Generics

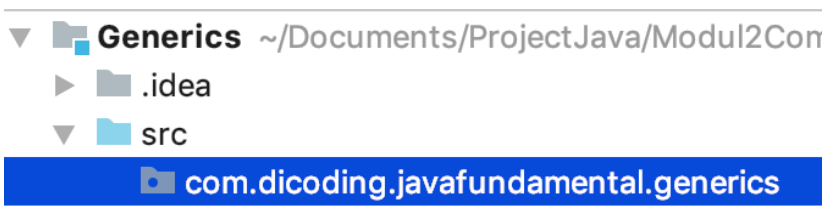
Generics adalah salah satu fitur yang terdapat di JDK 1.5. Ia sangat ditunggu-tunggu. Salah satunya untuk menyederhanakan penulisan kode dari *type-casting* juga untuk *compile-time type safety*. Generics dalam kode program bisa dikenali dengan *type-parameter*. Contoh penggunaan Generics paling umum adalah Collection.

Sebelumnya di materi Collection kita sudah menggunakan fitur Generics yaitu ketika kita deklarasi Set<String>, Set dengan *type-parameter* <String>. Kita juga deklarasi Map<String, Planet>, Map dengan *type-parameter* <String, Planet>. Tetapi untuk List kita deklarasi tanpa *type-parameter*. Apa perbedaannya? Ayo kita lihat dengan coding!

Codelab Generics

Mari kita praktikan Codelab berikut agar lebih paham mengenai Generics.

1. Buatlah proyek baru dengan nama Generics dengan nama package `com.dicoding.javafundamental.generics` di dalamnya:



2. Buatlah sebuah kelas baru di dalamnya dengan nama `Planet`, kemudian tambahkan kode berikut:

```
1.  package com.dicoding.javafundamental.generics;
2.
3.  class Planet {
4.      private String name;
5.      private double mass;
6.
7.      public Planet(String name, double mass) {
8.          this.name = name;
9.          this.mass = mass;
10.     }
11.
12.     public void print() {
13.         System.out.println("Planet " + name + ", mass: " + mass);
14.     }
15. }
```

3. Selanjutnya buatlah kelas `Main` dan tambahkan kode berikut:

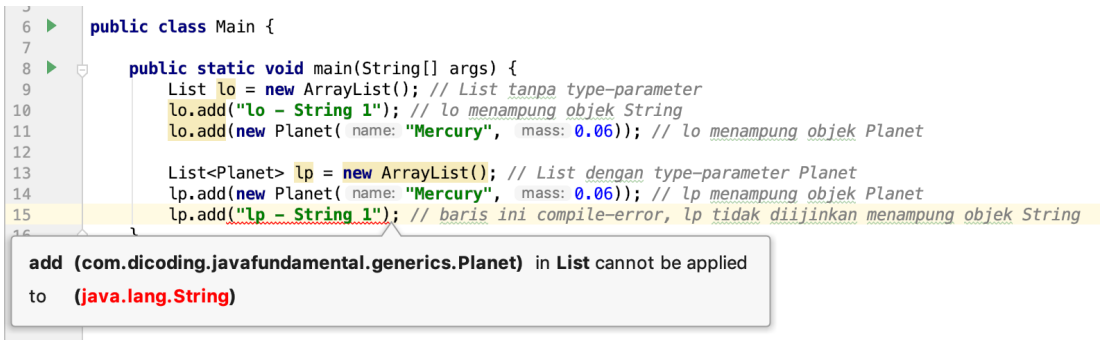
```java
package com.dicoding.javafundamental.generics;

import java.util.ArrayList;
import java.util.List;

public class Main {

    public static void main(String[] args) {
        List lo = new ArrayList(); // List tanpa type-parameter
        lo.add("lo - String 1"); // lo menampung objek String
        lo.add(new Planet("Mercury", 0.06)); // lo menampung objek Planet

        List<Planet> lp = new ArrayList(); // List dengan type-parameter Planet
        lp.add(new Planet("Mercury", 0.06)); // lp menampung objek Planet
        lp.add("lp - String 1"); // baris ini compile-error, lp tidak diijinkan menampung objek String
```



Dari kode di atas terlihat List<Planet> lp tidak diizinkan menampung objek selain Planet. Dalam kasus ini List<Planet> lp dilindungi *compile-time type safety* di mana jika ada objek lain yang dimasukkan ke List<Planet> lp . Tetapi dengan tipe selain Planet , seketika *compile error,* Artinya deteksilah lebih dahulu sebelum *runtime* (program dijalankan). Bandingkan dengan List lo yang bisa menampung objek String ataupun Planet (bahkan semua jenis objek). Sebagai contoh, kita ingin loop kedua List tersebut untuk memanggil method print dari class `Planet`.

