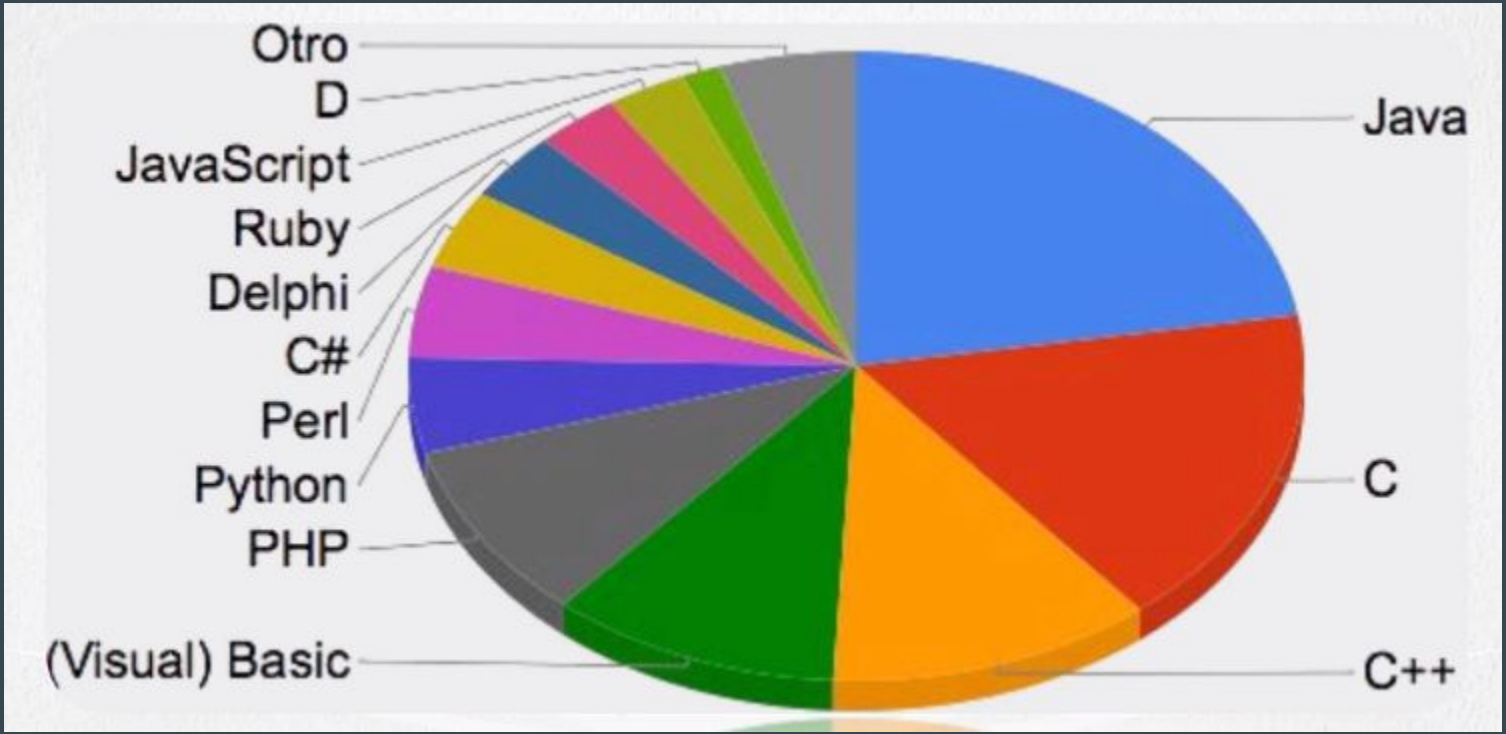
Carrera de Java Programmer SE8

•••

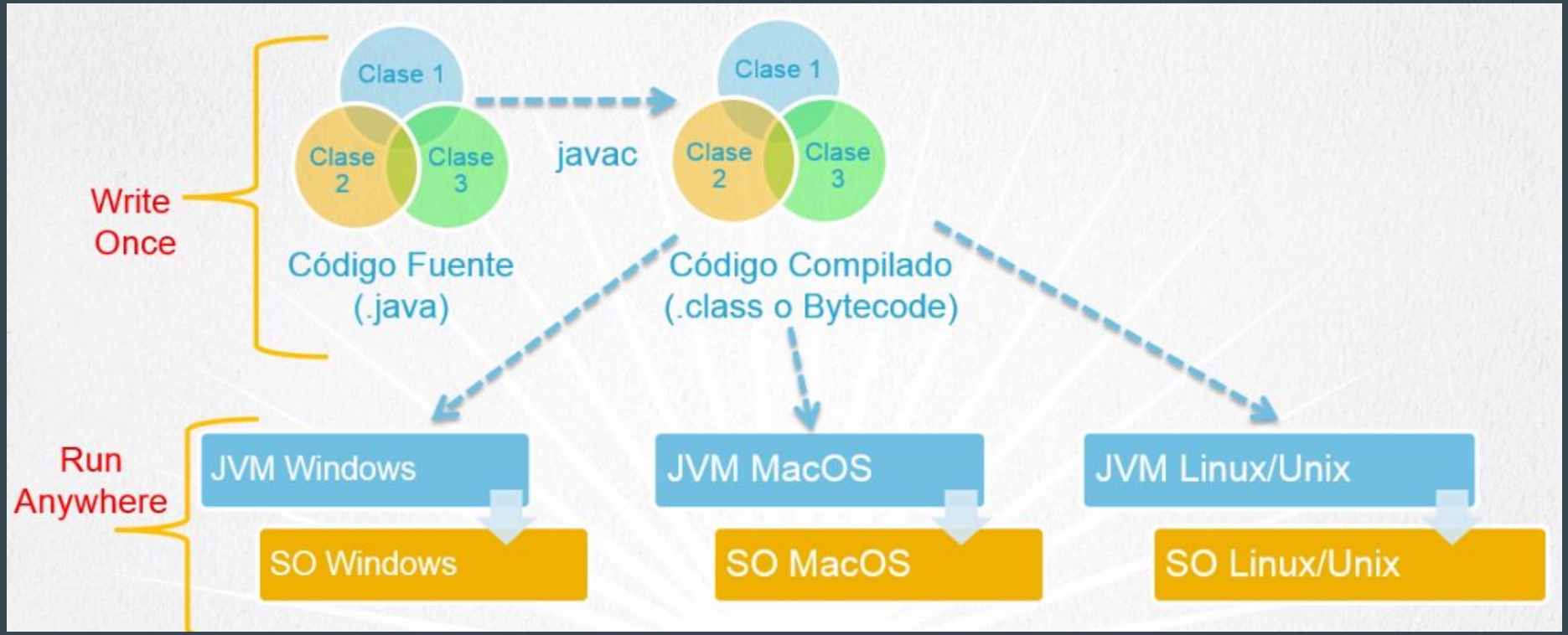
Módulo 2 : Lenguaje de Programación Java

