Herencia

En este conjunto de ejercicios, exploraremos la creación de jerarquías de clases utilizando herencia y la implementación de métodos sobrescritos en clases derivadas.

Ejercicio 1: Definición de una jerarquía de vehículos

Objetivo: Crear una jerarquía de clases que represente vehículos con características comunes.

1. Define una clase base `Vehiculo` con atributos como `marca`, `modelo` y `velocidadMaxima`.
2. Crea una subclase `Automovil` con un atributo adicional `cantidadPuertas`.
3. Crea otra subclase `Motocicleta` con un atributo adicional `tipoManillar`.
4. Sobrescribe el método `toString()` en cada clase para mostrar la información relevante.
5. Implementa una clase de prueba para instanciar objetos y mostrar sus características.

```
class Vehiculo {
    protected String marca;
    protected String modelo;
    protected int velocidadMaxima;

    public Vehiculo(String marca, String modelo, int velocidadMaxima) {
        this.marca = marca;
        this.modelo = modelo;
        this.velocidadMaxima = velocidadMaxima;
    }

    @Override
    public String toString() {
        return "Marca: " + marca + ", Modelo: " + modelo + ", Velocidad Máxima: "
+ velocidadMaxima + " km/h";
    }
}

class Automovil extends Vehiculo {
    private int cantidadPuertas;

    public Automovil(String marca, String modelo, int velocidadMaxima, int
cantidadPuertas) {
        super(marca, modelo, velocidadMaxima);
        this.cantidadPuertas = cantidadPuertas;
    }

    @Override
    public String toString() {
        return super.toString() + ", Puertas: " + cantidadPuertas;
    }
}

class Motocicleta extends Vehiculo {
```

```
    private String tipoManillar;

    public Motocicleta(String marca, String modelo, int velocidadMaxima, String
tipoManillar) {
        super(marca, modelo, velocidadMaxima);
        this.tipoManillar = tipoManillar;
    }

    @Override
    public String toString() {
        return super.toString() + ", Tipo de Manillar: " + tipoManillar;
    }
}

public class Main {
    public static void main(String[] args) {
        Automovil auto = new Automovil("Toyota", "Corolla", 180, 4);
        Motocicleta moto = new Motocicleta("Honda", "CBR", 220, "Deportivo");
        System.out.println(auto);
        System.out.println(moto);
    }
}
```

Ejercicio 2: Animales y sonidos

Objetivo: Comprender cómo sobrescribir métodos en clases derivadas.

1. Crea una clase `Animal` con un método `hacerSonido()`, que retorne un sonido genérico.
2. Crea subclases `Perro` y `Gato`.
3. Sobrescribe `hacerSonido()` para que el perro devuelva "Guau" y el gato "Miau".
4. Implementa una clase de prueba para crear instancias y llamar a `hacerSonido()`.

```
class Animal {
    public String hacerSonido() {
        return "Sonido genérico";
    }
}

class Perro extends Animal {
    @Override
    public String hacerSonido() {
        return "Guau";
    }
}

class Gato extends Animal {
    @Override
    public String hacerSonido() {
        return "Miau";
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Animal perro = new Perro();
        Animal gato = new Gato();
        System.out.println(perro.hacerSonido());
        System.out.println(gato.hacerSonido());
    }
}
```

Ejercicio 3: Figuras geométricas

Objetivo: Aplicar herencia en un contexto matemático.

1. Crea una clase `Figura` con un método `calcularArea()`, que retorne `0` por defecto.
2. Crea subclases `Circulo` y `Rectangulo`.
3. En `Circulo`, implementa `calcularArea()` con la fórmula `π * radio^2`.
4. En `Rectangulo`, implementa `calcularArea()` con la fórmula `base * altura`.
5. Implementa una clase de prueba para crear objetos y calcular áreas.

```
abstract class Figura {
    public abstract double calcularArea();
}

class Circulo extends Figura {
    private double radio;

    public Circulo(double radio) {
        this.radio = radio;
    }

    @Override
    public double calcularArea() {
        return Math.PI * Math.pow(radio, 2);
    }
}

class Rectangulo extends Figura {
    private double base, altura;

    public Rectangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return base * altura;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Figura circulo = new Circulo(5);
        Figura rectangulo = new Rectangulo(4, 6);
        System.out.println("Área del círculo: " + circulo.calcularArea());
        System.out.println("Área del rectángulo: " + rectangulo.calcularArea());
    }
}
```

Ejercicio 4: Empleados y sueldos

Objetivo: Manejo de herencia en un contexto empresarial.

