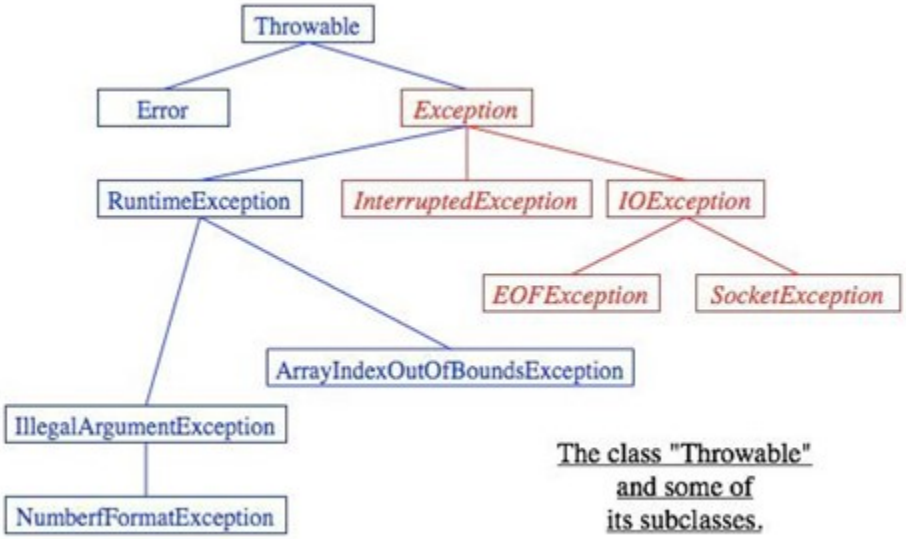
| | | |
|---|---|---|
| *Session date* | : | 23 October 2019 |
| *Semester* | : | Gasal |
| *Subject* | : | 1132102 – Pemrograman Berorientasi Objek |
| *Week/Session* | : | 7/2 |
| *Key topics* | : | Exception Handling |
| *Objectives* | : | Students understand exception handling concept in Java |
| *Duration* | : | 100 *minutes* |
| *Delivery* | : | |
| *Deadline of delivery* | : | |
| *Place of delivery* | : | https://ecourse.del.ac.id |
| *Lecturer* | : | TMP |
| *Instructor* | : | SEP |

## 1. Hierarki Kelas Throwable

Kelas *Throwable* adalah *superclass* dari *error* dan *exception* yang terdapat pada JAVA. Kelas *error* adalah kelas yang sebaiknya tidak ditangkap. Baris kode yang melempar objek *error* sebaiknya diperbaiki, tidak ditangkap ketika melempar objek *error*. Berbeda dengan *Exception*, baris kode yang akan melempar objek *exception* harus ditangkap dan ditangani, kecuali objek yang merupakan keturunan *RuntimeException*. Dapat dilihat pada Gambar 1. merupakan hierarki kelas *Throwable*.




**Gambar 1. Hierarki Kelas Throwable**

## 2. Exception Handling

*Exception* mengindikasikan terjadinya sebuah masalah pada saat eksekusi program. *Exception* ini dapat ditangkap agar eksekusi dari aplikasi tersebut dapat berjalan.

Perhatikan *snippet* dibawah ini:

```
public class Driver {
    public static void main(String[] args) {
        int a = 6;
        int b = 3;
        double c = bagi(a,b);
        System.out.println("Hasil "+a+" : "+b+" = "+c);
        System.out.println("Aplikasi selesai");
    }
    public static double bagi(int a, int b){
        System.out.println("Method dipanggil dengan a = "+a+" b = "+b);
        double hasil = a/b;
        System.out.println("Method bagi selesai dan akan return hasil");
        return hasil;
    }
}
```

Pada *snippet* tersebut terdapat *static method* bagi yang memiliki dua argumen, yaitu a dan b dengan tipe data *int*. *Return* dari *method* tersebut adalah pembagian dari a dan b. Hasil eksekusi dari *snippet* tersebut adalah:

```
Method dipanggil dengan a = 6 b = 3
Method bagi selesai dan akan return hasil
Hasil 6 : 3 = 2.0
Aplikasi selesai
BUILD SUCCESSFUL (total time: 3 seconds)
```

Eksekusi tersebut tidak akan menemui masalah. Tetapi bagaimana jika nilai dari variable b diubah menjadi angka 0? Apa yang akan terjadi? Selidiki program hasil modifikasi Anda apakah mengalami *exception*? Sertakan jawaban Anda dalam laporan!

