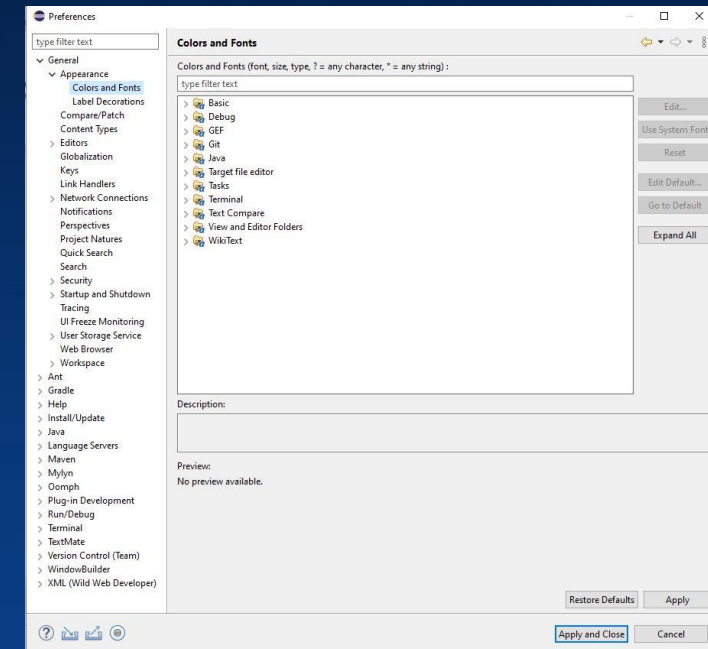
- Ejercicio 1
 - Realiza en Eclipse la clase Java HolaMundo que muestra por pantalla el mensaje “Hola Mundo”. Ejecuta la clase usando eclipse.

```
1 package ud1; //Indica el paquete al que pertenece
2
3 public class Ejer01HolaMundo { //Nombre de la clase, al ser public es accesible desde fuera de la clase
4
5     public static void main(String[] args) { //Main, método principal necesario para la ejecución del programa
6         System.out.println("Hola Mundo"); //Saca por consola el texto "Hola Mundo" y hace un salto de línea
7     }
8 }
9
10 }
```

- Tras escribir el código, debemos pulsar el botón **Save** y una vez guardado el botón **Run**.

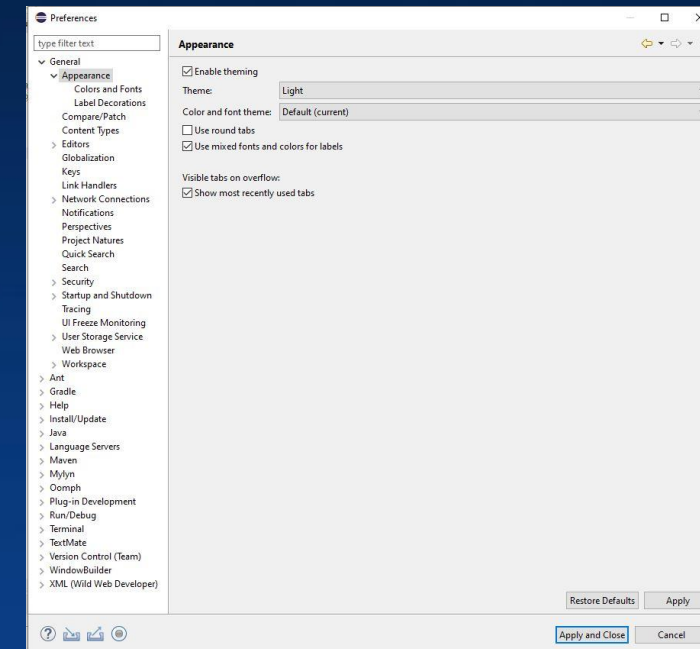
14. Eclipse. Personalizar editor

- Window → Preferences → General → Appearance → Color and Fonts, Basics: Cambiar el tamaño de la fuente.
- Window → Preferences → Java → Code Style → Formatter: Cambiar el tamaño de las tabulaciones. Cambiar el nombre del perfil por Eclipse – PROG. En la pestaña Indentation, Tab Size 2.



14. Eclipse. Personalizar editor

- Window → Preferences → General → Appearance, Theme. Cambiar el color de la aplicación.
- Window → Preferences → General → Appearance → Color and Fonts, debug: Personalizar la fuente de la consola.



