

Universidad Autónoma de Nuevo León.
Facultad de Ciencias Físico Matemáticas.

Práctica #07 – Manejo de excepciones.

Laboratorio de Programación Orientada a Objetos.

Alumno: Jeremy Uriel Rossell Segura

Matrícula: 2173396

Materia: Laboratorio de Programación Orientada a Objetos.

Docente: Jorge Alberto Islas Pineda

Compilación

El programa compiló adecuadamente.

```
import java.util.ArrayList;
public class Main {
    public static void main(String[] args) {
        ArrayList<Transporte> transportes = new ArrayList<>();
        // objetos con distintas capacidades
        Transporte mar = new TransporteMaritimo(id: "BAR-01", capacidad: 50);
        Transporte tierra = new TransporteTerrestre(id: "BUS-22", capacidad: 50);
        Transporte aire = new TransporteAereo(id: "AV-777", capacidad: 10);
        Transporte tren = new TransporteFerrovial(id: "TRN-05", capacidad: 10);
    }
}

public abstract class Transporte {
    private String id; 3 usages
    private int capacidad; 4 usages
    public Transporte(String id, int capacidad) { 4 usages
        setId(id);
        setCapacidad(capacidad);
    }
    public String getId() { no usages
        return id;
    }
}

public abstract class Transporte {
    private String id; 3 usages
    private int capacidad; 4 usages
    public Transporte(String id, int capacidad) { 4 usages
        setId(id);
        setCapacidad(capacidad);
    }
    public String getId() { no usages
        return id;
    }
}

Process finished with exit code 0
```

Depuración

El programa se depuró adecuadamente.

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

public abstract class Transporte {
    private String id; 3 usages
    private int capacidad; 4 usages
    public Transporte(String id, int capacidad) { 4 usages
        setId(id);
        setCapacidad(capacidad);
    }
    public String getId() { no usages
        return id;
    }
}

public abstract class Transporte {
    private String id; 3 usages
    private int capacidad; 4 usages
    public Transporte(String id, int capacidad) { 4 usages
        setId(id);
        setCapacidad(capacidad);
    }
    public String getId() { no usages
        return id;
    }
}

Process finished with exit code 0
```

Código

Main.java y Transporte.java.

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<Transporte> transportes = new ArrayList<>();

        // objetos con distintas capacidades
        Transporte mar = new TransporteMaritimo(id: "BAR-01", capacidad: 100);
        Transporte tierra = new TransporteTerrestre(id: "BUS-22", capacidad: 50);
        Transporte aire = new TransporteAereo(id: "AV-777", capacidad: 120);
        Transporte tren = new TransporteFerroviario(id: "TRN-05", capacidad: 200); // ...

        // polimorfismo
        transportes.add(mar);
        transportes.add(tierra);
        transportes.add(aire);
        transportes.add(tren);

        int pasajeros = 60;

        for (Transporte t : transportes) {
            t.mover();
            try {
                t.transportar(pasajeros);
            } catch (CapacidadExcedidaException e) {
                System.out.println("[!] ERROR: " + e.getMessage());
            } finally {
                if (t instanceof Operable op) {
                    op.realizarMantenimiento();
                }
            }
            System.out.println("----");
        }
    }
}
```

```
public abstract class Transporte { 4 inheritors
    public void setId(String id) {
        ...
        this.id = id;
    }

    public int getCapacidad() {
        return capacidad;
    }

    public void setCapacidad(int capacidad) {
        if (capacidad <= 0) {
            throw new IllegalArgumentException("[!] La capacidad debe ser mayor que 0");
        }
        if (capacidad > 500) {
            // para evitar valores ilógicos o irreales
            throw new IllegalArgumentException("[!] Capacidad no válida (>500). Revisa tu código.");
        }
        this.capacidad = capacidad;
    }

    public abstract String tipo(); 4 implementations
    public abstract void mover(); 4 implementations

    public void transportar(int pasajeros) throws CapacidadExcedidaException {
        if (pasajeros > capacidad) {
            throw new CapacidadExcedidaException(
                "[!] Pasajeros (" + pasajeros + ") exceden capacidad (" + capacidad + ")"
            );
        } else {
            System.out.println("[!] Transportando " + pasajeros + " pasajeros en " + tipo());
        }
    }
}
```

CapacidadExcedidaException, Operable, & BasicExceptions.

```
public class CapacidadExcedidaException extends Exception {
    public CapacidadExcedidaException(String mensaje) {
        super(mensaje);
    }
}
```

```
public interface Operable {
    void realizarMantenimiento();
}
```

```
public class BasicExceptions {
    public int division(int a, int b) {
        return a / b;
    }

    public void age(int a) throws Exception {
        if (a < 0) {
            throw new Exception("Algo anda mal");
        }
    }
}
```

El resto de las clases de los diferentes tipos de transportes.

```
Main.java
public class Main {
    public static void main(String[] args) {
        TransporteAereo avion = new TransporteAereo("Avión 1", 10);
        TransporteFerrovial tren = new TransporteFerrovial("Tren 1", 10);
        TransporteMaritimo barco = new TransporteMaritimo("Barco 1", 10);
        TransporteTerrestre camion = new TransporteTerrestre("Camión 1", 10);

        System.out.println(avion.tipo());
        System.out.println(tren.tipo());
        System.out.println(barco.tipo());
        System.out.println(camion.tipo());

        avion.mover();
        tren.mover();
        barco.mover();
        camion.mover();

        avion.realizarMantenimiento();
        tren.realizarMantenimiento();
        barco.realizarMantenimiento();
        camion.realizarMantenimiento();
    }
}

TransporteAereo.java
public class TransporteAereo extends Vehiculo {
    public TransporteAereo(String id, int capacidad) {
        super(id, capacidad);
    }

    @Override 2 usages
    public String tipo() {
        return "Aéreo";
    }

    @Override 1 usage
    public void mover() {
        System.out.println("Volando en el aire");
    }

    @Override 1 usage
    public void realizarMantenimiento() {
        System.out.println("Mantenimiento aéreo");
    }
}

TransporteFerrovial.java
public class TransporteFerrovial extends Vehiculo {
    public TransporteFerrovial(String id, int capacidad) {
        super(id, capacidad);
    }

    @Override 2 usages
    public String tipo() {
        return "Ferrovíario";
    }

    @Override 1 usage
    public void mover() {
        System.out.println("Circulariendo sobre rieles");
    }

    @Override 1 usage
    public void realizarMantenimiento() {
        System.out.println("Mantenimiento ferroviario");
    }
}

TransporteMaritimo.java
public class TransporteMaritimo extends Vehiculo {
    public TransporteMaritimo(String id, int capacidad) {
        super(id, capacidad);
    }

    @Override 2 usages
    public String tipo() {
        return "Marítimo";
    }

    @Override 1 usage
    public void mover() {
        System.out.println("Navegando en el mar");
    }

    @Override 1 usage
    public void realizarMantenimiento() {
        System.out.println("Mantenimiento marítimo");
    }
}

TransporteTerrestre.java
public class TransporteTerrestre extends Vehiculo {
    public TransporteTerrestre(String id, int capacidad) {
        super(id, capacidad);
    }

    @Override 2 usages
    public String tipo() {
        return "Terrestre";
    }

    @Override 1 usage
    public void mover() {
        System.out.println("Rodando sobre ruedas");
    }

    @Override 1 usage
    public void realizarMantenimiento() {
        System.out.println("Mantenimiento terrestre");
    }
}
```