Selanjutnya perhatikan *snippet* berikut:

```
public class Test3 {
    public static void main(String[] args) {
        String[] bilangans = new String[4];
        bilangans[0] = "1";
        System.out.println("Nilai indeks 1 dari Array: "+bilangans[0]);
        bilangans[1] = "2";
        System.out.println("Nilai indeks 2 dari Array: "+bilangans[1]);

    }
}
```

Hasil eksekusi *snippet* tersebut adalah:

```
Nilai indeks 1 dari Array: 1
Nilai indeks 2 dari Array: 2
BUILD SUCCESSFUL (total time: 4 seconds)
```

Eksekusi tersebut tidak akan menemui masalah. Tetapi bagaimana jika Anda mencoba untuk mengakses array indeks ke 4? Apa yang akan terjadi? Selidiki hasil modifikasi Anda apakah mengalami *exception*? Sertakan jawaban Anda dalam laporan!

### 3. Try-Catch

Untuk menangani *exception*, digunakan blok *try-catch*. Struktur dari *try-catch* adalah sebagai berikut:

```
try
{
// blok kode yang akan dimonitor
}
catch (ExceptionType1 exOb) {
//handler exception untuk Exception1
}
catch (ExceptionType2 exOb) {
// handler untuk Exception2
}
// ...
finally {
//blok yang dieksekusi setalah try dieksekusi atay terjadinya catch eksepsi
//baris ini selalu dieksekusi
}
```

Pada contoh pertama, untuk menangani metode pembagian dapat dilakukan seperti *snippet* berikut:

```
public class Test4 {
    public static void main(String[] args) {
        int a = 6;
        int b = 0;
        double c = bagi(a, b);
        System.out.println("Hasil "+a+" : "+b+" = "+c);
        System.out.println("Aplikasi selesai");
    }
    public static double bagi (int a, int b){
        double hasil = 0;
        System.out.println("Method bagi dipanggil a = "+a+" b = "+b);
        try
        {
            hasil = a/b;
        }
        catch (ArithmeticException ae)
        {
            System.out.println("PESAN_ERROR: "+ae.getMessage());
        }
        System.out.println("Method bagi selesai dan akan return hasil");
        return hasil;
    }
}
```

Hasil eksekusi dari baris tersebut sebagai berikut:

```
Method bagi dipanggil a = 6 b = 0
PESAN_ERROR: / by zero
Method bagi selesai dan akan return hasil
Hasil 6 : 0 = 0.0
Aplikasi selesai
BUILD SUCCESSFUL (total time: 2 seconds)
```

Setelah menangkap *exception* pada method bagi, aplikasi tersebut tidak lagi mengalami masalah menyelesaikan semua baris perintah yang ada. Tetapi terjadi PESAN_ERROR : / by zero, kenapa masalah tersebut muncul ? Jadi bagaimana penyelesaiannya ? Method bagi daripada me-return angka 0 ke pemanggilannya, sebaiknya melempar langsung objek exception tersebut. Perhatikan baris berikut:

```
public class Test5 {
    public static void main(String[] args) {
        int a = 6;
        int b = 0;
        try{
            double c = bagi(a, b);
            System.out.println("Hasil "+a+" : "+b+" = "+c);
        }
        catch (ArithmeticException ie)
        {
            System.out.println("PESAN EKSEPSI: "+ie.getMessage());
        }
        System.out.println("Aplikasi selesai");
    }
    public static double bagi (int a, int b){
        double hasil = 0;
        System.out.println("Method bagi dipanggil a = "+a+" b = "+b);
        //melempar langsung objek eksepsi
        if (b==0) throw new ArithmeticException("Pembagian 0");
        {
            hasil = a/b;
        }

        System.out.println("Method bagi selesai dan akan return hasil");
        return hasil;
    }
}
```

Pada *source code* di atas, aka dicek pada *method* bagi, apakah b=0, dan langsung melempar objek *ArithmeticException* ke pemanggilannya di *method main*. Sehingga menghindari method mengembalikan nilai hasil pembagian yang tidak benar. Pada *method main*, ketika *method* bagi

dipanggil, maka akan dilakukan penangkapan menggunakan basis *try catch*. Hasil eksekusi dari baris tersebut sebagai berikut:

```
Method bagi dipanggil a = 6 b = 0
PESAN EKSEPSI: Pembagian 0
Aplikasi selesai
BUILD SUCCESSFUL (total time: 2 seconds)
```

## 4. Checked exception

Kelas *exception* dapat dibagi dua yaitu: *uncheck exception* dan *checked exceptions*. Kelas turunan dari *RuntimeException* bertipe *uncheck exception*. Dalam artian, tidak diperlukan penanganan pada *exception* tersebut. *Source code* akan lolos tahap kompilasi, tetapi pada saat di run, objek *exception* mungkin saja terjadi. *Exception* jenis ini sebaiknya diperbaiki, tidak ditangani menggunakan *try catch*. Sedangkan *checked exception* adalah *exception* yang harus ditangani karena tahap kompilasi tidak akan dilewati tanpa menangani *exception* ini. Contoh *checked exeption*:

```
public class Test6 {
    public static void main(String[] args) {
        try{
            ServerSocket myServer = new ServerSocket();
        }

        catch(IOException ex)
        {
            ex.printStackTrace();
        }
    }
}
```

*Snippet* di atas adalah untuk menciptakan sebuah objek *ServerSocket*. Penciptaan objek *ServerSocket* tersebut harus menangani kemungkinan terjadinya pelemparan sebuah objek dari kelas IOException. Jika tidak ditangani, maka kompilasi akan gagal.

