Técnicas de Programación Avanzada (Java)

Curso 2022-2023

S_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;

}

Dra. Nieves Cubo Mateo

Sept 2022

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.

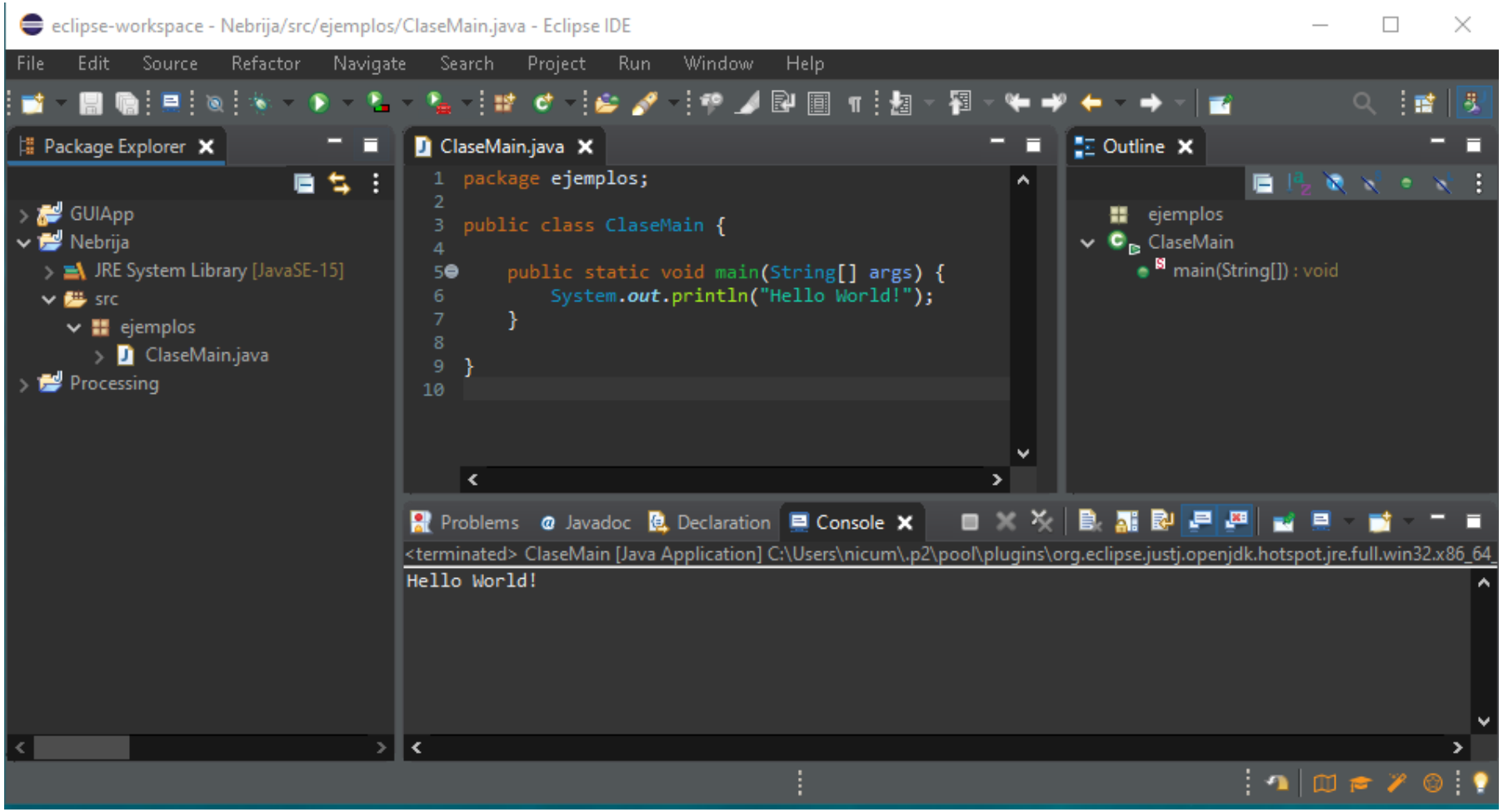
5.2 Inversión de Control. Definición y ejemplos. Inyección de dependencias.

