Bab 13
Pengenalan Generics

13.1 Tujuan

Release Java terbaru menyediakan langkah terbesar dalam pemrograman Java dibandingkan dengan versi-versi lain sebelumnya. Ini terdiri atas ekstensi yang cukup signifikan terhadap source language syntax. Bagian yang paling terlihat yaitu penambahan generic types.

Modul ini mengenalkan Anda konsep dasar mengenai Java generic types.

Setelah melengkapi pembahasan ini, anda harus dapat :
1. Mengapresiasikan keuntungan dari generic types
2. Mendeklarasikan class generic
3. Menggunakan constrained generics
4. Mendeklarasikan method generic

13.2 Mengapa Generics?

Satu dari penyebab adanya bugs yang paling signifikan dalam Bahasa pemrograman Java adalah keperluan akan pernyataan typecast atau downcast untuk lebih mengkhususkan tipe data dari tipe staticnya secara terus-menerus. Sebagai contoh, Sebuah object arraylist memungkinkan kita untuk menambahkan beberapa referensi tipe object pada list tapi ketika kita mendapatkan kembali elemet ini , kita perlu untuk typecast object ke tipe referensi khusus yang sesuai dengan keperluan kita. Downcasting adalah hotspot yang potensial untuk *ClassCastException*. Yang juga membuat kode kita menjadi lebih panjang, jadi, menjadi lebih sedikit dapat terbaca. Lebih dari itu, downcasting juga efektif merusak manfaat dari sebuah kekuatan bahasa yang diketikkan sejak dia menghapuskan keamanan yang disediakan perusahaan berupa pemeriksaan tipe(type checking).

Tujuan utama dari penambahan generics pada Java adalah untuk memecahkan masalah ini. tipe Generic memungkinkan subuah single class untuk bekerja dengan pilihan tipe yang beraneka ragam. Ini adalah jalan yang alami untuk mengeliminasi keperluan untuk pemilihan (casting).

Mari pertama-tama mempertimbangkan sebuah object *ArrayList* dan lihat bagaimana type generic akan membantu dalam peningkatan kode kita. Seperti yang sudah Anda ketahui, sebuah object arraylist memiliki kemampuan untuk menyimpan elemen-elemen dari beberapa tipe referensi untuk list ini. Sebuah instance arraylist, bagaimanapun, selalu memaksa kita untuk men-downcast object-object kita membantu mendapatkan kembali dari list. Pertimbangkan pernyataan berikut :

```
String myString = (String) myArrayList.get(0);
```

Versi generic dari class *ArrayList* didesain untuk bekerja secara asli dengan beberapa tipe class. Sama seperti, dia juga mempertahankan manfaat dari pengecekan tipe (type

checking). Kita dapat melanjutkannya dengan keperluan memiliki typecast elemen yang kita dapatkan dari list dan memiliki pernyataan berikut terhadap pernyataan sebelumnya :

```
String myString = myArrayList.get(0);
```

Walaupun downcasting sudah terhapus, ini bukan berarti bahwa Anda dapat menandai segala sesuatu sebagai return value dari method *get* dan melanjutkannya dengan typecasting semuanya. Jika Anda menandai sesuatu yang lain disamping sebuah *String* untuk output dari method *get*, Anda akan menghadapi sebuah waktu mengcompile type tidak sesuai seperti pesan berikut ini :

```
found: java.lang.String
required: java.lang.Integer
Integer data = myArrayList.get(0);
```

Untuk Anda agar hanya memiliki ide bagaimana type-type generic digunakan sebelum materi ini digali lebih dalam, pertimbangkan potongan kode berikut ini :

```
ArrayList <String> genArrList = new ArrayList <String>();
genArrList.add("A generic string");
String myString = genArrList.get(0);
JoptionPane.showMessageDialog(this, myString);
```

Amatilah melalui pernyataan, anda mungkin mengamati kata <*String*> segera terlihat setelah referensi tipe data arraylist. Anda dapat menerjemahkan pernyataan pertama sebagai instantiasi sebuah versi generic dari class *ArrayList dan versi generic* ini terdiri dari object-object dari tipe *String*. *genArrList* adalah batas dari tipe *String*. Oleh sebab itu, mengikat sebuah Integer atau beberapa tipe lain bukan String untuk hasil dari get function akan menjadi illegal. Pernyataan berikut ini adalah illegal.

```
int myInt = genArrList.get();
```

