**LAPORAN PRAKTIKUM 12**

**PEMROGRAMAN BERORIENTASI OBJEK (PBO)**



Nama : Alya Gustiani Nur ‘Afifah

NIM : 231511035

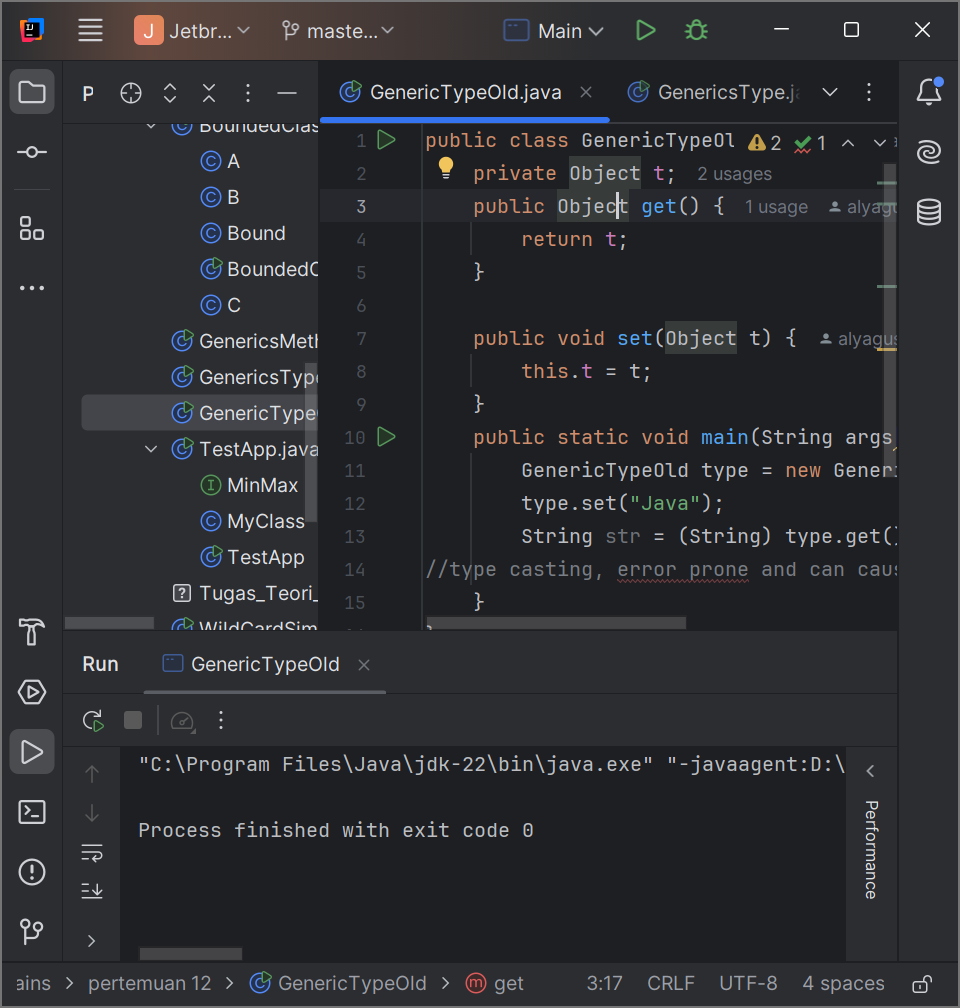
Kelas/Prodi : 2B/D3 Teknik Informatika

**Politeknik Negeri Bandung**

**2024**

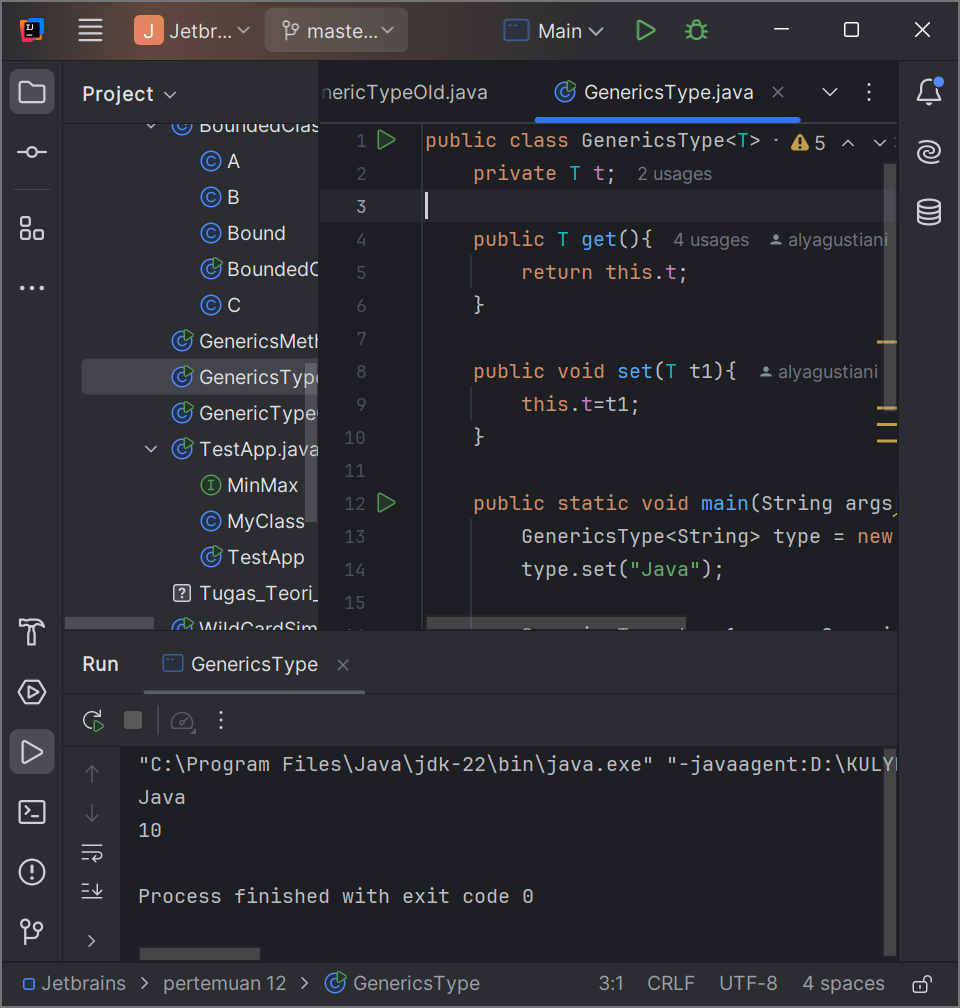
1. Link Repository GitHub : <https://github.com/alyagustiani/PBOSem2> (pertemuan 12) penjelasan ada di paling bawah
2. GenericsTypeOld

public class GenericTypeOld {  
 private Object t;  
 public Object get() {  
 return t;  
 }  
  
 public void set(Object t) {  
 this.t = t;  
 }  
 public static void main(String args[]){  
 GenericTypeOld type = new GenericTypeOld();  
 type.set("Java");  
 String str = (String) type.get();  
//type casting, error prone and can cause ClassCastException  
 }  
}

Hasil akhir :

