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

ECYS | DIDT
DESARROLLO DE TRANSFERENCIA TECNOLÓGICA

Día, Fecha:	Jueves, 31/08/2023
Hora de inicio:	07:10

Introducción a la programación y computación 1 [F]

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.

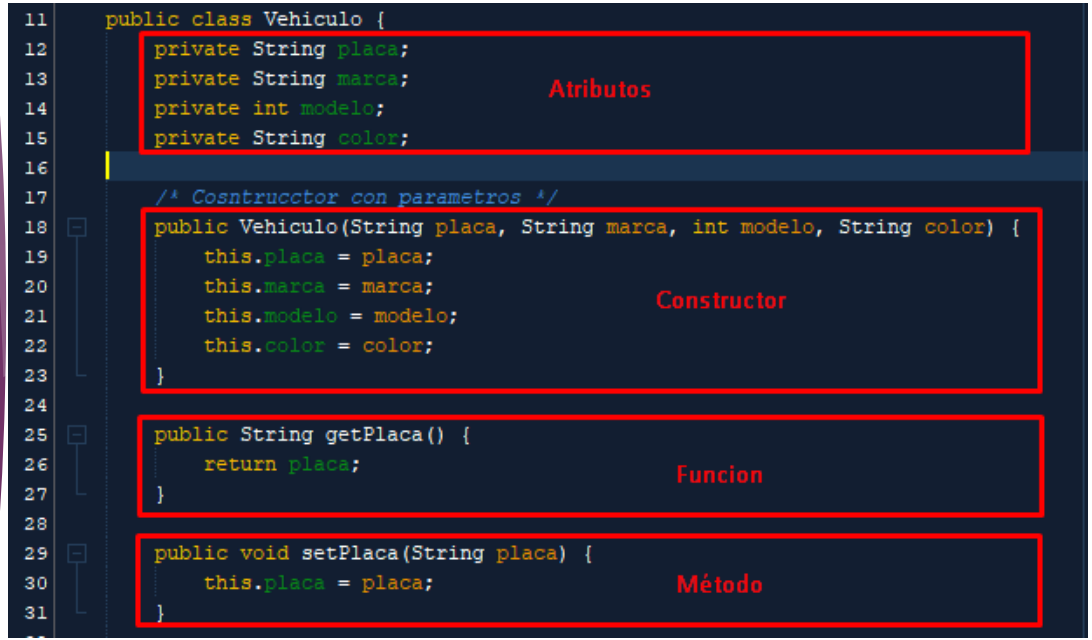
Ejemplo

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

```java
public class Vehiculo {
    private String placa;
    private String marca;
    private String modelo;
    private String color;

    public Vehiculo(String placa, String marca, String modelo, String color) {
        this.placa = placa;
        this.marca = marca;
        this.modelo = modelo;
        this.color = color;
    }

    public String getPlaca() {
        return placa;
    }

    public void setPlaca(String placa) {
        this.placa = placa;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
```

Los componentes 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 atributos, métodos o funciones, constructores.



Constructores con parámetros 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.

Ejemplo de constructores

```java
public class Vehiculo {
    private String placa;
    private String marca;
    private int modelo;
    private String color;

    /* Constructor vacio */
    public Vehiculo() {
        this.placa = "P123ABC";
        this.marca = "Toyota";
        this.modelo = 2013;
        this.color = "Negro";
    }
    /* Cosntrucctor con parametros */
    public Vehiculo(String placa, String marca, int modelo, String color) {
        this.placa = placa;
        this.marca = marca;
        this.modelo = modelo;
        this.color = color;
    }
```

Uso de constructores

- Para crear un objeto y asignarle valores iniciales, se utiliza la palabra clave new seguida del nombre del constructor
- Los constructores son una parte importante de la programación orientada a objetos en Java, y permiten crear objetos con valores iniciales personalizados o predeterminados.

Ejemplo

```
    public static void main(String[] args) {
        // TODO code application logic here
        Vehiculo vehiculo1 = new Vehiculo();
        Vehiculo vehiculo2 = new Vehiculo( placa: "P589GSK",  marca: "Mazda",  modelo: 2020,  color: "Azul");

        vehiculo1.mostrarDatos();
        vehiculo2.mostrarDatos();
    }

}
```

Salida en consola

```
Placa: P123ABC  Marca: Toyota   Modelo: 2013    Color: Negro
Placa: P589GSK  Marca: Mazda    Modelo: 2020    Color: Azul
```

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**
- **Protected**
- **Private**

Ejemplo

```
5   package Clases;
6
7   /**...4 lines */
11  public class Figura {
12      /* Variable que podra ser accedida desde cualquie parte
13          del codigo                                          */
14      public String descripcion;
15      /*  Variable que podra ser accedida desde:
16              - La clase a la que pertenece
17              - El mismo paquete
18              - Subclases                                     */
19      protected double area;
20      /* Variable que podra ser accedida solo en la clase a la
21          que pertenece                                       */
22      private int lados;
23
24      public Figura(double area, int lados, String descripcion) {
25          this.area = area;
26          this.lados = lados;
27          this.descripcion = descripcion;
28      }
29
```

