Técnicas de programación avanzada

Curso 2023-2024

Tema 1. Objetos y memoria

Dra. Nieves Cubo Mateo (@Nicuma3)
Alejandro Alonso Puig (@mundobot)

Tema 1: Objetos y memoria.

1.1 Características básicas del lenguaje. Primer programa. Compilación y Ejecución. IDE.

1.2 Sentencias de control. Secuencia, selección e iteración.

1.3 Abstracción. Clases, objetos, métodos y atributos.

1.4 Sobrecarga de métodos y encapsulamiento.

Tema 2. Otros conceptos fundamentales de la Programación Orientada a Objetos

2.1 Herencia. Interfaces y clases abstractas. Agregación.

2.2 Polimorfismo.

2.3 Gestión de Excepciones.

2.4 Genericidad y plantillas.

2.5 Utilidades. Entrada y Salida.

2.6 Anotaciones.

Tema 3. Patrones de Diseño.

3.1 Concepto de Patrones de Diseño.

3.2 Patrones de creación.

3.3 Patrones estructurales.

3.4 Patrones de comportamiento.

Tema 4. Programación de Interfaces.

4.1 Interfaces Gráficas de Usuario.

4.2 Gestión de eventos.

Tema 5. Temas Avanzados.

5.1 Concurrencia.

5.2 Inversión de Control. Definición y ejemplos. Inyección de dependencias.

5.3 Expresiones avanzadas del lenguaje.

1.1. Características básicas del lenguaje

1.1 Características básicas del lenguaje.

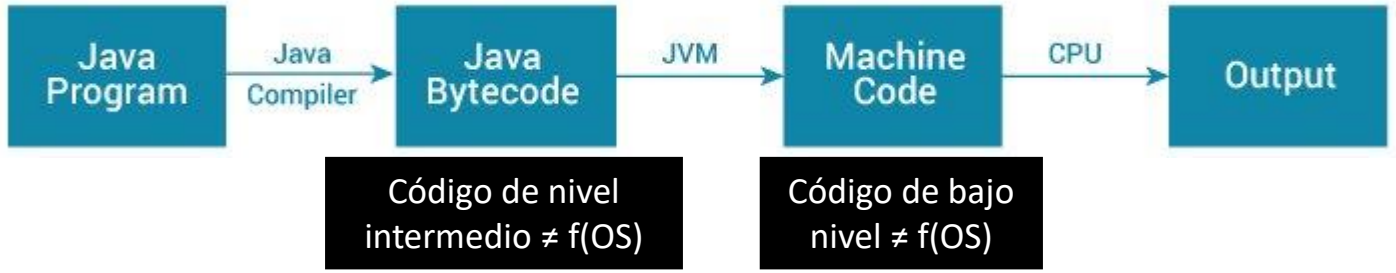
IDE

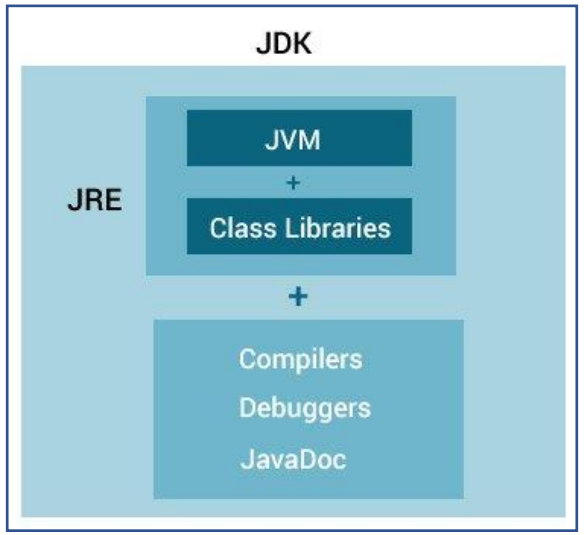
Java es un lenguaje de programación que tiene las siguientes características:

- **Independencia de la plataforma:** las aplicaciones Java se compilan en un código de bytes que se almacena en archivos de clase y se carga en una JVM. Dado que las aplicaciones se ejecutan en una JVM, se pueden ejecutar en muchos sistemas operativos y dispositivos diferentes.
- **Orientado a objetos:** Java es un lenguaje orientado a objetos que toma muchas de las características de C y C ++ y las mejora.
- **Recolección automática de basura:** Java asigna y desasigna memoria automáticamente para que los programas no tengan que cargar con esa tarea.
- **Biblioteca estándar enriquecida:** Java incluye una gran cantidad de objetos prefabricados que se pueden usar para realizar tareas como entrada / salida, redes y manipulación de fechas.)

1.1 Características básicas del lenguaje.

IDE





JDK (Java Development Kit)
JRE (Java Runtime Environment)
JVM (Java Virtual Machine)

Hay tres componentes principales de Java: JVM, JDK y JRE.

JDK o Java Development Kit es donde los desarrolladores escriben su código y lo ejecutan a través de JRE o Java Runtime Environment.

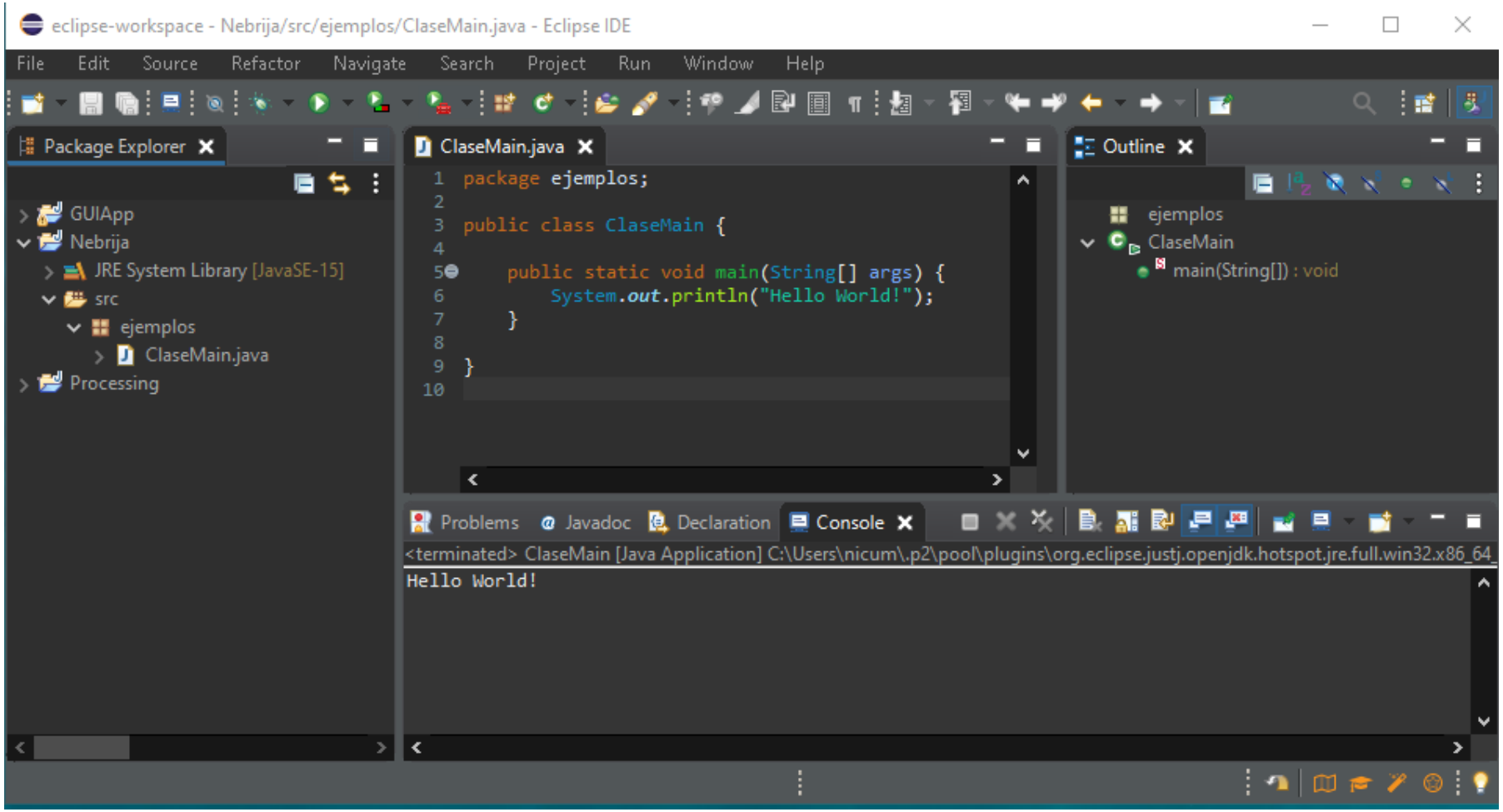
JVM: La máquina abstracta donde se ejecutan los bytecodes de Java. Consiste en un documento de especificación que describe la implementación de JVM, el programa de implementación real y la instancia de JVM (tiempo de ejecución) donde puede ejecutar su programa principal.

JRE: Implementación física (instancia de tiempo de ejecución) de JVM. Contiene los paquetes de la biblioteca y los archivos de soporte que JVM usa para ejecutar un programa.