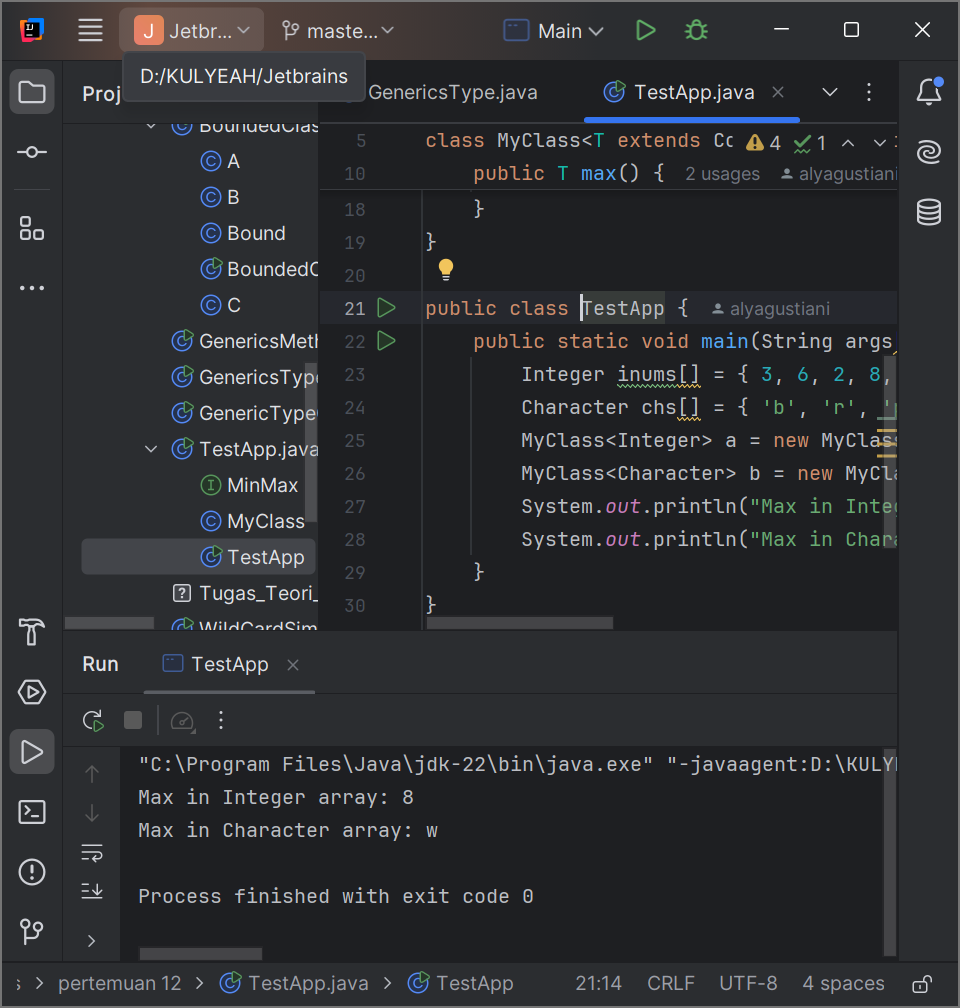
1. GenericsType<T>

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(); //raw type  
 type1.set("Java");  
 type1.set(10);  
  
 System.*out*.println(type.get());  