5.3 Expresiones avanzadas del lenguaje.

1.1 Características básicas del lenguaje.

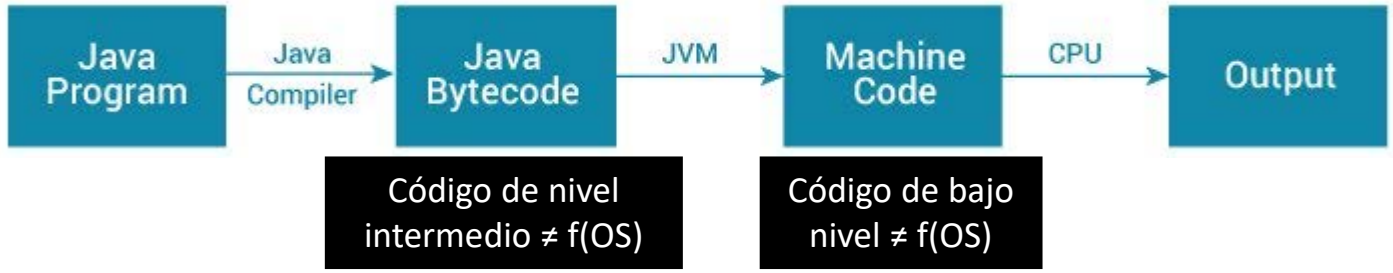
IDE

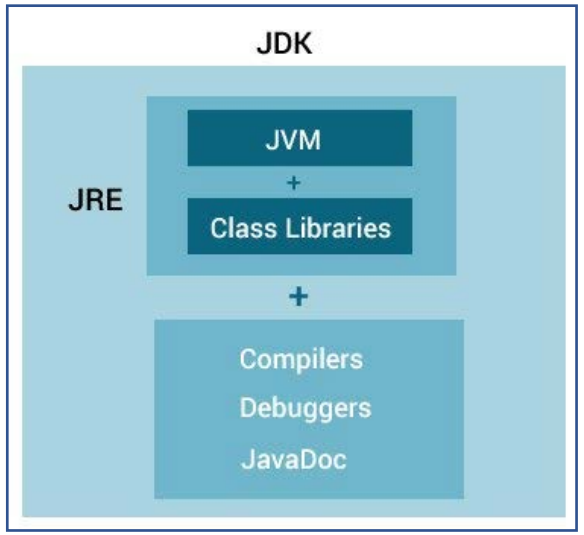
- JDK Installation Guide – Java SE (Standard Edition)
- Eclipse IDE – For Java Developers



1.1 Características básicas del lenguaje.

IDE





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

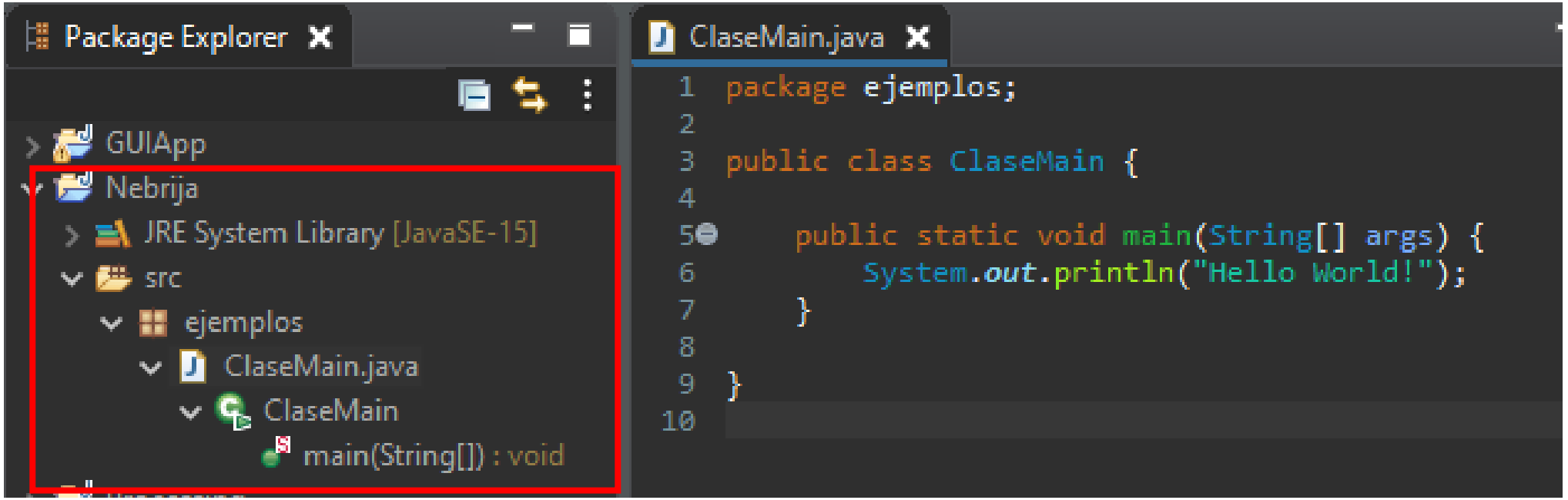
Java es un lenguaje de programación que tiene las siguientes características:

- **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.

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!");
	}
}
```

El archivo debe llamarse igual que la clase principal (.java)

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 (podría haber otros si se cambian los argumentos de entrada, sobrecarga):