Repaso de la Plataforma Java

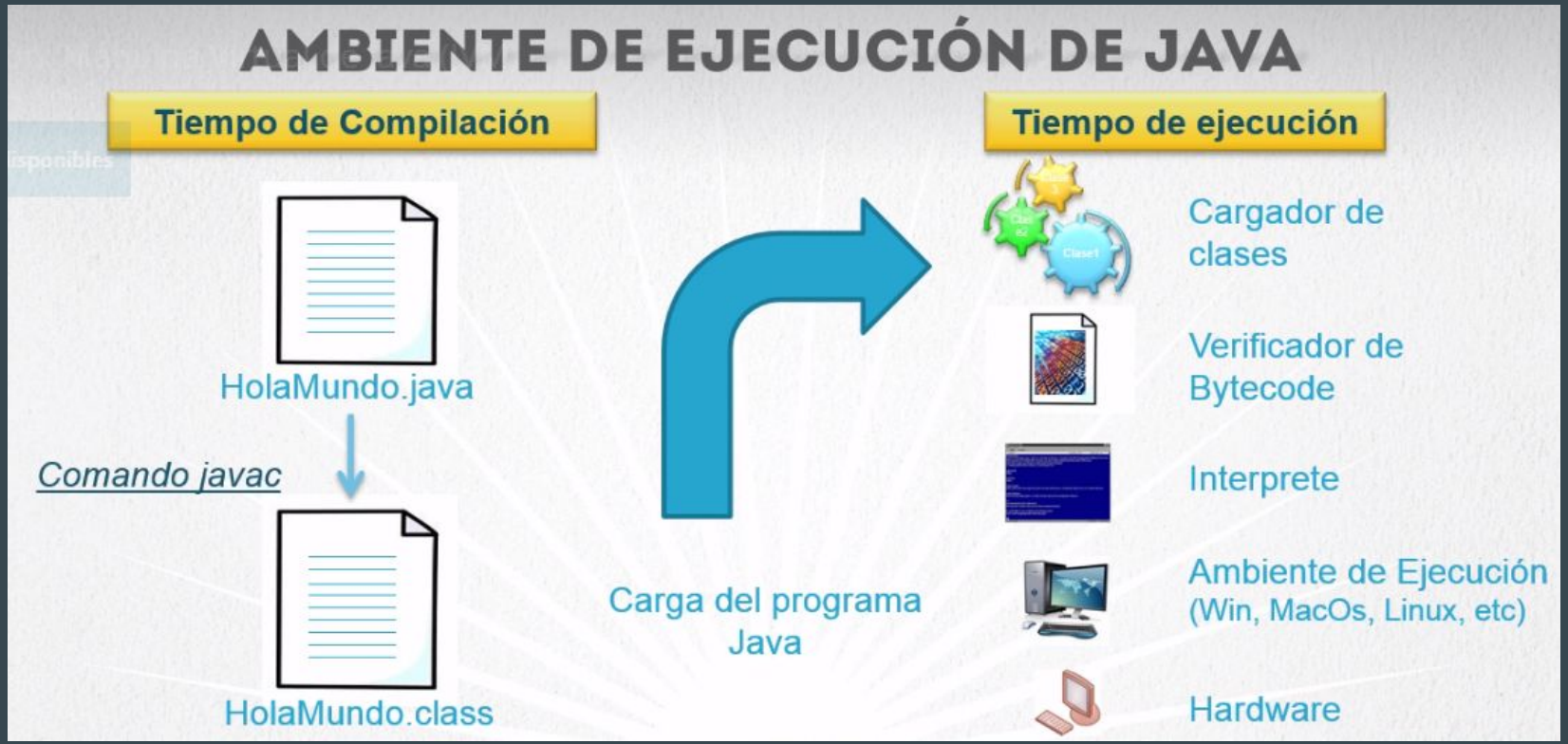
Java y la Industria de Software



Máquina Virtual de Java (JVM)



Ambientes en Java



Productos de Java

- JSE (Java Standard Edition):
  - Soluciones de escritorio.
  - Aplicaciones independientes.
  - Aplicaciones distribuidas.
- JEE / J2EE (Java Enterprise Edition):
  - Soluciones empresariales.
  - Aplicaciones empresariales con alto nivel transaccional.
  - Comercio electrónico.
- JME / J2ME (Java Micro Edition):
  - Soluciones de consumo
  - Celulares, PDAs, TVs, microchips, entre otros.

