Técnicas de programación avanzada

Curso 2023-2024

Tema 5.3 Expresiones avanzadas del lenguaje

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.

5.3 Expresiones avanzadas del lenguaje.

Clases anónimas

Las clases anónimas en Java son una forma de extender una clase existente o implementar una interfaz de manera rápida y sin necesidad de crear una subclase de forma explícita.

Son especialmente útiles cuando necesitas una implementación única de una interfaz o clase con pocos métodos, y no quieres la sobrecarga de crear una nueva clase en tu código.

```java
/**
 * Interfaz Boton que define la acción de presionar el botón.
 */
interface Boton {
    void presionar();
}
```

```java
public class Main {
    public static void main(String[] args) {
        /**
         * Creamos una instancia de Boton usando una clase anónima.
         * Aquí definimos el comportamiento del método presionar al vuelo.
         */
        Boton boton = new Boton() {
            @Override
            public void presionar() {
                System.out.println("La luz se enciende.");
            }
        };

        // Usamos el botón con su comportamiento único.
        boton.presionar();
    }
}
```

https://github.com/aalonsopuig/Java_Ejemplos_5/tree/main/src/f_clases_anonimas_simple_con

5.3 Expresiones avanzadas del lenguaje.

Clases anónimas

Sin clases anónimas

https://github.com/aalonsopuig/Java_Ejemplos_5/tree/main/src/f_clases_anonimas_simple_sin

```
interface Boton {
    void presionar();
}
```

```
class BotonLuz implements Boton {
    @Override
    public void presionar() {
        System.out.println("La luz se enciende.");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        // Creación de una instancia de BotonLuz
        Boton boton = new BotonLuz();
        boton.presionar();
    }
}
```

Con clases anónimas

https://github.com/aalonsopuig/Java_Ejemplos_5/tree/main/src/f_clases_anonimas_simple_con

```
interface Boton {
    void presionar();
}
```

```
public class Main {
    public static void main(String[] args) {
        Boton boton = new Boton() {
            @Override
            public void presionar() {
                System.out.println("La luz se enciende.");
            }
        };
        boton.presionar();
    }
}
```

Ventajas de las Clases Anónimas

- **Implementaciones concisas y claras**: Las clases anónimas permiten escribir implementaciones de interfaces o clases abstractas de manera concisa y clara, directamente en el lugar donde se necesitan. Esto puede hacer que el código sea más fácil de leer y mantener si la implementación es breve y se usa solo en ese lugar.

- **Menor Sobrecarga de Código**: Al usar clases anónimas, evitas la necesidad de crear una clase concreta completa, lo que puede ser útil para reducir la sobrecarga de tener demasiadas clases pequeñas y específicas en tu código.

- **Facilidad para Implementar Listeners y Manejadores de Eventos**: En el desarrollo de interfaces gráficas, las clases anónimas son muy útiles para implementar oyentes (listeners) y manejadores de eventos de manera rápida y directa.

- **Ideal para Implementaciones de Único Uso**: Si necesitas una implementación de una interfaz o una extensión de una clase que sabes que no se reutilizará en otra parte del código, una clase anónima puede ser la mejor opción.

5.3 Expresiones avanzadas del lenguaje.

Clases anónimas

Cuando Usar Clases Anónimas

- **Implementaciones Sencillas y de Único Uso**: Son ideales para implementaciones que son demasiado simples para justificar la creación de una clase concreta y que solo se usarán en un único lugar.
- **Desarrollo de Interfaces Gráficas de Usuario (GUI)**: En aplicaciones GUI, las clases anónimas son útiles para crear oyentes de eventos (como acciones de botones) de manera rápida y localizada.
- **Implementaciones Temporales o de Prueba**: Cuando estás experimentando o probando una implementación y no quieres crear una clase completa, las clases anónimas pueden ser una buena opción.