```java
public static void main(String[] args) {}
public static void main(int x) {}
```

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:

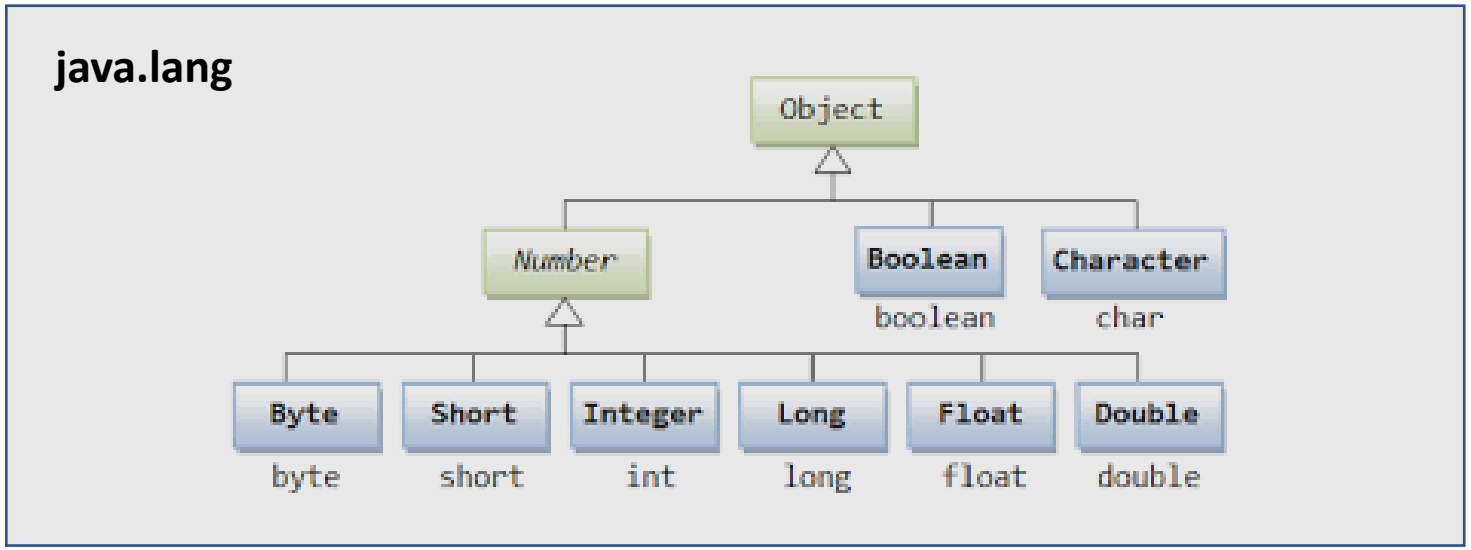
// En una línea

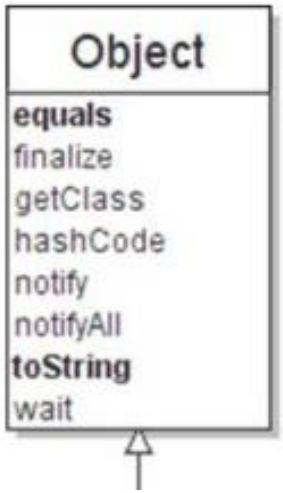
```
/*   En varias
* líneas
* Asteriscos extra opcionales
*/
```

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

1.1 Características básicas del lenguaje.

Primer programa. Compilación y Ejecución.

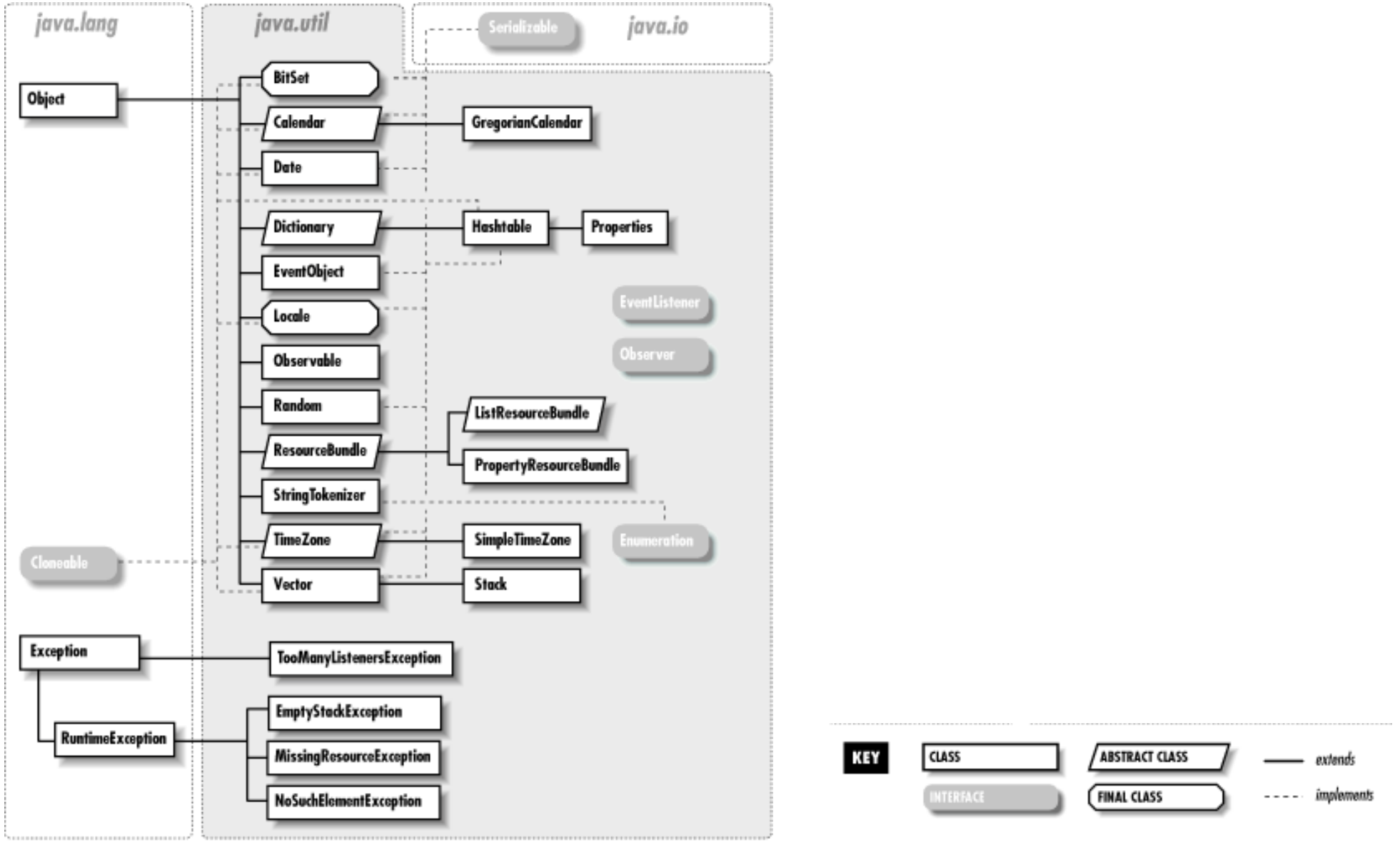




- Paquete **java.lang** contiene las clases más importantes del lenguaje de Java.
