BAB 2
Exceptions dan Assertions

2.1 Tujuan

Dasar penanganan exception telah dikenalkan pada Anda di pelajaran pemrograman pertama. Bab ini membahas secara lebih dalam mengenai exception dan sedikit menyinggung tentang assertion.

Pada akhir pembahasan, diharapkan pembaca dapat :

1. Menangani exception dengan menggunakan try, catch dan finally
2. Membedakan penggunaan antara throw dengan throws
3. Menggunakan exception class yang berbeda – beda
4. Membedakan antara checked exceptions dan unchecked exceptions
5. Membuat exception class tersendiri
6. Menjelaskan keunggulan penggunaan assertions
7. Menggunakan assertions

2.2 Apa itu Exception?

2.2.1 Pendahuluan

Bugs dan error dalam sebuah program sangat sering muncul meskipun program tersebut dibuat oleh programmer berkemampuan tinggi. Untuk menghindari pemborosan waktu pada proses error-checking, Java menyediakan mekanisme penanganan exception.

Exception adalah singkatan dari Exceptional Events. Kesalahan (errors) yang terjadi saat runtime, menyebabkan gangguan pada alur eksekusi program. Terdapat beberapa tipe error yang dapat muncul. Sebagai contoh adalah error pembagian 0, mengakses elemen di luar jangkauan sebuah array, input yang tidak benar dan membuka file yang tidak ada.

2.2.2 Error dan Exception Classes

Seluruh exceptions adalah subclasses, baik secara langsung maupun tidak langsung, dari sebuah root class *Throwable*. Kemudian, dalam class ini terdapat dua kategori umum : Error class dan Exception class.

Exception class menunjukkan kondisi yang dapat diterima oleh user program. Umumnya hal tersebut disebabkan oleh beberapa kesalahan pada kode program. Contoh dari exceptions adalah pembagian oleh 0 dan error di luar jangkauan array.

Error class digunakan oleh Java run-time untuk menangani error yang muncul pada saat dijalankan. Secara umum hal ini di luar control user karena kemunculannya disebabkan oleh run-time environment. Sebagai contoh adalah *out of memory* dan *harddisk crash.*

2.2.3 Sebuah Contoh

Perhatikan contoh program berikut :

```
class DivByZero {
       public static void main(String args[]) {
              System.out.println(3/0);
              System.out.println(“Cetak.”);
       }
}
```

Jika kode tersebut dijalankan, akan didapatkan pesan kesalahan sebagai berikut :

```
Exception in thread "main" java.lang.ArithmeticException: / by
zero at DivByZero.main(DivByZero.java:3)
```

Pesan tersebut menginformasikan tipe exception yang terjadi pada baris dimana exception itu berasal. Inilah aksi default yang terjadi bila terjadi exception yang tidak tertangani. Jika tidak terdapat kode yang menangani exception yang terjadi, aksi default akan bekerja otomatis. Aksi tersebut pertama-tama akan menampilkan deskripsi exception yang terjadi. Kemudian akan ditampilkan stack trace yang mengindentifikasi method dimana exception terjadi. Pada bagian akhir, aksi default tersebut akan menghentikan program secara paksa.

Bagaimana jika Anda ingin melakukan penanganan atas exception dengan cara yang berbeda? Untungnya, bahasa pemrograman Java memiliki 3 keywords penting dalam penanganan exception, yaitu *try, catch* dan *finally*.

2.3 Menangkap Exception

2.3.1 Try - Catch

Seperti yang telah dijelaskan sebelumnya, keyword *try, catch* dan *finally* digunakan dalam menangani bermacam tipe exception. 3 Keyword tersebut digunakan bersama, namun *finally* bersifat opsional. Akan lebih baik jika memfokuskan pada dua keyword pertama, kemudian membahas *finally* pada bagian akhir.