Sintaxis de Java y Repaso de POO

Clases

```java
public class Example {

    public static void main(String[] args) {
        // Programa escrito
    }

}
```

```java
public class NewClass {
    // Elementos de clase
}
```

Variables, constantes y atributos

```java
public class Example {

    public static void main(String[] args) {
        // Variables
        // Primitivas
        int number; // Declarada.
        boolean status = false; // Declarada e iniciada
        // De referencia
        String name = "Javier";
        NewClass nc = new NewClass();
        // Constantes
        final float CONSTANT = 300.67F;
    }

}
```

Operadores

```java
public class Example {

    public static void main(String[] args) {
        // Operadores aritméticos
        System.out.println(20 + 5); // 25
        System.out.println(10 - 15); // -5
        System.out.println(20 * 5); // 100
        System.out.println(60 / 2); // 30
        System.out.println(60 % 2); // 0
    }

}
```

Operadores

```java
public class Example {

    public static void main(String[] args) {
        // Operadores de relación
        System.out.println(20 == 5); // false
        System.out.println(10 > 15); // false
        System.out.println(20 < 5); // false
        System.out.println(60 >= 2); // true
        System.out.println(60 <= 2); // false
    }

}
```

Operadores

```java
public class Example {

    public static void main(String[] args) {
        // Operadores lógicos
        System.out.println(true && true); // true
        System.out.println(true && false); // false
        System.out.println(true || true); // true
        System.out.println(true || false); // true
        System.out.println(!true); // false
    }

}
```

Manipulación de Cadenas

```
public class Example {

    public static void main(String[] args) {
        // Crear y manipular cadenas de caractéres
        // Inmutables
        String cadena1 = "Hola";
        cadena1 += " Mundo.";
        // Mutables
        StringBuilder cadena2 = new StringBuilder();
        cadena2.append("Otro Hola").append("Mundo");

        System.out.println(cadena1.toUpperCase()); // hola mundo
        System.out.println(cadena2.equals(cadena1)); // false
    }

}
```

Estructuras de Control

```
public class Example {

    public static void main(String[] args) {
        // Condicional simple
        if((3*3) == 9) { // 9 es igual 9
            System.out.println("SÍ Se cumplió la condición.");
        }
        // Condicional doble
        if((3*3) == 5) { // 9 no es igal a 5
            System.out.println("SÍ Se cumplió la condición.");
        } else {
            System.out.println("NO se cumplió la condición.");
        }
    }

}
```

Estructuras de Control

```java
public class Example {

    public static void main(String[] args) {
        int option = 3;
        // Condicional multiple
        if(option == 1) {
            System.out.println("Ha escogido la opción #1.");
        } else if(option == 2) {
            System.out.println("Ha escogido la opción #2.");
        } else if(option == 3) {
            System.out.println("Ha escogido la opción #3.");
        } else {
            System.out.println("Ninguna de las anteriores.");
        }
    }
}
```

Estructuras de Control

```java
public class Example {

    public static void main(String[] args) {
        int option = 3;
        // Switch case
        switch (option) {
            case 1:
                System.out.println("Ha escogido la opción #1.");
                break;
            case 2:
                System.out.println("Ha escogido la opción #2.");
                break;
            case 3:
                System.out.println("Ha escogido la opción #3.");
                break;
            default:
                System.out.println("Ninguna de las anteriores.");
                break;
        }
    }
}
```

Iteraciones con Ciclos

```java
public class Example {

    public static void main(String[] args) {
        boolean verify = true;
        int i = 0;

        // Ciclo While
        while(verify) {
            i++;
            System.out.println("2 * " + i + " = " + (2*i));
            if(i == 10) {
                verify = false;
            }
        }
    }
}
```

Iteraciones con Ciclos

```java
public class Example {

    public static void main(String[] args) {
        boolean verify = true;
        int i = 0;

        // Ciclo DoWhile
        do {
            i++;
            System.out.println("2 * " + i + " = " + (2*i));
            if(i == 10) {
                verify = false;
            }
        } while(verify);
    }
}
```

Iteraciones con Ciclos

```java
public class Example {

    public static void main(String[] args) {
        // Ciclo For
        for(int i = 1; i <= 10; i++) {
            System.out.println("2 * " + i + " = " + (2*i));
        }
    }
}
```

Iteraciones con Ciclos

```java
public class Example {

    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        // Ciclo Foreach
        for(int n : numbers) {
            System.out.println("2 * " + n + " = " + (2*n));
        }
    }
}
```

Arreglos

```java
public class Example {

    public static void main(String[] args) {
        // Arreglos unidimensionales (Vectores)
        // Declaración
        String[] students;
        // inicialización
        students = new String[5];
        // Declaración e inicialización
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // Asignación manual
        // Asignación por manual indice de posición
        students[0] = "Luis Ochoa";
        students[1] = "Sofia Vargas";
        students[2] = "Carlos Pérez";
        students[3] = "Celeste Rodriguez";
        students[4] = "Erika López";
        // Muestra
        System.out.println(students[3]);// Celeste Rodríguez
        // Para llenar con ciclos
        System.out.println(numbers.length);//10 - cantidad de posiciones.
    }
}
```