- La clase **Object** **es la raíz de toda la jerarquía de clases en Java** (ultimate superclass). Es decir, cualquier clase es implicitamente hija de Object.
- Los datos primitivos son los únicos que no son objetos. Si se quieren tratar como tales, hay que usar sus wrappers (envoltorios).
- En Java todo se basa en punteros de forma implícita, por lo que los objetos (excepto los primitivos) se pasan por referencia (se pasa su dirección, no su valor).

1.1 Características básicas del lenguaje.

Primer programa. Compilación y Ejecución.



1.1 Características básicas del lenguaje.

Repaso diagramas UML

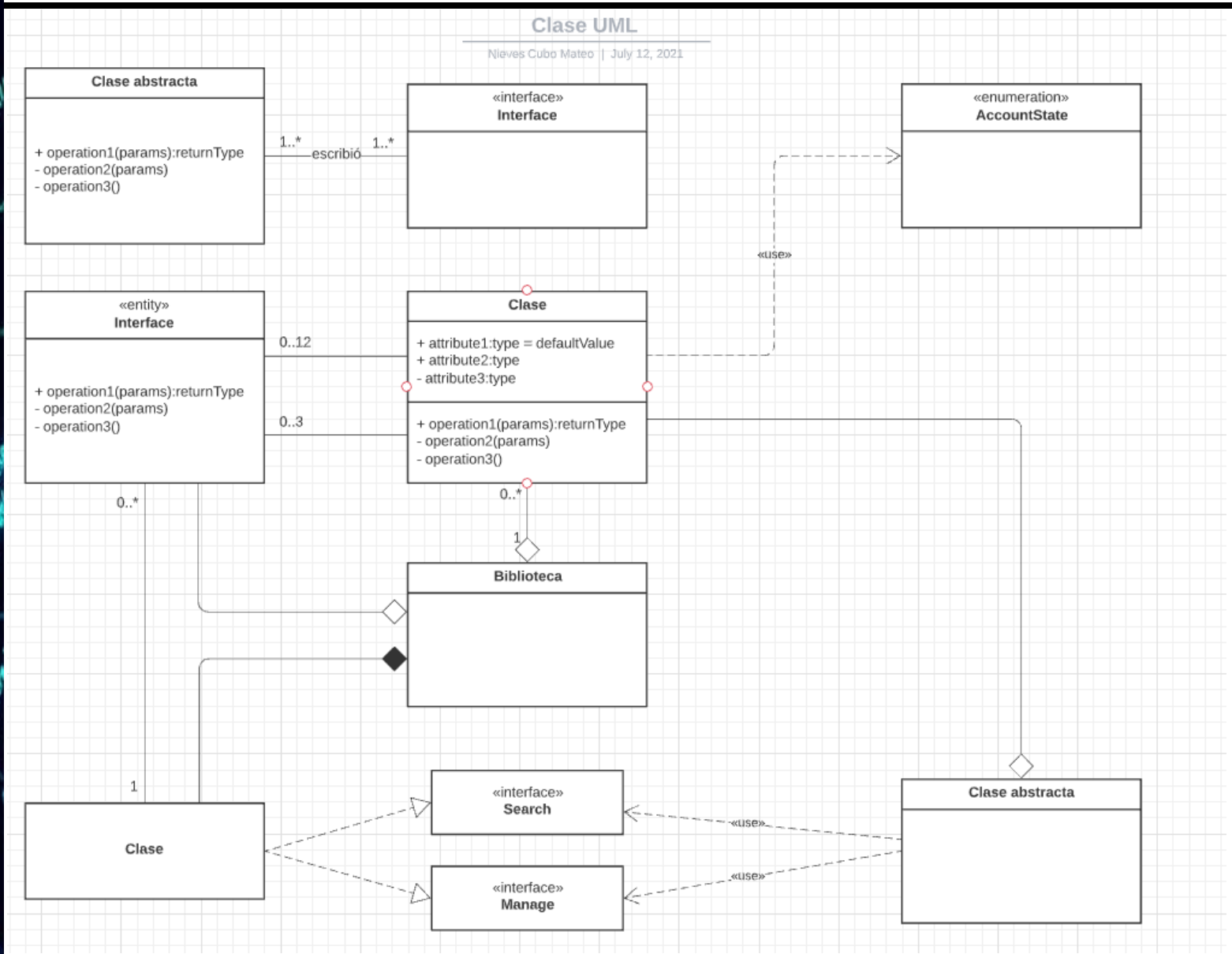


Diagrama de clases. UML.

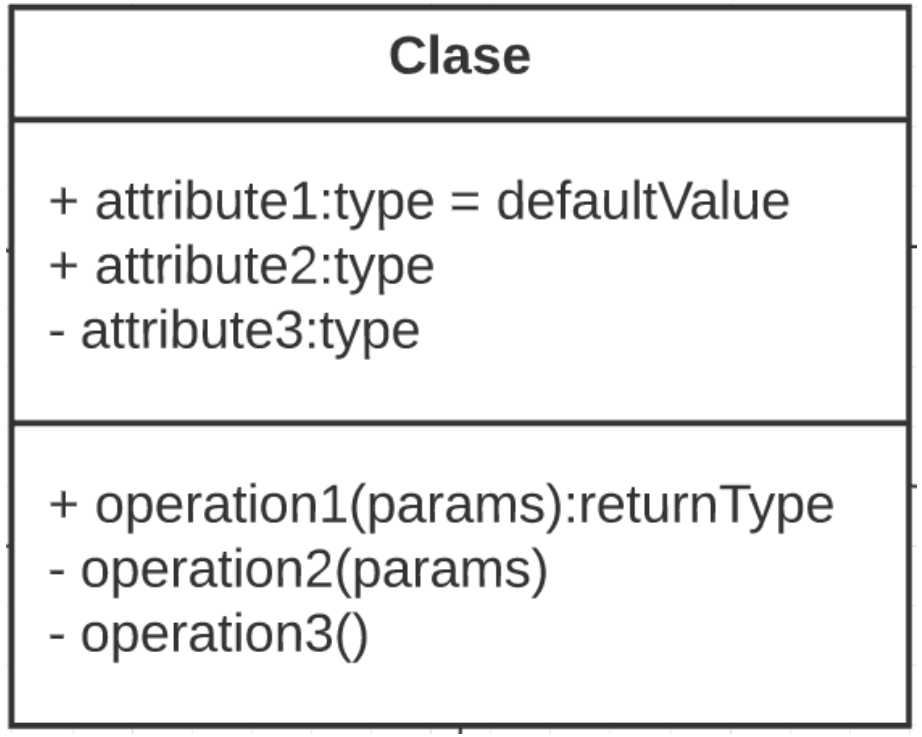
Repaso + Plataforma para crear diagramas (Lucidachart):

https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml/#top

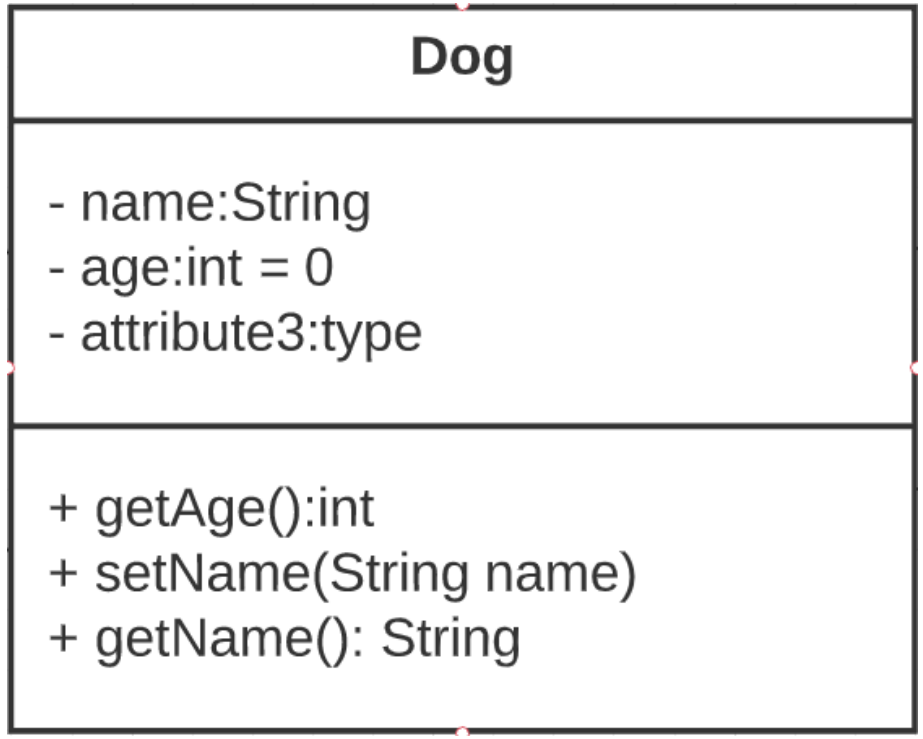
1.1 Características básicas del lenguaje.

Repaso diagramas UML

Genérico



Ejemplo



Niveles de acceso:

- **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

1.2 Sentencias de control.

Secuencia, selección e iteración/repetición.

