Ana Chun Gómez De Castro DAM1

Ejercicio de Programación Teórico

1. Introducción Teórica: Un breve documento explicando los conceptos de polimorfismo, herencia, sobrecarga de métodos, polimorfismo paramétrico (generics en Java), y polimorfismo de inclusión (también conocido como subtipado o polimorfismo de subclases).

- **Polimorfismo**: Forma parte de la programación orientada a objetos que se pueden crear y utilizar dos o más métodos con el mismo nombre para ejecutar funciones diferentes

- **Herencia**: Es una propiedad fundamental en la POO, implica que una superclase o clase herede sus funciones y sus atributos a una clase o subclase. Los tipos que hay son herencia simple, múltiple, etc. Crea una jerarquía de clase. Heredar significa que un objeto puede aprovechar las propiedades de otro.

- **Sobrecarga de métodos**: Este concepto se refiere a que utiliza varios métodos en una clase con el mismo nombre, pero con diferentes tipos o números de parámetros.

- **Polimorfismo paramétrico (generics en Java)**: Es un tipo de polimorfismo que permite que una función acepte como número infinito. Sirve para dar flexibilidad al programador dando una función que pueda recibir argumentos de diferentes tipos.

- **Polimorfismo inclusión (subtipado o polimorfismo de subclases)**: Permite a una subclase modificar completamente el comportamiento de un método heredado de su superclase. Esto significa que la subclase puede proporcionar una implementación específica del método que difiere de la implementación

de la superclase, permitiendo adaptar el comportamiento según las necesidades particulares de cada clase descendiente.

2. Ejemplos de Código: Implementación de ejemplos en Java que demuestren cada uno de estos conceptos claramente, incluyendo:

2.1) Herencia: Crear una clase base y derivadas que demuestren el uso de herencia.

```
package Ejemplos_de_Código;

/*1) Herencia*/

// Clase base que representa un animal genérico
       public class Animal {
       String nombre;

   // Constructor que recibe el nombre del animal
   public Animal(String nombre) {
        this.nombre = nombre;
    }

// Método que imprime un mensaje genérico indicando que el animal emite un sonido
  public void emitirSonido() {
      System.out.println("El animal emite sonido");
  }

// Representa una vaca y hereda de la clase Animal
   public class vaca extends Animal {
   // Constructor que llama al constructor de la clase base (Animal)
   public vaca (String nombre) {
       super(nombre);
   }

// Sobrescribe el método emitirSonido para imprimir el sonido característico
//de una vaca
   @Override
   public void emitirSonido() {
   System.out.println("¡Muuu, muuu!");
  }
}

// Clase interna que representa un gallo y hereda de la clase Animal
  public class gallo extends Animal {
       // Constructor que llama al constructor de la clase base (Animal)
       public gallo(String nombre) {
       super(nombre);
```

```
}

// Sobrescribe el método emitirSonido para imprimir el sonido característico
//de un gallo
   @Override
    public void emitirSonido() {
        System.out.println(¡¡¡"Quiquiriquí!!!");
}
```

2.2) Polimorfismo de Inclusión: Demostrar cómo un objeto de una clase derivada puede ser tratado como un objeto de su clase base.

```
package Ejemplos_de_Código;

/*2)Polimorfismo de Inclusión: */

public class presentarAnimal {
// Método que recibe un objeto de tipo Animal
public static void presentarAnimal(Animal a) {
// Llama al método emitirSonido() del objeto
a.emitirSonido();
}

public static void main(String[] args) {
// Creación de objetos de tipo Vaca y Gallo
Vaca p = new Vaca("Highland");
Gallo g = new Gallo("Gallo blue");

// Llamada al método presentarAnimal con diferentes objetos
presentarAnimal(p); // La vaca Highland hace ¡Muuu, muuu!
presentarAnimal(g); // El gallo blue hace Quiquiriquí!!!
}
```

2.3) Sobrecarga (Overloading): Implementar métodos sobrecargados dentro de una clase.

```
package Ejemplos_de_Código;

/*3) Polimorfismo Overloading*/

public class Calculadora {

// Método suma en enteros
public int suma(int a, int b) {
return a + b;
}

// Método suma en decimales
```

```
public double suma(double a, double b) {
return a + b;
}

public static void main(String[] args) {
Calculadora calculadora = new Calculadora ();

// Uso de los métodos sobrecargados
System.out.println("Suma de dos enteros: " + calculadora.suma(5, 10));
System.out.println("Suma de dos números de punto flotante: " +
calculadora.suma(3.7, 2.3));
}
}
```