1.1 Características básicas del lenguaje.

IDE

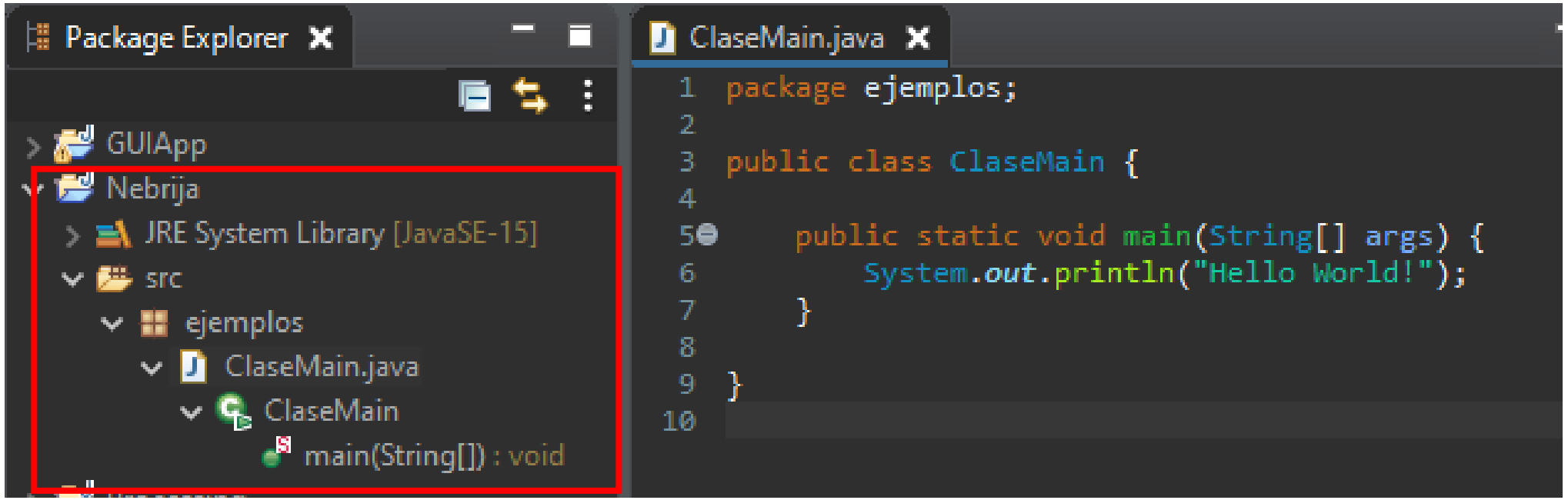
- JDK Installation Guide – Java SE (Standard Edition)
- Eclipse IDE – For Java Developers



1.1 Características básicas del lenguaje.

Primer programa. Compilación y Ejecución. IDE.

1) File > New > Java Project (UpperCase): Nebrija
2) File > New > Package (LowerCase): ejemplos
3) File > New > Class (UpperCase): ClaseMain



1.1 Características básicas del lenguaje.

Primer programa. Compilación y Ejecución.

```java
package ejemplos; //En este paquete se guardan nuestras clases.

public class ClaseMain {

	public static void main(String[] args) {
		System.out.println("Hello World!");
	}
}
```

PROGRAMA PRINCIPAL
Estructura básica

Método principal de la clase (main)

- Java se basa en clases (`class`). Sólo hay clases (métodos, atributos) e interacciones entre ellas.
- Debe haber al menos una defición de clase en el programa
- El programa principal (`main`) es una función/método público de una clase (`public class`). Sólo puede haber 1 `main(String[] args)` en el programa

1.1 Características básicas del lenguaje.

Comentarios y Javadoc Comments & Tags

```
/**
*  Comentario inicial para describir programa
*  Tipo Javadoc – Documentación automática en HTML
*  @author Nombre/Pseudónimo
*/


/**
*  Función calcular edad futura
*  @param years Número de años que pasarán
*  @return age Qué edad tendrás entonces
*/

        public int getFutAge (int years){
                return this.age + years;

}
```

COMENTARIOS

// En una línea

```
/*   En varias
* líneas
* Asteriscos extra opcionales
*/
```

```
/**
* Javadoc comments
* @tags (param, return,
throws e, see #x)
*
*/
```

1.1 Características básicas del lenguaje.

Variables y constantes

```
//VARIABLES

int importe, total, suma;
byte a = 127;

//CONSTANTES

final byte MAYOR_EDAD = 18;
```

tipo	descripción	rango de valores
boolean	valor lógico	true o false
char	carácter unicode (16 bits)	los caracteres internacionales
byte	entero de 8 bits con signo	-128..127
short	entero de 16 bits con signo	-32768..32767
int	entero de 32 bits con signo	-2.147.483.648.. 2.147.483.647
long	entero de 64 bits con signo	aprox. $9.0*10^{18}$
float	nº real de 32 bits	unos 6 dígitos
double	nº real de 64 bits	unos 15 dígitos

1.1 Características básicas del lenguaje.

Operadores

Operadores aritméticos

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
-	operador unario de cambio de signo	`-4`	`-4`
+	Suma	`2.5 + 7.1`	`9.6`
-	Resta	`235.6 - 103.5`	`132.1`
*	Producto	`1.2 * 1.1`	`1.32`
/	División (tanto entera como real)	`0.050 / 0.2` `7 / 2`	`0.25` `3`
%	Resto de la división entera	`20 % 7`	`6`

Operadores aritméticos incrementales

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

++	Incremento
	`i++` primero se utiliza la variable y luego se incrementa su valor
	`++i` primero se incrementa el valor de la variable y luego se utiliza
		`4++`
		`a=5;`
		`b=a++;`
		`a=5;`
		`b=++a;`
			`5`
			
			`a vale 6 y b vale 5`
			
			`a vale 6 y b vale 6`

--	decremento	`4--`	`3`

1.1 Características básicas del lenguaje.

Operadores

Operadores aritméticos combinados

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+=	Suma combinada	a+=b	a=a+b
-=	Resta combinada	a-=b	a=a-b
*=	Producto combinado	a*=b	a=a*b
/=	División combinada	a/=b	a=a/b
%=	Resto combinado	a%=b	a=a%b

Operadores de relación

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	7 == 38	false
!=	distinto que	'a' != 'k'	true
<	menor que	'G' < 'B'	false
>	mayor que	'b' > 'a'	true
<=	menor o igual que	7.5 <= 7.38	false
>=	mayor o igual que	38 >= 7	true

1.1 Características básicas del lenguaje.

Operadores

Operadores lógicos o booleanos

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	`!false` `!(5==5)`	`true` `false`
|	Suma lógica – OR (binario)	`true | false` `(5==5)|(5<4)`	`true` `true`
^	Suma lógica exclusiva – XOR (binario)	`true ^ false` `(5==5)|(5<4)`	`true` `true`
&	Producto lógico – AND (binario)	`true & false` `(5==5)&(5<4)`	`false` `false`
||	Suma lógica con cortocircuito: si el primer operando es `true` entonces el segundo se salta y el resultado es `true`	`true || false` `(5==5)||(5<4)`	`true` `true`
&&	Producto lógico con cortocircuito: si el primer operando es `false` entonces el segundo se salta y el resultado es `false`	`false && true` `(5==5)&&(5<4)`	`false` `false`

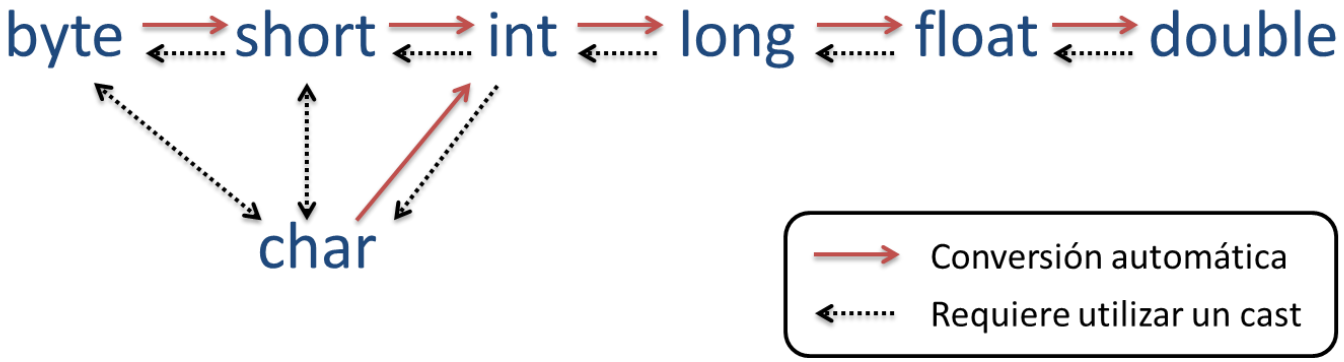
Operador condicional o ternario

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

?:	operador condicional	`a = 4;`
		`b = a == 4 ? a+5 : 6-a;`
		`b = a > 4 ? a*7 : a+8;`
			`b vale 9`
			`b vale 12`

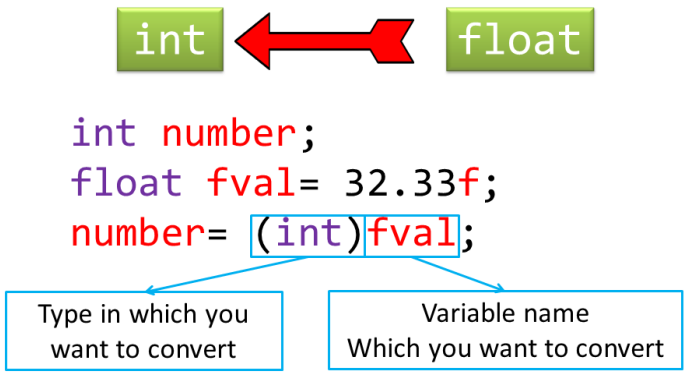
1.1 Características básicas del lenguaje.

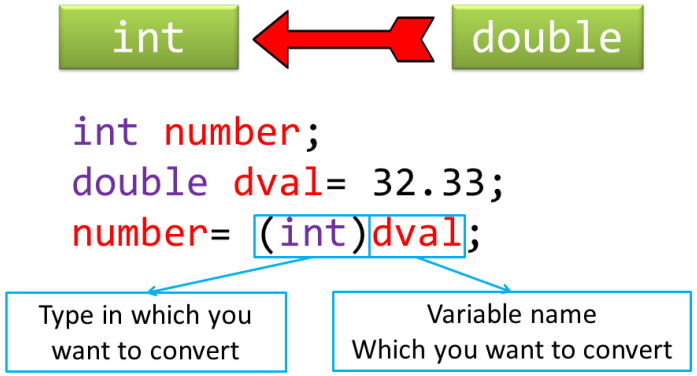
Conversión de tipos

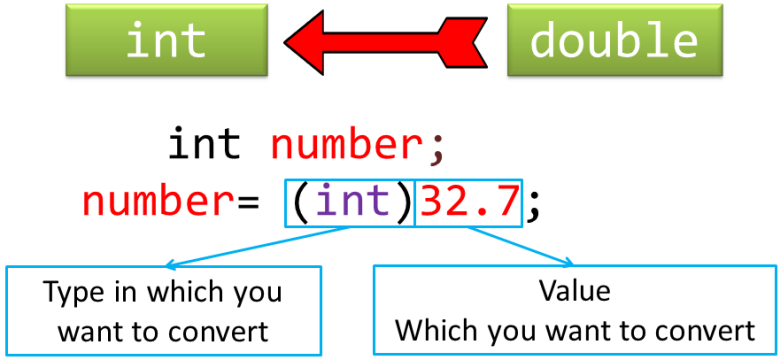


COMENTARIOS

Tenemos dos clases de conversiones de tipo: la promoción (conversión de tipo automática) y el casting (conversión forzada).