1. Define una clase `Empleado` con atributos `nombre` y `salarioBase`.
2. Crea subclases `Gerente` y `Desarrollador`, agregando bonificaciones específicas.
3. Sobrescribe un método `calcularSalarioTotal()` en cada subclase.
4. Crea una clase de prueba para calcular el salario total de diferentes empleados.

```
class Empleado {
    protected String nombre;
    protected double salarioBase;

    public Empleado(String nombre, double salarioBase) {
        this.nombre = nombre;
        this.salarioBase = salarioBase;
    }

    public double calcularSalarioTotal() {
        return salarioBase;
    }
}

class Gerente extends Empleado {
    private double bonificacion;

    public Gerente(String nombre, double salarioBase, double bonificacion) {
        super(nombre, salarioBase);
        this.bonificacion = bonificacion;
    }

    @Override
    public double calcularSalarioTotal() {
        return salarioBase + bonificacion;
    }
}

class Desarrollador extends Empleado {
    private double bonoTecnologico;

    public Desarrollador(String nombre, double salarioBase, double
```

```
bonoTecnologico) {
        super(nombre, salarioBase);
        this.bonoTecnologico = bonoTecnologico;
    }

    @Override
    public double calcularSalarioTotal() {
        return salarioBase + bonoTecnologico;
    }
}

public class Main {
    public static void main(String[] args) {
        Empleado gerente = new Gerente("Carlos", 3000, 1000);
        Empleado desarrollador = new Desarrollador("Ana", 2500, 500);
        System.out.println("Salario del gerente: " +
gerente.calcularSalarioTotal());
        System.out.println("Salario del desarrollador: " +
desarrollador.calcularSalarioTotal());
    }
}
```

Ejercicio 5: Biblioteca y libros

Objetivo: Aplicar herencia en el contexto de gestión de libros.

1. Crea una clase `Libro` con atributos `titulo`, `autor` y `anioPublicacion`.
2. Crea subclases `LibroFisico` y `LibroDigital`, con atributos específicos como `numeroPaginas` y `tamanoArchivo`.
3. Sobrescribe un método `obtenerDescripcion()` en cada subclase.
4. Implementa una clase de prueba que cree libros y muestre sus descripciones.

```
class Libro {
    protected String titulo;
    protected String autor;
    protected int anioPublicacion;

    public Libro(String titulo, String autor, int anioPublicacion) {
        this.titulo = titulo;
        this.autor = autor;
        this.anioPublicacion = anioPublicacion;
    }

    public String obtenerDescripcion() {
        return titulo + " de " + autor + ", publicado en " + anioPublicacion;
    }
}

class LibroFisico extends Libro {
    private int numeroPaginas;
```

```
    public LibroFisico(String titulo, String autor, int anioPublicacion, int
numeroPaginas) {
        super(titulo, autor, anioPublicacion);
        this.numeroPaginas = numeroPaginas;
    }

    @Override
    public String obtenerDescripcion() {
        return super.obtenerDescripcion() + ", " + numeroPaginas + " páginas";
    }
}

class LibroDigital extends Libro {
    private double tamanoArchivo;

    public LibroDigital(String titulo, String autor, int anioPublicacion, double
tamanoArchivo) {
        super(titulo, autor, anioPublicacion);
        this.tamanoArchivo = tamanoArchivo;
    }

    @Override
    public String obtenerDescripcion() {
        return super.obtenerDescripcion() + ", tamaño de archivo: " +
tamanoArchivo + "MB";
    }
}

public class BibliotecaTest {
    public static void main(String[] args) {
        LibroFisico libro1 = new LibroFisico("El Quijote", "Cervantes", 1605,
500);
        LibroDigital libro2 = new LibroDigital("1984", "George Orwell", 1949,
1.5);

        System.out.println(libro1.obtenerDescripcion());
        System.out.println(libro2.obtenerDescripcion());
    }
}
```

Ejercicio 6: Instrumentos musicales

Objetivo: Sobrescribir métodos para diferenciar comportamientos.

1. Crea una clase base `Instrumento` con un método `tocar()`.
2. Crea subclases `Guitarra` y `Piano`.
3. Sobrescribe `tocar()` para que cada instrumento emita un sonido diferente.
4. Crea una clase de prueba para demostrar la funcionalidad.

```
class Instrumento {
    public void tocar() {
```

```
        System.out.println("Sonido genérico de un instrumento");
    }
}

class Guitarra extends Instrumento {
    @Override
    public void tocar() {
        System.out.println("La guitarra suena con acordes vibrantes");
    }
}

class Piano extends Instrumento {
    @Override
    public void tocar() {
        System.out.println("El piano suena con notas melódicas");
    }
}

public class InstrumentoTest {
    public static void main(String[] args) {
        Instrumento guitarra = new Guitarra();
        Instrumento piano = new Piano();

        guitarra.tocar();
        piano.tocar();
    }
}
```

Ejercicio 7: Vehículos eléctricos y de combustión

Objetivo: Diferenciar tipos de vehículos con herencia.

