1. Introducción Teórica:

- Polimorfismo: propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. Aunque el mensaje sea el mismo, diferentes objetos pueden responder a él de manera única y específica. Esta característica permite que, sin alterar ni tocar el código existente, se puedan incorporar nuevos comportamientos y funciones (es decir la interfaz sintáctica se mantiene inalterada pero cambia el comportamiento en función de qué objeto estamos usando en cada momento). El único requisito es que los objetos deben ser capaces de responder al mensaje que se les envía, garantizando así una flexibilidad y extensibilidad en el diseño del software.
- Herencia: La herencia facilita la creación de objetos a partir de otros ya existentes e implica que una subclase obtiene todo el comportamiento (métodos) y finalmente los atributos (variables) de su superclase.
  Es la relación entre una clase general y otra clase más específica. Por ejemplo: Si declaramos una clase párrafo derivada de una clase texto, todos los métodos y variables asociadas con la clase texto, son automáticamente heredados por la subclase párrafo.
- Sobrecarga de métodos: capacidad de un lenguaje de programación, que permite nombrar con el mismo identificador diferentes variables u operaciones.
  Se refiere a la posibilidad de tener dos o más funciones con el mismo nombre pero funcionalidad diferente. Es decir, dos o más funciones con el mismo nombre realizan acciones diferentes. El compilador usará una u otra dependiendo de los parámetros usados.
- Polimorfismos paramétrico: es aquel en el que el código no incluye ningún tipo de especificación sobre el tipo de datos sobre el que se trabaja. Así, puede ser utilizado a todo tipo de datos compatible.
- Polimorfismo de inclusión o subtipado o poli.de subclases: Se refiere a la capacidad de objetos de diferentes clases de responder de manera uniforme a un mismo mensaje o método.

2. Ejemplos de Código:

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

```
class Figura {
    double area;

    public Figura() {
        this.area = 0;
    }

    // Método para calcular el área (por defecto, 0 para Figura)
    public void calcularArea() {
        System.out.println("Área de la figura: " + area);
    }
}
```

```
// Clase derivada (subclase) - Triángulo
class Triangulo extends Figura {
    double base;
    double altura;

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

    // Sobrescritura del método calcularArea para calcular el área de un triángulo
    @Override
    public void calcularArea() {
        area = (base * altura) / 2;
        System.out.println("Área del triángulo: " + area);
    }
}

// Clase derivada (subclase) - Rectángulo
class Rectangulo extends Figura {
    double ladoA;
    double ladoB;

    public Rectangulo(double ladoA, double ladoB) {
        this.ladoA = ladoA;
        this.ladoB = ladoB;
    }

    // Sobrescritura del método calcularArea para calcular el área de un rectángulo
    @Override
    public void calcularArea() {
        area = ladoA * ladoB;
        System.out.println("Área del rectángulo: " + area);
    }
}

public class Main {
    public static void main(String[] args) {
        // Crear instancia de la subclase Triangulo
        Triangulo miTriangulo = new Triangulo(5, 3);
        miTriangulo.calcularArea(); // Calcular y mostrar área del triángulo

        // Crear instancia de la subclase Rectangulo
        Rectangulo miRectangulo = new Rectangulo(4, 6);
```

```
        miRectangulo.calcularArea(); // Calcular y mostrar área del rectángulo

        // Crear instancia de la superclase Figura (polimorfismo de inclusión)
        Figura miFigura = new Figura();
        miFigura.calcularArea(); // Mostrar área de la figura (0)
    }
}
```

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

```
// Clase base (superclase)
class Animal {
    public void hacerSonido() {
        System.out.println("Haciendo sonido genérico...");
    }
}

// Clase derivada (subclase) - Perro
class Perro extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("El perro ladra: Woof!");
    }

    public void perseguirCola() {
        System.out.println("El perro está persiguiendo su cola.");
    }
}

public class Main {
    public static void main(String[] args) {
        // Crear un objeto de la clase derivada Perro
        Perro miPerro = new Perro();

        // Tratar el objeto Perro como un objeto de la clase base Animal
        Animal miAnimal = miPerro;

        // Llamar al método hacerSonido() del objeto Perro
        miPerro.hacerSonido();  // Salida: El perro ladra: Woof!

        // Llamar al método hacerSonido() del objeto Animal (polimorfismo)
        miAnimal.hacerSonido(); // Salida: El perro ladra: Woof!
```

```
        // No se puede llamar al método perseguirCola() del objeto Animal
        // miAnimal.perseguirCola(); // Esto daría un error en tiempo de compilación
    }
}
```