1.1 Características básicas del lenguaje.

Entrada y salida de datos

```java
import java.util.Scanner;

public class ClaseMain {

  public static void main(String[] args)
  {
     System.out.print("Enter a number: ");

     // create an object of Scanner
     Scanner input = new Scanner(System.in);

     // take input from the user
     int number = input.nextInt();

     System.out.println("You entered: " + number);

     // closes the scanner
     input.close();

   }
}
```

- Clase `Scanner` de java.util se usa para obtener entradas del teclado, archivos, usuarios, etc.
- `System.out` ***[PrintStream]*** indica salida standard (nuestra pantalla), por consola.
- ***print*** *vs* ***println*** ***(new line)***
  - ***Llamada implícita a toString()***
- `System.in` ***[InputStream]*** indica que la entrada es standard (nuestro teclado).

1.1 Características básicas del lenguaje.

String

La clase **String** está orientada a manejar cadenas de caracteres.

Métodos comunes

cad1.equals(cad2)

Retorna true si el contenido de caracteres del parámetro cad1 es exactamente igual a cad2.

cad1.equalsIgnoreCase(cad2)

No tiene en cuenta mayúsculas y minúsculas (si comparamos 'Ana' y 'ana' el método equalsIgnoreCase retorna true)

cad1.compareTo(cad2)

Este método retorna un 0 si el contenido de cad1 es exactamente igual a cad2. Retorna un valor >0 si el contenido de cad1 es mayor alfabéticamente al parámetro cad2.

cad1.charAt(int pos)

Retorna un caracter del String cad1

cad1.length()

Retorna la cantidad de caracteres almacenados en el String cad1.

cad1.substring(int pos1,int pos2)

Retorna un substring a partir de la posición indicada en el parámetro pos1 hasta la posición pos2 sin incluir dicha posición.

cad1.indexOf(cad2)

Retorna -1 si el String cad2 no está contenido en cad1. En caso de que se encuentre contenido, retorna la posición.

cad1. toUpperCase()

Retorna un String con el contenido convertido todo a mayúsculas.

cad1. toLowerCase()

Retorna un String con el contenido convertido todo a minúsculas.

Fuente: https://www.tutorialesprogramacionya.com/javaya/

1.1 Características básicas del lenguaje.

Arrays

Es usual tener la necesidad de almacenar una lista de valores para después procesarlos.
Una posibilidad es asociar a cada valor una variable, pero esto sería ineficiente y engorroso.
Un **array** es una variable que guarda una lista de valores de mismo tipo.
El array se almacena en posiciones continuas de memoria y el acceso a los elementos se realiza mediante índices.
Para declarar un array se requiere el tipo de dato de los elementos a almacenar y un nombre para el array.

Sintaxis:

```
double[] data; // declara var. array data
```

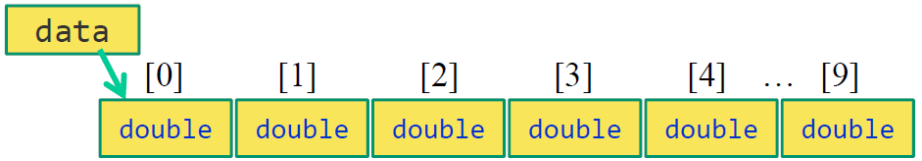
o

```
double data[];
```

Tipo	Corchetes	Nombre Array	punto y coma
double	[ ]	data	;

Después de declarar un array es necesario reservar memoria para todos los elementos.
Se especifica el número de elementos del array a través de un método constructor (new).

Nombre Array		Pal.reservada	Tipo	Tamaño	punto y coma
data	=	new	double	[10]	;



Se puede declarar y crear un array al mismo tiempo

Tipo	Corchetes	Nombre Array		Constructor	Tipo	Tamaño	pyc
double	[]	data	=	new	double	[10]	;

Fuente: https://personales.unican.es/corcuerp/java/

1.1 Características básicas del lenguaje.

Arrays

Se puede declarar y asignar el valor inicial de todos los elementos:

Tipo	Corchetes	Nombre Array		Lista del contenido	pyc
int	[ ]	primos	=	{ 2, 3, 5, 7}	;

Índice va de 0 a tamaño_array − 1

Recorrer arrays (for-each)
Permite acceder a cada elemento del array secuencialmente.

Métodos con arrays

.length → Número de elementos
Arrays.toString(*array*) → Convierte array a string
Arrays.equals(*a1*, *a2*) → Compara dos arraysA
Arrays.sort(*array*) → Ordena un array
Arrays. copyOf(*array*, *n*) → Copia de n primeros
Arrays.copyOfRange(*a,n1,n2*) → Copia rango

```java
String [] nombres = new String [7];
nombres[0] = "Mary";

String [] nombres2 = {"Mary", "Angel", "Joy", "Monica"};

System.out.println(nombres2.length);

for (String item:nombres2) {
        System.out.println(item);
}

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
int x = myNumbers[1][2];
System.out.println(x);
```

Los arrays pueden ser multidimensionales

Los arrays pueden pasarse a funciones

```java
public class prueba {
    static void printThisArray(int[] data) {
        for (int item:data) {
            System.out.println(item);
        }
    }
    public static void main(String[] args) {
        int[] arr = {3,4,5};
        printThisArray (arr);
    }
}
```

Fuente: https://personales.unican.es/corcuerp/java/

Ejercicios

1.1 Características básicas del lenguaje.

Ejercicios

1. Diseñar un programa que pida un número al usuario (por teclado) y a continuación lo muestre
2. Diseñar un programa que calcule el perímetro y el área de una circunferencia, cuyo radio se introducirá por teclado (con decimales)
3. Diseñar un programa que calcule la media aritmética (decimal) de tres valores enteros

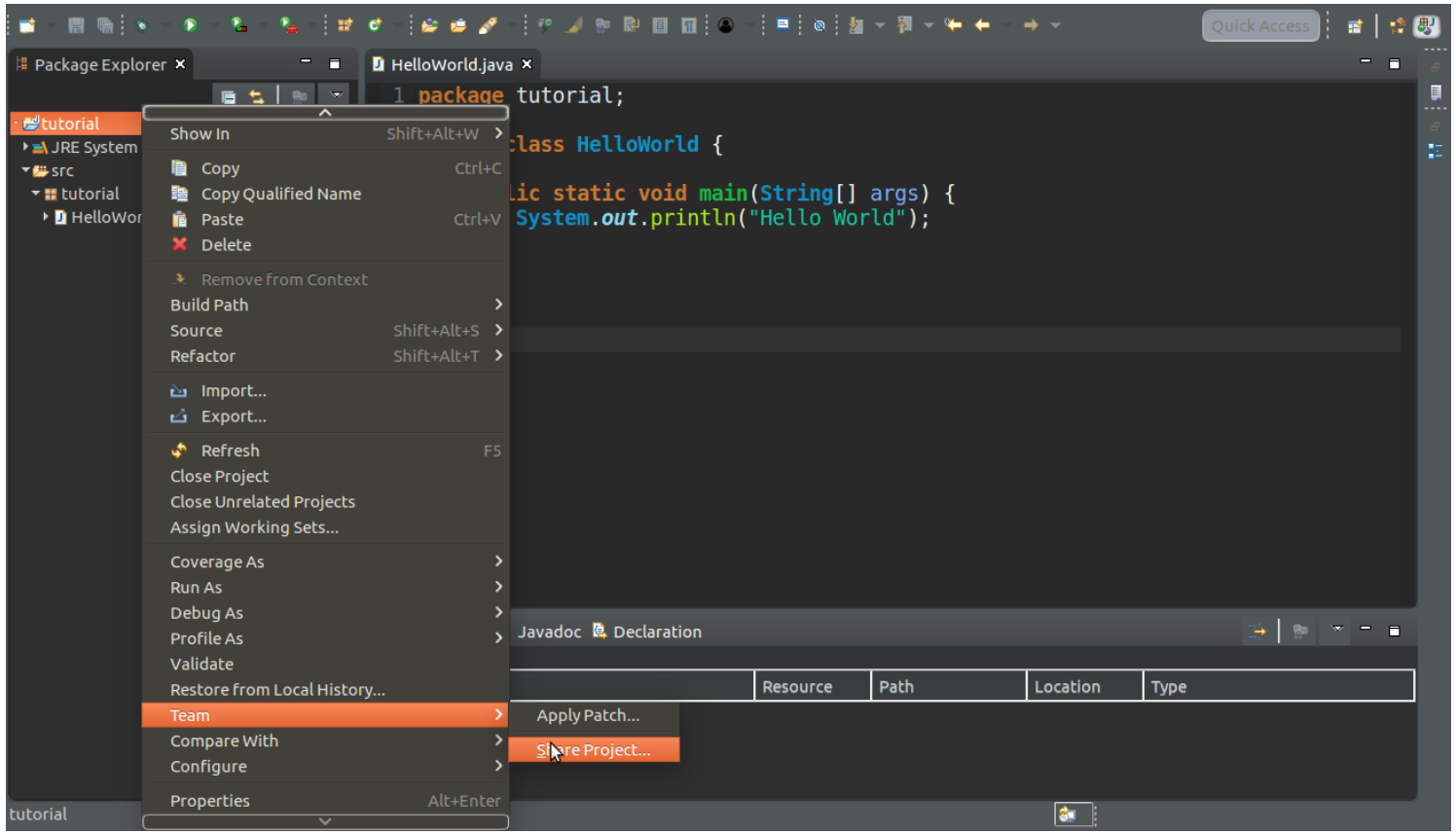
Review. Repositorios (GIT)

