# Exception Handling

# Exception Handling

# Definisi Exception

- Suatu mekanisme penanganan kesalahan.
- Event yang terjadi ketika program menemui kesalahan saat instruksi program dijalankan.
- Exception sering digunakan dalam akses sumberdaya non memori.
- Catatan:
  - Exception = untuk menangani kesalahan ringan (mild error).
  - Error = mengindikasikan bahwa error yang terjadi adalah fatal error (severe problem) dimana proses recovery sangat sulit dilakukan bahkan tidak mungkin dilakukan (Contoh : program running out of memory)

# Contoh kesalahan yang terjadi:

- Pembagian bilangan dengan 0
- Pengisian elemen array diluar ukuran array
- Kegagalan koneksi database
- File yang akan dibuka tidak exist
- Operand yg akan dimanipulasi out of prescribed range
- Mengakses obyek y g ang belum diinisialisasi

# Common Exception

- ArithmeticException
  - Hasil dari operasi divide-by-zero pada integer
  - Misal : int i = 12/0;
- NullPointerException
  - Mencoba mengakses atribut atau method suatu object padahal object belum dibuat.
  - Misal : Date d = null;
    System.out.println(d.toString());
- NegativeArraySizeException
  - Mencoba membuat array dengan ukuran negatif.
- ArrayIndexOutOfBoundsException
  - Mencoba mengakses elemen array dimana index nya melebihi ukuran array.
- SecurityException
  - Biasanya dilempar ke browser, class security manager melempar exception untuk applet yang mencoba melakukan:
    - Mengakses lokal file
    - Open socket ke host yg berbeda dgn host yg di open oleh applet

# Contoh Exception

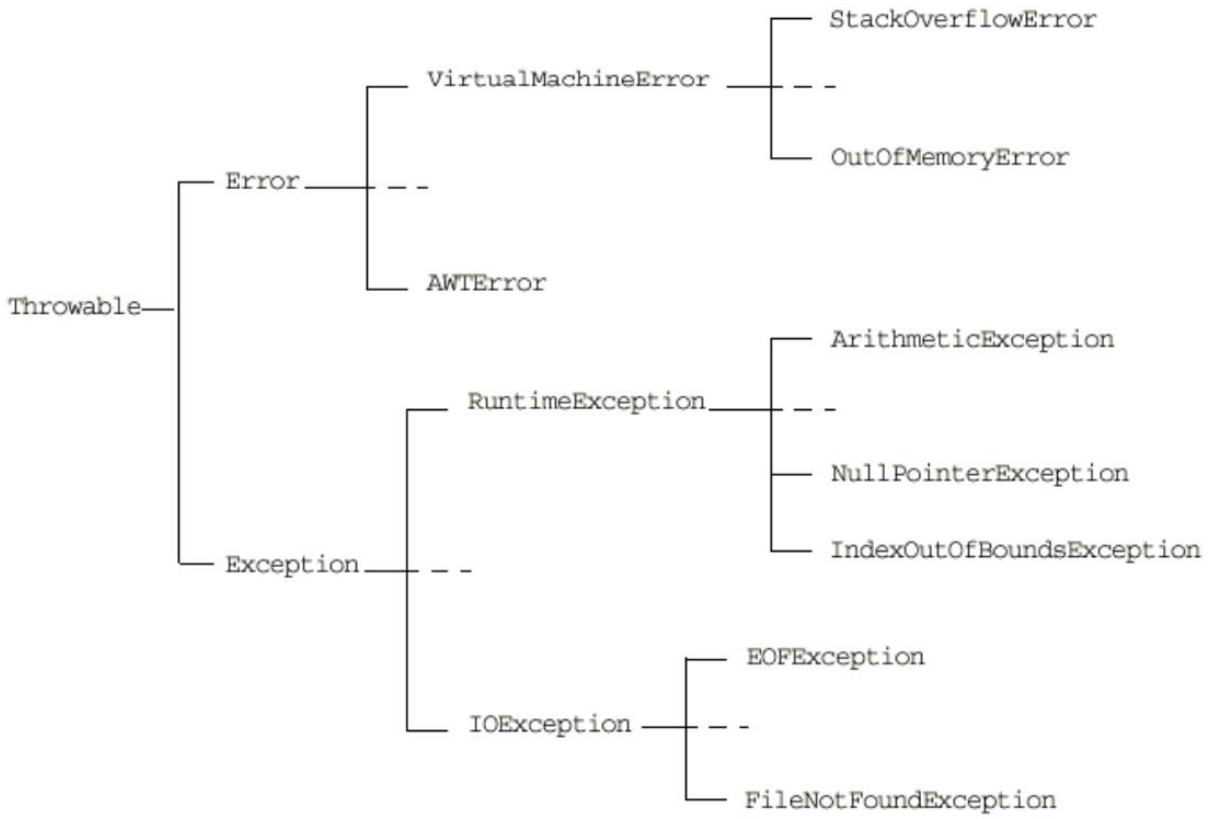
```
Class DivByZero {
    public static void main(String args[]) {
        System.out.println(3/0);
        System.out.println("Pls. print me.");
    }
}
```