Consideraciones

- **Legibilidad**: Para implementaciones más complejas o largas, las clases anónimas pueden hacer que el código sea difícil de leer y mantener. En estos casos, es mejor usar clases internas o clases regulares.
- **Limitaciones**: Las clases anónimas no pueden tener constructores nombrados, no pueden extender múltiples clases, ni implementar múltiples interfaces, y no pueden ser reutilizadas.

Ejercicios

5.3 Expresiones avanzadas del lenguaje.

Ejercicios

Sistema de Control para la Colonización de la Luna con Clases Anónimas

Crear un programa en Java que utilice clases anónimas para simular un sistema de control en una misión de colonización de la Luna. El sistema controlará diferentes aspectos del viaje y la colonia lunar, como el mantenimiento de la nave espacial y la gestión de recursos en la colonia.

Interfaz de Control:

- Definir una interfaz ControlMision con un método ejecutar.
- El método ejecutar debe aceptar un parámetro String que describe la tarea a realizar.

Implementación con Clases Anónimas:

- Utilizar clases anónimas para implementar diferentes funciones del control de la misión.
- Crear al menos dos implementaciones distintas: una para el control de la nave y otra para la gestión de recursos en la Luna.

Ejecución de Tareas:

- En el método main, crear instancias de la interfaz ControlMision utilizando clases anónimas y ejecutar tareas específicas.

```
Controlando la nave para: mantenimiento del motor
Gestionando recursos lunares para: construcción de hábitat
```

5.3 Expresiones avanzadas del lenguaje.

Expresiones Lambda

Una expresión lambda es una bloque corto de código que toma parámetros y devuelve un valor. Las expresiones lambda son similares a los métodos, pero no necesitan un nombre y se pueden implementar directamente en el cuerpo de un método.

```
((parameter) -> {expression});
```

Puedes pensar en una expresión lambda como una forma corta de definir un método anónimo. Por ejemplo, imagina que tienes una lista de números y quieres filtrar solo los números pares. Sin lambda, tendrías que escribir una implementación completa de un interfaz para hacer esto. Con lambda, puedes hacerlo en una sola línea de código.

Lambda para Convertir un String a Mayúsculas:
Esta lambda toma un String como entrada y devuelve el mismo String en mayúsculas.

```
str -> str.toUpperCase()
```

Lambda para Sumar dos Números:
Esta lambda toma dos enteros a y b como entrada y devuelve su suma.

```
(int a, int b) -> a + b
```

Lambda para Comprobar si un Número es Par:
Esta lambda toma un número entero n como entrada y devuelve true si n es par, y false si no lo es.