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.

```
5   package ejemplos;
6
7   import Clases.Figura;
8   /**...4 lines */
12  public class Ejemplos {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18
19          Figura f = new Figura( area: 0,  lados: 0,  descripcion: "");
20          System.out.println("Descripcion de la figura "+f.descripcion);
21      }
22  }
```

protected

Los miembros declarados como protected son accesibles desde la clase en la que se declararon, el mismo paquete o 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.

```
5   package ejemplos;
6
7   import Clases.Figura;
8   /**...4 lines */
12  public class Ejemplos {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18
19          Figura f = new Figura( area: 0,  lados: 0,  descripcion: "");
20          System.out.println("Descripcion de la figura "+f.descripcion);
            System.out.println("Area de la figura "+f.area);
22      }
23  }
```

```
5   package Clases;
6
7   /**...4 lines */
11  public class ListaFiguras {
12
13      public ListaFiguras() {
14          Figura f = new Figura( area: 0,  lados: 0,  descripcion: "");
15          System.out.println("Area de la figura es "+f.area);
16      }
17  }
```

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.

```
5   package ejemplos;
6
7   import Clases.Figura;
8   /**...4 lines */
12  public class Ejemplos {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18
19          Figura f = new Figura( area: 0,  lados: 0,  descripcion: "");
            System.out.println("La cantidad de lados es "+f.lados);
21      }
22  }
```

```
5   package Clases;
6
7   /**...4 lines */
11  public class ListaFiguras {
12
13      public ListaFiguras() {
14          Figura f = new Figura( area: 0,  lados: 0,  descripcion: "");
            System.out.println("La cantidad de lados es "+f.lados);
16      }
17  }
```

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.

Excepciones

- Las excepciones son eventos inusuales o errores que pueden ocurrir durante la ejecución de un programa, y el manejo adecuado de estas excepciones es esencial para garantizar la robustez y la confiabilidad del software.
- las excepciones representan situaciones en las que el programa no puede continuar de manera regular debido a un problema o error. Estas situaciones pueden ser diversas, desde errores de programación hasta condiciones imprevistas en el entorno de ejecución.

Manejo de excepciones en java

- 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.

Try-catch

- **Try**: Bloque de sentencias que se utiliza para envolver el código que podría lanzar excepciones. Puedes tener uno o más bloques catch que manejen las excepciones específicas que pueden ocurrir.
- **Catch**: Bloque de sentencias que controla un tipo de error especifico.
- **Finally**: Bloque de sentencias, es opcional y se usa para contener el código que debe ejecutarse independientemente de si se lanza una excepción o no.

```
public static void main(String[] args) {

    try {
        /* Sentencisa a ejecutar */
    } catch (NullPointerException e) {
        /* Sentencisa a ejecutar */
    } catch (ArithmeticException e) {
        /* Sentencisa a ejecutar */
    } finally{
        /* Sentencisa a ejecutar */
    }
}
```

Excepciones

Excepción	Descripción
NullPointerException	Lanzada cuando intentas acceder a un objeto o método en una referencia null.
IllegalArgumentException	Lanzada cuando se pasa un argumento ilegal o inapropiado a un método.
ArrayIndexOutOfBoundsException	Lanzada cuando intentas acceder a un índice fuera del rango válido en un array.
ArithmeticException	Lanzada cuando ocurre un error aritmético, como la división por cero.
FileNotFoundException	Lanzada cuando intentas abrir o leer un archivo que no existe.
IOException	Excepción base para problemas de entrada/salida, como manipulación de archivos y flujos.
ClassNotFoundException	Lanzada cuando una clase no se encuentra durante la carga en tiempo de ejecución.
InterruptedException	Lanzada cuando un hilo es interrumpido mientras está en espera o durmiendo.
NumberFormatException	Lanzada al intentar convertir una cadena a un tipo numérico y la cadena no tiene el formato correcto.
RuntimeException	Clase base para excepciones no comprobadas, como IndexOutOfBoundsException y NullPointerException.

Ejemplo

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

    int numerador = 10;
    int denominador = 0;

    try {
        int resultado = numerador / denominador; // Intenta la división
        System.out.println("Resultado: " + resultado);
    } catch (ArithmeticException e) {
        System.out.println( x: "Error: División por cero no permitida.");
    }

    System.out.println( x: "Fin del programa.");
}
```

Importante

El manejo adecuado de excepciones es crucial para garantizar que tu programa sea resistente a fallos y proporcione mensajes de error útiles para los usuarios o desarrolladores. Debes considerar cuidadosamente cómo manejar y propagar las excepciones en tu código para lograr un comportamiento robusto y controlado.

Dudas y Preguntas

Parte práctica