Ejercicios de Polimorfismo en Java

Ejercicio 1: Figuras geométricas

Objetivo: Aplicar polimorfismo con clases abstractas.

Instrucciones:

1. Crea una clase abstracta `Figura` con un método abstracto `calcularArea()`.
2. Crea subclases `Circulo` y `Rectangulo`.
3. En `Circulo`, `calcularArea()` debe devolver el área de un círculo.
4. En `Rectangulo`, `calcularArea()` debe devolver el área de un rectángulo.
5. Implementa una clase de prueba que cree instancias y calcule las áreas.

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

class Circulo extends Figura {
    double radio;

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

    @Override
    double calcularArea() {
        return Math.PI * radio * radio;
    }
}

class Rectangulo extends Figura {
    double ancho, alto;

    Rectangulo(double ancho, double alto) {
        this.ancho = ancho;
        this.alto = alto;
    }

    @Override
    double calcularArea() {
        return ancho * alto;
    }
}

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

```
        System.out.println("Área del rectángulo: " + r.calcularArea());
    }
}
```

Ejercicio 2: Animales y sonidos

Objetivo: Aplicar polimorfismo con métodos sobrescritos.

Instrucciones:

1. Crea una clase abstracta `Animal` con un método `hacerSonido()`.
2. Crea subclases `Perro` y `Gato`.
3. En `Perro`, `hacerSonido()` debe imprimir "Guau".
4. En `Gato`, `hacerSonido()` debe imprimir "Miau".
5. Implementa una clase de prueba con instancias de cada animal.

```
abstract class Animal {
    abstract void hacerSonido();
}

class Perro extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("Guau");
    }
}

class Gato extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("Miau");
    }
}

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

Ejercicio 3: Empleados y salarios

Objetivo: Aplicar polimorfismo en cálculo de salarios.

Instrucciones:

1. Crea una clase abstracta `Empleado` con un método `calcularSalario()`.
2. Crea subclases `EmpleadoTiempoCompleto` y `EmpleadoPorHoras`.
3. `EmpleadoTiempoCompleto` devuelve un salario fijo.
4. `EmpleadoPorHoras` calcula el salario según las horas trabajadas.
5. Implementa una clase de prueba para probar ambos casos.

```
abstract class Empleado {
    abstract double calcularSalario();
}

class EmpleadoTiempoCompleto extends Empleado {
    @Override
    double calcularSalario() {
        return 3000;
    }
}

class EmpleadoPorHoras extends Empleado {
    int horasTrabajadas;
    double tarifaHora;

    EmpleadoPorHoras(int horasTrabajadas, double tarifaHora) {
        this.horasTrabajadas = horasTrabajadas;
        this.tarifaHora = tarifaHora;
    }

    @Override
    double calcularSalario() {
        return horasTrabajadas * tarifaHora;
    }
}

public class PruebaEmpleados {
    public static void main(String[] args) {
        Empleado emp1 = new EmpleadoTiempoCompleto();
        Empleado emp2 = new EmpleadoPorHoras(40, 15);
        System.out.println("Salario empleado tiempo completo: " +
emp1.calcularSalario());
        System.out.println("Salario empleado por horas: " +
emp2.calcularSalario());
    }
}
```

Ejercicio 4: Vehículos y métodos de conducción

Objetivo: Demostrar polimorfismo con distintos tipos de vehículos.

Instrucciones:

1. Crea una clase abstracta `Vehiculo` con un método `conducir()`.

2. Crea subclases `Auto` y `Moto`.
3. En `Auto`, `conducir()` imprime "Conduciendo un auto".
4. En `Moto`, `conducir()` imprime "Conduciendo una moto".
5. Implementa una clase de prueba para probar ambas clases.

```
abstract class Dispositivo {
    abstract void encender();
}

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

class Radio extends Dispositivo {
    @Override
    void encender() {
        System.out.println("La radio se está encendiendo");
    }
}

public class PruebaDispositivos {
    public static void main(String[] args) {
        Dispositivo tv = new Televisor();
        Dispositivo radio = new Radio();
        tv.encender();
        radio.encender();
    }
}
```

Ejercicio 5: Instrumentos musicales

Objetivo: Aplicar polimorfismo en sonidos de instrumentos.

Instrucciones:

1. Crea una clase abstracta `InstrumentoMusical` con un método `tocar()`.
2. Crea subclases `Guitarra` y `Violin`.
3. `Guitarra.tocar()` imprime "La guitarra suena con acordes".
4. `Violin.tocar()` imprime "El violín emite un sonido melodioso".
5. Implementa una clase de prueba con instancias de ambas clases.

```
abstract class Vehiculo {
    abstract String obtenerTipoCombustible();
}

class AutoElectrico extends Vehiculo {
```

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

class AutoGasolina extends Vehiculo {
    @Override
    String obtenerTipoCombustible() {
        return "Gasolina";
    }
}

public class PruebaVehiculos {
    public static void main(String[] args) {
        Vehiculo electrico = new AutoElectrico();
        Vehiculo gasolina = new AutoGasolina();
        System.out.println("Auto eléctrico usa: " +
electrico.obtenerTipoCombustible());
        System.out.println("Auto de gasolina usa: " +
gasolina.obtenerTipoCombustible());
    }
}
```

Ejercicio 6: Métodos de pago

Objetivo: Aplicar polimorfismo en distintos métodos de pago.