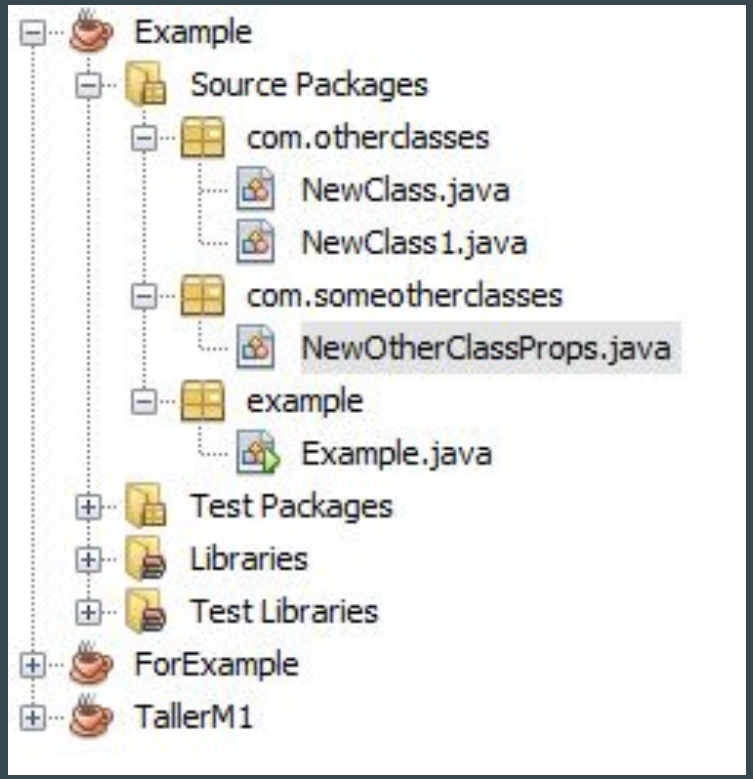
Arreglos

```java
public class Example {
    public static void main(String[] args) {
        // Arreglos bidimensionales (Matrices)
        // Declaración
        String[][] codes;
        // inicialización
        codes = new String[2][2];
        // Declaración e inicialización
        int[][] tables = {
       /*0*/{1, 2, 3, 4, 5},// 00, 01, 02, 03, 04
       /*1*/{2, 4, 6, 8, 10},// 10, 11, 12, 13, 14
       /*2*/{3, 6, 9, 12, 15}// 20, 21, 22, 23, 24
        }; // Asignación manual
        // Asignación por manual índice de posición
        codes[0][0] = "JUY";
        codes[0][1] = "ADW";
        codes[1][0]= "XVN";
        codes[1][1] = "PLH";
        // Muestra
        System.out.println(codes[0][1]);// ADW
        // Para llenar con ciclos
        System.out.println(codes[0].length);// 2 - cantidad de posiciones.
    }
}
```

Clase para POO

```java
public class NewClass {
    // Atributos
    int NumberOne;
    int NumberTwo;
    String Operate;
    // Métodos
    public NewClass(int no, int nt, String o) {
        this.NumberOne = no;
        this.NumberTwo = nt;
        this.Operate = o;
    } // Constructor
    public int sumNumbers() {
        int result = 0;
        switch(this.Operate) {
            case "sumar": result = this.NumberOne + this.NumberTwo; break;
            case "restar": result = this.NumberOne - this.NumberTwo; break;
            case "multiplicar": result = this.NumberOne * this.NumberTwo; break;
            case "dividir": result = this.NumberOne / this.NumberTwo; break;
        }
        return result;
    }
}
```

Encapsulamiento



Encapsulamiento

```java
public class NewOtherClassProps {
    // Attributos
    private int PropertyOne; // Modificador de acceso privado
    protected float PropertyTwo; // Modificador de acceso protegido
    public int PropertyThree; // Modificador de acceso público

    // Métodos
    public int getPropertyOne() {
        return PropertyOne;
    }
    public void setPropertyOne(int PropertyOne) {
        this.PropertyOne = PropertyOne;
    }
    public float getPropertyTwo() {
        return PropertyTwo;
    }
    public void setPropertyTwo(float PropertyTwo) {
        this.PropertyTwo = PropertyTwo;
    }
}
```

Encapsulamiento

```java
public class NewOtherClass extends NewOtherClassProps {
    public void firstMethod() {
        // Cuerpo del método
    }

    public boolean secondtMethod() {
        // Cuerpo del método
        return true;
    }

    public void thirdMethod() {
        // Cuerpo del método
    }

    public String fourthMethod() {
        // Cuerpo del método
        return "ÉXITO!";
    }
}
```

Objetos Inmutables

Inmutabilidad: algo es inmutable cuando no se puede modificar.

En programación, decimos que una variable es inmutable cuando su valor no se puede modificar, así como decimos que un objeto es inmutable cuando el estado del mismo no puede cambiar luego de su creación.

Sirve para asegurarse que los objetos no van a ser modificados en lugares inesperados del programa.

