Catatan:

Sebagaian dari source code dan materi disini diambil dari tutorial Java di website Sun Microsystem, kecuali disebutkan lain.

Pengertian Exception

- Kependekan dari "Exception Event"
- Merupakan suatu event yang muncul pada saat eksekusi dari program, mengacaukan aliran normal dari program
- "Throwing an exception" adalah istilah untuk menunjukkan penciptaan suatu obyek pada saat terjadi kesalahan serta memberikannya ke sistem yang sedang berjalan
- "Exception handler" merupakan blok kode yang digunakan untuk menangani exception event.
- Jika terjadi exception dan tidak ada exception handler maka program berhenti.

Mengapa Perlu Exceptions Handling?

1. Memisahkan kode untuk penanganan kesalahan dengan kode "reguler".

Misal pseudo code untuk membaca file berikut ini:

```
readFile {
  open the file;
  determine its size;
  allocate that much memory;
  read the file into memory;
  close the file;
}
```

Apa yang terjadi jika:
- file tidak bisa dibuka?
- panjang file tidak bisa diketahui?
- tidak bisa mengalokasikan memory secukupnya?
- pembacaan gagal?
- file tidak bisa ditutup?

Penanganan kesalahan jika menggunakan cara tradisional, tanpa exception handling:

```
errorCodeType readFile {
  initialize errorCode = 0;

  open the file;
  if (theFileIsOpen) {
    determine the length of the file;
    if (gotTheFileLength) {
      allocate that much memory;
      if (gotEnoughMemory) {
        read the file into memory;
        if (readFailed) {
          errorCode = -1;
        }
      } else {
        errorCode = -2;
      }
    } else {
```

```
      errorCode = -3;
    }
    close the file;
    if (theFileDidntClose && errorCode == 0) {
      errorCode = -4;
    } else {
      errorCode = errorCode and -4;
    }
  } else {
    errorCode = -5;
  }
  return errorCode;
}
```

Menggunakan exceptions handling:

```
readFile {
  try {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
  } catch (fileOpenFailed) {
    doSomething;
  } catch (sizeDeterminationFailed) {
    doSomething;
  } catch (memoryAllocationFailed) {
    doSomething;
  } catch (readFailed) {
    doSomething;
  } catch (fileCloseFailed) {
    doSomething;
  }
}
```

2. Memungkinkan penanganan error melewati stack pemanggilan

Misal terdapat urutan method: method1 -> method2 -> method3 -> readFile

```
method1 {
  call method2;
}

method2 {
  call method3;
}

method3 {
  call readFile;
}
```

Penanganan Error:

```
method1 {
  errorCodeType error;
  error = call method2;
```

```
  if (error)
    doErrorProcessing;
  else
    proceed;
}

errorCodeType method2 {
  errorCodeType error;
  error = call method3;
  if (error)
    return error;
  else
    proceed;
}

errorCodeType method3 {
  errorCodeType error;
  error = call readFile;
  if (error)
    return error;
  else
    proceed;
}
```

Menggunakan exception handling:

```
method1 {
  try {
    call method2;
  } catch (exception e) {
    doErrorProcessing;
  }
}

method2 throws exception {
  call method3;
}

method3 throws exception {
  call readFile;
}
```

3. Pengelompokan dan pembedaan tipe error

Semua exception merupakan obyek yang diberikan ke runtime system saat program berjalan. Dengan demikian terdapat kelas yang menangani error ini. Exception dikelompokkan, misalnya java.io -- IOException -- FileNotFoundException.

Jenis-jenis Exception

1. Checked Exception: Exception diantisipasi pada saat kompilasi. Harus diletakkan pada blok try ... catch.
2. Unchecked Exception: Error (misalnya hardware error) dan Runtime exception (misalnya kesalahan penggunaan API, kesalahan logika, misal kesalahan yang menyebabkan nama file menjadi null sehingga terjadi kesalahan pembacaan file. Ini disebabkan karena bug di dalam aplikasi yang dibuat / bug dari sisi logik).

