Sobreescritura y sobrecarga de métodos

En Java, la **sobreescritura de métodos** (overriding) y la **sobrecarga de métodos** (overloading) son dos técnicas clave en la programación orientada a objetos que permiten la reutilización y personalización del código.

- **Sobreescritura de Métodos:** Permite que una subclase proporcione una implementación específica de un método que ya está definido en su superclase. Esto es útil cuando se desea que una subclase tenga un comportamiento diferente para un método heredado. Para sobreescribir un método, el nombre, los parámetros y el tipo de retorno del método deben coincidir con los del método en la superclase.
- **Sobrecarga de Métodos:** Permite definir múltiples métodos con el mismo nombre pero diferentes parámetros dentro de la misma clase. Esto facilita la creación de métodos que realizan operaciones similares pero con diferentes tipos o cantidades de argumentos.

Estos conceptos son esenciales para aprovechar al máximo la herencia y el polimorfismo en Java, permitiendo escribir código más flexible y mantenible.

Veamos un poco más en detalle.

Sobreescritura de Métodos (Method Overriding)

Como ya se mencionó la sobreescritura de métodos ocurre cuando una subclase proporciona una implementación específica de un método que ya está definido en su superclase. Es una forma de garantizar que una subclase pueda proporcionar una versión especializada del método. Para que ocurra la sobreescritura, deben cumplirse ciertas condiciones:

1. **Mismo nombre del método:** El nombre del método en la subclase debe ser el mismo que en la superclase.
2. **Misma lista de parámetros:** La lista de parámetros (la cantidad y el tipo) debe ser la misma.
3. **Mismo tipo de retorno o un subtipo compatible:** El tipo de retorno debe ser el mismo o un subtipo compatible.
4. **Visibilidad:** El método en la subclase no puede tener una visibilidad más restrictiva que el método en la superclase.

Ejemplo de Sobreescritura de Métodos:

```
public class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido");
    }
}
```

```
public class Perro extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El perro ladra");
    }
}
```

```
public class Principal {
    public static void main(String[] args) {
        Animal miAnimal = new Animal();
```

```
        miAnimal.hacerSonido(); // Imprime "El animal hace un     sonido"

        Perro miPerro = new Perro();
        miPerro.hacerSonido(); // Imprime "El perro ladra"
    }
}
```

En este ejemplo, la clase `Perro` sobrescribe el método `hacerSonido` de la clase `Animal`.

Sobrecarga de Métodos (Method Overloading)

La sobrecarga de métodos se produce cuando en una clase hay más de un método con el mismo nombre pero con diferentes listas de parámetros (cantidad o tipo de parámetros). La sobrecarga permite crear varios métodos con el mismo nombre pero que se diferencian en la cantidad o el tipo de argumentos que aceptan.

Ejemplo de Sobrecarga de Métodos:

```
public class Calculadora {
   public int sumar(int a, int b) {
      return a + b;
   }

    public void sumar(int a, int b, int c, String op) {
      sum= a + b+ c;
      System.out.printLn("La suma es: "+sum);
    }

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

   public int sumar(int a, int b, int c) {
      return a + b + c;
    }
}
```

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

        System.out.println(calc.sumar(2, 3)); // Imprime 5
        System.out.println(calc.sumar(2.5, 3.5)); // Imprime 6.0
        System.out.println(calc.sumar(1, 2, 3)); // Imprime 6

        calc.sumar(3, 2, 6, "Hola");
    }
}
```

En este ejemplo, la clase `Calculadora` tiene tres métodos `sumar` sobrecargados, cada uno con diferentes listas de parámetros.

Diferencias Clave

- **Sobreescritura:** Ocurre entre una superclase y una subclase. La subclase redefine el comportamiento de un método heredado.
- **Sobrecarga:** Ocurre dentro de la misma clase. Permite tener múltiples métodos con el mismo nombre pero diferentes parámetros.

Estos conceptos son fundamentales en la programación orientada a objetos, ya que proporcionan flexibilidad y permiten implementar polimorfismo y técnicas de diseño limpio en Java.

Instructor: Cristian David Henao Hoyos