15. Salida de datos. Concatenación de valores

- Se puede usar `System.out.println` para mostrar por pantalla el valor de una o varias variables mezclado con uno o varios textos con el operador de concatenación (+):

- `System.out.println ("El valor de la variable es " + numero);`

- Ejercicios 2-5

- Es necesario tabular para hacer la lectura del código más fácil. Del mismo modo, dejando líneas en blanco nos permite agrupar código en grupos para su mejor comprensión.

- `Ctrl + Shift + F`: Autotabulación.

```
public class Students {  
    public static void main(String[] args) {  
  
        String s1 = "Java Programming";  
        String s2 = "at seu";  
        String s3 = "Type your name";  
        System.out.print(s1);  
        System.out.print(" ");  
        System.out.println(s2.toUpperCase());  
        System.out.println(s3);  
        System.out.println("This is my first Assignment");  
        System.out.println("+++++++");  
        System.out.print(s1.replace("java", "CS140"));  
        System.out.print(" ");  
        System.out.println(s2.toLowerCase());  
        System.out.print(8 + 8);  
        System.out.print(" = ");  
        System.out.println(s1.length(5));  
    }  
}
```

16. Salida de datos con formato

- Si queremos mostrar por pantalla datos en un formato determinado usaremos:
 - `System.out.printf`
- El método `printf` usa, entre otros, los siguientes identificadores de formato:
 - `%d` datos de tipo `int`
 - `%f` datos de tipo `float`
 - `%f` datos de tipo `double`
 - `%c` datos de tipo `char`
 - `%s` datos de tipo `String`
- Si queremos que aparezcan dos decimales debemos usar `"%.2f"`.

PRINT FORMATTING: `PRINTF()`

Conversion Type Characters ::

Formatting String

Output ::

| | |
|--|-----------|
| <code>System.out.printf("%d", 10);</code> | 10 |
| <code>System.out.printf("%f", 10.1);</code> | 10.100000 |
| <code>System.out.printf("%c", 'a');</code> | a |
| <code>System.out.printf("%C", 'a');</code> | A |
| <code>System.out.printf("%s", "hello");</code> | hello |
| <code>System.out.printf("%S", "hello");</code> | HELLO |
| <code>System.out.printf("%b", 5 < 4);</code> | false |
| <code>System.out.printf("%B", 5 < 4);</code> | FALSE |
| <code>System.out.printf("%b", null);</code> | false |
| <code>System.out.printf("%b", "cow");</code> | |

16. Salida de datos con formato

- Ejercicio 6
 - Realiza la clase Java MostrarReal2 que coge el valor de una variable de tipo real con decimales y lo muestra con 2 decimales.

```
1 package ud1;
2
3 public class Ejer06MostrarReal2 {
4
5     public static void main(String[] args) {
6
7         double pi = 3.1416;
8         System.out.printf("%.2f", pi);           //para usar printf se deben pasar como argumento al método 2 datos, un string con el formato entre comillas y la variable
9         System.out.printf("El numero pi es %.2f", pi); //otro ejemplo en el que se muestra que podemos escribir un texto antes del formato que aparecerá por pantalla
10
11     }
12
13 }
```

17. Entrada de datos en Java

- Consiste en introducir datos por teclado.
- Para ello usaremos la clase **Scanner** que se encuentra dentro de la clase **java.util**. Pero **java.util** no se encuentra entre las clases que se cargan por defecto al programar en Java, por lo que la debemos **importar**.
- Para importarla debemos escribir antes de usar la clase Scanner:
 - **import java.util.Scanner;**

```
package com.mcnz.example;

import java.util.Scanner; // explicit import

public class JavaScannerImportExample {

    public static void main(String args[]) {


        System.out.println("What is your favorite color?");
        Scanner stringScanner = new Scanner(System.in);
        String color = stringScanner.next();
        System.out.println(color + " is my favorite color too!");
        stringScanner.close();

    }
}
```

17. Entrada de datos en Java

- Dentro de nuestra función main definiremos un objeto de tipo Scanner:
 - `Scanner teclado = new Scanner(System.in);`
- Esto hace que podemos leer datos por teclado.
- Una vez terminamos de pedir datos por teclado debemos cerrar el objeto Scanner, para que no afecte a otras aplicaciones en uso.
 - `teclado.close();`

```
5 public class ScannerInput {  
6  
7     public static void main(String[] args) {  
8         Scanner scnr = new Scanner(System.in);  
9         int num;  
10        String sentence;  
11  
12        System.out.println("Enter a number: ");  
13        num = scnr.nextInt();  
14  
15        System.out.println("Enter a sentence: ");  
16        sentence = scnr.nextLine();  
17        num = scnr.nextLine();  
18  
19    }  
20 }  
21 }  
22
```