 System.*out*.println(type1.get());  
 }  
}

Hasil akhir :

1. Main (TestApp)

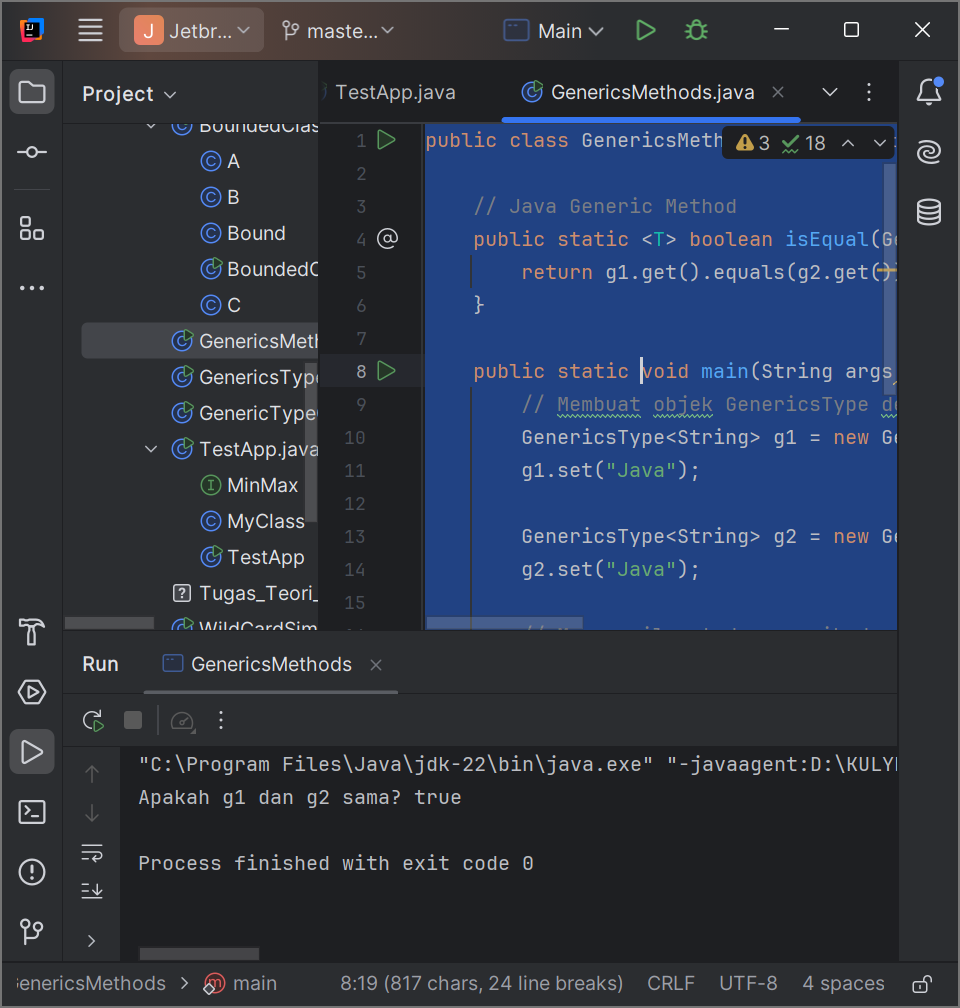
interface MinMax<T extends Comparable<T>> {  
 T max();  
}  
  
