Guia de Repaso

Herencia, Clases Abstractas e Interfaces

Semana 01 - Martes 10 de Febrero de 2026

Academia Java - Ciudad de Mexico

Temas del dia:

1. Herencia con clases abstractas
2. Polimorfismo con List y herencia
3. Polimorfismo avanzado: clases abstractas
4. Interfaces en Java
5. Clase abstracta vs Interfaz

Instructor: Miguel Rugerio

1. Herencia con Clases Abstractas

La herencia permite que una clase hija extienda a una clase padre, heredando sus campos y metodos. Cuando la clase padre es abstracta, define un contrato que las subclases DEBEN cumplir implementando los metodos abstractos.

Proyecto: herencia / v1

Se crea una clase abstracta Operacion con un metodo abstracto ejecuta(int x, int y). Tres subclases concretas implementan la operacion correspondiente: Suma, Division y Multiplicacion.

Jerarquia de clases

```
Operacion (abstract)
          abstract int ejecuta(int x, int y)
           /         |            \
        Suma      Division   Multiplicacion
       x + y       x / y         x * y
```

Clase abstracta: Operacion.java

```
public abstract class Operacion {

    abstract int ejecuta(int x, int y);

}
```

> Clase abstracta: No se puede instanciar directamente (new Operacion() da error de compilacion). Solo sirve como plantilla para las subclases.

Subclases concretas

```
public class Suma extends Operacion {
    @Override
    int ejecuta(int x, int y) {
        return x + y;       // Implementa la suma
    }
}

public class Division extends Operacion {
    @Override
    int ejecuta(int x, int y) {
        return x / y;       // Implementa la division
    }
}

public class Multiplicacion extends Operacion {
    @Override
    int ejecuta(int x, int y) {
        return x * y;       // Implementa la multiplicacion
    }
}
```

! Cada subclase DEBE implementar ejecuta(). Si no lo hace, la subclase tambien debe declararse abstract.

2. Polimorfismo con List y Herencia

El metodo Principal demuestra como usar una List<Operacion> para tratar todas las subclases de forma uniforme. Esto es polimorfismo en accion.

Proyecto: herencia / v1 - Principal.java

```
public class Principal {

    public static void main(String[] args) {
        List<Operacion> operaciones = new ArrayList<>();

        Operacion ope1 = new Suma();           // Upcasting
        Operacion ope2 = new Division();       // Upcasting
        Operacion ope3 = new Multiplicacion(); // Upcasting

        operaciones.add(ope3);  // Multiplicacion
        operaciones.add(ope1);  // Suma
        operaciones.add(ope2);  // Division

        show(operaciones);
    }

    static void show(List<Operacion> operaciones) {
        for (Operacion o : operaciones) {
            System.out.println(o.getClass().getSimpleName());
            System.out.println(o.ejecuta(8, 4));
        }
    }
}
```

Salida del programa

```
Multiplicacion
32
Suma
12
Division
2
```

> Reflexion con getClass(): o.getClass().getSimpleName() obtiene el nombre de la clase REAL del objeto en tiempo de ejecucion, no el tipo de la variable.

Conceptos clave

Concepto	Ejemplo en el codigo	Explicacion
Clase abstracta	abstract class Operacion	Define contrato, no se instancia
Metodo abstracto	abstract int ejecuta()	Sin cuerpo, las hijas implementan
Herencia	class Suma extends Operacion	Suma hereda de Operacion
@Override	@Override int ejecuta()	Garantiza sobreescritura correcta
Polimorfismo	List<Operacion>	Lista con diferentes subclases
Dynamic dispatch	o.ejecuta(8, 4)	Ejecuta metodo del objeto real

3. Polimorfismo Avanzado: Clases Abstractas

En la version 4 del proyecto polimorfismo, se usa una clase abstracta Ave como tipo base. Se combina con arreglos polimorficos y seleccion aleatoria para demostrar dynamic dispatch en un escenario mas real.