Cómo subir mi Proyecto a Github fácilmente

UNIVERSIDAD NEBRIJA

Desde ECLIPSE

Tutorial GIT para ECLIPSE: https://github.com/utnfrrojava/eclipse-git-tutorial

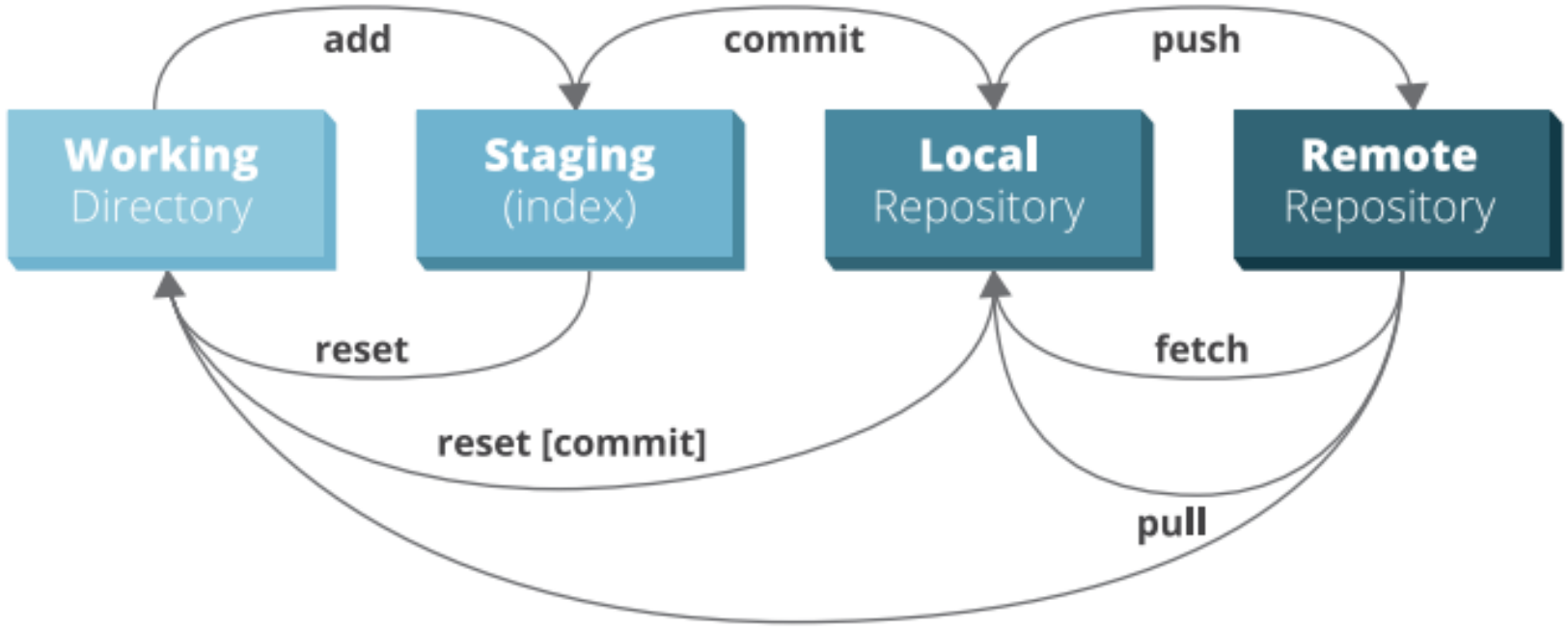


Cómo subir mi Proyecto a Github fácilmente

UNIVERSIDAD NEBRIJA

Desde Consola

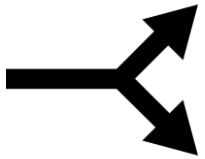
Github desktop: https://desktop.github.com/



1.2. Sentencias de control

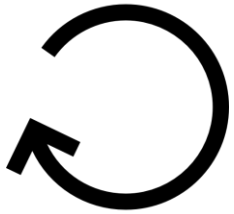
1.2 Sentencias de control.

Secuencia, selección e iteración.



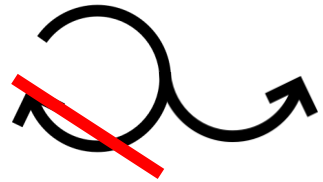
Condicional (selección):

if...else (bifurcación) // switch Statement (varias opciones)



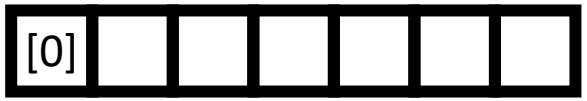
Bucles (iteración/repetición):

- for / for-each Loop (n iteraciones conocido)
- while Loop (condición)



Salir de bucles:

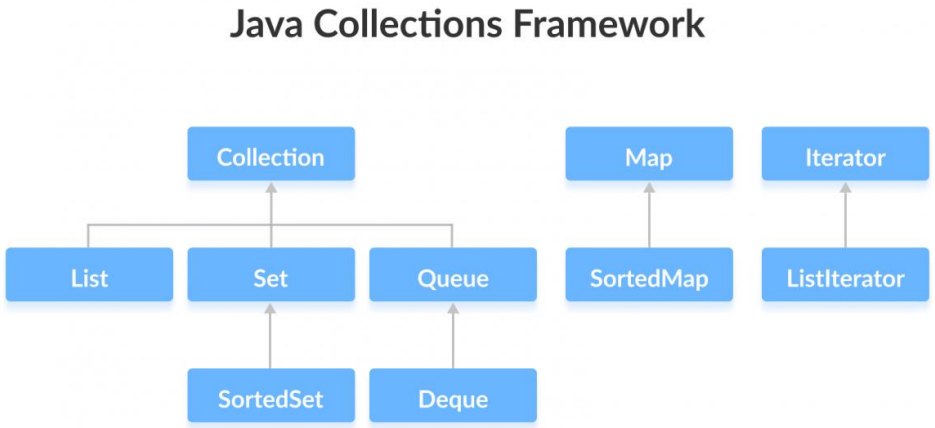
- break
- continue



Elementos donde son importantes los bucles:

- arrays, listas, listas enlazadas…

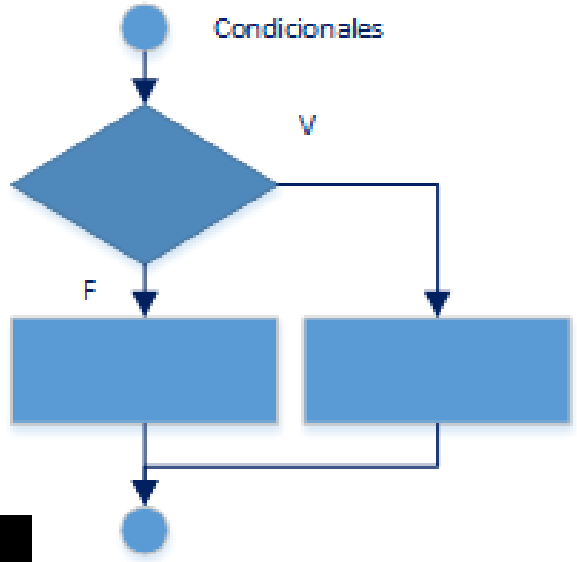
Otras estructuras de datos avanzadas:



1.2 Sentencias de control.

if...else

```
if (condition) {
        // code in if block
}
 else {
        // code in else block
}
```




```
if (condition1) {
        // code
} else if(condition2) {
                // code
        } else if (condition3) {
                // code
                } . . else {
                        // code
                }
```

```
if (x<100) {
    System.out.println("n<100");
    if (x<75) {
        System.out.println("n<75");
        if (x<50) {
            System.out.println("n<50");
        }
    }
}
```

```
result = (condition) ? return_if_True : return_if_False ;
```

OPERADOR TERNARIO

"inline if statement"

```
int x=7, y=5;
int mayor=(x>y)?x:y;
```

1.1 Características básicas del lenguaje.

Operadores booleanos (Repaso)

Operadores de relación

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	`7 == 38`	`false`
!=	distinto que	`'a' != 'k'`	`true`
<	menor que	`'G' < 'B'`	`false`
>	mayor que	`'b' > 'a'`	`true`
<=	menor o igual que	`7.5 <= 7.38`	`false`
>=	mayor o igual que	`38 >= 7`	`true`

Operadores lógicos o booleanos

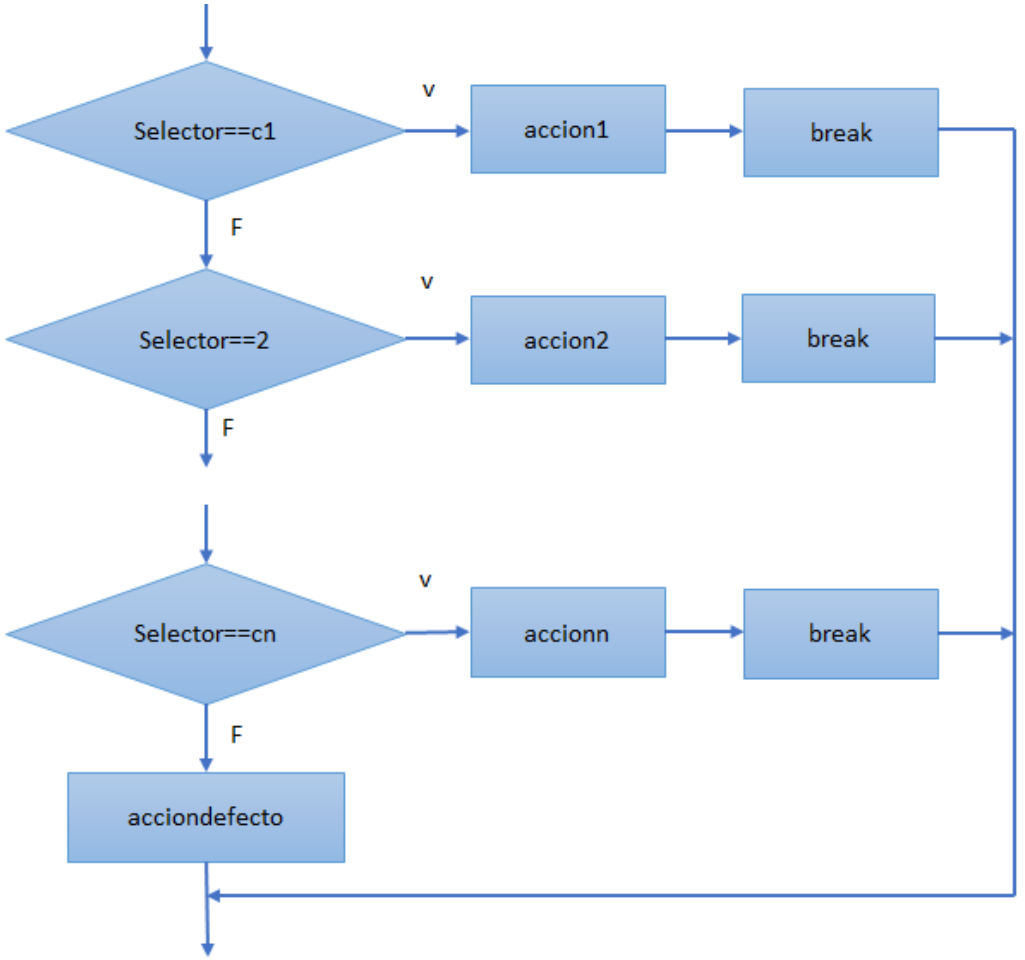
Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	`!false` `!(5==5)`	`true` `false`
|	Suma lógica – OR (binario)	`true | false` `(5==5)|(5<4)`	`true` `true`
^	Suma lógica exclusiva – XOR (binario)	`true ^ false` `(5==5)|(5<4)`	`true` `true`
&	Producto lógico – AND (binario)	`true & false` `(5==5)&(5<4)`	`false` `false`
||	Suma lógica con cortocircuito: si el primer operando es `true` entonces el segundo se salta y el resultado es `true`	`true || false` `(5==5)||(5<4)`	`true` `true`
&&	Producto lógico con cortocircuito: si el primer operando es `false` entonces el segundo se salta y el resultado es `false`	`false && true` `(5==5)&&(5<4)`	`false` `false`