Objetos Inmutables

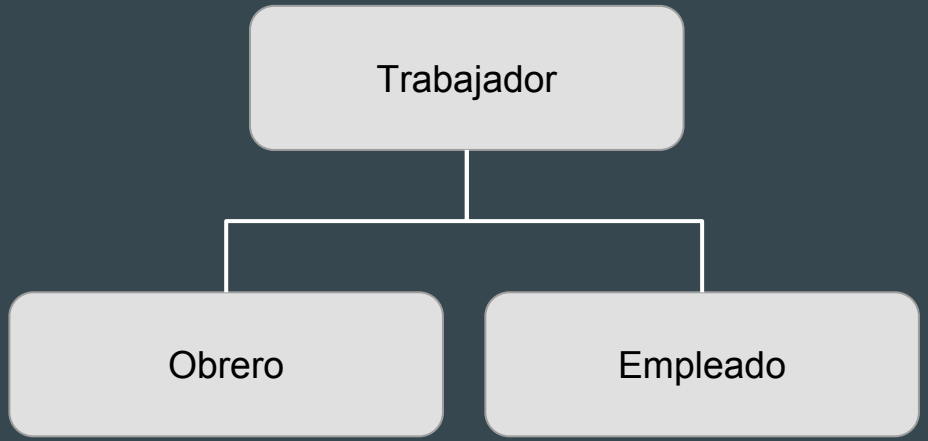
1. No proporcionar ningún método que permita modificar el objeto (como p. ej. los setter).
2. Usar visibilidad **private** para los atributos.
3. Definir los atributos como **final**.
4. Usar copia defensiva para las referencias internas a objetos mutables. Esto implica:
    1. No inicializar nunca referencias internas simplemente copiando una referencia externa, por ejemplo, en los constructores.
    2. No devolver nunca una referencia interna tal cual, por ejemplo, en los getters.
5. Declarar la clase como **final**.

Abstracción

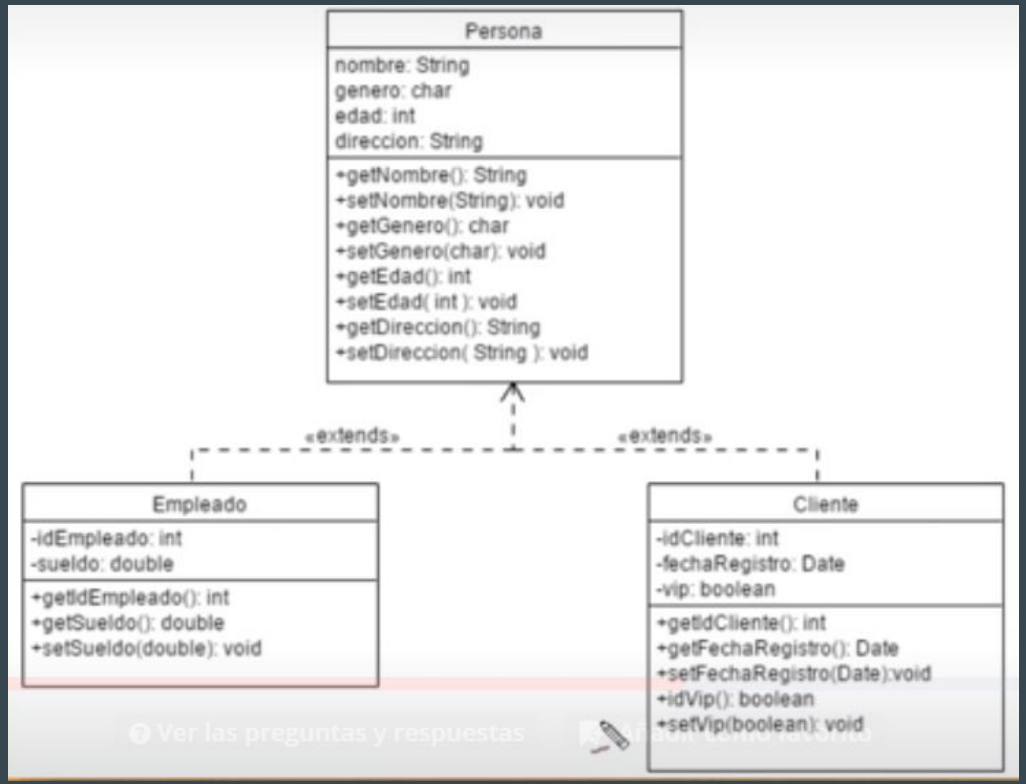
Resaltar la parte más representativa de algo, ignorando detalles para centrarse en lo principal.

Herencia

- Utilizar la estructura o modelo de una clase padre.
- La clase hija se puede acceder a los métodos y atributos de la clase padre.
- Se usa la palabra reservada “extends”, seguido del nombre de la clase de la cual se quiere heredar.
- Representar comportamiento común.
- Evitar duplicación de código.
- Es un característica de la POO.
- Se utiliza la palabra ”super” para acceder a los componentes del padre.