4. Ubahlah kelas `Main` menjadi seperti ini:

```java
package com.dicoding.javafundamental.generics;

import java.util.ArrayList;
import java.util.List;

public class Main {

    public static void main(String[] args) {
        List lo = new ArrayList(); // List tanpa type-parameter
        lo.add("lo - String 1"); // lo menampung objek String
        lo.add(new Planet("Mercury", 0.06)); // lo menampung objek Planet

        for (Object o : lo) {
            Planet p = (Planet) o; // perlu type-casting dari Object ke Planet
            p.print();
```

5. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan output seperti ini.

```
Exception in thread "main" java.lang.ClassCastException: class java.lang.String cannot be cast to class com.dicoding.javafundamental.generics.Planet (java.lang.String is in module java.base of loader 'bootstrap'; com.dicoding.javafundamental.generics.Planet is in unnamed module of loader 'app')
at com.dicoding.javafundamental.generics.Main.main(Main.java:14)
```

Perhatikan kode untuk loop masing-masing List. Untuk List lo perlu dilakukan *type-casting* sedangkan untuk List<Planet> lp tidak perlu. Dalam kasus ini terlihat penggunaan Generics membuat kode lebih sederhana seperti yang sudah di bahas di paragraf awal. Dari kode di atas tidak ada *compile-error* tetapi saat kita jalankan akan terjadi *runtime-error.* Sebabnya, dalam List lo ada objek yang tidak bisa di-*cast* ke Planet.

Suatu saat kode program kita akan menjadi besar, ribuan baris kode! Bayangkan jika kita tidak menggunakan *type-parameter* saat deklarasi objek Collection. Lalu objek tersebut digunakan di tempat lainnya, misal dikirim sebagai parameter ke suatu method. Bisa saja tanpa sadar kita memasukkan tipe objek yang salah dan baru akan ketahuan ketika program dijalankan. Pusing kan? Nah, gunakan *type-parameter.* Paling tidak kita akan terbantu karena masalah terdeteksi lebih dini saat *compile*.

Wildcards

Perhatikan baris kode di bawah ini.

```
1.  package com.dicoding.javafundamental.generics;
2.
3.  import java.util.ArrayList;
4.  import java.util.Collection;
5.  import java.util.List;
6.
7.  public class Wildcard {
8.      public static void main(String[] args) {
9.          List<String> ls = new ArrayList();
10.         ls.add("String1");
11.         ls.add("String2");
12.
13.         print(ls); // Apakah baris ini valid?
14.
15.         Collection<Planet> cp = new ArrayList();
```

Apakah baris yang ditanyakan di atas valid? Coba pindahkan ke IntelliJ. Jawabannya adalah tidak valid. Loh kenapa begitu? Berdasarkan materi Inheritance, bukankah class String dan class Planet adalah turunan dari class Object?

Ya benar tetapi List<String> dan Collection<Planet> bukan turunan (*subtype*) dari Collection<Object> atau Collection<Object> bukan *supertype* dari Collection<Planet> dan List<String>. Lalu apa *supertype* dari semua tipe Collection? Hal ini dikenal dengan nama *wildcard type*, ditulis dengan *syntax* Collection<?> yang artinya *collection of unknown*. Kita tulis ulang sebagian kode di atas menggunakan *wildcard type*.

```
1.  package com.dicoding.javafundamental.generics;
2.
3.  import java.util.ArrayList;
4.  import java.util.Collection;
5.  import java.util.List;
6.
7.  public class Wildcard {
8.      public static void main(String[] args) {
9.          List<String> ls = new ArrayList();
10.         ls.add("String1");
11.         ls.add("String2");
12.
13.         print(ls); // Baris ini valid
14.
15.         Collection<Planet> cp = new ArrayList();
```

Sekarang kode kita sebelumnya menjadi valid. Perhatikan sekarang method `print()` menjadi bisa dipanggil dengan tipe Collection yang berbeda atau *reuseable*.

