06/02/2024

PROGRAMACIÓN

Requisitos del Trabajo

1. Introducción Teórica:

Polimorfismo

Es la capacidad de un objeto de comportarse de diferentes maneras.Con la programación orientada a objetos, esto se puede lograr de varias maneras:
Sobrecarga de métodos: es el uso de varios métodos con el mismo nombre, pero con diferentes parámetros. Permite que un objeto responda a diferentes tipos de mensajes de la misma manera.

Ejemplo:

```
public void unMetodo(Strings){//Método sobrecargado
}
```

puede tener:

- Diferentes tipos de retorno.
- Diferentes modificadores de acceso.
- Lanzar diferentes excepciones.

Polimorfismo paramétrico(genéricos en Java): Es el uso de tipos de datos genéricos para crear métodos que pueden trabajar con diferentes tipos de datos. Permite que un objeto sea reutilizable para diferentes tipos de datos.
Polimorfismo de inclusión(subtipado o polimorfismo de subclases): Es el uso de subclases para reemplazar superclases. Permite que un objeto se use de la misma manera que superclases,con un comportamiento adicional.

Herencia

Es un mecanismo que permite a una clase heredar las características de otra clase.Permite que las clases se reutilicen y se extiendan.
Sobrecarga de métodos : es el uso de varios métodos con el mismo nombre, pero con diferentes parámetros.Permite que un objeto responda a diferentes tipos de mensajes de la misma manera.
Polimorfismo paramétrico(genérico en java): Es el uso de tipos de datos genéricos para crear métodos que pueden trabajar con diferentes tipos de datos. Permite que un objeto sea reutilizable para diferentes tipos de datos.
Polimorfismo de inclusión(subtipado o polimorfismo de subclases): Es el uso de subclases para reemplazar superclases.Permite que un objeto use de la misma manera que su superclase, pero con un comportamiento adicinal.

06/02/2024

2. Ejemplos de Código:

Herencia

```
// Clase base
class Animal {
    protected String nombre;
    public Animal(String nombre) {
        this.nombre = nombre;
    }
    public void comer() {
        System.out.println("El animal " + nombre + " está comiendo");
    }
}
// Clase derivada
class Perro extends Animal {
    public Perro(String nombre) {
        super(nombre);
    }
    public void ladrar() {
        System.out.println("El perro " + nombre + " está ladrando");
    }
}
// Ejemplo de uso
public class Main {
    public static void main(String[] args) {
        Perro perro = new Perro("Toby");
        perro.comer(); // Invoca el método comer de la clase Animal
        perro.ladrar(); // Invoca el método ladrar de la clase Perro
    }
}
```

Polimorfismo de inclusión

```
// Clase base
class Figura {
    public abstract double Area();
}
// Clases derivadas
class Circulo extends Figura {
    private double Radio;
    public Circulo(double Radio) {
        this.Radio = Radio;
    }
    @Override
    public double Area() {
        return Math.PI * Math.pow(Radio, 2);
    }
}
class Rectangulo extends Figura {
    private double Largo;
    private double Ancho;
    public Rectangulo(double Largo, double Ancho) {
        this.Largo = Largo;
        this.Ancho = Ancho;
    }
```

```
    @Override
    public double Area() {
        return Largo * Ancho;
    }
}
// Ejemplo de uso
public class Main {
    public static void main(String[] args) {
        Figura[] Figuras = {new Circulo(5), new Rectangulo(4, 3)};
        for (Figura Figura : Figuras) {
            System.out.println("Área de la figura: " + Figura.Area());
        }
    }
}
```

Sobrecarga(Overloading):

```
class Calculadora {
    public int sumar(int a, int b) {
        return a + b;
    }
    public double sumar(double a, double b) {
        return a + b;
    }
    public String sumar(String a, String b) {
        return a + b;
    }
    // Ejemplo de uso
    public static void main(String[] args) {
        Calculadora calculadora = new Calculadora();
        System.out.println(calculadora.sumar(1, 2)); // Imprime 3
        System.out.println(calculadora.sumar(1.5, 2.5)); // Imprime 4.0
        System.out.println(calculadora.sumar("Hola", "Mundo cruel")); // Imprime Hola
Mundo cruel
    }
}
```

Polimorfismo paramétrico:

```
class Contenedor<T> {
    private T Contenido;
    public Contenedor(T Contenido) {
        this.Contenido = Contenido;
    }
    public T getContenido() {
        return Contenido;
    }
    public void setContenido(T Contenido) {
        this.Contenido = Contenido;
    }
    // Ejemplo de uso
    public static void main(String[] args) {
        Contenedor<Integer> ContenedorInt = new Contenedor<>(10);
        System.out.println(ContenedorInt.getContenido()); // Imprime 10
        Contenedor<String> ContenedorString = new Contenedor<>("Hola que tal");
```

```
            System.out.println(ContenedorString.getContenido()); // Imprime Hola que tal
        }
    }
```

Polimorfismo (en general)

```
    interface Animal {
        public void comer();
    }
    class Perro implements Animal {
        @Override
        public void comer() {
            System.out.println("El perro está comiendo");
        }
    }
    class Gato implements Animal {
        @Override
        public void comer() {
            System.out.println("El gato está comiendo");
        }
    }
    // Ejemplo de uso
    public class Main {
        public static void alimentar(Animal animal) {
            animal.comer();
        }
        public static void main(String[] args) {
            Perro perro = new Perro();
            Gato gato = new Gato();
            alimentar(perro);
}
```

3. Análisis Comparativo:

- Explicar las diferencias entre polimorfismo y sobrecarga de métodos.

- **El polimorfismo se centra en el comportamiento del objeto,puede ser de dos tipos y se resuelve en tiempo de ejecución.**
- **En cambio la sobrecarga se centra en la selección del método, es un único concepto y se resuelve en tiempo de compilación.**

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

- **La sobrecarga**(overloading) **se da en la misma clase y se basa en la signatura del método,no implica herencia, es decir es una forma de comportamiento ad-hoc.**
- **En cambio la redefinición**(overriding) **se da en diferentes clases relacionadas por herencia y se basa en el nombre del método y la clase en la que se invoca, es decir es una forma de comportamiento polimórfico.**

06/02/2024

Preguntas:

- ¿Qué es el término firma? Es un método que se refiere a la combinación

de su nombre y la lista de tipos de sus parámetros.

- ¿Diferencias entre los términos Overloading y Overriding?
Overloading se refiere a tener varios métodos con el mismo nombre en la misma clase pero con diferentes firmas, mientras que con Overriding se refiere a implementar un método en una clase hija con la firma que el método en la clase padre

- ¿Se pueden sobrecargar métodos estáticos?

Si, si se pueden tener métodos estáticos con el mismo nombre pero con diferente parámetro. Pero sin embargo , no pueden ser redefinidos.

- ¿Es posible sobrecargar la clase main() en Java?
Si, si se puede sobrecargar el método main mientras que los parámetros sean diferentes.