

LAPORAN PRAKTIKUM PBO



Disusun Oleh:
Muhamad Haikal Hariyanto (231511081)
Jurusan Teknik Komputer dan Informatika

Program Studi D-3 Teknik Informatika
Politeknik Negeri Bandung

10 November 2024

Judul : Pemrograman Berorientasi Object

Hari : Minggu, 10 November 2024

Program:

GenericsTypeOld.java

```
GenericsTypeOld.java > GenericsTypeOld
public class GenericsTypeOld {
    private Object t;

    public Object get() {
        return t;
    }

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

    Run | Debug
    public static void main(String args[]) {
        GenericsTypeOld type = new GenericsTypeOld();
        type.set(t:"Java");
        String str = (String) type.get();
        System.out.println(str);
        // type casting, error prone and can cause ClassCastException
    }
}
```

Output:

```
PS C:\Users\Asus\OneDrive - Politeknik Negeri Bandung\semester 3\PBO\PBO-M.Haikal_Hariyanto\Pertemuan11\Pratikum11> cd "c:\Users\Asus\OneDrive - Politeknik Negeri Bandung\semester 3\PBO\PBO-M.Haikal_Hariyanto\Pertemuan11\Pratikum11\src\" ; if ($?) { javac GenericsTypeOld.java } ; if ($?) { java GenericsTypeOld }
Java
PS C:\Users\Asus\OneDrive - Politeknik Negeri Bandung\semester 3\PBO\PBO-M.Haikal_Hariyanto\Pertemuan11\Pratikum11\src>
```

Penjelasan:

Kelas GenericsTypeOld memiliki variabel `t` dengan tipe `Object`, yang dapat menyimpan data dari tipe apapun. Metode `set` menginisialisasi variabel `t` dengan nilai yang diberikan, sementara metode `get` mengembalikan nilai `t`. Karena tidak ada generics, tipe data yang disimpan tidak diketahui secara pasti pada saat kompilasi, sehingga untuk mengambil nilai yang benar, harus melakukan casting. Ini menyebabkan program rentan terhadap kesalahan runtime seperti `ClassCastException` jika casting dilakukan ke tipe yang salah. Selain itu, pendekatan ini tidak type-safe, yang berarti pengecekan tipe dilakukan saat runtime, bukan saat kompilasi. Pada akhirnya, program mencetak "Value stored in type: Java".

Program:

GenericsType.java

```

GenericsType.java > GenericsType<T>
public class GenericsType<T> {
    private T t;

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

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

    Run | Debug
    public static void main(String args[]) {
        GenericsType<String> type = new GenericsType<>();
        type.set(t1:"Java"); // valid
        System.out.println("Value stored in type: " + type.get());
        GenericsType type1 = new GenericsType(); // raw type
        type1.set(t1:"Java"); // valid
        System.out.println("Value stored in type1: " + type1.get());
        type1.set(t1:10); // valid and autoboxing support

        System.out.println("Value stored in type1: " + type1.get());
    }
}

```

Output:

```

mester 3\PB0\PB0-M.Haikal_Hariyanto\Pertemuan11\Pratikum11\src> if ($?) { javac GenericsType.java } ; if ($?) { java GenericsType }
Note: GenericsType.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Value stored in type: Java
Value stored in type1: Java
Value stored in type1: 10
PS C:\Users\Asus\OneDrive - Politeknik Negeri Bandung\semester 3\PB0\PB0-M.Haikal_Hariyanto\Pertemuan11\Pratikum11\src>

```

Penjelasan

kelas `GenericsType<T>` menggunakan parameter tipe `T` untuk membuat kelas lebih fleksibel dan type-safe. Variabel `t` sekarang memiliki tipe `T`, yang ditentukan saat instansiasi kelas. Sebagai contoh, `GenericsType<String>` menetapkan bahwa tipe `T` adalah `String`, yang memastikan hanya `String` yang bisa disimpan dan diambil dari objek ini. Ini menghindari kebutuhan casting eksplisit dan memungkinkan pengecekan tipe pada saat kompilasi, yang mengurangi kemungkinan kesalahan runtime. Di dalam `main`, program menunjukkan dua cara penggunaan: dengan menentukan tipe (contoh `GenericsType<String>`) dan tanpa menentukan tipe (contoh `GenericsType type1` atau "raw type"). Penggunaan raw type tidak dianjurkan karena menghilangkan keunggulan type-safe dari generics. Program ini menghasilkan output berupa "Value stored in type: Java" dan "Value stored in type1: 10", yang menunjukkan bahwa `type1` dapat menyimpan tipe data apapun.

