1. Introducción Teórica

a. Polimorfismo

El polimorfismo en Java es la capacidad de una entidad (como un método o una clase) de tomar muchas formas diferentes. En Java, el polimorfismo se logra a través de la herencia y la implementación de interfaces.

b. Herencia

La herencia en Java permite que una clase (subclase o clase derivada) herede los campos y métodos de otra clase (superclase o clase base). Esto promueve la reutilización de código y permite la creación de jerarquías de clases.

c. Sobrecarga de Métodos

La sobrecarga de métodos es cuando una clase tiene múltiples métodos con el mismo nombre, pero con diferentes firmas (número o tipo de parámetros). Esto permite que se pueda invocar un método utilizando diferentes argumentos.

d. Polimorfismo Paramétrico (Generics en Java)

Permiten definir clases o métodos que pueden funcionar con múltiples tipos de datos sin necesidad de especificar un tipo concreto.

e. Polimorfismo de Inclusión (Subtipado o Polimorfismo de Subclases)

Es cuando una subclase es tratada como su superclase. Los objetos de la subclase pueden ser sustituidos donde se espera un objeto de la superclase.

2. Ejemplos de Código

1) Ejemplo de código de herencia: Crearemos una clase base llamada ***Animal*** y dos clases derivadas: ***Perro*** y ***Gato***. Ambas clases derivadas heredarán propiedades y métodos de la clase base.

```
public class Animal {
    public void emitirSonido() {
        System.out.println("Sonido genérico de animal");
    }
}
public class Perro extends Animal {
    @Override
    public void emitirSonido() {
        System.out.println("Guau guau");
    }
}
```

```
public class Gato extends Animal {
    @Override
    public void emitirSonido() {
        System.out.println("Miau miau");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal miAnimal = new Perro();
        miAnimal.emitirSonido(); // Salida: "Guau guau"
    }
}
```

2) Ejemplo de Polimorfismo de Inclusión: Demostraremos cómo un objeto de una clase derivada puede ser tratado como un objeto de su clase base.

```
public class Figura {
    public void dibujar() {
        System.out.println("Dibujando una figura genérica");
    }
}
public class Circulo extends Figura {
    @Override
    public void dibujar() {
        System.out.println("Dibujando un círculo");
    }
}
public class Main {
    public static void main(String[] args) {
        Figura miFigura = new Circulo();
        miFigura.dibujar(); // Salida: "Dibujando un círculo"
    }
 }
```

3) Ejemplo de Sobrecarga (Overloading)

```
public class Calculadora {
    public int sumar(int a, int b) {
        return a + b;
    }
    public double sumar(double a, double b) {
        return a + b;
    }
}
public class Main {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        System.out.println(calc.sumar(7, 3)); // Salida: 10
        System.out.println(calc.sumar(2.5, 1.5)); // Salida: 4.0
    }
}
```

4) Ejemplo de Polimorfismo Paramétrico

```
public class Contenedor<T> {
    private T elemento;
    public Contenedor(T elemento) {
        this.elemento = elemento;
    }
    public T obtenerElemento() {
        return elemento;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Contenedor<String> contenedorString = new Contenedor<>("Hola, mundo");
        System.out.println(contenedorString.obtenerElemento()); // Salida: "Hola, mundo"
        Contenedor<Integer> contenedorInt = new Contenedor<>(50);
        System.out.println(contenedorInt.obtenerElemento()); // Salida: 50
    }
}
```

5) Ejemplo de Polimorfismo (en general)

```
interface InstrumentoMusical {
    void tocar();
}
// Piano va a implementar la interfaz InstrumentoMusical
public class Piano implements InstrumentoMusical {
    @Override
    public void tocar() {
        System.out.println("Tocando el piano");
    }
}
// Guitarra va a implementar la interfaz InstrumentoMusical
public class Guitarra implements InstrumentoMusical {
    @Override
    public void tocar() {
        System.out.println("Tocando la guitarra");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        InstrumentoMusical miInstrumento = new Piano();
        miInstrumento.tocar(); // Salida: "Tocando el piano"
        miInstrumento = new Guitarra();
        miInstrumento.tocar(); // Salida: "Tocando la guitarra"
    }
}
```

3. Análisis Comparativo

- **Diferencias entre Polimorfismo y Sobrecarga de Métodos:** El polimorfismo permite que un objeto se comporte de diferentes maneras según su tipo, mientras que la sobrecarga de métodos permite definir varios métodos con el mismo nombre, pero diferentes listas de parámetros en la misma clase.
- **Diferenciar entre Sobrecarga (Overloading) y Redefinición (Overriding) de Métodos:** La sobrecarga de métodos ocurre dentro de una misma clase y se refiere a tener varios métodos con el mismo nombre, pero diferentes parámetros. La redefinición de métodos (overriding) ocurre en una relación de herencia, donde una subclase proporciona una implementación específica de un método que ya está definido en su superclase.

Preguntas Adicionales

- **¿Qué es el término firma?**

  La firma de un método consiste en su nombre y la lista de tipos de parámetros que acepta. No incluye el tipo de retorno ni el nombre de la clase.

- **¿Diferencias entre los términos Overloading y Overriding?**

  Overloading se refiere a tener varios métodos con el mismo nombre, pero diferentes listas de parámetros, mientras que Overriding se refiere a proporcionar una implementación específica de un método que ya está definido en una superclase.

- **¿Se pueden sobrecargar métodos estáticos?**

  Sí, se pueden sobrecargar métodos estáticos en Java, pero no pueden ser redefinidos (overridden).

- **¿Es posible sobrecargar la clase main() en Java?**

  No, la clase **main()** no se puede sobrecargar en Java. Debe tener exactamente la firma estándar **public static void main(String[] args)**.