1.2 Sentencias de control.

switch

```
switch (expression) {
  case value1:
    // code
    break;
  case value2:
    // code
    break;
  …
    …
  default:
    // default statements
}
```

- Si no se añade el **'break',** el resto de casos (tras elegir el valor verdadero) se ejecutarán.
- **'default':** Cuando no se ha cumplido ninguna opción anterior



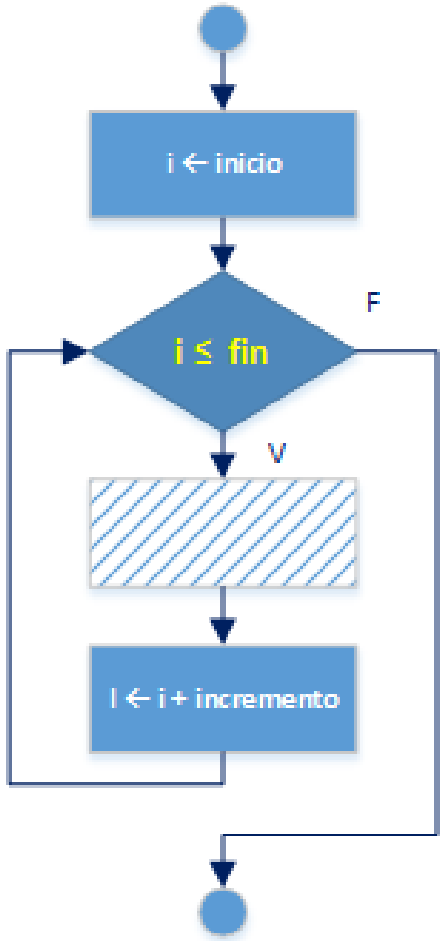
1.2 Sentencias de control.

for

Syntax

```
for (initialization; condition; increment) {
  // statement(s);
}
```

```
class Main {
    public static void main(String[] args) {
          int n = 5;

          // for loop
          for (int i = 1; i <= n; ++i) {
                System.out.println("n is " + n );
           }
      }
}
```



1.2 Sentencias de control.

for-each

❖ FOR each…

Equivalente a (para recorrer arrays y colecciones):

```
for (int i = 0; i < numbers.length; ++ i)
```

```
for (dataType item : array) {
 …
}
```

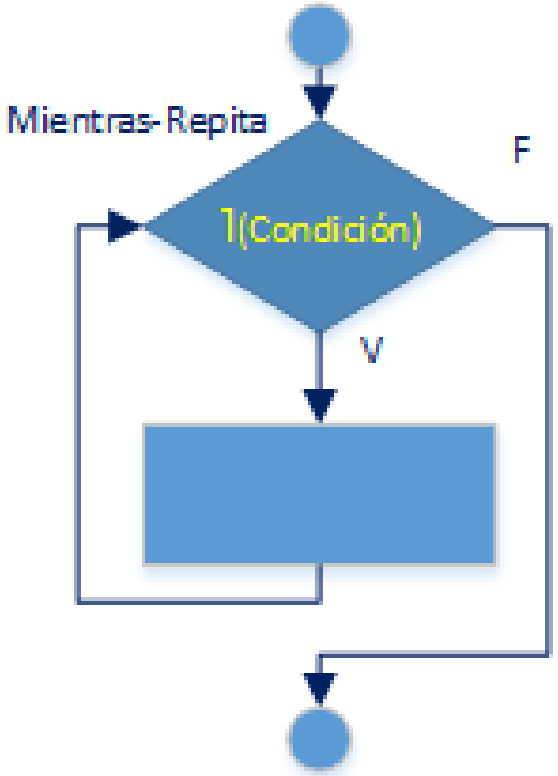
```
// create an array
int[] numbers = {3, 7, 5, -5};

// iterating through the array
for (int i: numbers) {
   System.out.println(i);
}
```

1.2 Sentencias de control.

while

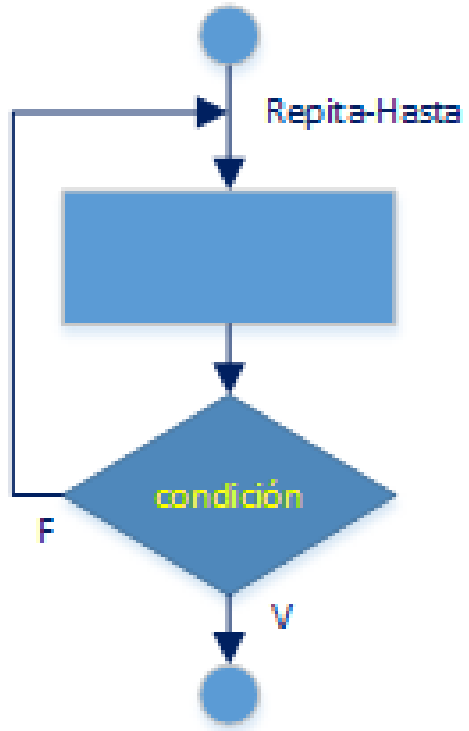
```
while (testExpression) {
    // body of loop
}
```

```
while(i <= n) {
  System.out.println(i);
  i++;
}
```



1.2 Sentencias de control.

do…while




```
do {
    // body of loop
} while (testExpression)
```

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

do…while() al menos se ejecuta una vez

1.2 Sentencias de control.

break & continue

```
while(true) {
  System.out.println(i);
  i++;

  if (i == 5) {
      i++;
      continue;
  }
}
```

- **Elementos terminación bucles**:
  - break: sale del bucle
  - continue: sale de la iteración en curso
- **labeled break/continue:**
  - Útil para bucles anidados (Avanzado)

```
while(i<10) {
   if (i == 5) {

      break;
   }
   System.out.println(i);
   i++;
}
```

¿i?

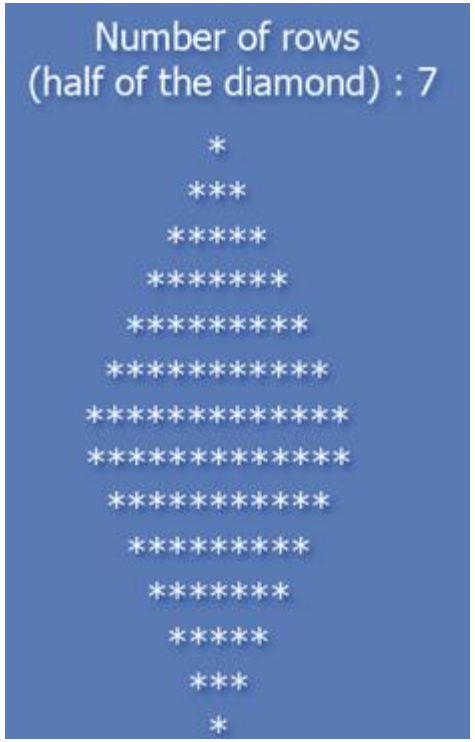
```
while(i<10) {
   if (i == 5) {
        i++;
        continue;
   }
   System.out.println(i);
   i++;
}
```

Ejercicios

1.2 Sentencias de control

Ejercicios

1. Escriba un programa Java que requiera que el usuario ingrese un solo carácter del alfabeto. Imprima Vocal o Consonante, dependiendo de la entrada del usuario. Si la entrada del usuario no es una letra (entre a y z o A y Z), o es una cadena de longitud > 1, imprime un mensaje de error. (if)
2. Escriba un programa en Java que pida un número entero y muestre un patrón de diamante culla altura sea el doble que dicho número. Ej para 7 (for)
3. Escriba un programa Java para encontrar el número de días en un mes. Como datos de entrada pedirá el número de año y el número de mes. (switch)
4. Subir a GIT



1.2 Sentencias de control

Ejercicios

Mira este Código. En cada caso, crees que dará true o false? Pruébalo. ¿entiendes qué pasa en cada caso?

```
public static void main(String[] args)
{
String s1 = "HELLO";
String s2 = "HELLO"; //String constant pool
String s3 = new String("HELLO");
String s4 = s1;

System.out.println("s1 == s2, is \t" + (s1 == s2));
System.out.println("s1 == s3, is \t" + (s1 == s3));
System.out.println("s1 == s4, is \t" + (s1 == s4));

System.out.println("s1 equals s2, is " + (s1.equals(s2)));
System.out.println("s1 equals s3, is " + (s1.equals(s3)));
System.out.println("s1 equals s4, is " + (s1.equals(s4)));

System.out.println("Adress s1: " + System.identityHashCode(s1));
System.out.println("Adress s2: " + System.identityHashCode(s2));
System.out.println("Adress s3: " + System.identityHashCode(s3));
System.out.println("Adress s4: " + System.identityHashCode(s4));


int a = 5;
int b = 5;
int c = a;

System.out.println("\na == b, is \t" + (a == b));
System.out.println("a == c, is \t" + (a == c));
//System.out.println(a.equals(b)); ERROR: can not invoke that
method for primitive types
```

(continuación código…)

```
System.out.println("Adress a: " + System.identityHashCode(a));
System.out.println("Adress b: " + System.identityHashCode(b));
System.out.println("Adress c: " + System.identityHashCode(c));

Integer d = 5;
Integer e = 5; //Integer constant pool
Integer f = new Integer(5); //deprecated
Integer g = d;


System.out.println("\nd == e, is \t" + (d == e));
System.out.println("d == f, is \t" + (d == f));
System.out.println("d == g, is \t" + (d == g));
System.out.println("d equals e, is \t" + (d.equals(e)));
System.out.println("d equals f, is \t" + (d.equals(f)));
System.out.println("d equals g, is \t" + (d.equals(g)));

System.out.println("Adress d: " + System.identityHashCode(d));
System.out.println("Adress e: " + System.identityHashCode(e));
System.out.println("Adress f: " + System.identityHashCode(f));
System.out.println("Adress g: " + System.identityHashCode(g));
}
```

1.3 Abstracción. Clases, objetos, métodos y atributos.

1.3 Abstracción. Clases, objetos, métodos y atributos

Clases

Concepto
La programación orientada a objetos se basa en la programación de **clases**; a diferencia de la programación estructurada, que está centrada en las funciones.