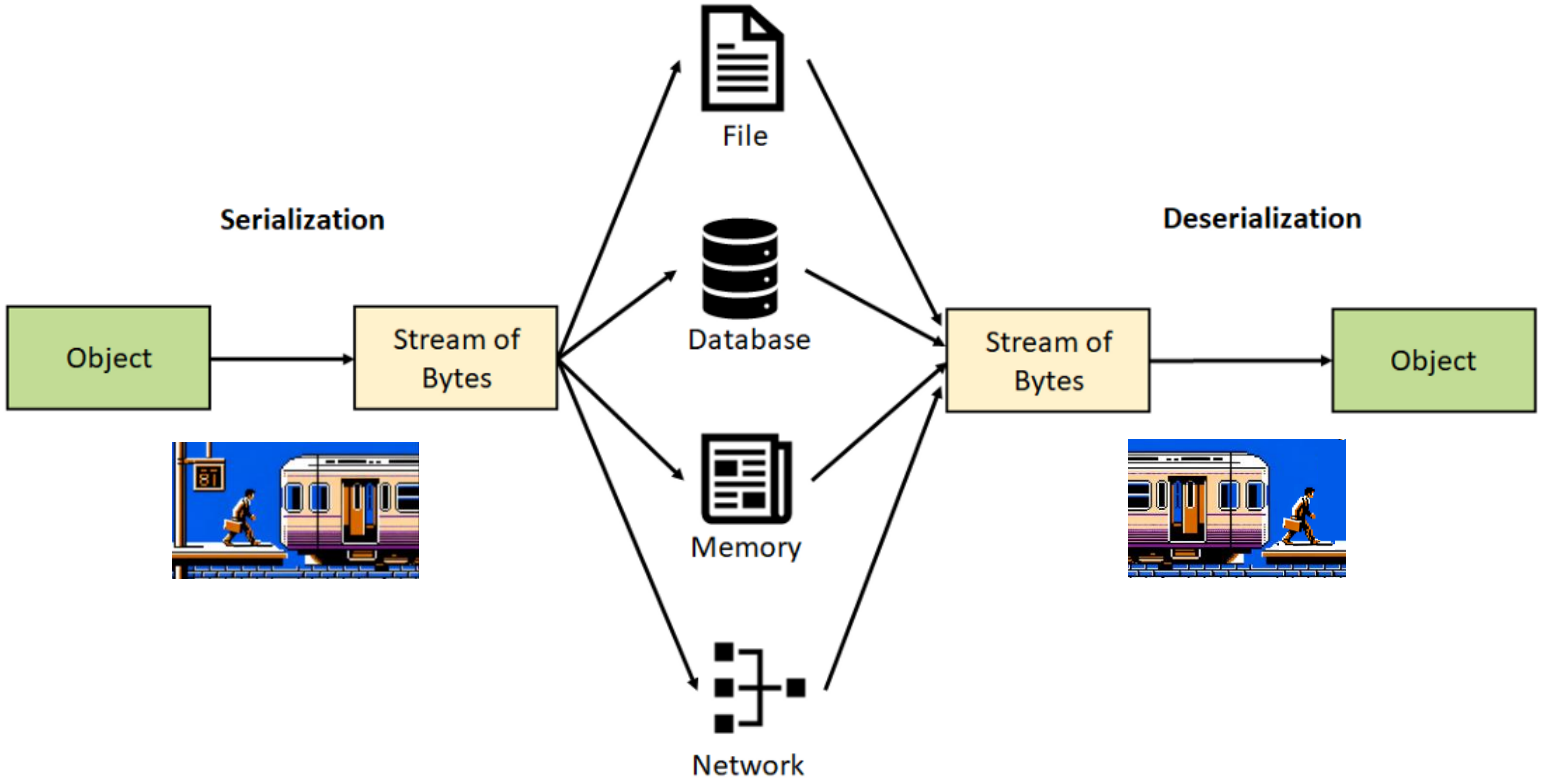
```
n -> n % 2 == 0
```

5.3 Expresiones avanzadas del lenguaje.

Clases serializables

Las clases serializables en Java permiten convertir objetos en una secuencia de bytes, lo que facilita su almacenamiento en archivos o transmisión a través de redes.

Ejemplo: *Piensa en un objeto en tu programa como un rompecabezas. La serialización es como tomar ese rompecabezas, guardarlo en una caja (convirtiéndolo en una secuencia de bytes) y luego poder abrir esa caja en otro lugar o en otro momento para reconstruir el rompecabezas (deserialización).*



5.3 Expresiones avanzadas del lenguaje.

Clases serializables

El proceso de serialización asocia una identificación con cada clase serializable que se conoce como ***SerialVersionUID***. Se utiliza para verificar el remitente y el receptor del objeto serializado, que deben tener el mismo SerialVersionUID; de lo contrario, se lanzará *InvalidClassException* cuando deserialice el objeto.

Para serializar un objeto, llamamos al método writeObject () de la clase ObjectOutputStream, y para la deserialización llamamos al método readObject () de la clase ObjectInputStream.