- Menampilkan pesan error

  **Exception in thread "main" java.lang.ArithmeticException: / by zero**

  **at DivByZero.main(DivByZero.java:3)**

# Kategori Exception



# Apa yang terjadi jika terjadi kesalahan?

- Secara otomatis akan dilempar sebuah object yang disebut dgn *exception*.
- Exception dapat diproses lebih lanjut oleh fungsi-fungsi yang siap menangani kesalahan.
- Proses pelemparan exception disebut dgn *throwing exception*.
- Proses penerimaan exception di b t sebut dengan *catch exception*

# Contoh kejadian error – Cara lama (loading file from the disk)

```
int status = loadTexfile();
If (status != 1) {
    // something unusual happened, describe it
    switch (status) {
            case 2:
                    // file not found
                    break;
            case 3:
                    //disk error
                    break;
            case 4:
                    //file corrupted
                    break;
            default:
                    // other error
    }
} else {
    // file loaded OK, continue with program
}
```

# Contoh program

- Fungsi bacaFile
  - BukaFile
  - BacaBarisFileSampaiHabis
  - TutupFile

# Ditambahkan program untuk pengecekan berhasil tidaknya pembacaan file

- Fungsi bacaFile
  - BukaFile
  - Jika Gagal Buka File
    - Lakukan Sesuatu
  - Jika Berhasil Buka File
    - BacaBarisFileSampaiHabis
  - TutupFile

- Bagaimana bila ditambahkan program untuk pengecekan terhadap status pembacaan file?
- Bagaimana bila ditambahkan program untuk pengecekan terhadap status penutupan file?
- Maka program akan menjadi sangat panjang dan banyak terdapat nested if-else.

# Solusi?

- Gunakan exception
  - Bentuk:

```
try {
.....
} catch (ExceptionType x) {
.....
}
```

- Blok try : digunakan untuk menempatkan kodekode program java yang mengandung kode program yang mungkin melemparkan exception.
- Blok catch : digunakan untuk menempatkan kode-kode program java yang digunakan untuk menangani sebuah exception tertentu.

# Implementasi 1

```
try {
        Fungsi bacaFile
                BukaFile
                BacaBarisFileSampaiHabis
                TutupFile
} catch (KesalahanBukaFile) {
        // lakukan sesuatu
}
```

# Try dgn banyak catch

- Dapat digunakan beberapa blok catch untuk satu blok try.
- Exception dalam satu program bisa mengatasi banyak exception.
- Contoh implementasi:
  - Misal dalam satu blok try terdapat kemungkinan terjadi:
    - NullPointerException
    - IndexOutOfBoundsException
    - ArithmeticException

```
try {
        ……..
} catch (ExceptionType1 x1) {
        ……..
} catch (ExceptionType2 x2) {
        ……..
}
```