Proyecto: polimorfismo / v4

Clase abstracta Ave

```
public abstract class Ave {

    abstract void volar();  // Contrato: toda Ave debe definir como vuela

}
```

Subclases concretas

```
public class Pato extends Ave {
    void volar() {
        System.out.println("Volar Pato");
    }
}

public class Perico extends Ave {
    void volar() {
        System.out.println("Volar Perico");
    }
}

public class Pinguino extends Ave {
    void volar() {
        System.out.println("No volar Pinguino");  // No vuela!
    }
}

public class Murcielago extends Ave {
    void volar() {
        System.out.println("Volar Murcielago");
    }
}
```

! No se puede hacer new Ave() porque es abstracta. La linea //new Ave(), //MAL esta comentada en el codigo para recordar esta restriccion.

Principal.java - Arreglo polimorfico + Random

```
public class Principal {

    public static void main(String[] args) {
        System.out.println("Abstract");
        Ave ave = getAve();           // Retorna un Ave aleatorio
        comportamientoVolar(ave);     // Ejecuta volar() del real
    }

    static void comportamientoVolar(Ave ave) {
        ave.volar();  // Dynamic dispatch
    }

    private static Ave getAve() {
        Ave[] aves = {new Pato(),
                      //new Ave(),  // MAL: no instanciar abstract
                      new Murcielago(),
                      new Perico(),
                      new Pinguino()
        };
        int aleatorio = new Random().nextInt(aves.length);
        return aves[aleatorio];
    }
}
```

Arreglo polimorfico: Ave[] puede contener cualquier subclase de Ave mezclada. El tipo del arreglo es el padre, pero cada posicion contiene un objeto concreto diferente.

Flujo de ejecucion

Paso	Que sucede
1	getAve() crea un arreglo con 4 subclases de Ave
2	Random genera un indice aleatorio (0 a 3)
3	Se retorna el Ave en esa posicion (tipo real desconocido)
4	comportamientoVolar() recibe el Ave y llama volar()
5	Java ejecuta el volar() del objeto REAL (dynamic dispatch)

Cada ejecucion puede producir una salida diferente:

```
Abstract
Volar Perico   // o "Volar Pato", "Volar Murcielago", "No volar Pinguino"
```

4. Interfaces en Java

Una interfaz define un contrato puro: solo declara metodos (sin implementacion). Las clases que la implementan usan implements en lugar de extends.

Proyecto: polimorfismo / v5

Se refactoriza el proyecto v4 cambiando la clase abstracta Ave por una interfaz. Las subclases ahora implementan la interfaz en lugar de extender una clase.

Interfaz Ave

```
public interface Ave {

    //public abstract       // Todos los metodos de una interfaz son
    void volar();           // public abstract por defecto

}
```

> Metodos de interfaz: En una interfaz, todos los metodos son public abstract por defecto. No es necesario escribirlo explicitamente, pero Java lo asume.

Implementaciones con implements

```
public class Pato implements Ave {        // implements, no extends
    public void volar() {                 // DEBE ser public
        System.out.println("Volar Pato");
    }
}

public class Perico implements Ave {
    public void volar() {
        System.out.println("Volar Perico");
    }
}

public class Pinguino implements Ave {
    public void volar() {
        System.out.println("No volar Pinguino");
    }
}

public class Murcielago implements Ave {
    public void volar() {
        System.out.println("Volar Murcielago");
    }
}
```

! Al implementar una interfaz, los metodos DEBEN ser public. En la clase abstracta (v4) no era necesario porque el metodo original tenia visibilidad default (package-private).