Berikut ini adalah penulisan *try-catch* secara umum :

```
try {
       <code to be monitored for exceptions>
} catch (<ExceptionType1> <ObjName>) {
       <handler if ExceptionType1 occurs>
}
...
} catch (<ExceptionTypeN> <ObjName>) {
       <handler if ExceptionTypeN occurs>
}
```

Petunjuk Penulisan Program :

Blok catch dimulai setelah kurung kurawal dari kode try atau catch terkait. Penulisan kode dalam blok yang dimasukkan

Gunakan contoh kode tersebut pada program DivByZero yang telah dibuat sebelumnya :

```
class DivByZero {
       public static void main(String args[]) {
              try {
                     System.out.println(3/0);
                     System.out.println("Cetak.");
              } catch (ArithmeticException exc) {
                     //Reaksi atas kejadian
                     System.out.println(exc);
              }
              System.out.println("Setelah Exception.");
       }
}
```

Kesalahan pembagian dengan bilangan 0 adalah salah satu contoh dari *ArithmethicException*. Tipe exception kemudian mengindikasikan klausa *catch* pada class ini. Program tersebut menangani kesalahan yang terjadi dengan menampilkan deskripsi dari permasalahan.

Output program saat eksekusi akan terlihat sebagai berikut :

```
java.lang.ArithmeticException: / by zero
After exception.
```

Bagian kode yang terdapat pada blok *try* dapat menyebabkan lebih dari satu tipe exception. Dalam hal ini, terjadinya bermacam tipe kesalahan dapat ditangani menggunakan beberapa blok *catch.* Perlu dicatat bahwa blok *try* dapat hanya menyebabkan sebuah exception pada satu waktu, namun dapat pula menampikan tipe exception yang berbeda di lain waktu.

Berikut adalah contoh kode yang menangani lebih dari satu exception :

```
class MultipleCatch {
       public static void main(String args[]) {
              try {
                     int den = Integer.parseInt(args[0]); //baris 4
                     System.out.println(3/den); //baris 5
              } catch (ArithmeticException exc) {
                     System.out.println(“Nilai Pembagi 0.”);
              } catch (ArrayIndexOutOfBoundsException exc2) {
                     System.out.println(“Missing argument.”);
              }
              System.out.println(“After exception.”);
       }
}
```

Pada contoh ini, baris ke-4 akan menghasilkan kesalahan berupa *ArrayIndexOutOfBoundsException* bilamana seorang user alpa dalam memasukkan argument, sedang baris ke-5 akan menghasilkan kesalahan *ArithmethicException* jika pengguna memasukkan nilai 0 sebagai sebuah argument.

Pelajari apakah yang akan terjadi terhadap program bila argumen – argumen berikut dimasukkan oleh user :

a) Tidak ada argument
b) 1
c) 0

Penggunaan *try* bersarang diperbolehkan dalam pemrograman Java.

```
class NestedTryDemo {
       public static void main(String args[]){
         try {
           int a = Integer.parseInt(args[0]);
           try {
             int b = Integer.parseInt(args[1]);
             System.out.println(a/b);
           } catch (ArithmeticException e) {
             System.out.println(“Divide by zero error!");
           }
         } catch (ArrayIndexOutOfBoundsException) {
           System.out.println(“2 parameters are required!");
         }
       }
```

```
    }
```

Pelajari apa yang akan terjadi pada program jika argument – argument berikut dimasukkan :

a) Tidak ada argumen
b) 15
c) 15 3
d) 15 0

Kode berikut menggunakan *try* bersarang tergabung dengan penggunaan method.

```
class NestedTryDemo2 {
      static void nestedTry(String args[]) {
           try {
                int a = Integer.parseInt(args[0]);
                int b = Integer.parseInt(args[1]);
                System.out.println(a/b);
           } catch (ArithmeticException e) {
                System.out.println("Divide by zero error!");
           }
      }

      public static void main(String args[]){
           try {
                 nestedTry(args);
           } catch (ArrayIndexOutOfBoundsException e) {
                 System.out.println("2 parameters are required!");
           }
      }
}
```

Bagaimana output program tersebut jika diimplementasikan terhadap argument – argument berikut :

a) Tidak ada argumen
b) 15
c) 15 3
d) 15 0

2.3.2 Keyword Finally

Saatnya Anda mengimplementasikan *finally* dalam blok *try-cacth*. Berikut ini cara penggunaan keyword tersebut :

```
try {
      <kode monitor exception>
} catch (<ExceptionType1> <ObjName>) {
      <penanganan jika ExceptionType1 terjadi>
} ...
} finally {
      <kode yang akan dieksekusi saat blok try berakhir>
}
```

Petunjuk Penulisan Program :

Sekali lagi, ketentuan penulisan program juga mengatur penggunaan finally seperti halnya pada blok catch. Penggunaan finally dimulai setelah kurung kurawal penutup blok catch terkait. Penulisan dalam blok tersebut juga mengalami indentasi.