Instrucciones:

1. Crea una clase abstracta `MetodoPago` con un método `pagar()`.
2. Crea subclases `PagoTarjeta` y `PagoEfectivo`.
3. `PagoTarjeta.pagar()` imprime "Pago realizado con tarjeta".
4. `PagoEfectivo.pagar()` imprime "Pago realizado en efectivo".
5. Implementa una clase de prueba con ambos métodos de pago.

```
abstract class MetodoPago {
    abstract void procesarPago(double monto);
}

class TarjetaCredito extends MetodoPago {
    @Override
    void procesarPago(double monto) {
        System.out.println("Pago de " + monto + " procesado con tarjeta de
crédito.");
    }
}

class PayPal extends MetodoPago {
    @Override
    void procesarPago(double monto) {
```

```
        System.out.println("Pago de " + monto + " procesado con PayPal.");
    }
}

public class PruebaPagos {
    public static void main(String[] args) {
        MetodoPago pago1 = new TarjetaCredito();
        MetodoPago pago2 = new PayPal();
        pago1.procesarPago(100.50);
        pago2.procesarPago(200.75);
    }
}
```

Ejercicio 7: Dispositivos electrónicos

Objetivo: Aplicar polimorfismo con dispositivos que encienden de forma distinta.

Instrucciones:

1. Crea una clase abstracta `DispositivoElectronico` con un método `encender()`.
2. Crea subclases `Television` y `Celular`.
3. `Television.encender()` imprime "Encendiendo la TV".
4. `Celular.encender()` imprime "Encendiendo el celular".
5. Implementa una clase de prueba con instancias de ambas clases.

```
abstract class Documento {
    abstract void imprimir();
}

class PDF extends Documento {
    @Override
    void imprimir() {
        System.out.println("Imprimiendo documento PDF...");
    }
}

class Word extends Documento {
    @Override
    void imprimir() {
        System.out.println("Imprimiendo documento Word...");
    }
}

public class PruebaDocumentos {
    public static void main(String[] args) {
        Documento doc1 = new PDF();
        Documento doc2 = new Word();
        doc1.imprimir();
        doc2.imprimir();
```

```
    }
}
```

Ejercicio 8: Alimentos y consumo

Objetivo: Aplicar polimorfismo con alimentos y cómo se consumen.

Instrucciones:

1. Crea una clase abstracta `Alimento` con un método `consumir()`.
2. Crea subclases `Manzana` y `Pizza`.
3. `Manzana.consumir()` imprime "Morder la manzana".
4. `Pizza.consumir()` imprime "Comiendo la pizza en porciones".
5. Implementa una clase de prueba con ambos alimentos.

```
abstract class TransportePublico {
    abstract void anunciarParada();
}

class Autobus extends TransportePublico {
    @Override
    void anunciarParada() {
        System.out.println("Próxima parada: Centro");
    }
}

class Tren extends TransportePublico {
    @Override
    void anunciarParada() {
        System.out.println("Próxima estación: Terminal");
    }
}

public class PruebaTransporte {
    public static void main(String[] args) {
        TransportePublico t1 = new Autobus();
        TransportePublico t2 = new Tren();
        t1.anunciarParada();
        t2.anunciarParada();
    }
}
```

Ejercicio 9: Transporte y velocidad

Objetivo: Aplicar polimorfismo con distintos medios de transporte.

Instrucciones:

1. Crea una clase abstracta `Transporte` con un método `velocidadMaxima()`.
2. Crea subclases `Avion` y `Bicicleta`.
3. `Avion.velocidadMaxima()` devuelve "900 km/h".
4. `Bicicleta.velocidadMaxima()` devuelve "30 km/h".
5. Implementa una clase de prueba con ambos transportes.

```
abstract class Personaje {
    abstract void atacar();
}

class Guerrero extends Personaje {
    @Override
    void atacar() {
        System.out.println("El guerrero ataca con su espada");
    }
}

class Mago extends Personaje {
    @Override
    void atacar() {
        System.out.println("El mago lanza un hechizo");
    }
}

public class PruebaPersonajes {
    public static void main(String[] args) {
        Personaje p1 = new Guerrero();
        Personaje p2 = new Mago();
        p1.atacar();
        p2.atacar();
    }
}
```

Ejercicio 10: Personajes y habilidades

Objetivo: Aplicar polimorfismo con distintos personajes y sus habilidades.

Instrucciones:

1. Crea una clase abstracta `Personaje` con un método `realizarHabilidad()`.
2. Crea subclases `Guerrero` y `Mago`.
3. `Guerrero.realizarHabilidad()` imprime "El guerrero ataca con su espada".
4. `Mago.realizarHabilidad()` imprime "El mago lanza un hechizo".
5. Implementa una clase de prueba con ambos personajes.

```
abstract class InstrumentoMusical {
    abstract void tocar();
}
```

```
class Violin extends InstrumentoMusical {
    @Override
    void tocar() {
        System.out.println("El violín está tocando una melodía");
    }
}

class Flauta extends InstrumentoMusical {
    @Override
    void tocar() {
        System.out.println("La flauta emite un sonido suave");
    }
}

public class PruebaInstrumentos {
    public static void main(String[] args) {
        InstrumentoMusical i1 = new Violin();
        InstrumentoMusical i2 = new Flauta();
        i1.tocar();
        i2.tocar();
    }
}
```