TUGAS : GENERIC PROGRAMMING

MINGGU PERTEMUAN 11

PEMROGRAMAN BERORIENTASI OBJEK (PR)

Disusun oleh :

Muhammad Wildan Gumilang (231511087)

PROGRAM DIPLOMA III TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

DAFTAR ISI

Generic Class

```
public class GenericsType<T> {
    private T t;

    public T get() {
        return this.t;
    }

    public void set(T t1) {
        this.t = t1;
    }

    public static void main(String args[]) {
        GenericsType<String> type = new GenericsType<>();
        type.set("Java");
        GenericsType type1 = new GenericsType();
        type1.set("Java");
        type1.set(10);
        String str = (String) type.get();
        int str1 = (int) type1.get();
        System.out.println(str);
    }
}
```

Jadi, generic class adalah kelas yang fleksibel untuk bekerja dengan berbagai tipe data dan mengurangi risiko kesalahan casting.

Dalam studi kasus ini terdapat class GenericsType<T> yang dapat digunakan dengan tipe data apa saja dengan adanya parameter tipe <T>, yang di mana <T> adalah parameter tipe yang memungkinkan kelas ini bekerja dengan tipe data apa pun yang kita inginkan. Jadi, pada class ini dapat digunakan tipe data yang lebih spesifik (dalam kasus ini string) saat membuat objek dari kelas tersebut. Dalam GenericsType<String> type, variabel t hanya dapat menyimpan data String, sehingga tidak perlu melakukan casting secara manual. Sehingga generic class ini dapat mengurangi resiko kesalahan tipe data (ClassCastException).

Namun jika kita membuat objek tanpa menetapkan tipe data spesifik, atau yang disebut raw type, kita kehilangan keamanan tipe tersebut. Hal ini terjadi pada objek GenericsType type1, yang tidak memiliki batasan tipe, sehingga kita bisa menyimpan nilai apa pun ke dalam variabel t. Namun, ketika kita mencoba mengambil nilainya, kita harus melakukan casting secara manual, yang berpotensi menyebabkan kesalahan runtime jika tipe data yang kita gunakan tidak sesuai.

Generic Methods

```
public class GenericsMethods {
    // Java Generic Method
    public static <T> boolean isEqual(GenericsType<T> g1, GenericsType<T> g2) {
        return g1.get().equals(g2.get());
    }

    public static void main(String args[]) {
        GenericsType<String> g1 = new GenericsType<>();
        g1.set("Java");
        GenericsType<String> g2 = new GenericsType<>();
        g2.set("Java");
        boolean isEqual = GenericsMethods.<String>isEqual(g1, g2);
        // above statement can be written simply as
        isEqual = GenericsMethods.isEqual(g1, g2);
```

```
        System.out.println(isEqual);

        /*
         * This feature, known as type inference, allows you to invoke
         * a generic method as an ordinary method, without specifying a type
         * between angle brackets
         */
        // Compiler will infer the type that is needed
    }

}
```

Generic Method adalah metode yang dideklarasikan dengan parameter tipe, yang memungkinkan metode ini bekerja dengan berbagai jenis data. Dengan method ini, kita dapat membuat fungsi yang lebih fleksibel karena parameter tipe T akan digantikan dengan tipe data yang spesifik saat metode ini dipanggil. Yang dimana dalam studi kasus ini, terdapat Generic Method isEqual, yang mengecek apakah dua objek dari tipe GenericsType<T> mempunyai nilai yang sama. Parameter tipe <T> terdapat pada sebelum tipe kembaliannya yaitu boolean, yang menunjukkan bahwa method ini adalah generic method dan dapat menerima berbagai tipe data.

Pada kode ini, mehtod isEqual menerima dua objek GenericsType<T> sebagai argumen dan membandingkan nilai di dalamnya menggunakan fungsi bawaan equals(). Dan pada main programnya, dibuat dua objek GenericsType<String>, g1 dan g2, dengan tipe String dan mengatur nilai keduanya menjadi "Java". Ketika isEqual dipanggil dengan argumen g1 dan g2, method akan ini mengembalikan true karena kedua objek mempunyai nilai yang sama.

Generic Interface

```
interface MinMax<T extends Comparable<T>> {
    T max(); /* w w w .java2 s . co m */
}

class MyClass<T extends Comparable<T>> implements MinMax<T> {
    T[] vals;

    MyClass(T[] o) {
        vals = o;
    }

    public T max() {
        T v = vals[0];
        for (int i = 1; i < vals.length; i++) {
            if (vals[i].compareTo(v) > 0) {
                v = vals[i];
            }
        }
        return v;
    }
}

public class Main {
    public static void main(String args[]) {
        Integer inums[] = { 3, 6, 2, 8, 6 };
        Character chs[] = { 'b', 'r', 'p', 'w' };
        MyClass<Integer> a = new MyClass<Integer>(inums);
        MyClass<Character> b = new MyClass<Character>(chs);
        System.out.println(a.max());
        System.out.println(b.max());
    }
}
```

Generic interface adalah interface yang didefinisikan dengan parameter tipe, sehingga dapat diimplementasikan dengan berbagai tipe data. Dalam kasus ini, terdapat generic interface MinMax<T>, yang mendefinisikan satu method max(), tipe T harus merupakan kelas yang mengimplementasikan interface Comparable.