## 5. Multiple catch

Perhatikan *source code* berikut:

```
public class Test7 {
    public static void main(String[] args){
        try
        {
            int a = 0;
            int b = 10;
            int cs[] = { 1, 2, 3 };
            double d = b / a;
            System.out.println(d);
            System.out.println(cs[5]);
        }
        catch(Exception e)
        {
            System.out.println("Error terjadi");
            System.out.println(e);
        }
    }

}
```

Pada *snippet, exception* yang ditangkap lebih general menggunakan kelas *exception*. Terkadang dibutuhkan penanganan spesifik ketika *exception* terjadi. Hal ini dapat dilakukan dengan *multiple catch*. Perhatian *snippet* berikut:

```
public class Test8 {
    public static void main(String[] args){
        try
        {
            int a = 0;
            int b = 10;
            int cs[] = { 1, 2, 3 };
            double d = b / a;
            System.out.println(d);
            System.out.println(cs[5]);
        }
        catch (ArrayIndexOutOfBoundsException aibe)
        {
            System.out.println("ERROR PENGAKSESAN ARRAY TERJADI");
            System.out.println(aibe);
        }
        catch(ArithmeticException ae)
        {
            System.out.println("Error terjadi");
            System.out.println(ae);
        }
    }

}
```

Selidiki apa yang terjadi pada program tersebut? Sertakan jawaban Anda dalam laporan!

## 6. Finally keyword

Pada blok *try catch* terdapat satu *keyword* yang sangat penting, yaitu *finally*. Blok pada *finally* akan selalu dieksekusi baik pada saat aplikasi berjalan normal ataupun terjadi *exception*. Perhatikan contoh dibawah ini:

```
public class Test9 {
    public static void main(String[] args){
        try
        {
            int cs[] = { 1, 2, 3 };
            System.out.println(cs[5]);
        }
        catch (ArrayIndexOutOfBoundsException aibe)
        {
            System.out.println("ERROR PENGAKSESAN ARRAY TERJADI");
            System.out.println(aibe);
        }
        finally
        {
            System.out.println("Blok ini akan selalu dieksekusi apapun yang terjadi");
        }
    }
}
```

Hasil eksekusi dari *snippet* tersebut adalah:

```
ERROR PENGAKSESAN ARRAY TERJADI
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
Blok ini akan selalu dieksekusi apapun yang terjadi
BUILD SUCCESSFUL (total time: 1 second)
```

## 7. Menciptakan sub-class dari exception

Terkadang Anda akan butuh untuk menciptakan *sub-class* dari *exception*, agar aplikasi yang Anda ciptakan dapat melempar *exception* ketika terjadi situasi yang tidak diinginkan. Hal ini dapat dicapai dengan meng-*extends* kelas *exception*. Perhatikan *snippet* berikut:

```java
package org.myprogram.model;

public class Dana {
    private int jumlahUang;

    public Dana(int jumlahUang) {
        this.jumlahUang = jumlahUang;
    }

    public int getJumlahUang() {
        return jumlahUang;
    }

    public void setJumlahUang(int jumlahUang) {
        this.jumlahUang = jumlahUang;
    }

    public int mintaDuluSumbangan(int jumlahSumbangan){
        jumlahUang -= jumlahSumbangan;
        return jumlahSumbangan;
    }

    @Override
    public String toString()
    {
        return "Dana saat ini: "+jumlahUang;
    }
}
```

Kelas Dana tersebut mempunyai sebuah *entity* jumlahUang bertipe int. Kemudian terdapat *method* mintaDuluSumbangan yang akan mengurangi jumlah uang yang ada ketika ada yang memanggilnya. *Method* tersebut juga akan mengembalikan jumlah uang yang disumbang. Perhatikan *driver* berikut ini:

```java
public class Driver {
    public static void main(String[] args) {
        Dana myDana = new Dana(5000);
        System.out.println(myDana);
        System.out.println("MEMINTA DANA: "+myDana.mintaDuluSumbangan(2000));
        System.out.println(myDana);
        System.out.println("MEMINTA DANA: "+myDana.mintaDuluSumbangan(4000));
        System.out.println(myDana);
    }
}
```

