



ESCUELA DE  
INGENIERÍA EN CIENCIAS Y SISTEMAS  
FACULTAD DE INGENIERÍA  
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



**Día, Fecha:**

Miércoles, 22/02/2023

**Hora de inicio:**

07:10


# Introducción a la programación y computación 1 [D]

*José Eduardo Morales García*

# El principio del encapsulamiento

- ▶ El principio del encapsulamiento consiste en ocultar la complejidad interna de un objeto y proporcionar una interfaz pública que permita interactuar con él.
- ▶ El encapsulamiento tiene varios beneficios, como: proteger la integridad de los datos del objeto, facilitar el mantenimiento y actualización del código, y reducir la complejidad del código.

# Beneficios del encapsulamiento en la programación orientada a objetos.



**PROTECCIÓN DE LOS DATOS:** LOS DATOS ESTÁN OCULTOS Y NO PUEDEN SER ACCEDIDOS NI MODIFICADOS DESDE FUERA DEL OBJETO.

**MODULARIDAD:** CADA OBJETO SE CONVIERTE EN UN MÓDULO INDEPENDIENTE Y AUTÓNOMO QUE PUEDE SER UTILIZADO EN OTROS CONTEXTOS.

**REUTILIZACIÓN:** LOS OBJETOS ENCAPSULADOS PUEDEN SER REUTILIZADOS EN OTROS PROGRAMAS O PROYECTOS.

**FLEXIBILIDAD:** EL CAMBIO INTERNO DE UN OBJETO NO AFECTA A OTROS OBJETOS QUE LO UTILIZAN.

```
public class Coche {  
    private String marca;  
    private String modelo;  
    private int anio;  
  
    public String getMarca() {  
        return marca;  
    }  
  
    public void setMarca(String marca) {  
        this.marca = marca;  
    }  
  
    public String getModelo() {  
        return modelo;  
    }  
  
    public void setModelo(String modelo) {  
        this.modelo = modelo;  
    }  
  
    public int getAnio() {  
        return anio;  
    }  
  
    public void setAnio(int anio) {  
        this.anio = anio;  
    }  
}
```

En este ejemplo, las propiedades de la clase Coche son privadas, lo que significa que sólo se pueden acceder a ellas a través de los métodos públicos get y set.

# Los miembros de una clase

- ▶ Los miembros de una clase son las propiedades y métodos que definen el comportamiento de los objetos de esa clase.
- ▶ En Java, los miembros de una clase pueden ser de cuatro tipos: atributos, métodos, constructores y clases anidadas.

# Constructores y Constructores Vacíos en Java

- ▶ Los constructores son métodos especiales que se utilizan para inicializar objetos cuando se crean.
- ▶ Un constructor vacío es un constructor que no toma ningún argumento y no hace nada. Se utiliza para crear un objeto con valores predeterminados.

```
public class Persona {  
    private String nombre;  
    private int edad;  
  
    public Persona() { // Constructor vacío  
        this.nombre = "Sin nombre";  
        this.edad = 0;  
    }  
  
    public Persona(String nombre, int edad) { // Constructor con argumentos  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}
```

# Uso de constructores

Para crear un objeto y asignarle valores iniciales, se utiliza la palabra clave `new` seguida del nombre del constructor

```
// Crea un objeto con valores predeterminados
Persona p1 = new Persona();

// Crea un objeto con valores personalizados
Persona p2 = new Persona("Juan", 25);
```

Los constructores son una parte importante de la programación orientada a objetos en Java, y permiten crear objetos con valores iniciales personalizados o predeterminados.



```
public class Coche {  
    private String marca;  
    private String modelo;  
    private int anio;  
  
    public Coche(String marca, String modelo, int anio) {  
        this.marca = marca;  
        this.modelo = modelo;  
        this.anio = anio;  
    }  
  
    public void acelerar() {  
        // Lógica para acelerar el coche.  
    }  
  
    public void frenar() {  
        // Lógica para frenar el coche.  
    }  
  
    public String getMarca() {  
        return marca;  
    }  
}
```

```
public void setMarca(String marca) {  
    this.marca = marca;  
}  
  
// Clase anidada  
public static class Motor {  
    private int cilindrada;  
  
    public Motor(int cilindrada) {  
        this.cilindrada = cilindrada;  
    }  
  
    public int getCilindrada() {  
        return cilindrada;  
    }  
}  
}
```

# Modificadores de visibilidad

Los modificadores de visibilidad son palabras clave que se usan para controlar el acceso a los miembros de una clase. En Java, hay tres modificadores de visibilidad: `public`, `private` y `protected`.