Ejemplo



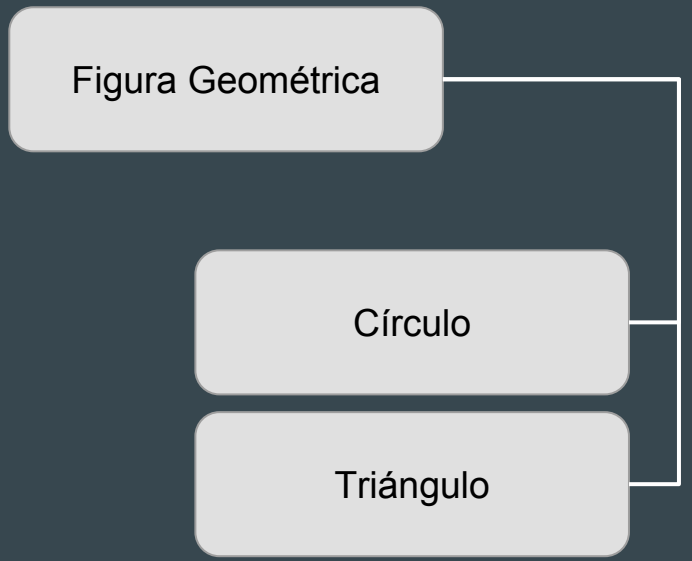
Clase Persona
+ getNombre()
+ setNombre()
+ getGenero(), etc

Clase Empleado
+getIdEmpleado()
+getSueldo(), etc

Polimorfismo

Es la capacidad que tienen los objetos de comportarse de múltiples formas. Eso significa que para aplicarlo se debe programar de manera general en vez de hacerlo de manera específica.

El objetivo de aplicar polimorfismo es el de poder crear un árbol de objetos en un proyecto, de tal manera que cada instancia de objeto que se cree pueda pertenecer a un mismo tipo (siempre y cuando el análisis del desarrollo de los objetos lo necesite).



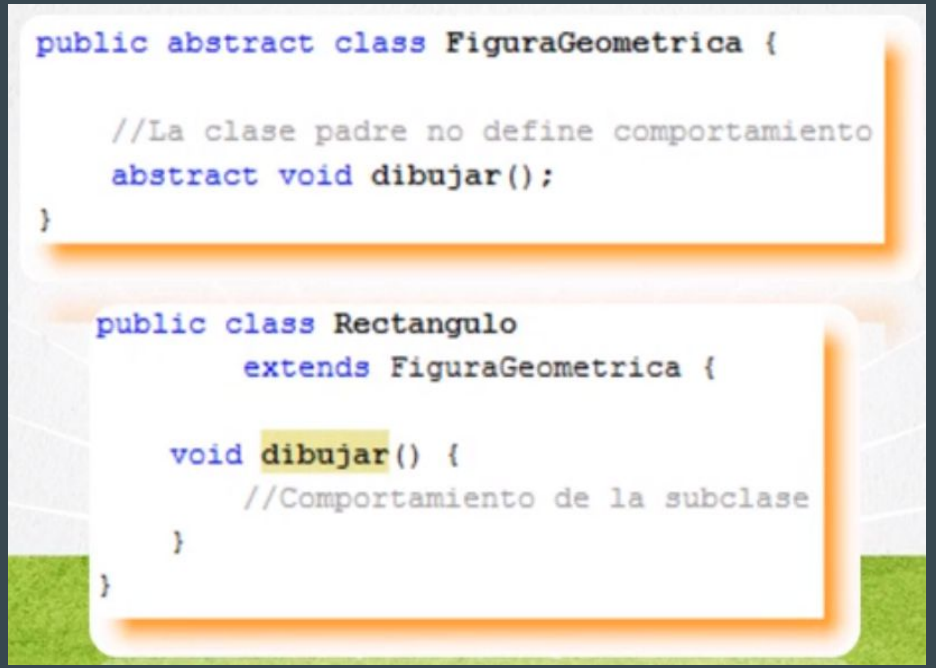
Clases Abstractas

Características:

- Definir una estructura.
- Aplicar el polimorfismo.
- Uso de la palabra reservada “extends”.

Restricciones:

- No se pueden instanciar.
- No definen comportamiento.
- La funcionalidad la establece las clases hijas.



```
public abstract class FiguraGeometrica {

    //La clase padre no define comportamiento
    abstract void dibujar();
}
```

```
public class Rectangulo
        extends FiguraGeometrica {

    void dibujar() {
        //Comportamiento de la subclase
    }
}
```

Miembros estáticos de una clase

- También se les conoce como miembros de clase. Son elementos que pertenecen a la clase como tal.
- Al momento de instanciar una clase, se monta en la memoria de trabajo una referencia, cuyo nombre almacena un objeto del tipo creado. Este contiene una copia de la estructura de la clase, incluyendo sus atributos y métodos, excluyendo los elementos estáticos.
- En resumen, los elementos estáticos pertenecen a la clase, no a los objetos.

Interfaces

- Declaración formal de un contrato.
- Son abstractos, es decir, no poseen implementación.
- Uso de la palabra reservada “implements”.
- Podemos implementar múltiples interfaces a una clase.

Estructura de una Interface

Definición de una interface en Java:

```
<modificadores>  interface <nombre_interface> [extends <interface padre>]
{
        <atributos>
        <métodos>
}
```

Uso de una interface en Java:

```
<modificadores> class <nombre_clase> [extends <superclase>] [implements
<interface1,interface2,etc>]
{
        <implementar_métodos_interface>
}
```