Menangani Exception dengan try-catch-finally

```
  try {
    int c = System.in.read();
  }
  catch (IOException e) {
    System.out.println(e.getMessage());
  }


  try {
    int c = System.in.read();
  }
  catch (IOException e) {
    System.out.println(e.getMessage());
  }
  finally {
    System.out.println("Will always execute this statement");
  }
```

Contoh tanpa Exception Handling:

```
/*
 * taken from: http://www.javabeginner.com/learn-java/understanding-java-exceptions
 *
 */

public class DividedExceptionUnchecked {

  public static void main(String[] args) {
    division(100,4);    // Line 1
    division(100,0);        // Line 2
    System.out.println("Exit main().");
  }

  public static void division(int totalSum, int totalNumber) {

    System.out.println("Computing Division.");
    int average  = totalSum/totalNumber;
    System.out.println("Average : "+ average);
  }
}

[bpdp@bpdp-arch src]$ javac DividedExceptionUnchecked.java
[bpdp@bpdp-arch src]$ java DividedExceptionUnchecked
Computing Division.
Average : 25
Computing Division.
Exception in thread "main" java.lang.ArithmeticException: / by zero
  at DividedExceptionUnchecked.division(DividedExceptionUnchecked.java:17)
  at DividedExceptionUnchecked.main(DividedExceptionUnchecked.java:10)
[bpdp@bpdp-arch src]$
```

Terdapat Exception Handling

```
public class DividedExceptionHandled {

  public static void main(String[] args) {
    int result  = division(100,0);        // Line 2
    System.out.println("result : "+result);
  }

  public static int division(int totalSum, int totalNumber) {
    int quotient = -1;
    System.out.println("Computing Division.");
    try {
      quotient  = totalSum/totalNumber;
    }
    catch (Exception e){
      System.out.println("Exception : "+ e.getMessage());
    }
    finally {
      if (quotient != -1) {
        System.out.println("Finally Block Executes");
        System.out.println("Result : "+ quotient);
      } else {
        System.out.println("Finally Block Executes. Exception Occurred");
        return quotient;
      }
    }
    return quotient;
  }
}

[bpdp@bpdp-arch src]$ javac DividedExceptionHandled.java
[bpdp@bpdp-arch src]$ java DividedExceptionHandled
Computing Division.
Exception : / by zero
Finally Block Executes. Exception Occurred
result : -1
[bpdp@bpdp-arch src]$
```

Membuat sendiri Exception

```
  public class NumberRangeException extends Exception {
    public NumberRangeException( String msg ) {
      super(msg);
    }
  }
```

Penggunaan:

```
  throw new NumberRangeException("An out of range value was specified");
```

Contoh:

```
/*     taken     from:     http://www.java-tips.org/java-se-tips/java.lang/creating-
application-specific-exceptions.html
 *
```

```
 */

class ApplicationException extends Exception {

  private int intError;

  ApplicationException(int intErrNo){
    intError = intErrNo;
  }

  ApplicationException(String strMessage){
    super(strMessage);
  }

  public String toString(){
    return "ApplicationException["+intError+"]";
  }
}

public class UserDefinedExceptionDemo {

  static void compute(int a) throws ApplicationException {

    System.out.println("called compute(" +a +" )");

    if (a>10) {
      throw new ApplicationException(a);
    }

    System.out.println("NORMAL EXIT");
  }

  public static void main(String[] args) {

    try {
      compute(1);
      compute(20);

    } catch(ApplicationException e){
      System.out.println("caught " + e);
    }
  }
}

[bpdp@bpdp-arch src]$ javac UserDefinedExceptionDemo.java
[bpdp@bpdp-arch src]$ java UserDefinedExceptionDemo
called compute(1 )
NORMAL EXIT
called compute(20 )
caught ApplicationException[20]
[bpdp@bpdp-arch src]$
```