13.3 Mendeklarasikan sebuah Class Generic

Untuk menjalankan potongan code sebelumnya, kita harus sudah mendefinisikan versi generic dari class *ArrayList*. Untungnya, versi java terbaru sudah menyediakan user dengan versi generic dari semua class-class Java *Collection*. Pada sesi ini, Anda akan mempelajari bagaimana untuk mendeklarasikan class generic anda sendiri.
Dripada berdiskusi lebihpanjang lagi tentang bagaimana untuk mendeklarasikan sebuah class generic , anda akan diberikan sebuah contoh sederhana tentang class generic untuk dipelajari bentuknya.

```
class BasicGeneric <A> {
   private A data;
   public BasicGeneric(A data) {
      this.data = data;
   }
   public A getData() {
      return data;
   }
}
```

```
public class GenSample {
   public String method(String input) {
      String data1 = input;
      BasicGeneric <String> basicGeneric = new
                                    BasicGeneric <String>(data1);
      String data2 = basicGeneric.getData();
      return data2;
   }
   public Integer method(int input) {
      Integer data1 = new Integer(input);
      BasicGeneric <Integer> basicGeneric = new
                                   BasicGeneric <Integer>(data1);
      Integer data2 = basicGeneric.getData();
      return data2;
   }
   public static void main(String args[]) {
      GenSample sample = new GenSample();
      System.out.println(sample.method("Some generic data"));
      System.out.println(sample.method(1234));
   }
}
```

Sekarang mari kita melalui bagian dari kode yang menggunakan syntax untuk type generic.
Untuk deklarasi dari class *BasicGeneric*,

```
class BasicGeneric <A>
```

nama class diikuti oleh sepasang kurung yang didalamnya terdapat huruf kapital *A*: *<A>*. Ini disebut dengan sebuah parameter tipe. Penggunaan kurung ini mengindikasikan bahwa class yang dideklarasikan adalah class generic. Ini berarti bahwa class tidak bekerja dengan beberapa type referensi khusus.
kemudian, amati bahwa sebuah field dari class dideklarasikan menjadi tipe A

```
private A data;
```

Deklarasi ini mengelompokkan bahwa field *data* adalah dari tipe generic, tergantung pada tipe data yang telah didesain untuk bekerja dengan object *BasicGeneric*.

Ketika mendeklarasikan sebuah instance dari class, anda harus mengelompokkan tipe referensi dengan yang mana yang anda inginkan untuk bekerja sama.

```
BasicGeneric <String> basicGeneric = new
                                    BasicGeneric <String>(data1);
```

Syntax <*String*> setelah mendeklarasi *BasicGeneric* mengelompokkan bahwa instance dari class ini akan bekerja dengan variabel-variabel bertipe *String*.

Anda juga dapat bekerja denga variabel-variabel bertipe *Integer* atau referensi tipe yang lain. Untuk bekrja dengan *Integer*, potongan kode memiliki pernyataan berikut ini :

```
BasicGeneric <Integer> basicGeneric = new
                                   BasicGeneric <Integer>(data1);
```

Anda mungkin dapat menerjemahkan sisa dari kode dengan pemahaman anda sendiri. Mempertimbangkan deklarasi dari method *getData* .

```
public A getData() {
   return data;
}
```

Method *getData* mengembalikan sebuah nilai dari tipe *A*, Sebuah Tipe type. Ini bukan berarti bahwa method tidak akan memiliki tipe data runtime, atau even pada waktu meng-compile. Setelah Anda mendeklarasikan sebuah object yang bertipe *BasicGeneric*, *A* adalah pengikat ke sebuah tipe data yang spesifik. Instance ini akan berlaku sebagai jika ini dideklarasikan untuk memiliki tipe data spesifik ini dan tipe ini hanya dari bagian sangat awal.

Pada kode yang diberikan, dua instances dari class *BasicGeneric* terbentuk.

```
BasicGeneric <String> basicGeneric = new
                                    BasicGeneric <String>(data1);
String data2 = basicGeneric.getData();

BasicGeneric <Integer> basicGeneric = new
                                   BasicGeneric <Integer>(data1);
Integer data2 = basicGeneric.getData();
```

Perlu diperhatikan perhatian bahwa instantiasi dari class generic adalah hanya sama dengan instantiasi sebuah class normal kecuali bahwa tipe data khusus berada dalam <> menggantikan nama konstruktor. Informasi tambahan ini mengindikasikan tipe dari data anda akan bekerja dengan siapa untuk bagian instance ini dari class *BasicGeneric*.Setelah instantiasi, anda dapat mengakses anggota dari class melalui instance sekarang. Tidak ada yang lebih diperlukan untuk typecast nilai pengembalian dari method *getData* sejak diputuskan bahwa ini akan bekerja dengan sebuah referensi tipe data yang spesifik.

13.3.1 Pembatasan"Primitive"

Sebuah pembatasan type generic dalam Java adalah mereka dibatasi oleh tipe referensi dan tidak akan bekerja dengan tipe data primitive.

Sebagai contoh, pernyataan berikut akan menjadi illegal sejak *int* adalah sebuah tipe data primitive.

```
BasicGeneric <int> basicGeneric = new
                                      BasicGeneric <int>(data1);
```

Petama-tama Anda akan menyelesaikan type primitive sebelum menggunakan mereka sebagai arguments ke sebuah type generic.