Program:

Main.java

```

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 {
    Run | Debug
    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());
    }
}

```

Output:

```

PS C:\Users\Asus\OneDrive - Politeknik Negeri Bandung\semester 3\PBO\PBO-M.Haikal_Hariyanto\Pertemuan11\Pratikum11\src> cd
mester 3\PBO\PBO-M.Haikal_Hariyanto\Pertemuan11\Pratikum11\src\ ; if ($?) { javac Main.java } ; if ($?) { java Main }
8
w
PS C:\Users\Asus\OneDrive - Politeknik Negeri Bandung\semester 3\PBO\PBO-M.Haikal_Hariyanto\Pertemuan11\Pratikum11\src>

```

Penjelasan

Program ini menunjukkan penggunaan interface generic dengan parameter tipe yang dibatasi (bounded type parameter). Interface `MinMax<T>` mendeklarasikan metode `max()` untuk mencari nilai maksimum, dan kelas `MyClass<T>` mengimplementasikannya. Parameter tipe `T` dibatasi untuk tipe yang mengimplementasikan `Comparable`, sehingga memastikan bahwa tipe `T` mendukung operasi perbandingan melalui metode `compareTo()`. Dengan demikian, metode `max()` dalam `MyClass` dapat mencari nilai tertinggi dalam array `vals` dengan tipe `T`. Di dalam `main`, dua instance `MyClass` dibuat, satu untuk array `Integer` dan satu untuk `Character`. Program ini menghasilkan output "Maximum integer value: 8" dan "Maximum character value: w", menunjukkan nilai maksimum dalam masing-masing array. Bounded generic di sini memberikan fleksibilitas tipe sambil memastikan tipe-tipe tersebut kompatibel dengan `Comparable`.

Program:

GenericsMethods.java

```
GenericsMethods.java 7 GenericsMethods
public class GenericsMethods {
    // Java Generic Method
    public static <T> boolean isEqual(GenericsType<T> g1, GenericsType<T> g2) {
        return g1.get().equals(g2.get());
    }

    Run | Debug
    public static void main(String args[]) {
        GenericsType<String> g1 = new GenericsType<>();
        g1.set(t1:"Java");
        GenericsType<String> g2 = new GenericsType<>();
        g2.set(t1:"Java");
        boolean isEqual = GenericsMethods.<String>isEqual(g1, g2);
        // above statement can be written simply as
        isEqual = GenericsMethods.isEqual(g1, g2);
        /*
         * 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
        System.out.println("Are the values equal? " + isEqual);
    }
}
```

Output:

```
PS C:\Users\Asus\OneDrive - Politeknik Negeri Bandung\semester 3\PBO\PBO-M.Haikal_Hariyanto\Pertem
mester 3\PBO\PBO-M.Haikal_Hariyanto\Pertemuan11\Pratikum11\src\ ; if ($?) { javac GenericsMethod
Are the values equal? true
PS C:\Users\Asus\OneDrive - Politeknik Negeri Bandung\semester 3\PBO\PBO-M.Haikal_Hariyanto\Pert
```

Penjelasan

Program ini memperkenalkan metode generic melalui metode `isEqual()`, yang menerima dua objek `GenericsType<T>` dan membandingkan nilai mereka menggunakan metode `equals()`. Parameter tipe `<T>` pada metode ini memungkinkan metode untuk bekerja dengan tipe data apapun tanpa mengikat tipe spesifik pada kelasnya. Metode ini memastikan keamanan tipe (type-safe) pada level metode dan meminimalkan kemungkinan kesalahan runtime karena casting. Di dalam `main`, dua objek `GenericsType<String>` dibuat, masing-masing berisi nilai

"Java". Metode isEqual() kemudian digunakan untuk membandingkan kedua objek dan mengembalikan true jika nilainya sama. Program ini menghasilkan output "Are the values equal? true" yang menunjukkan bahwa kedua objek memiliki nilai yang sama.