Generic Methods

Perhatikan kode di bawah ini.

```
1.  static void arrayToCollection(Object[] a, Collection<?> c) {
2.     for (Object o : a) {
3.         c.add(o); // baris ini tidak valid
4.     }
5.  }
```

Kode di atas tidak valid karena Collection<?> c adalah *collection of unknown type* dan kita menambahkan tipe Object o. Sekarang kita ubah kode kita menjadi sebagai berikut.

```
1.  static <T> void arrayToCollection(T[] a, Collection<T> c) {
2.     for (T o : a) {
3.         c.add(o); // baris ini valid
4.     }
5.  }
```

Kode kita menjadi menarik bukan? Perhatikan penambahan huruf T. Kita bisa mengganti dengan huruf apapun, tetapi anjuran *coding convention* menggunakan huruf T, mengacu ke *type*. Cara penulisan kode seperti di atas dikenal dengan istilah *Generic Methods*. Lalu kita bisa gunakan method di atas dengan cara seperti di bawah ini.

```
1.  package com.dicoding.javafundamental.generics;
2.
3.  import java.util.ArrayList;
4.  import java.util.Collection;
5.
6.  public class GenericsMethod {
7.
8.      private static <T> void arrayToCollection(T[] a, Collection<T> c) {
9.          for (T o : a) {
10.             c.add(o); // baris ini valid
11.         }
12.     }
13.
14.     public static void main(String[] args) {
15.         Object[] oa = new Object[100];
```

Wildcards vs Generic Methods

Setelah kita pelajari *wildcards* dan *generic methods* timbul pertanyaan baru. Kapan kita menggunakan *wildcards* dan kapan menggunakan *generic methods*? Jawaban sederhananya adalah: ketika balikan (*return-type*) dari suatu method tidak bergantung kepada tipe parameter (*parameter-type*). Atau kita hanya ingin memanfaatkan fitur polymorphism untuk tipe parameter method tersebut maka gunakanlah *wildcards*. Jika ada keterkaitan antara *return-type* dan *parameter-type* maka gunakanlah *generic methods*. Untuk memperjelas maksud kalimat di atas mari kita simak kode di bawah ini.

```
1.  package com.dicoding.javafundamental.generics;
2.
3.  import java.util.ArrayList;
4.  import java.util.Collection;
5.
6.  public class WildcardVSGenericsMethod {
7.      // mengunakan wildcards
8.      static void print(Collection<?> col) {
9.          for (Object o : col) {
10.             System.out.println(o);
11.         }
12.     }
13.
14.     // menggunakan generic methods
15.     static <T> Collection arrayToCollection(T[] a) {
```

Pada metode `print()`, kita menggunakan wildcard sebagai parameternya agar dapat bekerja dengan beragam tipe data. Penggunaan wildcard juga dapat kita lakukan saat menuliskan kode metode generic class yang tidak bergantung pada tipe parameternya. Misalnya, `List.size` atau `List.clear`. Sementara pada metode `arrayToCollection`, kita mengembalikan nilai dengan tipe Collection dan untuk tipe parameter kita gunakan T. Tanpa harus menuliskan <String>arrayToCollection(sa). Fitur ini dinamakan *type inference,* memungkinkan kita untuk memanggil generic method sebagai metode biasa, tanpa menentukan jenis antara kurung sudut.

Sebelumnya

Selanjutnya

dicoding

Dicoding Space
Jl. Batik Kumeli No.50, Sukaluyu,
Kec. Cibeunying Kaler, Kota Bandung
Jawa Barat 40123

Penghargaan

image
click bila
belum muncul

image
click bila
belum muncul

Decode Ideas
Discover Potential

Blog
Reward
Showcase
Tentang Kami
Hubungi Kami
FAQ