```
package processingPckg;

import processing.core.PApplet;

public class MySketch extends PApplet {

public void settings(){
            size(500, 500);
}

public void draw(){
            ellipse(mouseX, mouseY, 50, 50);
}

public void mousePressed(){
            background(64);
}


public static void main(String[] args) {

            String[] processingArgs = {"MySketch"};
            MySketch mySketch = new MySketch();
            PApplet.runSketch(processingArgs, mySketch);

}

}
```

SECUENCIA (integrada en Java)

TRANSFERENCIA DE CONTROL -

Estructuras de control:

- if...else
- switch Statement

SELECCIÓN

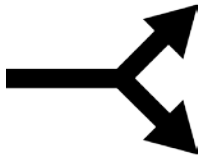
- for / for-each Loop
- while Loop

(+break & continue statements)

REPETICIÓN

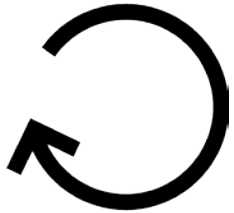
1.2 Sentencias de control.

Secuencia, selección e iteración.



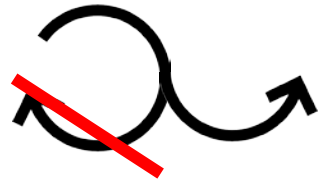
Condicional (selección):

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



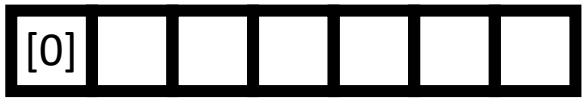
Bucles (iteración/repetición):

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



Salir de bucles:

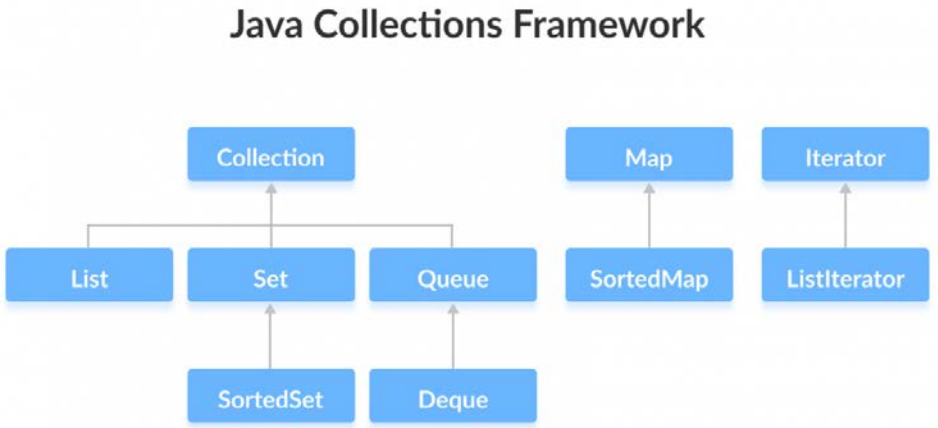
- break
- continue



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
}
```

- Igual que en C++.
- Admite "if else" y anidación

```
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;
```

1.2 Sentencias de control.

Boolean conditionals

- true / false
- `boolean bol = true;`

- AND: A && B Deben dares ambas condiciones
- OR: A || B Debe darse al menos una de las condiciones

```
if ((x>5)&&(x<10)){
    …
}
```

1.2 Sentencias de control.

Switch

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

- Igual que en C++.
- 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 & for-each

- Igual que en C++. Se usa cuando el número de iteraciones es conocido.

```
for (initialExpression; testExpression; updateExpression) {
   // body of the loop
}
```

```
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 );
            }
      }
}
```

- FOR each…

```
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);
}
```

Equivalente a (para recorrer arrays y colecciones):

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

1.2 Sentencias de control.

while & do…while

- Igual que en C++

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

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

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

```
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++;
}
```

1.2 Sentencias de control.

Arrays

Reserva de espacio: new

- Devuelve una dirección de memoria, la referencia (puntero) a la información creada (objeto).

- Invoca al contructor de la clase

```
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)

object[] mixedArray = new object[10];
```

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).

Ejercicios

1.1 y 1.2

1. Crea un programa que pida al usuario que introduzca dos números (primero un número, después otro). Llama al método suma e imprime por pantalla el resultado.

- *Habrá que crear también la clase "Calculadora" que tenga 2 atributos privados: x e y, los corresponientes setters y getters y el método "suma".*
- *Pueden incluirse todas las clases en el mismo archivo o crear un archivo.java para cada una*

```
package __________;

public class _________ {

	public static void main(String[] args) {
		…
	}

}
```

Ejercicios

1.1 y 1.2

2. Crea un programa que pida al usuario que introduzca un número entero y le diga si es primo o no. (Controlar si el número introducido es entero o no).