Principal.java - Identico comportamiento

```
public class Principal {

    public static void main(String[] args) {
        System.out.println("Interface"); // Indica que usa interfaz
        Ave ave = getAve();
        comportamientoVolar(ave);
    }

    static void comportamientoVolar(Ave ave) {
        ave.volar();    // Mismo dynamic dispatch que v4
    }

    private static Ave getAve() {
        Ave[] aves = {new Pato(),
                      //new Ave(),  // MAL: no instanciar interfaz
                      new Murcielago(),
                      new Perico(),
                      new Pinguino()
        };
        int aleatorio = new Random().nextInt(aves.length);
        return aves[aleatorio];
    }
}
```

Transparencia polimorfica: El codigo de Principal es practicamente identico en v4 y v5. El cambio de clase abstracta a interfaz no afecta al codigo que usa el tipo Ave. Esto demuestra el poder del polimorfismo.

5. Clase Abstracta vs Interfaz

La diferencia principal entre v4 y v5 es como se define el tipo base Ave. El comportamiento polimorfico es el mismo, pero la eleccion tiene implicaciones de diseno.

Cambios entre v4 y v5

Aspecto	v4 (Clase abstracta)	v5 (Interfaz)
Definicion de Ave	abstract class Ave	interface Ave
Subclases	extends Ave	implements Ave
Visibilidad del metodo	Default (package-private)	public (obligatorio)
Instanciar Ave?	NO (es abstract)	NO (es interface)
Herencia multiple?	NO (solo 1 extends)	SI (multiples implements)

Comparativa general

Caracteristica	Clase abstracta	Interfaz
Palabra clave	abstract class	interface
Herencia	extends (solo 1)	implements (multiples)
Metodos concretos	SI puede tener	Solo con default (Java 8+)
Constructores	SI puede tener	NO
Campos de instancia	SI	Solo constantes (static final)
Cuando usarla	IS-A con codigo compartido	Contrato puro, capacidades

! Regla practica: usa interfaz cuando solo necesitas definir un contrato (que hacer). Usa clase abstracta cuando ademas necesitas compartir codigo comun (como hacerlo) entre las subclases.

6. Resumen del Dia

Proyecto	Tema principal	Concepto clave
herencia v1	Herencia + clases abstractas	abstract class, @Override, List
polimorfismo v4	Polimorfismo con abstract	Arreglos polimorficos, Random
polimorfismo v5	Polimorfismo con interfaces	interface, implements, public

Progresion del polimorfismo en el curso

Version	Concepto	Novedad
v0	Polimorfismo basico	Upcasting automatico
v1	Variable polimorfica	Una referencia, multiples tipos
v2	Parametros polimorficos	Metodo generico con tipo padre
v3	Arreglos + Random	Ave como clase concreta
v4 (nuevo)	Clase abstracta	Ave ya no se puede instanciar
v5 (nuevo)	Interfaz	Ave es contrato puro

Estructura de los proyectos

```
herencia/
  src/com/curso/v1/
    Operacion.java       -- Clase abstracta (padre)
    Suma.java            -- Subclase: x + y
    Division.java        -- Subclase: x / y
    Multiplicacion.java  -- Subclase: x * y
    Principal.java       -- Demo con List polimorfico

polimorfismo/
  src/com/curso/v4/
    Ave.java             -- Clase ABSTRACTA
    Pato.java            -- extends Ave
    Perico.java          -- extends Ave
    Pinguino.java        -- extends Ave
    Murcielago.java      -- extends Ave
    Principal.java       -- Arreglo polimorfico + Random

  src/com/curso/v5/
    Ave.java             -- INTERFAZ
    Pato.java            -- implements Ave
    Perico.java          -- implements Ave
    Pinguino.java        -- implements Ave
    Murcielago.java      -- implements Ave
    Principal.java       -- Mismo comportamiento que v4
```

Para repasar

1. Compara Operacion.java (herencia) con Ave.java (polimorfismo v4): ambas son abstractas pero con diferente proposito.

2. Compara v4 (extends) con v5 (implements): observa los cambios minimos necesarios.

3. Intenta agregar una nueva operacion (ej. Resta) al proyecto herencia sin modificar Principal.java.

4. Intenta agregar una nueva ave al proyecto polimorfismo y observa que Principal la soporta automaticamente.

5. Preguntate: si Ave necesitara un campo nombre, podria ser interfaz? (Pista: no directamente).