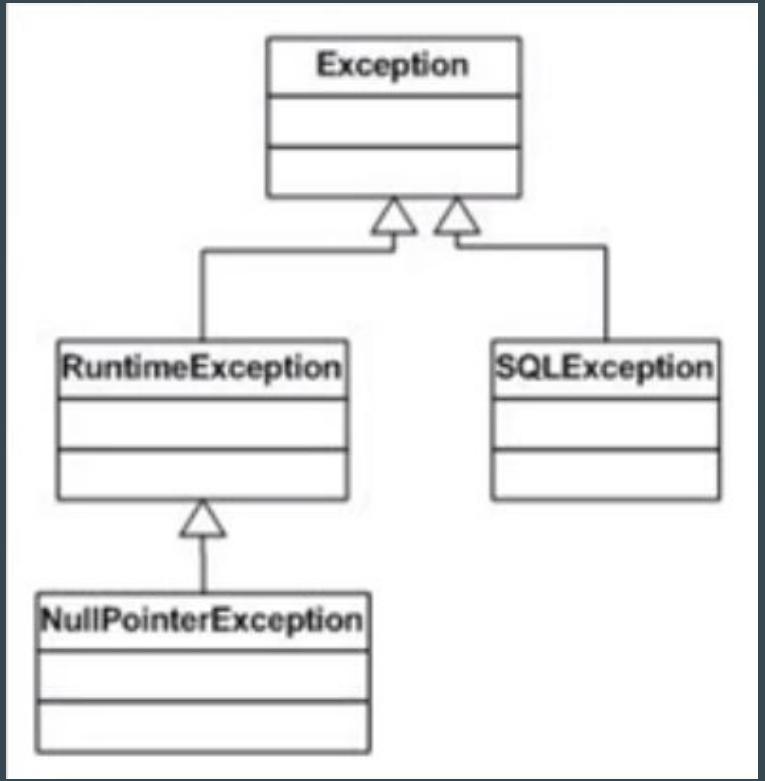
Excepciones

Check Exception

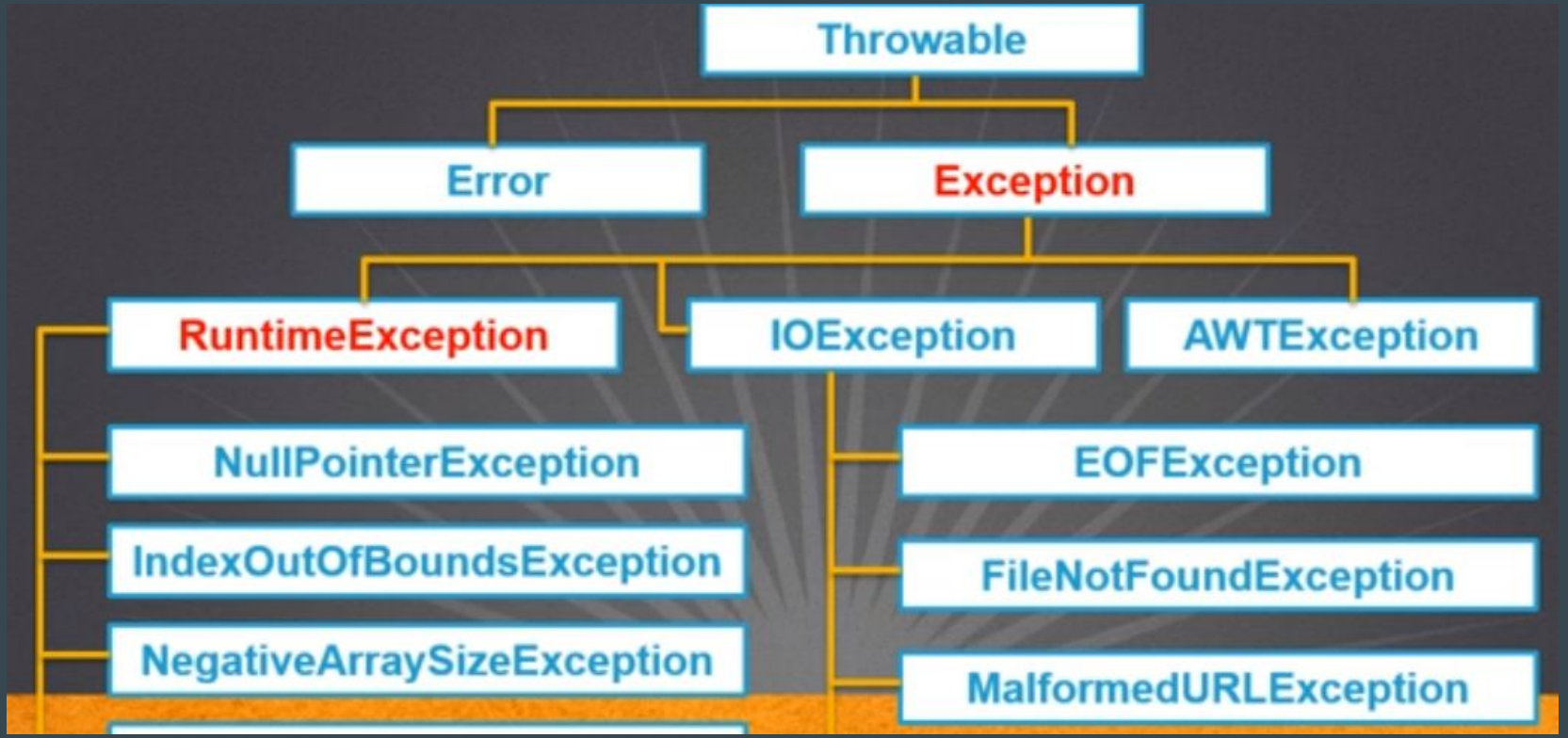
- Hereden de la clase Exception.
- Es obligatorio tratarlas.
- SQLException.