3. Crea un programa que pida al usuario un número entero y mostrar todos los números de la serie de Fibonacci que estén por debajo.

4. Escribe un programa que pida al usuario introducir números (double) en un array hasta que lo desee (detener introducción de datos con un Centinela, ej: núm. Negativo, o letra). Y devolver la suma de los números al final.

Lo que sabéis hacer en C++, deberíais saber hacerlo en Java rápido…

Más ejercicios con solución para practicar (+ editor online!):
https://www.w3resource.com/java-exercises/

Buen lugar de donde sacar ejercicios de examen…

Ejercicios

Segunda clase

1. **Haz un programa que pida al usuario que introduzca números enteros, y ve guardándolos en un array hasta que introduzca una 'x'. Después recorre el array usando un for normal e imprime todos los números introducidos en la misma línea, separados por comas.**

2. **Repite el mismo programa, ahora con un for-each.**

3. **Haz un programa que pida al usuario que introduzca una palabra. Después deberá sacar por pantalla, separando en diferentes líneas:**
   - **la palabra, del revés**
   - **el número de letras que tiene la palabra**
   - **la palabra sin sus vocales**

   (*Util: métodos de la clase String - https://www.w3schools.com/java/java_ref_string.asp)*

4. **Crea una clase abstrayendo un objeto real, marca sus parámetros (min. 3) (privados o públicos) y los métodos correspondientes que consideres. En la clase principal, crea un objeto de esa clase e invoca alguno de sus métodos.**

Ejercicios

Segunda clase

5. 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
```

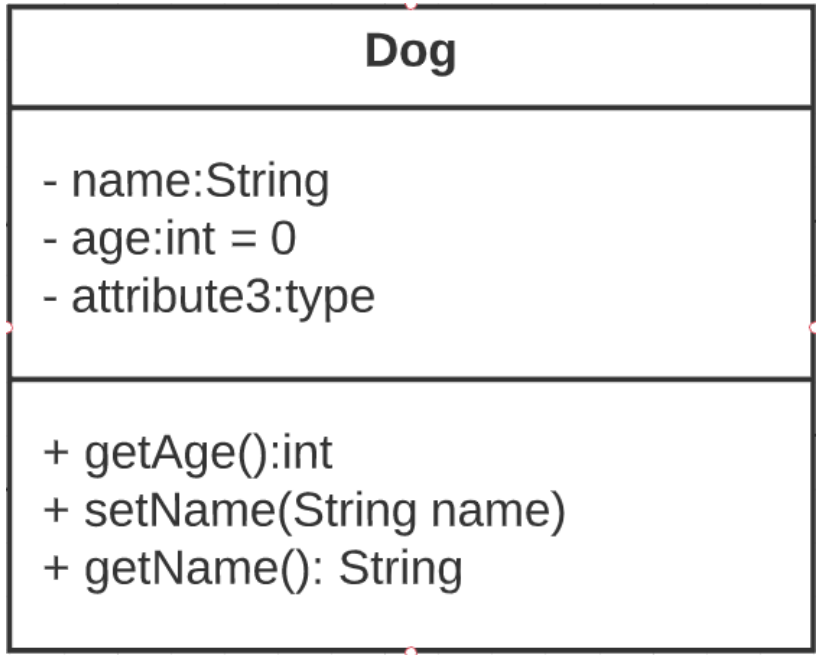
(continuación código…)

```
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 y encapsulamiento

Clases, objetos, métodos y atributos.