class MyClass<T extends Comparable<T>> implements MinMax<T> {  
 T[] vals;  
 MyClass(T[] o) {  
 vals = o;  
 }  
 public T max() {  
 T v = vals[0];  
 for (int i = 1; i < vals.length; i++) {  
 if (vals[i].compareTo(v) > 0) {  
 v = vals[i];  
 }  
 }  
 return v;  
 }  
}  
  
public class TestApp {  
 public static void main(String args[]) {  
 Integer inums[] = { 3, 6, 2, 8, 6 };  
 Character chs[] = { 'b', 'r', 'p', 'w' };  
 MyClass<Integer> a = new MyClass<>(inums);  
 MyClass<Character> b = new MyClass<>(chs);  
 System.*out*.println("Max in Integer array: " + a.max());  
 System.*out*.println("Max in Character array: " + b.max());  
 }  
}

Hasil akhir :

1. GenericsMethods

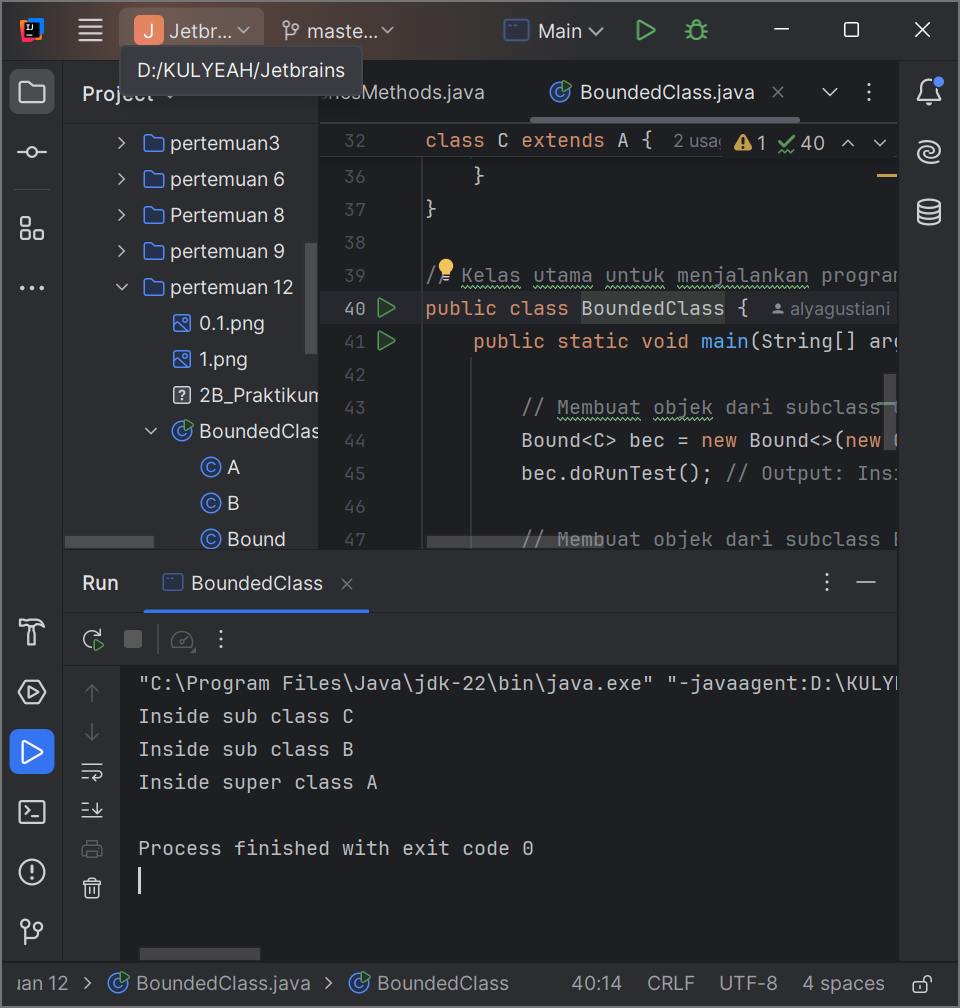
public class GenericsMethods {  
  
 // Java Generic Method  
 public static <T> boolean isEqual(GenericsType<T> g1, GenericsType<T> g2) {  
 return g1.get().equals(g2.get());  
 }  
  
 public static void main(String args[]) {  
 // Membuat objek GenericsType dengan tipe String  
 GenericsType<String> g1 = new GenericsType<>();  
 g1.set("Java");  
  
 GenericsType<String> g2 = new GenericsType<>();  
 g2.set("Java");  
  
 // Memanggil metode generik dengan eksplisit tipe parameter  
 boolean isEqual = GenericsMethods.<String>*isEqual*(g1, g2);  
  
 // Panggilan yang sama tanpa menyebutkan tipe parameter, compiler akan melakukan type inference  
 isEqual = GenericsMethods.*isEqual*(g1, g2);  
  
 System.*out*.println("Apakah g1 dan g2 sama? " + isEqual);  
 }  
}

Hasil akhir :