1. Crea una clase `Vehiculo` con un método `obtenerTipoCombustible()`.
2. Crea subclases `Electrico` y `Combustion`.
3. Sobrescribe `obtenerTipoCombustible()` para que devuelva "Electricidad" o "Gasolina".
4. Implementa una clase de prueba para comprobar el comportamiento.

```
class Vehiculo {
    public String obtenerTipoCombustible() {
        return "Desconocido";
    }
}

class Electrico extends Vehiculo {
    @Override
    public String obtenerTipoCombustible() {
        return "Electricidad";
    }
}

class Combustion extends Vehiculo {
```

```
    @Override
    public String obtenerTipoCombustible() {
        return "Gasolina";
    }
}

public class VehiculoTest {
    public static void main(String[] args) {
        Vehiculo tesla = new Electrico();
        Vehiculo ford = new Combustion();

        System.out.println("Tesla usa: " + tesla.obtenerTipoCombustible());
        System.out.println("Ford usa: " + ford.obtenerTipoCombustible());
    }
}
```

Ejercicio 8: Usuarios y administradores

Objetivo: Aplicar herencia en la gestión de usuarios.

1. Crea una clase `Usuario` con atributos `nombre` y `correo`.
2. Crea una subclase `Administrador` que tenga un atributo `nivelAcceso`.
3. Sobrescribe un método `obtenerDetalles()` en `Administrador`.
4. Implementa una clase de prueba con diferentes tipos de usuarios.

```
class Usuario {
    protected String nombre;
    protected String correo;

    public Usuario(String nombre, String correo) {
        this.nombre = nombre;
        this.correo = correo;
    }

    public String obtenerDetalles() {
        return "Usuario: " + nombre + " - Correo: " + correo;
    }
}

class Administrador extends Usuario {
    private int nivelAcceso;

    public Administrador(String nombre, String correo, int nivelAcceso) {
        super(nombre, correo);
        this.nivelAcceso = nivelAcceso;
    }

    @Override
    public String obtenerDetalles() {
        return super.obtenerDetalles() + " - Nivel de acceso: " + nivelAcceso;
    }
```

```
}

public class UsuarioTest {
    public static void main(String[] args) {
        Usuario usuario = new Usuario("Juan", "juan@example.com");
        Administrador admin = new Administrador("Ana", "ana@example.com", 5);

        System.out.println(usuario.obtenerDetalles());
        System.out.println(admin.obtenerDetalles());
    }
}
```

Ejercicio 9: Dispositivos electrónicos

Objetivo: Aplicar herencia en tecnología.

1. Crea una clase `Dispositivo` con un método `encender()`.
2. Crea subclases `Computadora` y `Telefono`.
3. Sobrescribe `encender()` para mostrar mensajes personalizados.
4. Crea una clase de prueba para demostrar la funcionalidad.

```
class Dispositivo {
    public void encender() {
        System.out.println("El dispositivo se está encendiendo");
    }
}

class Computadora extends Dispositivo {
    @Override
    public void encender() {
        System.out.println("La computadora está iniciando");
    }
}

class Telefono extends Dispositivo {
    @Override
    public void encender() {
        System.out.println("El teléfono se está encendiendo");
    }
}

public class DispositivoTest {
    public static void main(String[] args) {
        Dispositivo pc = new Computadora();
        Dispositivo movil = new Telefono();

        pc.encender();
        movil.encender();
    }
}
```

Ejercicio 10: Transporte y velocidades

Objetivo: Aplicar herencia en distintos tipos de transporte.

1. Crea una clase `Transporte` con un método `obtenerVelocidadMaxima()`.
2. Crea subclases `Bicicleta` y `Avion`.
3. Sobrescribe `obtenerVelocidadMaxima()` con valores realistas.
4. Implementa una clase de prueba para comparar las velocidades.

```
class Transporte {
    public int obtenerVelocidadMaxima() {
        return 0;
    }
}

class Bicicleta extends Transporte {
    @Override
    public int obtenerVelocidadMaxima() {
        return 25;
    }
}

class Avion extends Transporte {
    @Override
    public int obtenerVelocidadMaxima() {
        return 900;
    }
}

public class TransporteTest {
    public static void main(String[] args) {
        Transporte bici = new Bicicleta();
        Transporte avion = new Avion();

        System.out.println("Velocidad de Bicicleta: " +
bici.obtenerVelocidadMaxima() + " km/h");
        System.out.println("Velocidad de Avión: " + avion.obtenerVelocidadMaxima()
+ " km/h");
    }
}
```