Unchecked Exception

- Hereden de la clase Exception.
- Es opcional tratarla.
- RuntimeException.
    - NullPointerException.

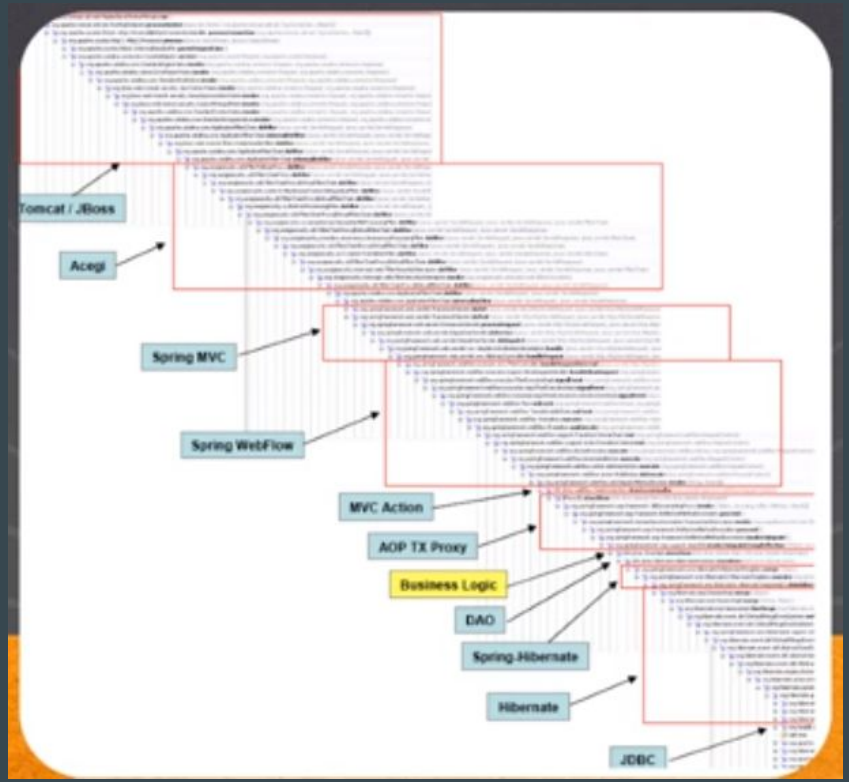


Excepciones más comunes



Stacktrace en Java

- Pilas de errores.
- Traza del error del inicio al fin.
- Flujo del error:
  - Método arroja un error.
  - Si no atrapa la excepción, la propaga hasta que alguna clase la maneja.
  - En caso de no manejarla el método main se satura
  - El programa se finaliza de manera anormal.



Manejo de Excepciones

```
public void verificaExcepciones() {
   try {
      // código que lanza excepciones
   } catch (Exception ex) {
      //Bloque de código que maneja la excepción
      ex.printStackTrace();
   }
   finally{
      //Bloque de código opcional, pero
      //que se ejecuta siempre
   }
}
```

Cláusula Throws

```java
public class ArrojarExcepcion {

  public void metodoX() throws Exception {
    throw new Exception("Mensaje de error");
  }
}
```

```java
public class TestArrojarExcepcion {

  public static void main(String args[]) throws Exception {
    ArrojarExcepcion ae = new ArrojarExcepcion();
    ae.metodoX();
  }
}
```

Creación de Nuestras Excepciones

```
public class MiExcepcion extends Exception{

  public MiExcepcion(String mensaje){
    super(mensaje);
  }
}
```

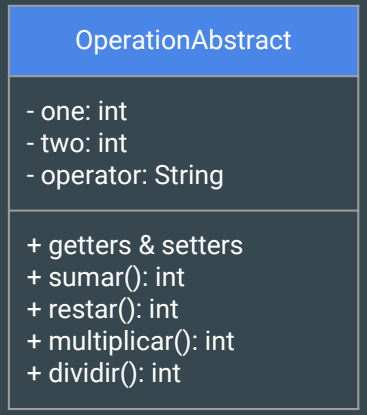
```
public class ArrojarExcepcion2 {

  public void metodoX() throws MiExcepcion {
    throw new MiExcepcion("Mi mensaje de error");
  }
}
```

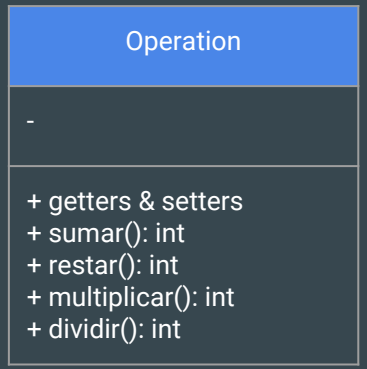
Práctica

Súper clase

abstract



clase



Subclase

Indicaciones

Hacer un programa llamado **Calculadora**, nos pedirá 2 operados y un signo aritmético, según este último se realizará la operación correspondiente y mostrará el resultado. Para realizar este ejercicio se deben seguir las siguientes pautas:

1. Crear los siguientes paquetes:
   a. com.saime.main
   b. com.saime.structures
   c. com.saime.models
2. Crear una clase ejecutable dentro de **com.saime.main** llamada **EntryPoint**
3. Definir una clase abstracta llamada **OperationAbstract** dentro de **com.saime.structure**
4. Definir una clase llamada **Operation** dentro de **com.saime.models**
5. Desarrollar la funcionalidad restante en **EntryPoint**