Ejemplo serialización

```java
import java.io.Serializable;

/**
 * Clase Persona que implementa la interfaz Serializable.
 * Esto permite que objetos de Persona se puedan serializar.
 */
class Persona implements Serializable {
    private static final long serialVersionUID = 1L;
    private String nombre;
    private int edad;

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

    // Métodos getters y setters
    public String getNombre() { return nombre; }
    public int getEdad() { return edad; }
}
```

```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SerializacionEjemplo {
    public static void main(String[] args) {
        Persona persona = new Persona("Juan", 30);

        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("persona.ser"))) {
            oos.writeObject(persona);
            System.out.println("Objeto Persona serializado con éxito.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
Objeto Persona serializado con éxito.
```

https://github.com/aalonsopuig/Java_Ejemplos_5/tree/main/src/h_clases_serializables_ejemplo

5.3 Expresiones avanzadas del lenguaje.

Clases serializables

Ejemplo serialización

```java
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class DeserializacionEjemplo {
    public static void main(String[] args) {
        // Nombre del archivo donde está almacenado el objeto serializado
        String nombreArchivo = "persona.ser";

        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nombreArchivo))) {
            // Leyendo el objeto desde el archivo y convirtiéndolo de nuevo en un objeto Persona
            Persona personaDeserializada = (Persona) ois.readObject();

            // Mostrando información del objeto deserializado
            System.out.println("Objeto Persona deserializado con éxito.");
            System.out.println("Nombre: " + personaDeserializada.getNombre());
            System.out.println("Edad: " + personaDeserializada.getEdad());
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

https://github.com/aalonsopuig/Java_Ejemplos_5/tree/main/src/h_clases_serializables_ejemplo

Para la deserialización llamamos al método readObject () de la clase ObjectInputStream.

```
Objeto Persona deserializado con éxito.
Nombre: Juan
Edad: 30
```

5.3 Expresiones avanzadas del lenguaje.

Clases serializables

Ventajas de las Clases Serializables

- **Persistencia de Datos**: Permite guardar objetos en un medio persistente como archivos o bases de datos binarias para su uso posterior.
- **Comunicación a través de Redes**: Facilita el envío de objetos entre aplicaciones a través de una red, como en el caso de aplicaciones cliente-servidor.

Cuándo Usar Serialización

- **Guardar Estado de Objetos**: Cuando necesitas guardar el estado de un objeto para recuperarlo más tarde.
- **Comunicación entre Aplicaciones**: Para enviar objetos entre diferentes aplicaciones o servicios, especialmente en aplicaciones distribuidas.

Consideraciones

- **Seguridad**: La serialización puede presentar problemas de seguridad, como la ejecución de código malicioso durante la deserialización. Se debe tener cuidado con lo que se serializa y deserializa.
- **Compatibilidad de Versiones**: Si cambias la estructura de una clase serializable, puede que no sea compatible con versiones anteriores o posteriores.
- **Rendimiento**: La serialización y deserialización pueden ser procesos costosos en términos de rendimiento y uso de memoria.

Ejercicios

5.3 Expresiones avanzadas del lenguaje.

Ejercicios

Gestión de Motocicletas con Serialización

Desarrollar un programa en Java que demuestre el proceso de serialización y deserialización utilizando una clase que representa una motocicleta. El programa deberá ser capaz de guardar el estado de un objeto Motocicleta en un archivo y luego recuperarlo..

Definir la Clase Motocicleta:

- La clase **Motocicleta** debe tener al menos tres atributos: **marca**, **modelo** y **cilindrada**.
- Implementar métodos **getters** para cada atributo.
- Incluir un método **toString** para mostrar la información de la motocicleta.
- La clase debe implementar la interfaz **Serializable**.

Serializar el Objeto Motocicleta:

- Crear una instancia de **Motocicleta** con valores específicos.
- Serializar el objeto en un archivo denominado **motocicleta.ser**.

Deserializar el Objeto Motocicleta:

- Leer el archivo **motocicleta.ser**.
- Deserializar el objeto y mostrar su estado recuperado en consola.

```
Motocicleta deserializada con éxito: Motocicleta{marca='Yamaha', modelo='YZF-R1', cilindrada=1000}
```

Bibliografía

- Deitel, H. M. & Deitel, P. J.(2008). *Java: como programar*. Pearson education. Séptima edición.
- 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!
```