13.3.2 Meng-compilasi Generics

Untuk meng-compile source codes Java denagn type generic menggunakan JDK (v. 1.5.0), gunakan syntax berikut ini :

```
javac -version -source "1.5" -sourcepath src -d classes
src/SwapClass.java
```

Dimana *src* mengarah pada lokasi dari source code java sementara *class* mengarah pada lokasi dimana file class akan disimpan.
Beriut ini sebuah contoh :

```
javac -version -source "1.5" -sourcepath c:\temp -d c:\temp
c:/temp/SwapClass.java
```

13.4 Constrained Generics

Dalam contoh yang diberikan terdahulu, type parameter dari class *BasicGeneric* dapat memiliki beberapa referensi tipe data. Ada beberapa kasus, bagaimanapun, dimana anda ingin untuk membatasi tipe instantiasi yang potensial dari class generic. Java juga memungkinkan kita untuk membatasi set argument type yang mungkin untuk subtypes dari sebuah batas type yang diberikan.

Sebagai contoh, kita mungkin ingin untuk mendefinisikan sebuah class generic *ScrollPane yang* merupakan sebuah template untuk sebuah *Container* asli yang telah dilengkapi dengan fungsi scrolling. Tipe runtime dari sebuah instance dari class ini akan sering menjadi sebuah subclass dari *Container*, tapi tipe static atau general adalah *Container* yang lebih sederhana.

Untuk membatasi instantiasi tipe dari sebuah class, kita menggunakan kata kunci *extends* diikuti oleh class yang membatasi tipe generic sebagai bagian dari tipe parameter.

Contoh berikut ini membatasi instantiasi tipe dari class *ScrollPane* ke subtipe dari class *Container*.

```
class ScrollPane <MyPane extends Container> {
   ...
}

class TestScrollPane {
   public static void main(String args[]) {
      ScrollPane <Panel> scrollPane1 = new
                                        ScrollPane <Panel>();
      // pernyataan berikutnya adalah illegal
      ScrollPane <Button> scrollPane2 = new
                                        ScrollPane <Button>();
   }
}
```

Instantiasi dari *scrollPane1* bernilai valid sejak *Panel* menjadi sebuah subclass dari class *Container* sedangkan kreasi dari *scrollPane2* akan menyebabkan munculnya compile time error sejak *Button* bukan merupakan sebuah subclass dari *Container*.

menggunakan generic constrained yang dapat memberikan kita penambahan pengecekan tipe static. Sebagai sebuah hasil, kita akan menjamin bahwa setiap

instantiasi dari tipe generic yang melekat pada batas yang kita miliki.

Sejak kita yakin bahwa setiap tipe instantiasi adalah sebuah subclass dari batas yang dimiliki, kita dapat memanggil beberapa method secara aman yang ditemukan dalam objek static tipe. Jika kita belum menempatkan beberapa batas eksplisit pada parameternya, default batas adalah *Object*. Ini berarti bahwa kita tidak dapat menjalankan method pada sebuah instance dari batas yang tidak ditampilkan dalam class *Object*.

13.5 Mendeklarasikan sebuah Method Generic

Disamping mendeklarasikan sebuah class generic, Java juga memberi kita perlakuan khusus untuk mendeklarasikan sebuah method generic. ini disebut dengan method polymorphic, yang mana didefinisikan menjadi method yang diberi nilai parameter berupa tipe.

method parameterisasi sangat membantu kita kita ingin untuk menampilkan tugas dimana ketergantungan tipe antara argument dan nilai pengembalian aslinya merupakan generic, tapi generic sebenarnya tidak mempercayakan pada beberapa tipe level class informasi dan akan merubah dari method call ke method call.

Sebagai contoh, Andaikata kita menginginkan untuk menambahkan sebuah method *make* untuk sebuah class *ArrayList*. Method *static* ini akan diambil dalam sebuah argument tunggal, yang mana akan menjadi satu-satunya elemen dari object *ArrayList*. Untuk membuat generic *ArrayList* kita, sebagai penampung beberapa tipe elemen, argumen tunggal dalam method *make* harus memiliki sebuah tipe generic sebagai sebuah argumen dan sebagai sebuah tipe nilai kembalian.

Untuk mendeklarasikan tipe generic pada level method, pertimbangkan contoh berikut ini :

```
class Utilities {
   /* T secara implisit extends terhadap Object */
   public static <T> ArrayList<T> make(T first) {
      return new ArrayList<T>(first);
   }
}
```

Java juga menggunakan sebuah mekanisme type-inference untuk secara otomatis menyimpulkan tipe method polymorphic berdasarkan pada tipe-tipe argumennya. Ini mengurangi panjang dan kompleksitas dari sebuah alur untuk menjalankan suatu method.

Untuk membangun sebuah instance baru dari *ArrayList<Integer>*, kita mempunyai cara yang cukup sederhana yaitu pernyataan berikut :

```
Utilities.make(Integer(0));
```

13.6 Latihan

13.6.1 Swapping

Membentuk sebuah class dengan sebuah versi generic dari method *print*S*wapped*. Method ini menukar secara sederhana nilai dari parameternya secara lokal dan mencetak nilainya. Catatan bahwa pencetakan harus dilakukan pada method ini. Mencetak nilai dari argument dalam method lain tidak akan bekerja karena Java melewatkan object ke method melalui suatu nilai. Uji method ini pada object *Integer,* object *String* dan object *ArrayList*.