Blok *finally* mengandung kode penanganan setelah penggunaan *try* dan *catch.* Blok kode ini selalu tereksekusi walaupun sebuah exception terjadi atau tidak pada blok *try.* Blok kode tersebut juga akan menghasilkan nilai *true* meskipun *return, continue* ataupun *break* tereksekusi. Terdapat 4 kemungkinan skenario yang berbeda dalam blok *try-catch-finally*. Pertama, pemaksaan keluar program terjadi bila control program dipaksa untuk melewati blok *try* menggunakan *return*, *continue* ataupun *break*. Kedua, sebuah penyelesaian normal terjadi jika *try-catch-finally* tereksekusi secara normal tanpa terjadi error apapun. Ketiga, kode program memiliki spesifikasi tersendiri dalam blok *catch* terhadap exception yang terjadi. Yang terakhir, kebalikan skenario ketiga. Dalam hal ini, exception yang terjadi tidak terdefinisikan pada blok *catch* manapun. Contoh dari skenario – skenario tersebut terlihat pada kode berikut ini :

```
class FinallyDemo {
     static void myMethod(int n) throws Exception{
            try {
                  switch(n) {
                        case 1: System.out.println("case pertama");
                                return;
                        case 3: System.out.println("case ketiga");
                                throw new RuntimeException("demo case
                                ketiga");
                        case 4: System.out.println("case keempat");
                                throw new Exception("demo case
                                keempat");
                        case 2: System.out.println("case Kedua");
                  }
            } catch (RuntimeException e) {
                  System.out.print("RuntimeException terjadi: ");
                  System.out.println(e.getMessage());
            } finally {
                  System.out.println("try-block entered.");
            }
     }
     public static void main(String args[]){
            for (int i=1; i<=4; i++) {
                  try {
                        FinallyDemo.myMethod(i);
                  } catch (Exception e){
                        System.out.print("Exception terjadi: ");
                        System.out.println(e.getMessage());
                  }
                  System.out.println();
            }
     }
}
```

2.4 Melempar Exception

2.4.1 Keyword Throw

Disamping menangkap exception, Java juga mengijinkan seorang user untuk melempar sebuah exception. Sintaks pelemparan exception cukup sederhana.

```
throw <exception object>;
```

Perhatikan contoh berikut ini.

```
/* Melempar exception jika terjadi kesalahan input */
class ThrowDemo {
   public static void main(String args[]){
          String input = “invalid input”;
          try {
                if (input.equals(“invalid input”)) {
                      throw new RuntimeException("throw demo");
                } else {
                      System.out.println(input);
                }
                System.out.println("After throwing");
                } catch (RuntimeException e) {
                      System.out.println("Exception caught here.");
                      System.out.println(e);
          }
   }
}
```

2.4.2 Keyword Throws

Jika sebuah method dapat menyebabkan sebuah exception namun tidak menangkapnya, maka digunakan keyword *throws*. Aturan ini hanya berlaku pada checked exception. Anda akan mempelajari lebih lanjut tentang checked exception dan unchecked exception pada bagian selanjutnya, “Kategori Exception”.

Berikut ini penulisan syntax menggunakan keyword *throws* :

```
<type> <methodName> (<parameterList>) throws <exceptionList> {
      <methodBody>
}
```

Sebuah method perlu untuk menangkap ataupun mendaftar seluruh exceptions yang mungkin terjadi, namun hal itu dapat menghilangkan tipe Error, RuntimeException, ataupun subclass-nya.