Una **clase** es un molde del que luego se pueden crear múltiples objetos, con similares características.

Componentes básicos
Una clase es una plantilla (molde), que define **atributos** (variables) y **métodos** (funciones)

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

La estructura de una clase es:

```
class [nombre de la clase] {
  [atributos o variables de la clase]
  [métodos o funciones de la clase]
  [main]
}
```

Fuente: https://www.tutorialesprogramacionya.com/javaya/

1.3 Abstracción. Clases, objetos, métodos y atributos

Clases

```
import java.util.Scanner;
public class Persona {
    private String nombre;
    private int edad;

    public void inicializar() {
        Scanner teclado=new Scanner(System.in);
        System.out.print("Ingrese nombre:");
        nombre=teclado.next();
        System.out.print("Ingrese edad:");
        edad=teclado.nextInt();
    }

    public void imprimir() {
        System.out.println("Nombre:"+nombre);
        System.out.println("Edad:"+edad);
    }

    public void esMayorEdad() {
        if (edad>=18) {
            System.out.print(nombre+" es mayor de edad.");
        } else {
            System.out.print(nombre+" no es mayor de edad.");
        }
    }

    public static void main(String[] ar) {
        Persona persona1;
        persona1=new Persona();
        persona1.inicializar();
        persona1.imprimir();
        persona1.esMayorEdad();
    }
}
```

Ejemplo

Atributos:

nombre
edad

Métodos:

inicializar()
imprimir()
esMayorEdad()

Modificadores

private → Solo visible en propia clase (no en instancias)
ninguno → Visible en la propia clase, vecinas e instancias
public → Visible en clases vecinas, externas e instancias
static → valor fijo para todas las instancias

Fuente: https://www.tutorialesprogramacionya.com/javaya/

1.3 Abstracción. Clases, objetos, métodos y atributos

Clases

```
import java.util.Scanner;
public class Persona {
    private String nombre;
    private int edad;

    public void inicializar() {
        Scanner teclado=new Scanner(System.in);
        System.out.print("Ingrese nombre:");
        nombre=teclado.next();
        System.out.print("Ingrese edad:");
        edad=teclado.nextInt();
    }

    public void imprimir() {
        System.out.println("Nombre:"+nombre);
        System.out.println("Edad:"+edad);
    }

    public void esMayorEdad() {
        if (edad>=18) {
            System.out.print(nombre+" es mayor de edad.");
        } else {
            System.out.print(nombre+" no es mayor de edad.");
        }
    }

    public static void main(String[] ar) {
        Persona persona1;
        persona1=new Persona();
        persona1.inicializar();
        persona1.imprimir();
        persona1.esMayorEdad();
    }
}
```

Objetos o instancias

Debemos crear una clase antes de poder crear objetos (instancias) de esa clase.

Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

```
Persona estudiante;
estudiante = new Persona();

estudiante.inicializar();
estudiante.imprimir();
estudiante.esMayorEdad();
```

~~estudiante.nombre = “Javier”;~~
~~estudiante.edad = 18;~~

NO (Son *private*)

Fuente: https://www.tutorialesprogramacionya.com/javaya/

1.3 Abstracción. Clases, objetos, métodos y atributos

Clases

Constructores

Un constructor es un método especial que debe tener el mismo nombre que la clase. Se ejecuta inmediatamente después de crear el objeto (instancia). Puede usarse para inicializar valores de atributos y permite pasarle valores.

Es posible hacer constructores sobrecargados igual que métodos sobrecargados. (Se verá más adelante)

Persona estudiante;
estudiante = new Persona("Javier", 18);

```
Persona (String nombre, int edad)
{
    this.nombre = nombre;
    this.edad = edad;
}
```

Getters y Setters

Los métodos get y set se utilizan habitualmente para poder almacenar y recuperar datos de variables privadas desde un ámbito público, pero con control.

```
class Dog{
  //Attributes
  private String  name;
  private int age =0;

  //Methods
  public void set_name(String name) {
    this.name = name;
  }

  public String get_name() {
    return this.name;
  }

  public void set_age(int age) {
    if(age >=0 )this.age = age;
  }

  public int get_age() {
    return this.age;
  }
}
```

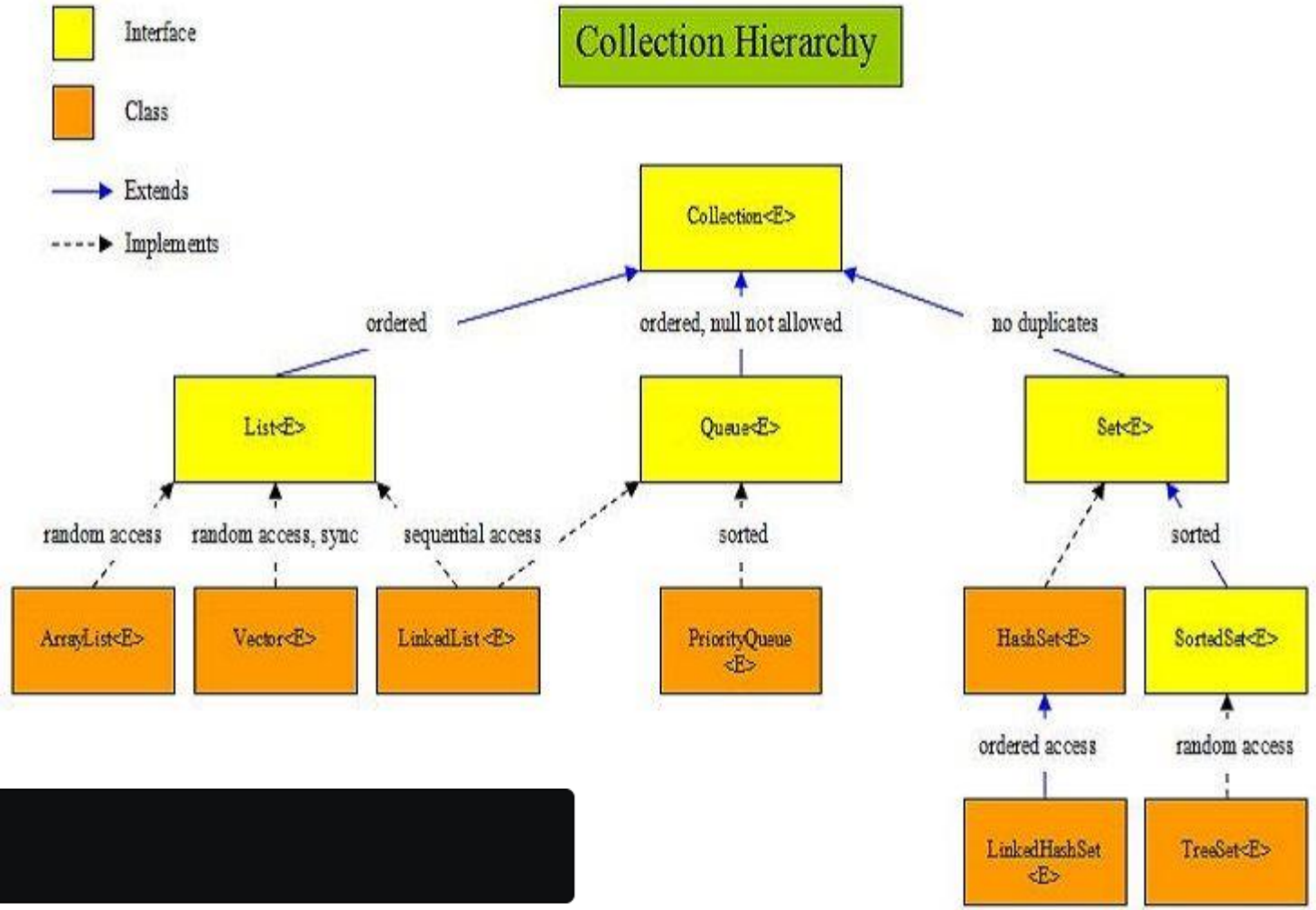
1.3 Abstracción. Clases, objetos, métodos y atributos

Listas

Las listas heredan de la interface Collection.

Esta interface o los objetos de esta, representan una colección ordenada de elementos, en la cual se tiene un control absoluto y preciso del lugar en el que se quiere insertar.

La interface List se encuentra en el paquete java.util, algunas de las clases que implementa esta interface son ArrayList, LinkedList, Vector.



Crear un ArrayList

```
List<String> lista = new ArrayList<>();
```

Fuente:https://blog.codmind.com/listas-en-java/

1.3 Abstracción. Clases, objetos, métodos y atributos

Listas

Cuando se escribe un programa que colecciona datos, no siempre se sabe cuántos valores se tendrá. En tal caso una lista ofrece ventajas:

- La lista puede crecer o disminuir como sea necesario.
- La clase ArrayList ofrece métodos para las operaciones comunes, tal como insertar o eliminar elementos.