ABSTRACCIÓN ¿Qué?
ENCAPSULAMIENTO ¿Cómo?




```java
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) {
    this.age = age;
  }

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

1.4 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.

- Similar a sobrecarga de constructores.

- Polimorfismo: igual, pero desde clases hijas a clase padre (super).

Paquetes y *module-info.java*

Paquete: agrupación de clases que 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).*

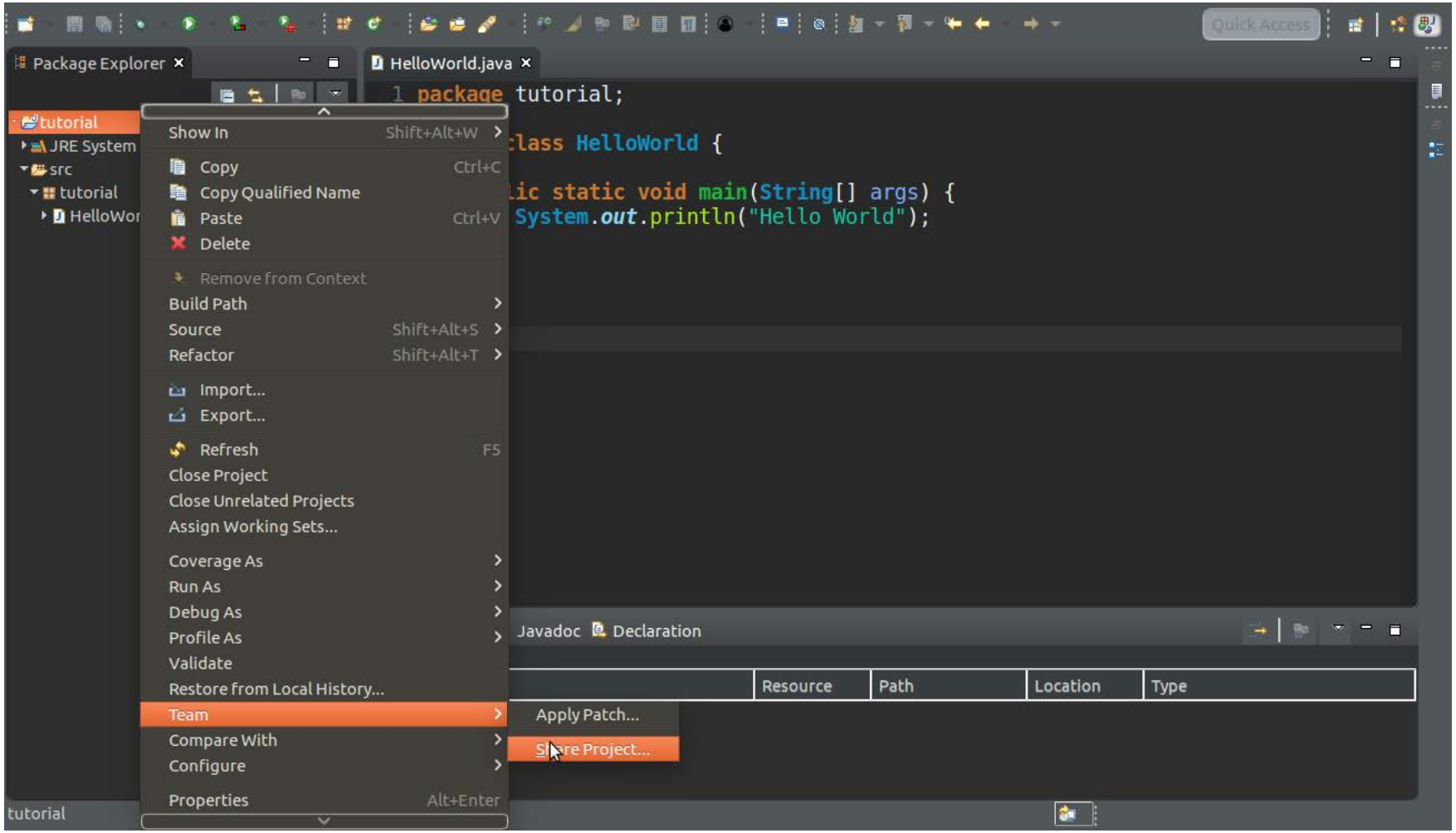
Una clase puede acceder a todas las clases públicas que están en su mismo paquete, sin necesidad de indicar el nombre de dicho paquete. *Si se desea acceder a otras que no están en su mismo paquete, se puede importar éste o indicar el nombre completo. Ejemplo:* `import java.awt.image.BufferStrategy;`

¿Para qué sirve el module-info.java? (Se crea por defecto)
Para exportar o importar paquetes, este fichero descriptor nos será de mucha utilidad para saber que exportamos o bien que importamos. Ejemplo:

```
module rpg2D {
    requires module.name;
    exports package.name;
}
```

Cómo subir mi Proyecto a Github fácilmente

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



Estructuras de datos

- **Array (built-in array):** tamano no se puede modificar. Hace falta crear otro array.
  - String [] cats = new String(5);
    cats[0]="Kitty";

- **ArrayList:** Se pueden anadir y eliminar elementos (set, get, remove, size, sort,)
  - import java.util.ArrayList;
    import java.util.Collections; //Para usar sort
    ArrayList<String> cats = new ArrayList<String>(); ← Objects, Wrappers
    cats.add("Kitty");

- **LinkedList**: similar a ArrayList (ambos implementan la interfaz List), pero son elementos enlazados (nodos) en orden (addFirst, addLast, removeFirst, removeLast, getFirst, getLast, pop, peek, ..), **no se puede acceder de forma random**.
  - import java.util.LinkedList;
    LinkedList<String> cats = new LinkedList<String>();
    cats.add("Kitty");

- **HashMap:** como los diccionarios <key/value>
- **HashSet:** como los pools, saco con items únicos

Java Enums

- 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
}
```

(←) **Implementación simple o completa** (↓)

```
enum Races{
  ELF("Elf"), DWARF("Dwarf"), ORC("Orc");

  private final String Race;

  Races (String Race) {
    this.Race = Race;
  }

  public String getRace() {
    return Race;
  }
}
```

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

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

Conversión de tipos: parse/cast

String → int	public static int parseInt(String s) public static int valueOf(s);	int i=Integer.parseInt("200"); int i=Integer.valueOf("200");
int → String	public static int parseInt(String s) public static String valueOf(s);	int i=Integer.parseInt("200"); String s=String.valueOf(Integer(200));

Box/Autoboxing - Wrappers

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);
```

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 (Java)

Curso 2021-2021

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

Dra. Nieves Cubo Mateo