Contoh berikut ini menunjukkan bahwa method *myMethod* tidak menangani *ClassNotFoundException*.

```
class ThrowingClass {
	static void myMethod() throws ClassNotFoundException {
		throw new ClassNotFoundException ("just a demo");
	}
}

class ThrowsDemo {
	public static void main(String args[]) {
		try {
			ThrowingClass.myMethod();
		} catch (ClassNotFoundException e) {
			System.out.println(e);
		}
	}
}
```

2.5 Kategori Exception

2.5.1 Exception Classes dan Hierarki

Seperti yang disebutkan sebelumnya, root class dari seluruh exception classes adalah Throwable class. Yang disebutkan dibawah ini adalah exception class hierarki. Seluruh exceptions ini terdefinisi pada package *java.lang.*

Exception Class Hierarchy			
Throwable	Error	LinkageError, ...	
		VirtualMachineError, ...	
	Exception	ClassNotFoundException,	
		CloneNotSupportedException,	
		IllegalAccessException,	
		InstantiationException,	
		InterruptedException,	
		IOException,	EOFException,
			FileNotFoundException,
			...
		RuntimeException,	ArithmeticException,
			ArrayStoreException,
			ClassCastException,
			IllegalArgumentException, (IllegalThreadStateException and NumberFormatException as subclasses)
			IllegalMonitorStateException,
			IndexOutOfBoundsException,
			NegativeArraySizeException,
			NullPointerException,
			SecurityException
		...	

Tabel 1.4. Hirarki Exception Class

Sekarang Anda sudah cukup familiar dengan beberapa exception classes, saatnya untuk mengenalkan aturan : catch lebih dari satu harus berurutan dari subclass ke superclass.

```
class MultipleCatchError {
	public static void main(String args[]){
		try {
			int a = Integer.parseInt(args [0]);
			int b = Integer.parseInt(args [1]);
			System.out.println(a/b);
		} catch (Exception e) {
			System.out.println(e);
		} catch (ArrayIndexOutOfBoundsException e2) {
			System.out.println(e2);
		}
		System.out.println("After try-catch-catch.");
	}
}
```

Setelah mengkompilasi kode tersebut akan menghasilkan pesan error jika Exception class adalah superclass dari *ArrayIndexOutOfBoundsException* class.

```
MultipleCatchError.java:9: exception
java.lang.ArrayIndexOutOfBoundsException has already been caught
} catch (ArrayIndexOutOfBoundsException e2) {
```

2.5.2 Checked dan Unchecked Exceptions

Exception terdiri atas checked dan unchecked exceptions.

Checked exceptions adalah exception yang diperiksa oleh Java compiler. Compiler memeriksa keseluruhan program apakah menangkap atau mendaftar exception yang terjadi dalam sintax *throws*. Apabila checked exception tidak didaftar ataupun ditangkap, maka compiler error akan ditampilkan.

Tidak seperti checked exceptions, unchecked exceptions tidak berupa compile-time checking dalam penanganan exceptions. Pondasi dasar dari unchecked exception classes adalah Error, RuntimeException dan subclass-nya.

2.5.3 User Defined Exceptions

Meskipun beberapa exception classes terdapat pada package *java.lang* namun tidak mencukupi untuk menampung seluruh kemungkinan tipe exception yang mungkin terjadi. Sehingga sangat mungkin bahwa Anda perlu untuk membuat tipe exception tersendiri.
Dalam pembuatan tipe exception Anda sendiri, Anda hanya perlu untuk membuat sebuah extended class terhadap RuntimeException class, maupun Exception class lain. Selanjutnya tergantung pada Anda dalam memodifikasi class sesuai permasalahan yang akan diselesaikan. Members dan constructors dapat dimasukkan pada exception class milik Anda.

