Exceptions dan Assertions

Objektif

- Menangani exception dengan menggunakan try, catch dan finally
- Membedakan penggunaan antara throw dengan throws
- Menggunakan exception class yang berbeda – beda
- Membedakan antara checked exceptions dan unchecked exceptions
- Membuat exception class tersendiri
- Menggunakan assertion

Exception

- Singkatan dari Exceptional Events.
- Kesalahan (errors) yang terjadi saat runtime, menyebabkan gangguan pada alur eksekusi program.
- Contoh
  - Error pembagian 0
  - Mengakses elemen di luar jangkauan sebuah array, input yang tidak benar
  - Membuka file yang tidak ada.

Exception

- Subclasses, baik secara langsung maupun tidak langsung,dari sebuah root class *Throwable*
- Dua kategori umum
  - Error class
  - Exception class

Exception class

- Menunjukkan kondisi yang dapat diterima oleh user program.
- Umumnya disebabkan oleh beberapa kesalahan pada kode program.
- Contoh
  - Pembagian oleh 0
  - Error di luar jangkauan array

Error class

- Untuk menangani error yang muncul pada saat dijalankan (run-time error)
- Kemunculannya di luar control user disebabkan oleh run-time environment.
- Contoh
  - *out of memory*
  - *harddisk crash*

Contoh

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

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

Menangkap Exception →*Try - Catch*

- 3 Keyword : *try, catch dan finally*.
- *Bentuk umum :*

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

```
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.");
        }
}
```

```
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.");
	}
}
```

```
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!");
        }
      }
```

```
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!");
            }
      }
}
```

Keyword Finally

- 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*

Keyword Finally

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

```
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();
        }
    }
}
```

Keyword Throw

- Disamping menangkap exception
- Java juga dapat melempar sebuah exception.
- Sintaks pelemparan exception cukup sederhana.
  - throw <exception object>;

```
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);
            }
    }
}
```

Keyword Throws

- Sebuah method dapat menyebabkan sebuah exception namun tidak menangkapnya, maka digunakan keyword *throws.*
- *Aturan ini hanya berlaku pada* checked exception.
- Berikut ini penulisan syntax menggunakan keyword *throws :*

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

```
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);
		}
	}
}
```

Exception Classes dan Hierarki

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
		...	

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.

Checked dan Unchecked Exceptions

- 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.

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.

User Defined Exceptions

- 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 dimasukkanpada exception class milik Anda.

```
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 +
                  ".");
            }
      }
}
```

Assertions

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.

Assertions

Defined Exceptions

- 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.

Assertions

Defined Exceptions

- 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 setiapkomentar 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 terjad*

Assertions

Mengaktifkan dan Menonaktifkan Exceptions

- Penggunaan assertions tidak perlu melakukan import package *java.util.assert.*
- Menggunakan assertions lebih tepat ditujukan untuk memeriksa parameter dari nonpublic 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.

Assertions

Mengaktifkan dan Menonaktifkan Exceptions

- 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.

Assertions

- 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

Sintaks Assertions

- Penulisan assertions memiliki dua bentuk. Bentuk yang paling sederhana terlihat sebagai berikut :
- assert <expression1>;
- dimana <expression1> adalah kondisi dimana assertion bernilai true.

Sintaks Assertions

- 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!
        =)");
        }
    }
}
```

Latihan

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.

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 :

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