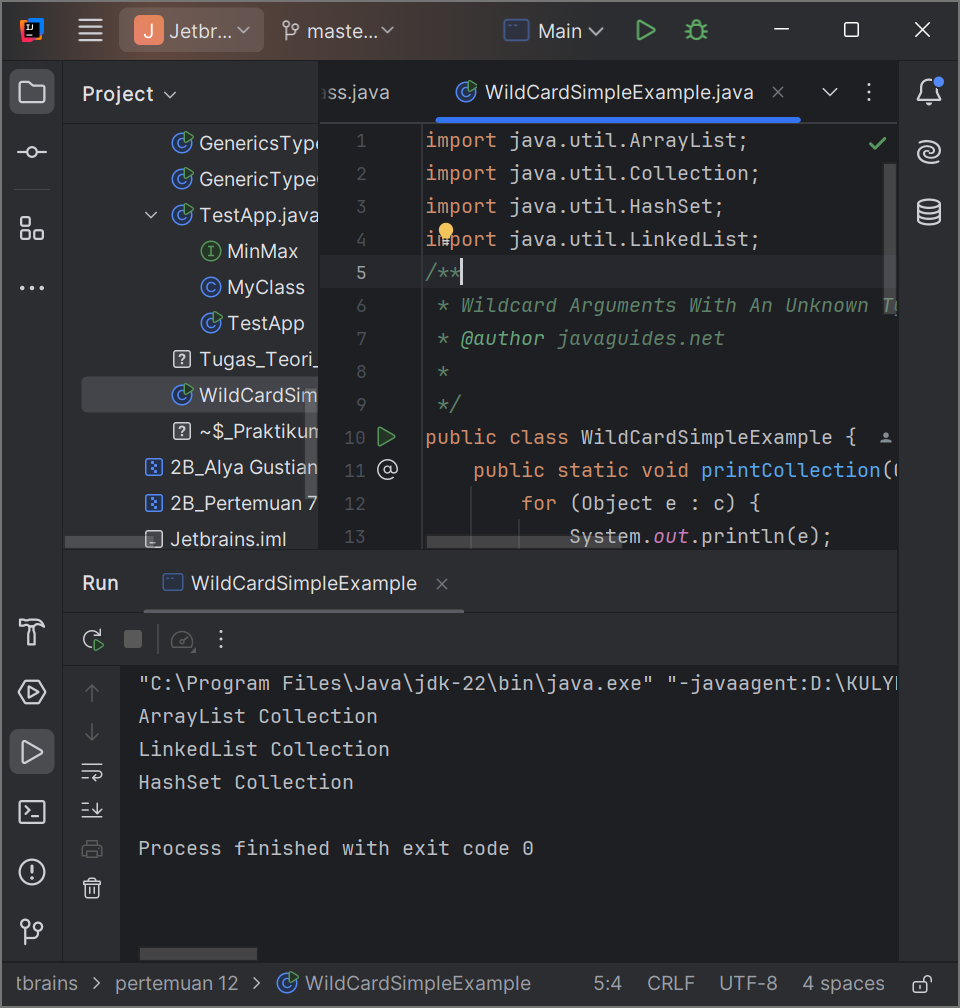
1. BoundedClass

// Kelas Bound dengan tipe parameter yang dibatasi pada kelas A atau turunannya  
class Bound<T extends A> {  
 private T objRef;  
  
 // Konstruktor untuk menginisialisasi objRef  
 public Bound(T obj) {  
 this.objRef = obj;  
 }  
  
 // Metode untuk menjalankan tes yang memanggil displayClass dari objRef  
 public void doRunTest() {  
 this.objRef.displayClass();  
 }  
}  
  
// Kelas A (superclass)  
class A {  
 public void displayClass() {  
 System.*out*.println("Inside super class A");  
 }  
}  
  
// Kelas B yang merupakan subclass dari A  
class B extends A {  
 @Override  
 public void displayClass() {  
 System.*out*.println("Inside sub class B");  
 }  
}  
  
// Kelas C yang juga merupakan subclass dari A  
class C extends A {  
 @Override  
 public void displayClass() {  
 System.*out*.println("Inside sub class C");  
 }  
}  
  
// Kelas utama untuk menjalankan program  
public class BoundedClass {  
 public static void main(String[] args) {  
  
 // Membuat objek dari subclass C dan melewatkannya ke Bound sebagai parameter tipe  
 Bound<C> bec = new Bound<>(new C());  
 bec.doRunTest(); // Output: Inside sub class C  
  
 // Membuat objek dari subclass B dan melewatkannya ke Bound sebagai parameter tipe  
 Bound<B> beb = new Bound<>(new B());  
 beb.doRunTest(); // Output: Inside sub class B  
  
 // Membuat objek dari superclass A dan melewatkannya ke Bound sebagai parameter tipe  
 Bound<A> bea = new Bound<>(new A());  
 bea.doRunTest(); // Output: Inside super class A  
 }  
}

Hasil akhir :

1. WildCardSimpleExample

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);  
 }  
}

Hasil akhir :

Penjelasan masing – masing generics

1. GenericsTypeOld

Tanpa generics, kita harus melakukan casting secara manual yang rentan terhadap kesalahan dan dapat menyebabkan ClassCastException. Pada contoh ini, Object digunakan sebagai tipe umum, tetapi ini tidak aman karena harus melakukan casting secara manual saat mendapatkan nilai.

1. GenericsType

Deklarasi kelas (<T>) dan metode (public void set(T t1)). Generics membuat kode lebih aman dan mudah dibaca dengan menghilangkan kebutuhan untuk casting dan memungkinkan penggunaan tipe tertentu.

1. MinMax dan My Class

Deklarasi antarmuka (<T extends Comparable<T>>) dan kelas (<T extends Comparable<T>>). Membatasi tipe yang digunakan sehingga hanya tipe yang mengimplementasikan Comparable dapat digunakan, memungkinkan operasi perbandingan di dalam metode max.

1. GenericsMethods

Deklarasi metode (<T>) dan parameter (GenericsType<T> g1). Memungkinkan metode isEqual bekerja dengan objek dari tipe yang berbeda tanpa kehilangan keamanan tipe.

1. BoundedClass

kelas (<T extends A>). Membatasi tipe parameter generics sehingga hanya kelas yang mewarisi A yang dapat digunakan, memungkinkan pemanggilan metode displayClass.

1. WildCardSimpleExaample

Parameter metode (Collection<?>). Wildcard ? memungkinkan metode printCollection menerima koleksi dari tipe apapun tanpa kehilangan fleksibilitas atau keamanan tipe.