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

```
public class Calculadora {
    // Método sobrecargado para sumar dos números enteros
    public int sumar(int num1, int num2) {
        return num1 + num2;
    }

    // Método sobrecargado para sumar dos números flotantes
    public float sumar(float num1, float num2) {
        return num1 + num2;
    }

    // Método sobrecargado para sumar tres números enteros
    public int sumar(int num1, int num2, int num3) {
        return num1 + num2 + num3;
    }

    // Método sobrecargado para concatenar dos cadenas
    public String concatenar(String cadena1, String cadena2) {
        return cadena1 + cadena2;
    }

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

        // Uso de los métodos sobrecargados
        System.out.println("Suma de enteros: " + calc.sumar(5, 3));
        System.out.println("Suma de flotantes: " + calc.sumar(2.5f, 3.7f));
        System.out.println("Suma de tres enteros: " + calc.sumar(5, 3, 2));
        System.out.println("Concatenación de cadenas: " + calc.concatenar("Hola",
" Mundo"));
    }
}
```

- Polimorfismo Paramétrico: Crear clases o métodos que utilicen genéricos para demostrar polimorfismo paramétrico.

```
// Clase genérica Pila
```

```
class Pila<T> {
    private T[] elementos;
    private int tope;

    // Constructor que inicializa la pila
    @SuppressWarnings("unchecked")
    public Pila(int capacidad) {
        elementos = (T[]) new Object[capacidad];
        tope = 0;
    }

    // Método para agregar un elemento a la pila
    public void push(T elemento) {
        elementos[tope] = elemento;
        tope++;
    }

    // Método para extraer y retornar el elemento del tope de la pila
    public T pop() {
        if (tope == 0) {
            throw new IllegalStateException("La pila está vacía");
        }
        tope--;
        return elementos[tope];
    }
}

public class Main {
    public static void main(String[] args) {
        // Crear una pila de enteros
        Pila<Integer> pilaEnteros = new Pila<>(5);
        pilaEnteros.push(1);
        pilaEnteros.push(2);
        pilaEnteros.push(3);
        System.out.println("Elemento extraído de la pila de enteros: " +
pilaEnteros.pop());

        // Crear una pila de cadenas
        Pila<String> pilaCadenas = new Pila<>(3);
        pilaCadenas.push("Hola");
        pilaCadenas.push("Mundo");
        System.out.println("Elemento extraído de la pila de cadenas: " +
pilaCadenas.pop());
    }
```

```
}
```

- Polimorfismo (en general): Demostrar el uso de polimorfismo en un escenario aplicado, donde se utilice la misma interfaz para diferentes tipos subyacentes.

```
// Clase genérica Pila
class Pila<T> {
    private T[] elementos;
    private int tope;

    // Constructor que inicializa la pila
    @SuppressWarnings("unchecked")
    public Pila(int capacidad) {
        elementos = (T[]) new Object[capacidad];
        tope = 0;
    }

    // Método para agregar un elemento a la pila
    public void push(T elemento) {
        elementos[tope] = elemento;
        tope++;
    }

    // Método para extraer y retornar el elemento del tope de la pila
    public T pop() {
        if (tope == 0) {
            throw new IllegalStateException("La pila está vacía");
        }
        tope--;
        return elementos[tope];
    }
}

public class Main {
    public static void main(String[] args) {
        // Crear una pila de enteros
        Pila<Integer> pilaEnteros = new Pila<>(5);
        pilaEnteros.push(1);
        pilaEnteros.push(2);
        pilaEnteros.push(3);
        System.out.println("Elemento extraído de la pila de enteros: " +
pilaEnteros.pop());

        // Crear una pila de cadenas
```

```
        Pila<String> pilaCadenas = new Pila<>(3);
        pilaCadenas.push("Hola");
        pilaCadenas.push("Mundo");
        System.out.println("Elemento extraído de la pila de cadenas: " +
pilaCadenas.pop());
    }
}
```

3. Análisis Comparativo:

- Diferencias entre polimorfismo y sobrecarga de métodos:
  el polimorfismo son las diferentes clases que pueden responder un mismo mensaje y la sobrecarga es la capacidad de poner diferentes métodos con un mismo nombre pero con diferentes parámetros dentro de una misma clase

- Diferencia entre sobrecarga(overloading) y redefinición(overriding) de métodos:
  el overloading la capacidad de poner diferentes métodos con un mismo nombre pero con diferentes parámetros dentro de una misma clase y el overriding es la capacidad de una subclase de implementar un método específico ya definido

4. Preguntas:

- ¿Que es el término firma?
  Combinación del nombre y sus parámetros, es única dentro de una clase y se usa para diferenciar métodos con el mismo nombre con los parámetros

- ¿Diferencias entre los términos overloading y overriding?
  El overloading es la capacidad de poner diferentes métodos con un mismo nombre pero con diferentes parámetros dentro de una misma clase y el overriding es la capacidad de una subclase de implementar un método específico ya definido.

- ¿Se pueden sobrecargar métodos estáticos?
  Si, ya que se basa en una lista de parámetros

- ¿Es posible sobrecargar la clase main() en java?
  No, porque es específica y estándar