2.4) Polimorfismo Paramétrico: Creard clases o métodos que utilicen genéricos para demostrar polimorfismo paramétrico.

```
package Ejemplos_de_Código;
/*4) Polimorfismo Paramétrico*/
abstract class Piece {
     public abstract void move(byte X, byte Y);
}

class Bishop extends Piece {
    @Override
    public void move(byte X, byte Y) {
  }
}
```

2.5) Polimorfismo (en general): Demostrar el uso en un escenario aplicado, donde se use la misma interfaz para diferentes tipos subyacentes.

```
package Ejemplos_de_Código;
/*5) Polimorfismo (en general) */

// Interfaz Figura:
interface Figura {
double calculoArea();
}

//Clase Circulo implementando la interfaz Figura

class Circulo implementos Figura {
/*
 * Representa un círculo e implementa la interfaz Figura. Tiene un constructor
```

```
 * que recibe el radio y una implementación del método calculoArea() para
 * calcular el área del círculo.
 */

private double radio;

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

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

// Clase Rectángulo implementando la interfaz Figura
class Rectangulo implements Figura {
/*
 * Representa un rectángulo y también implementa la interfaz Figura. Tiene un
 * constructor que recibe la longitud y la anchura, y una implementación del
 * método calculoArea() para calcular el área del rectángulo.
 */
private double longitud;
private double anchura;

public Rectangulo(double longitud, double anchura) {
this.longitud = longitud;
this.anchura = anchura;
}

@Override
public double calculoArea() {
return longitud * anchura;
}

// Clase principal para demostrar el polimorfismo
public class DemoPolimorfismo {
public static void main(String[] args) {
// Crear instancias de Circulo y Rectangulo
Figura circulo = new Circulo(2.4);
Figura rectangulo = new Rectangulo(3.754, 8.3);

// Calcular áreas sin preocuparse por el tipo específico de figura
double areaCirculo = circulo.calculoArea();
double areaRectangulo = rectangulo.calculoArea();

// Muestra los resultados
System.out.println("Área del círculo: " + areaCirculo);
System.out.println("Área del rectángulo: " + areaRectangulo);
}
}
}
```

3. Análisis Comparativo:

3.1) Explicar las diferencias entre polimorfismo y sobrecarga de métodos.

- El polimorfismo trata de diferentes tipos de objetos hace que un objeto tome varias maneras según el contexto, se logra mediante el uso de clases y objetos, y se basa en dos mecanismos que son la sobrecarga y la sobreescritura (override) de métodos.

- En la sobrecarga de métodos, el usuario crear varios métodos con el mismo nombre, pero tiene que ser diferentes parámetros. Además, se resuelve en tiempo de compilación usando los nombres de los métodos y los tipos de éstos

3.2) Diferenciar entre sobrecarga (overloading) y redefinición (overriding) de métodos.

- La sobrecarga de métodos(overloading) se trata de definir varios métodos con el mismo nombre, y se diferencian por los tipos de parámetros que reciben. Por ejemplo, el método llamado suma se sobrecarga con los diferentes tipos de datos.

(sumar (int a, int b): Suma dos números enteros.
sumar (double x, double y): Suma dos números de punto flotante (decimales).
sumar (String s1, String s2): Concatena dos cadenas de texto.)

- La redefinición (overriding) surge cuando una subclase declara un método con el mismo nombre, los mismos parámetros (o firma) y el mismo tipo de retorno que un método en su superclase. Por ejemplo, si se utiliza un objeto de la clase padre, se ejecutará en la clase padre y si es en un objeto de la subclase, también en la versión en la clase hija.

Además, dame respuesta a las siguientes preguntas en tus propias palabras:

- **¿Qué es el término firma?:** Es el método que hace referencia al nombre y sus parámetros o argumentos. Su función es incluir su nombre, los parámetros que acepta y su tipo de retorno.

- **¿Diferencias entre los términos Overloading y Overriding?:** Que en Overloading tiene mismos métodos, pero distinta firma y el Overriding es una subclase que tiene el mismo método definido en la subclase.

- **¿Se pueden sobrecargar métodos estáticos?:** Sí, puede tener varios métodos estáticos con el mismo nombre en una clase, siempre y cuando sus listas de parámetros sean distintas.

- **¿Es posible sobrecargar la clase main() en Java?:** Si ponemos bien los parámetros de entrada, sí.