Berikut ini contohnya :

```
class HateStringException extends RuntimeException{
	/* Tidak perlu memasukkan member ataupun construktor */
}

class TestHateString {
	public static void main(String args[]) {
		String input = "invalid input";
		try {
			if (input.equals("invalid input")) {
			throw new HateStringException();
		}
			System.out.println("String accepted.");
		} catch (HateStringException e) {
			System.out.println("I hate this string: " + input +
			".");
		}
	}
}
```

2.6 Assertions

2.6.1 User Defined Exceptions

Assertions mengijinkan programmer untuk menentukan asumsi yang dihadapi. Sebagai contoh, sebuah tanggal dengan area bulan tidak berada antara 1 hingga 12 dapat diputuskan bahwa data tersebut tidak valid. Programmer dapat menentukan bulan harus berada diantara area tersebut. Meskipun hal itu dimungkinkan untuk menggunakan contructor lain untuk mensimulasikan fungsi dari assertions, namun sulit untuk dilakukan karena fitur assertion dapat tidak digunakan. Hal yang menarik dari assertions adalah seorang user memiliki pilihan untuk digunakan atau tidak pada saat runtime.

Assertion dapat diartikan sebagai extensi atas komentar yang menginformasikan pembaca kode bahwa sebagian kondisi harus terpenuhi. Dengan menggunakan assertions, maka tidak perlu untuk membaca keseluruhan kode melalui setiap komentar untuk mencari asumsi yang dibuat dalam kode. Namun, menjalankan program tersebut akan memberitahu Anda tentang assertion yang dibuat benar atau salah. Jika assertion tersebut salah, maka *AssertionError* akan terjadi.

2.6.2 Mengaktifkan dan Menonaktifkan Exceptions

Penggunaan assertions tidak perlu melakukan import package *java.util.assert*. Menggunakan assertions lebih tepat ditujukan untuk memeriksa parameter dari non-public methods jika public methods dapat diakses oleh class lain. Hal itu mungkin terjadi bila penulis dari class lain tidak menyadari bahwa mereka dapat menonaktifkan assertions. Dalam hal ini program tidak dapat bekerja dengan baik. Pada non-public methods, hal tersebut tergunakan secara langsung oleh kode yang ditulis oleh programmer yang memiliki akses terhadap methods tersebut. Sehingga

mereka menyadari bahwa saat menjalankannya, assertion harus dalam keadaan aktif.

Untuk mengkompilasi file yang menggunakan assertions, sebuah tambahan parameter perintah diperlukan seperti yang terlihat dibawah ini :

```
javac –source 1.4 MyProgram.java
```

Jika Anda ingin untuk menjalankan program tanpa menggunakan fitur assertions, cukup jalankan program secara normal.

```
java MyProgram
```

Namun, jika Anda ingin mengaktifkan assertions, Anda perlu menggunakan parameter *–enableassertions* atau *–ea*.

```
java –enableassertions MyProgram
```

2.6.3 Sintaks Assertions

Penulisan assertions memiliki dua bentuk.

Bentuk yang paling sederhana terlihat sebagai berikut :

```
assert <expression1>;
```

dimana `<expression1>` adalah kondisi dimana assertion bernilai true.

Bentuk yang lain menggunakan dua ekspresi, berikut ini cara penulisannya :

```
assert <expression1> : <expression2>;
```

dimana `<expression1>` adalah kondisi assertion bernilai true dan `<expression2>` adalah informasi yang membantu pemeriksaan mengapa program mengalami kesalahan.

```
class AgeAssert {
    public static void main(String args[]) {
          int age = Integer.parseInt(args[0]);
          assert(age>0);
          /* jika masukan umur benar (misal, age>0) */
          if (age >= 18) {
                System.out.println(“Congrats! You're an adult!
          =)”);
          }
    }
}
```

2.7 Latihan

2.7.1 Heksadesimal ke Desimal

Tentukan sebuah angka heksadesimal sebagai input. Konversi angka tersebut menjadi bilangan desimal. Tentukan exception class Anda sendiri dan lakukan penanganan jika input dari user bukan berupa bilangan heksadesimal.

2.7.2 Menampilkan Sebuah Berlian

Tentukan nilai integer positif sebagai input. Tampilkan sebuah berlian menggunaakan karakter asterisk (*) sesuai angka yang diinput oleh user. Jika user memasukkan bilangan integer negatif, gunakan assertions untuk menanganinya.
Sebagai contoh, jika user memasukkan integer bernilai 3, program Anda harus menampilkan sebuah berlian sesuai bentuk berikut :

```
  *
 ***
*****
 ***
  *
```