Program:

Bounded.class

```
boundedClass.java / Bound<T> extends A>
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(x:"Inside super class A");
    }
}

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

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

public class BoundedClass {
    Run | Debug
    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();
    }
}
```

Output:

```
mester 3\PBO\PBO-M.Haikal_Hariyanto
Inside sub class C
Inside sub class B
Inside super class A
PS C:\Users\Asus\OneDrive - Politekn
```

Penjelasan

Program ini menunjukkan bounded type parameters dengan kelas `Bound<T extends A>`, yang memaksa parameter `T` untuk menjadi subclass dari kelas `A`. Di dalam kelas `Bound`, variabel `objRef` bertipe `T` menyimpan referensi objek, dan metode `doRunTest()` memanggil metode `displayClass()` dari `objRef`. Kelas `A` memiliki metode `displayClass()`, yang diturunkan ke subclass `B` dan `C`. Di dalam `main`, tiga instance `Bound` dibuat dengan tipe `C`, `B`, dan `A`, yang masing-masing memanggil metode `displayClass()` sesuai dengan kelas mereka. Output menunjukkan bahwa kelas `Bound` fleksibel untuk digunakan dengan `A` dan turunannya, menghasilkan output "Inside sub class C", "Inside sub class B", dan "Inside super class A". Program ini membatasi tipe `T` untuk tipe yang mewarisi `A`, menjaga keamanan tipe dan mencegah penggunaan tipe yang tidak sesuai.

Program:

WildcardSimpleExample.java

```
WildcardSimpleExample.java > ...
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);
        }
    }

    Run | Debug
    public static void main(String[] args) {
        Collection<String> collection = new ArrayList<>();
        collection.add(e:"ArrayList Collection");
        System.out.println(x:"ArrayList Collection:");
        printCollection(collection);
        Collection<String> collection2 = new LinkedList<>();
        collection2.add(e:"LinkedList Collection");
        System.out.println(x:"LinkedList Collection:");
        printCollection(collection2);
        Collection<String> collection3 = new HashSet<>();
        collection3.add(e:"HashSet Collection");
        System.out.println(x:"HashSet Collection:");
        printCollection(collection3);
    }
}
```

Output:

```
mester 3\PBO\PBO-M.Haikal_
ArrayList Collection:
ArrayList Collection
LinkedList Collection:
LinkedList Collection
HashSet Collection:
HashSet Collection
PS C:\Users\Asus\OneDrive
```

Penjelasan

Program ini menggunakan wildcard generic `<?>` dalam metode `printCollection(Collection<?>)`, yang memungkinkan metode tersebut untuk menerima koleksi dengan tipe data apapun. Dengan menggunakan wildcard `<?>`, metode ini fleksibel untuk menangani koleksi berbagai tipe (seperti `ArrayList`, `LinkedList`, dan `HashSet`) tanpa memerlukan tipe tertentu. Di dalam main, beberapa koleksi dibuat dengan berbagai jenis seperti `ArrayList<String>`, `LinkedList<String>`, dan `HashSet<String>`, dan masing-masing ditambahkan dengan elemen string. Setiap koleksi kemudian dicetak menggunakan metode `printCollection`, yang menghasilkan output seperti "ArrayList Collection: ArrayList Collection", "LinkedList Collection: LinkedList Collection", dan "HashSet Collection: HashSet Collection". Wildcard generic memberikan fleksibilitas ekstra untuk menerima berbagai jenis koleksi tanpa menetapkan tipe yang ketat, yang sangat bermanfaat dalam situasi di mana tipe spesifik tidak diperlukan atau diinginkan.