Interface MinMax<T> diimplementasikan oleh class MyClass<T>, yang juga menggunakan tipe geeric T. Dalam max(), ditetapkan nilai awal v sebagai elemen pertama vals, lalu membandingkan setiap elemen dengan v menggunakan compareTo(). Dan jika elemen tersebut lebih besar dari v, maka v di-update dengan nilai tersebut.

Bounded Type Parameters

```
class Bound<T extends A> {
    private T objRef;

    public Bound(T obj) {
        this.objRef = obj;
    }

    public void doRunTest() {
        this.objRef.displayClass();
    }
}

class A {
    public void displayClass() {
        System.out.println("Inside super class A");
    }
}

class B extends A {
    public void displayClass() {
        System.out.println("Inside sub class B");
    }
}

class C extends A {
    public void displayClass() {
        System.out.println("Inside sub class C");
    }
}

public class BoundedClass {
    public static void main(String a[]) {
        // Creating object of sub class C and
        // passing it to Bound as a type parameter.
        Bound<C> bec = new Bound<C>(new C());
        bec.doRunTest();
        // Creating object of sub class B and
        // passing it to Bound as a type parameter.
        Bound<B> beb = new Bound<B>(new B());
        beb.doRunTest();
        // similarly passing super class A
        Bound<A> bea = new Bound<A>(new A());
        bea.doRunTest();
    }
}
```

Dengan Bounded Type Parameters, dapat ditetapkan batasan pada tipe data yang dapat digunakan dalam generic parameter, memastikan bahwa hanya tipe data tertentu atau turunannya yang dapat diterima, sehingga mempermudah akses ke method yang spesifik dari

tipe tersebut. Pada kode ini, kelas Bound<T> memiliki variabel objRef bertipe T dan metode doRunTest() yang memanggil metode displayClass() dari objek objRef.

Kelas A didefinisikan dengan metode displayClass() yang menampilkan pesan "Inside super class A". Kelas B dan C adalah subclass dari A, dan masing-masing mengoverride metode displayClass() untuk menampilkan pesan yang berbeda, yaitu "Inside sub class B" dan "Inside sub class C".

Wildcard

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.LinkedList;

/**
 * Wildcard Arguments With An Unknown Type
 *
 * @author javaguides.net
 *
 */
public class WildCardSimpleExample {
    public static void printCollection(Collection<?> c) {
        for (Object e : c) {
            System.out.println(e);
        }
    }

    public static void main(String[] args) {
        Collection<String> collection = new ArrayList<>();
        collection.add("ArrayList Collection");
        printCollection(collection);
        Collection<String> collection2 = new LinkedList<>();
        collection2.add("LinkedList Collection");
        printCollection(collection2);
        Collection<String> collection3 = new HashSet<>();
        collection3.add("HashSet Collection");
        printCollection(collection3);
    }
}
```

Wildcard adalah simbol ? yang digunakan untuk mewakili tipe yang tidak diketahui atau tipe yang dapat bervariasi. Dengan wildcard dapat menulis kode yang lebih fleksibel, sehingga dapat digunakan dengan berbagai jenis koleksi tanpa perlu menentukan tipe elemen secara spesifik.

Dalam kasus ini terdapat method printCollection, yang dapat menerima koleksi dengan tipe elemen yang tidak diketahui. Wildcard ini memungkinkan method printCollection untuk bekerja dengan berbagai jenis koleksi seperti ArrayList, LinkedList, dan HashSet tanpa harus menentukan tipe elemen secara spesifik

Kesimpulan

Jadi, generic programming dapat diterapkan pada kasus ini, yaitu untuk membuat kode yang fleksibel dan bisa bekerja dengan berbagai tipe data. Generic class dapat membuat objek dengan tipe data yang ditentukan saat digunakan. Lalu generic methods dapat membuat sebuah

method untuk bekerja dengan berbagai tipe data tanpa harus menulis kode berbeda untuk setiap tipe. Lalu generic interfaces memberi interface yang bisa digunakan dengan berbagai tipe data. Bounded type parameters membatasi tipe data yang bisa digunakan, agar hanya tipe tertentu yang dapat diproses. Dan wildcard memungkinkan untuk bekerja dengan koleksi yang tipe elemennya tidak diketahui, tanpa mengorbankan keamanan tipe data.

Link Repository GitHub

https://github.com/WildanGumilang/PBO-praktek.git