Pada *driver* di atas, diciptakan objek Dana, dan memasukkan jumlah dana 5000. Terjadi peminjaman dua kali yaitu 2000 dan 4000. Ini tentu akan menjadi masalah karena jumlah uang akan menjadi -1000. Perhatikan hasil eksekusi *driver* tersebut di bawah ini:

```
Dana saat ini: 5000
MEMINTA DANA: 2000
Dana saat ini: 3000
MEMINTA DANA: 4000
Dana saat ini: -1000
BUILD SUCCESSFUL (total time: 2 seconds)
```

Ini harus dihindari. Hal ini dapat dilakukan dengan menciptakan *sub class* dari *exception* dan melempar objek dari *sub-class* tersebut ketika terjadi peminjaman uang yang melebihi jumlah uang yang tersedia. Perhatikan *snippet* berikut:

```
public class UangKurangEksepsi extends Exception{
    private int jumlahUangSaatIni;
    public UangKurangEksepsi(int jumlahUangSaatIni)
    {
        this.jumlahUangSaatIni = jumlahUangSaatIni;
    }

    public UangKurangEksepsi(int jumlahUangSaatIni, String message)
    {
        super(message);
        this.jumlahUangSaatIni = jumlahUangSaatIni;
    }

    public int getJumlahUangSaatIni()
    {
         return jumlahUangSaatIni;
    }
    public void setJumlahUangSaatIni(int jumlahUangSaatIni)
    {
         this.jumlahUangSaatIni = jumlahUangSaatIni;
    }

}
```

Sehingga akan terjadi perubahan pada *method* mintaDuluSumbangan seperti berikut ini:

```
public int mintaDuluSumbangan(int jumlahSumbangan) throws UangKurangEksepsi{
    if(jumlahUang > jumlahSumbangan){
        jumlahUang -= jumlahSumbangan;
    }
    else{
        throw new UangKurangEksepsi(jumlahUang);
    }
    return jumlahSumbangan;
}
```

Disini akan dilakukan pengecekan terhadap jumlah uang yang ada. Ketika jumlah uang lebih besar, maka sumbangan dilakukan. Ketika tidak, maka akan dilempar kepada sebuah objek dari kelas UangKurangEkspresi. Selain itu, pada *method* tersebut harus

dideklarasikan *method* yang memiliki kemungkinan akan melempar sebuah objek dari kelas UangKurangEkspresi menggunakan *keyword* ***throws***.

Pada *driver,* karena *method* tersebut melempar *exception*, maka mengharuskan pemanggilnya untuk menangani kemungkinan terjadinya *throw exception* tersebut. Perhatikan *snippet* Driver berikut:

```
public class Driver {
    public static void main(String[] args) throws UangKurangEksepsi {
        Dana myDana = new Dana(5000);
        System.out.println(myDana);
        try{
            System.out.println("MEMINTA DANA: "+myDana.mintaDuluSumbangan(2000));
            System.out.println(myDana);
            System.out.println("MEMINTA DANA: "+myDana.mintaDuluSumbangan(4000));
            System.out.println(myDana);
        }
        catch(UangKurangEksepsi uke)
        {
            System.out.println("Memang uang saat ini tinggal = "+
                    uke.getJumlahUangSaatIni());
        }
    }
}
```

Hasil eksekusi dari *snippet* di atas adalah sebagai berikut:

```
Dana saat ini: 5000
MEMINTA DANA: 2000
Dana saat ini: 3000
Memang uang saat ini tinggal = 3000
BUILD SUCCESSFUL (total time: 1 second)
```

Dengan terjadinya *throw exceptions*, maka permintaan sumbangan 4000 tidak akan dapat dilakukan karena jumlah uang yang ada hanya tinggal 3000 saja.

**Tugas!**

1. Terdapat sebuah kelas Toko yang memiliki method-method berikut ini beserta dengan *exception*-nya.
   - Cari barang berdasarkan ID barang

     Jika barang yang dicari tidak ada, maka akan ada *exception* dan penanganannya berupa informasi bahwa barang yang dicari tidak ada.
   - Membeli barang

     Ketika proses membeli barang, jika stok barang yang ingin dibeli kurang maka penanganannya berupa informasi bahwa stok barang tidak mencukupi untuk pembeliannya.
   - Pembelian

     Ketika proses membeli barang ada pengecekan jumlah uang pembeli. Jika uang pembeli kurang (lebih kecil) dari harga barang, maka penanganannya akan muncul informasi bahwa uangnya kurang.

     Sedangkan ketika uang pembeli melebihi (lebih besar) dari harga barang maka ada penanganan yang berisi informasi bahwa uangnya lebih dan ada kembalian sejumlah besarnya uang yang lebih.

     Untuk soal tersebut Anda harus membuat kelas sendiri yang meng-*extends* kelas *Exception*.