Entre < > se declara el tipo de la lista. Debe ser una clase. No permite tipos primitivo (int, double,..) pero si "wrappers"

Métodos útiles de ArrayList:

– add: añade un elemento
– get: retorna un elemento
– remove: elimina un elemento
– set: cambia un elemento
– size(): longitud del array

```
ArrayList<String> names = new ArrayList<String>();

names.add("Ann");
names.add("Cindy");

System.out.println(names);

names.add(1,"Bob");

names.remove(0);

names.set(0, "Bill");

String name = names.get(i);
String last = names.get(names.size() - 1);
```

- Para hacer una copia, pasar la referencia del ArrayList original al constructor del nuevo:

referencia

```
ArrayList<String> newNames = new ArrayList<String>(names);
```

Fuente: https://personales.unican.es/corcuerp/java/

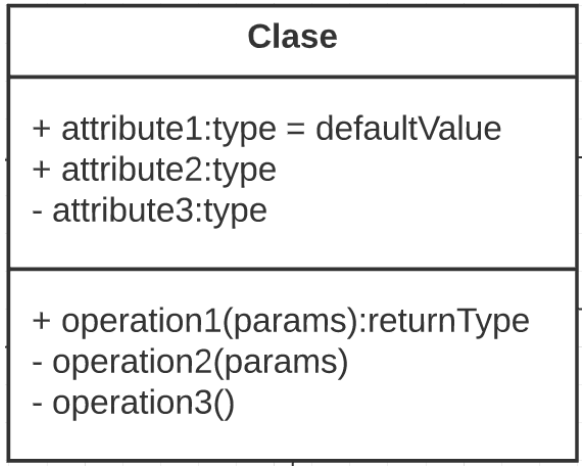
Review. Diagramas UML

1.3 Abstracción. Clases, objetos, métodos y atributos

Repaso diagramas UML

Concepto

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) puede ayudarte a modelar sistemas de diversas formas. Uno de los tipos más populares en el UML es el diagrama de clases. Como las clases son los componentes básicos de los objetos, los diagramas de clases son los componentes básicos del UML. Los diversos componentes en un diagrama de clases pueden representar las clases que se programarán en realidad, los objetos principales o la interacción entre clases y objetos.



Componentes básicos

En UML, una clase representa un objeto o un conjunto de objetos que comparte una estructura y un comportamiento comunes. Se representan con un rectángulo que incluye filas del nombre de la clase, sus atributos y sus operaciones. Al dibujar una clase en un diagrama de clases, solo se debe cumplimentar la fila superior. Las otras son opcionales y se usan si deseas agregar más detalles.

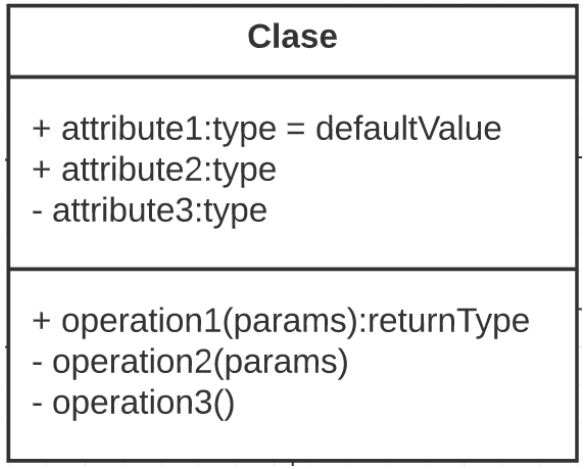
- Sección superior: Contiene el nombre de la clase. Esta sección siempre es necesaria, ya sea que estés hablando del clasificador o de un objeto.
- Sección central: Contiene los atributos de la clase. Usa esta sección para describir cualidades de la clase. Esto solo es necesario al describir una instancia específica de una clase.
- Sección inferior: Incluye operaciones de clases (métodos). Esto está organizado en un formato de lista. Cada operación requiere su propia línea. Las operaciones describen cómo una clase puede interactuar con los datos.

1.3 Abstracción. Clases, objetos, métodos y atributos

Repaso diagramas UML

Modificadores de acceso a miembros

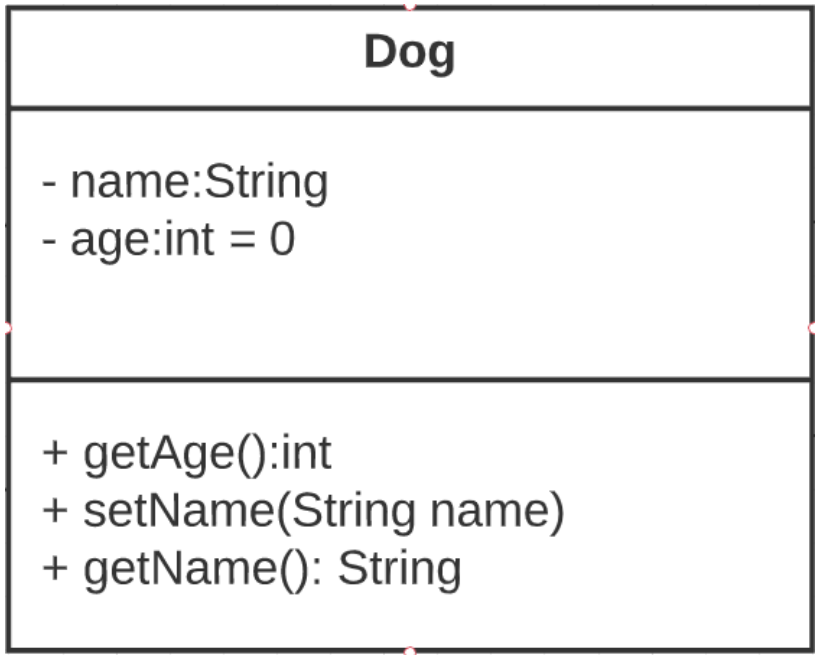
Todas las clases poseen diferentes niveles de acceso en función del modificador de acceso (visibilidad). A continuación, te mostramos los niveles de acceso con sus símbolos correspondientes:



- **Público/Public(+):** cualquiera tiene acceso
- **Privado/Private(-):** únicamente la clase puede acceder a la propiedad o método.
- **Protegido/Protected(#):** las clases del mismo paquete y que heredan de la clase pueden acceder a la propiedad o método.
- **Paquete / Package private (~)** (valor por defecto si no se indica ninguno): solo las clases en el mismo paquete pueden acceder a la propiedad o método.
- **Derivado/Derived property (/):** producido o calculado a partir del valor de otro atributo o método
- **Estático/ static (subrayado):** Se puede acceder directamente a una variable estática por el nombre de clase y no necesita ningún objeto

1.3 Abstracción. Clases, objetos, métodos y atributos

Repaso diagramas UML



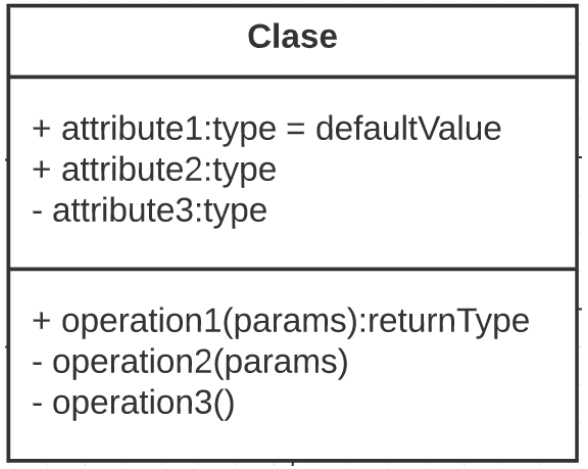

```java
class Dog{
  //Attributes
  private String  name;
  private int age =0;

  //Methods
  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return this.name;
  }

public int getAge() {
    return this.age;
  }
}
```

1.3 Abstracción. Clases, objetos, métodos y atributos

Repaso diagramas UML

Modificadores de acceso a miembros

Todas las clases poseen diferentes niveles de acceso en función del modificador de acceso (visibilidad). A continuación, te mostramos los niveles de acceso con sus símbolos correspondientes:



- **Público/Public(+):** cualquiera tiene acceso
- **Privado/Private(-):** únicamente la clase puede acceder a la propiedad o método.
- **Protegido/Protected(#):** las clases del mismo paquete y que heredan de la clase pueden acceder a la propiedad o método.
- **Paquete / Package private (~)** (valor por defecto si no se indica ninguno): solo las clases en el mismo paquete pueden acceder a la propiedad o método.
- **Derivado/Derived property (/):** producido o calculado a partir del valor de otro atributo o método
- **Estático/ static (subrayado):** Se puede acceder directamente a una variable estática por el nombre de clase y no necesita ningún objeto

1.3 Abstracción. Clases, objetos, métodos y atributos

Repaso diagramas UML

Relaciones

Una relación es un término general que abarca los tipos específicos de conexiones lógicas que se pueden encontrar en los diagramas de clases y objetos. UML presenta las siguientes relaciones:

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

Lo iremos viendo con más detalle durante el curso

1.3 Abstracción. Clases, objetos, métodos y atributos

Repaso diagramas UML

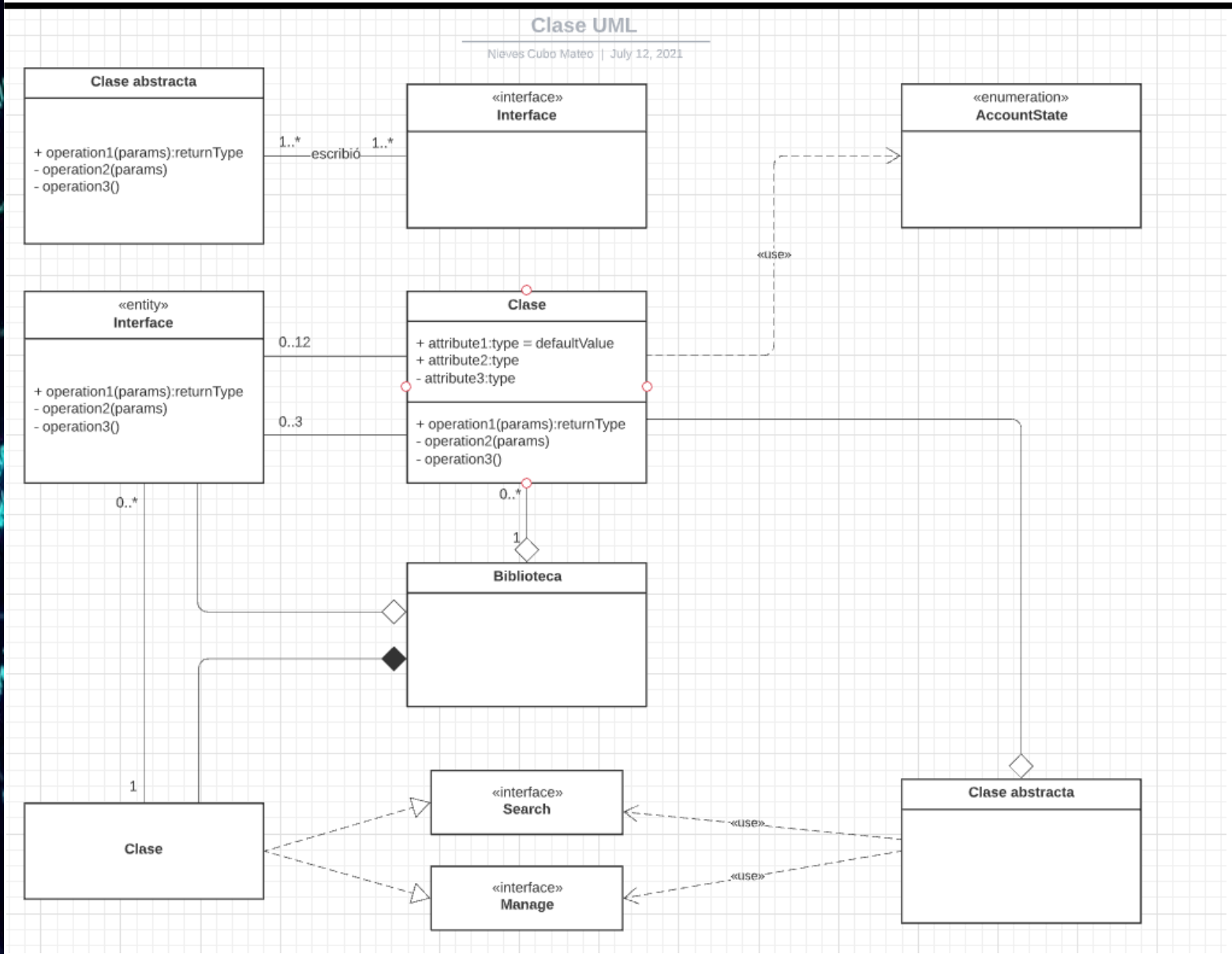


Diagrama de clases. UML.

Plataforma para crear diagramas (Lucidachart):

https://www.lucidchart.com/

Ejercicios

1.3 Abstracción. Clases, objetos, métodos y atributos

Ejercicios

1. Escriba un programa Java para crear una clase llamada "Persona" con un atributo de nombre y edad. Cree dos instancias de la clase "Persona", configure sus atributos usando el constructor e imprima su nombre y edad utilizando getters.
2. Escriba un programa Java para crear una clase llamada "Empleado" con un nombre, cargo y atributos de salario, y métodos para calcular y actualizar el salario.
3. Escriba un programa Java con una clase llamada “*Book*" con atributos para *title*, *author* e *ISBN*, y métodos para agregar y eliminar libros de una colección. (Pista: Utiliza ArrayList para almacenar los libros)

1.4 Clases. Sobrecarga y más

1.4 Sobrecarga y más

Sobrecarga de métodos

```
class Demo
{
  void multiply(int a, int b)
  {
    System.out.printIn("Result is"+(a*b)) ;
  }

  void multiply(int a, int b, int c)
  {
    System.out.printIn("Result is"+(a*b*c));
  }

  public static void main(String[] args)
  {
    Demo obj = new Demo();
    obj.multiply(8,5);
    obj.multiply(4,6,2);
  }
}
```

- Mismo método (nombre) dentro de una clase, pero: diferentes argumentos, en diferente orden.

1.4 Sobrecarga y más

Sobrecarga de constructores

```java
public class AverageClass {

    private double media = 0;

    public AverageClass (double a, double b){
        media = (a+b)/2;
    }

    public AverageClass (double a, double b, double c){
        media =(a+b+c)/3;
    }

    public double getMedia () {
        return (media);
    }
}
```

```java
package Sobrecarga_constructor;


public class ClaseMain {

    public static void main(String[] args) {

        AverageClass Media1 = new AverageClass(4.2, 2.5);
        AverageClass Media2 = new AverageClass(4,8.7,3);
        System.out.println(Media1.getMedia());
        System.out.println(Media2.getMedia());
    }
}
```

- Similar con sobrecarga de constructores.

1.4 Sobrecarga y más

Enumerados

- Similar a una clase, pero no puede ser instanciado y no puede extender otras clases (aunque sí implementar interfaces)
- Para representar valores constantes. Los elementos son públicos, estáticos y finales.
- Nos permite asegurar que los parámetros pasados a un método están dentro de una lista de posibilidades

```
enum Race{
   ELF,
   ORC,
   DWARF
}
```

```
public class ClaseMain {

    public static void setCharRace(Race race) {
            System.out.println(race);
        }

    public static void main(String[] args) {
        setCharRace(Race.DWARF);
    }
}
```

1.4 Sobrecarga y más

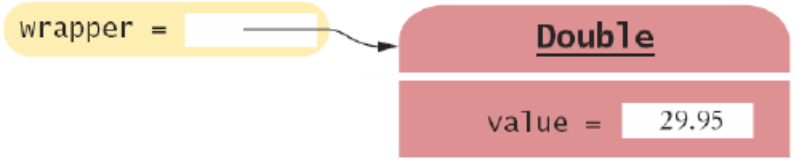
Wrappers y auto-boxing

Java ofrece las clases wrapper para tipos primitivos.

- Las conversiones son automáticas usando ***auto-boxing***
- Tipo primitivo a clase Wrapper

```
double x = 19.95;
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);            // boxing
data.add(x);                // boxing
double x = data.get(0);     // unboxing
```

```
double x = 29.95;
Double wrapper;
wrapper = x;  // boxing
```



- Clase Wrapper a tipo primitivo

```
double x;
Double wrapper = 29.95;
x = wrapper;  // unboxing
```

Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Fuente: https://personales.unican.es/corcuerp/java/

1.4 Sobrecarga y más

Wrappers y auto-boxing

primitivo → Wrapper

```
int x = 20;
Integer y = new Integer(x);
```

ò

```
Integer y = Integer.valueOf(x);
```

(idem para demás tipos) – Se hace un nuevo valor con el constructor del Wrapper o se invoca al valueOf.

Wrapper → primitive

```
Integer y = Integer.valueOf(34);
int x = y.intValue();
```

(idem para demás tipos) – Se invoca a intValue, doubleValue, booleanValue, etc.

Arrays: ArrayUtils.toPrimitive(Wrapper [] array) – Hay que importar

Ej.:

```
Double [] dW;
double[ ] d = ArrayUtils.toPrimitive(dW);
```

1.4 Sobrecarga y más

Paquetes

Concepto

Un **paquete** es un conjunto de clases que permite aislarlas de otras **clases externas**.

Estas clases comparten una temática o funcionalidad similar.

Evitar conflictos de nombres entre clases (diferentes paquetes pueden tener clases que se llamen igual, pero para acceder a ellas hay que poner a qué paquete pertenecen).

Las clases de un mismo paquete se denominan **clases vecinas**

Una clase puede acceder a todas las clases públicas que están en su mismo paquete, sin necesidad de indicar el nombre de dicho paquete. Si se desea acceder a otras que no están en su mismo paquete, se puede importar éste o indicar el nombre completo. Ejemplo: import java.awt.image.BufferStrategy;

```
package Pruebas;

public class ClaseMain {

    public static void setCharRace(Race race) {
            System.out.println(race);
        }

    public static void main(String[] args) {
        setCharRace(Race.DWARF);
    }
}
```

Ejercicios

1.4 Sobrecarga y más

Ejemplo

https://github.com/aalonsopuig/Ejemplos_Java.git

```java
package Sobrecarga_constructor;

public class AverageClass {

    private double media = 0;

    public AverageClass (double a, double b){
        media = (a+b)/2;
    }

    public AverageClass (double a, double b, double c){
        media =(a+b+c)/3;
    }

    public String getMedia () {
        return ("(Desde paquete Sobrecarga_constructor) " + media);
    }
}
```

```java
package Sobrecargas_y_enumerados;

public enum Semana {
    LUNES,
    MARTES,
    MIÉRCOLES,
    JUEVES,
    VIERNES,
    SÁBADO,
    DOMINGO
}
```

```java
package Sobrecargas_y_enumerados;

public class AverageClass {

    private double media = 0;

    public AverageClass (double [] valores){
        for (double num:valores) {
            media+=num;
        }
        media=media/valores.length;
    }

    public AverageClass (double a, double b){
        media = (a+b)/2;
    }

    public String getMedia () {
        return ("(Desde paquete Sobrecargas_y_enumerados) " + media);
    }
}
```

```java
package Sobrecargas_y_enumerados;

//import Sobrecarga_constructor.AverageClass;

public class ClaseMain {

    public static void main(String[] args) {

        //PARTE DEL CÓDIGO QUE CALCULA MEDIAS
        Sobrecarga_constructor.AverageClass Media1 = new Sobrecarga_constructor.AverageClass(4,2);
        Sobrecarga_constructor.AverageClass Media2 = new Sobrecarga_constructor.AverageClass(4,8,9);

        double [] valores = {4,8,9,6};
        AverageClass Media3 = new AverageClass(valores);
        AverageClass Media4 = new AverageClass(4,2);

        System.out.println(Media1.getMedia()); //Corresponde al paquete "Sobrecarga_constructor"
        System.out.println(Media2.getMedia()); //Corresponde al paquete "Sobrecarga_constructor"
        System.out.println(Media3.getMedia()); //Corresponde al paquete "Sobrecargas_y_enumerados"
        System.out.println(Media4.getMedia()); //Corresponde al paquete "Sobrecargas_y_enumerados"

        //PARTE DEL CÓDIGO QUE MUESTRA EL DÍA DE LA SEMANA
        Semana claseIngles;
        claseIngles = Semana.MARTES;
        if ((claseIngles != Semana.SÁBADO) && (claseIngles != Semana.DOMINGO)){
            System.out.println("Día de trabajo");
        }

    }
}
```

1.4 Sobrecarga y más

Ejercicios

1. Escribe un programa Java con dos paquetes, con clases y métodos de igual nombre, que incluyan sobrecarga de constructores y sobrecarga de métodos, así como Enumerados. Siéntete libre de hacer un programa demostrador todo lo complejo que desees.

Bibliografía

- Deitel, H. M. & Deitel, P. J.(2008). *Java: como programar*. Pearson education. Séptima edición.
- Sumérgete en los patrones de diseño. V2021-1.7. Alexander Shvets. https://refactoring.guru/es/design-patterns/book

  Versión online: https://refactoring.guru/es/design-patterns/catalog
- The Java tutorials. https://docs.oracle.com/javase/tutorial/
- Páginas de apoyo para la sintaxis, bibliotecas, etc.:
  - https://www.sololearn.com
  - https://www.w3schools.com/java/default.asp
  - https://docstore.mik.ua/orelly/java-ent/jnut/index.htm

Técnicas de Programación Avanzada

```
exit(); //Gracias!
```