18. Entrada de datos de tipo entero

- Para leer datos de tipo entero mediante la clase Scanner:
 - `numero = teclado.nextInt();`
- Ejercicio 7

```
1 package pruebas;
2
3 import java.util.Scanner;
4
5 public class NextIntNextLine {
6
7     public static void main(String[] args) {
8         Scanner s = new Scanner(System.in);
9
10        String cadena;
11        int numero;
12
13        System.out.println("Introduce tu nombre: ");
14        cadena = s.nextLine();
15        System.out.println("Introduce tu edad: ");
16        numero = s.nextInt();
17
18        System.out.println("Mi nombre es: " + cadena + " y mi edad: " + numero);
19
20        s.close();
21
22    }
23 }
24
```

```
<terminated> NextIntNextLine [Java Application] /L
Introduce tu nombre:
Manolo
Introduce tu edad:
35
Mi nombre es: Manolo y mi edad: 35
```

19. Entrada de datos de tipo real

- Para leer datos de tipo real mediante la clase Scanner:
 - `numero = teclado.nextDouble();`
- Cuando programamos debemos escribir los números decimales con “.” (3.14) aunque en la consola y en la configuración regional de Windows aparecen con “,” (3,14).

- Ejercicio 8

```
1  import java.util.Scanner;
2
3  //for Scanner
4  public class CelciusToFahrenheit {
5      public static void main(String[] args) {
6          double celsius, fahrenheit;
7          Scanner console = new Scanner(System.in);
8
9          System.out.print("Enter the temperature in degree in Celsius");
10         celsius = Scanner.nextDouble();
11         fahrenheit = (9.0/5.0) * celsius + 32;
12         System.out.printf("%f degrees Celsius is equals to %f degrees F
13     }
14 }
15
```

20. Entrada de datos de tipo carácter

- Para leer datos de tipo carácter mediante la clase Scanner:

- `letra = teclado.next().CharAt(0);`

- Ejercicio 9

```
public static void main(String[] args) {  
  
    Scanner user_input = new Scanner(System.in);  
  
    System.out.println( "Quit Y/N");  
  
    String aString = user_input.next();  
  
    char aChar = aString.charAt(0);  
  
    if (aChar == 'Y') {  
        System.out.println( "OK, BYE BYE");  
    }  
    else {  
        System.out.println( "Not Quitting");  
    }  
}
```

21. Entrada de datos de tipo String (cadena de caracteres)

- Para leer datos de tipo String mediante la clase Scanner:
 - `letra = teclado.next();`
- Ejercicios 10
- Ejercicio 11
 - Realiza la clase Java LeerStringCompuesto que lee un String por teclado y muestra por pantalla el mensaje “El valor de la variable introducida es “.

Scanner Methods

| NAME | USE |
|----------------------------|---|
| <code>nextInt();</code> | Returns the next integer value |
| <code>nextDouble();</code> | Returns the next double value |
| <code>nextFloat();</code> | Returns the next float value |
| <code>nextLong();</code> | Returns the next long value |
| <code>nextShort();</code> | Returns the next short value |
| <code>next();</code> | Returns the next one word String value |
| <code>nextLine();</code> | Returns the next multiple word String value |

```
1 package udl;
2
3 import java.util.Scanner;
4
5 public class Ejer11LeerStringCompuesto {
6
7     public static void main(String[] args) {
8         Scanner teclado = new Scanner(System.in);
9         String frase = "Ejercicio 11";
10
11         System.out.println("Introduzca una frase:");
12         frase = teclado.nextLine(); // guardamos en la variable frase la frase introducida por el usuario aunque tenga espacios
13         System.out.println("El valor de la variable introducida es " + frase);
14
15         teclado.close();
16
17     }
18
19 }
```


22. Forzado de tipo de datos. Casting

- El casting nos permite convertir un dato de un tipo a otro, por ejemplo, que un número real se comporte como un entero.
- Para ello se pone entre paréntesis el tipo nuevo antes del nombre de la variable:

```
double r;
```

```
int e;
```

```
e = (double) r;
```

- Ejercicios 12-20

- Widening Casting(Implicit)

byte → short → int → long → float → double

widening

- Narrowing Casting(Explicitly done)

double → float → long → int → short → byte

Narrowing