- ▶ **public**: El miembro es accesible desde cualquier parte del código.
- ▶ **private**: El miembro sólo es accesible desde dentro de la propia clase.
- ▶ **protected**: El miembro es accesible desde dentro de la propia clase y de las clases hijas (subclases).

```
public class Coche {
    private String marca;
    // Sólo accesible desde dentro de la propia clase.

    protected String modelo;
    // Accesible desde dentro de la propia clase y de las subclases.

    public int anio;
    // Accesible desde cualquier parte del código.

    // Constructor
    public Coche(String marca, String modelo, int anio) {
        this.marca = marca;
        this.modelo = modelo;
        this.anio = anio;
    }

    // Métodos
    private void metodoPrivado() {
        // Sólo accesible desde dentro de la propia clase.
    }

    protected void metodoProtegido() {
        // Accesible desde dentro de la propia clase y de las subclases.
    }

    public void metodoPublico() {
        // Accesible desde cualquier parte del código.
    }
}
```

En este ejemplo, la propiedad `marca` es privada y sólo se puede acceder a ella desde dentro de la propia clase.

La propiedad `modelo` es protegida y se puede acceder desde dentro de la propia clase y de las subclases.

La propiedad `anio` es pública y se puede acceder desde cualquier parte del código.

# public

```
public class EjemploPublic {  
    public int variablePublica;  
  
    public void metodoPublico() {  
        // Código del método  
    }  
}
```

public: Los miembros declarados como public son accesibles desde cualquier lugar, ya sea dentro o fuera de la clase, así como en cualquier otra clase del mismo paquete o de paquetes externos. Los miembros públicos deben ser utilizados en situaciones en las que se necesita acceso a una variable o método desde cualquier parte del código.

# private

private: Los miembros declarados como private son accesibles sólo desde dentro de la clase en la que se declararon. Los miembros privados deben ser utilizados en situaciones en las que se necesita proteger el acceso a una variable o método y no se desea que se modifique desde fuera de la clase.

```
public class EjemploPrivate {  
    private int variablePrivada;  
  
    private void metodoPrivado() {  
        // Código del método  
    }  
}
```

# protected

protected: Los miembros declarados como protected son accesibles desde la clase en la que se declararon y desde cualquier subclase de esa clase, ya sea dentro o fuera del paquete. Los miembros protegidos deben ser utilizados en situaciones en las que se necesita permitir el acceso a una variable o método sólo a las clases relacionadas o que heredan de la clase original.

```
public class EjemploProtected {
    protected int variableProtegida;

    protected void metodoProtegido() {
        // Código del método
    }
}

public class Subclase extends EjemploProtected {
    public void otroMetodo() {
        // Puedo acceder a la variable protegida y al método protegido
        variableProtegida = 10;
        metodoProtegido();
    }
}
```

# Importante

DEBEMOS RECORDAR QUE EL USO DE LOS MODIFICADORES DE VISIBILIDAD DEBE HACERSE DE MANERA CONSCIENTE Y CUIDADOSA, YA QUE UNA MALA UTILIZACIÓN DE ESTOS PUEDE LLEVAR A LA EXPOSICIÓN INDEBIDA DE DATOS O A UNA DIFICULTAD EN EL MANTENIMIENTO Y EVOLUCIÓN DEL CÓDIGO.

# Manejo de excepciones con try-catch

- ▶ En Java, las excepciones son situaciones de error que pueden ocurrir en tiempo de ejecución y que pueden detener el flujo normal del programa.
- ▶ Las excepciones pueden ser manejadas mediante la utilización de las sentencias try y catch.
- ▶ La sentencia try se utiliza para definir un bloque de código en el que se pueden producir excepciones.
- ▶ La sentencia catch se utiliza para definir un bloque de código que se ejecutará cuando se produzca una excepción.



```
public class EjemploTryCatch {
    public static void main(String[] args) {
        try {
            // Código que puede lanzar una excepción
            int resultado = dividir(10, 0);
            System.out.println("El resultado es: " + resultado);
        } catch (ArithmeticException e) {
            // Código que se ejecuta cuando se lanza una excepción
            System.out.println("Ha ocurrido un error: " + e.getMessage());
        }
    }

    public static int dividir(int a, int b) {
        return a / b;
    }
}
```

# Explicación

En este ejemplo, la función `dividir` realiza una operación de división entre dos números. Sin embargo, cuando se intenta dividir un número entre cero, se produce una excepción **`ArithmeticException`**.

En el bloque **`try`** se llama a la función `dividir` y se asigna el resultado a una variable `resultado`. Si se produce una excepción durante la ejecución de la función, se lanza la excepción y se ejecuta el bloque **`catch`**. En este bloque se muestra un mensaje de error con la información de la excepción utilizando el método `getMessage` de la clase `Exception`.

# Importante

ES VITAL RECORDAR QUE EL MANEJO DE EXCEPCIONES ES UNA PARTE IMPORTANTE DE LA PROGRAMACIÓN EN JAVA, YA QUE PERMITE DETECTAR Y SOLUCIONAR ERRORES EN TIEMPO DE EJECUCIÓN DE UNA MANERA ELEGANTE Y CONTROLADA.



# Parte práctica