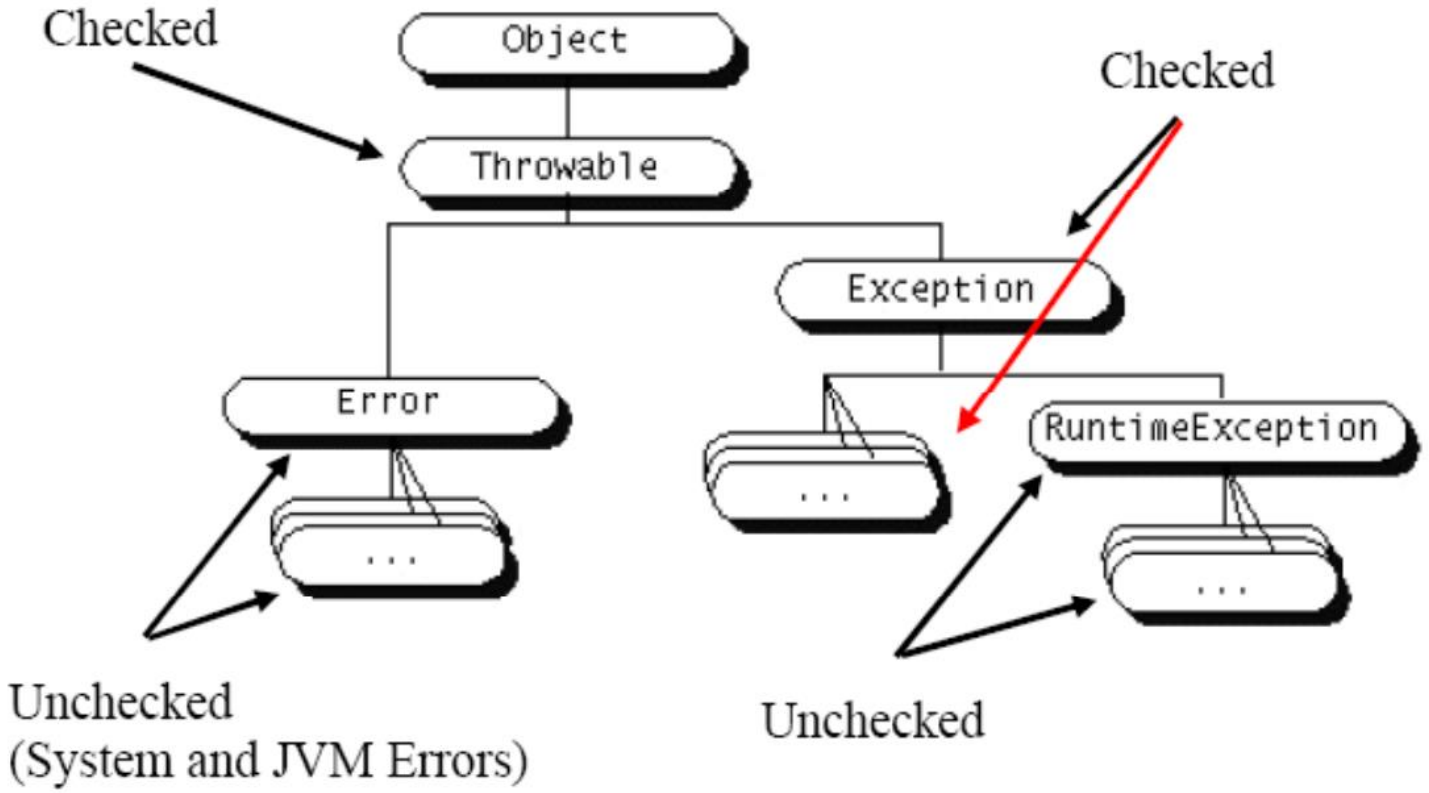
# Implementasi 2

```
try {
        Fungsi bacaFile
        BukaFile
        BacaBarisFileSampaiHabis
        TutupFile
} catch (KesalahanBukaFile) {
        // lakukan sesuatu
} catch (KesalahanAlokasiMemori) {
        // lakukan sesuatu
} catch (KesalahanTutupFile) {
        // lakukan sesuatu
}
```

# Checked/Unchecked Exceptions

- Exception bisa checked atau unchecked
  - Checked = dicek oleh the compiler
- Checked exception hanya dapat ditangani dalam try block atau method yang didesain untuk melempar exception.
  - Compiler akan memberitahu jika checked exception tidak ditangani secara tepat.
  - Contoh : IOException.
- Unchecked exception tidak memerlukan penanganan langsung . Pada saat dicompile tidak ada pemberitahuan kesalahan.
  - Contoh : RunTimeException dan turunannya

# Checked/Unchecked Exceptions



# Contoh: Tanpa Exception Handling

```
1   public class HelloWorld {
2      public static void main (String[] args) {
3         int i = 0;
4
5         String greetings [] = {
6            "Hello world!",
7            "No, I mean it!",
8            "HELLO WORLD!!"
9         };
10
11        while (i < 4) {
12           System.out.println (greetings[i]);
13           i++;
14        }
15     }
16  }
```

# Contoh: Dengan Exception Handling

```
1  public class HelloWorld {
2    public static void main(String[] args) {
3      int i = 0;
4
5      String[] greetings = {
6        "Hello world!",
7        "No, I mean it!",
8        "HELLO WORLD!!"
9      };
10
11     while (i < 4) {
12       try {
13         System.out.println(greetings[i]);
14         i++;
15       } catch (ArrayIndexOutOfBoundsException e){
16         System.out.println("Re-setting Index Value");
17         i = 0;
18       } finally {
19         System.out.println("This is always printed");
20       }
21     }
22   }
23 }
```

# Membuat class exception baru

- Sebuah subclass dari exception dapat dibuat sendiri oleh programmer untuk mendefinisikan sendiri secara lebih rinci tentang exception yang dapat terjadi.
- Class exception baru ini harus merupakan subclass dari java.lang.Exception.

# Membuat Exception

- Tujuan: mendefinisikan class exception yang lebih spesifik untuk keperluan tertentu.
- Untuk memuat class exception baru maka class itu harus merupakan subclass dari class Exception.

# Contoh 1: Membuat class exception baru

```
class Salah extends Exception{
        public Salah(){}
        public Salah(String pesan){
                super(pesan);
        }
}
```

```
public class TesSalah{
        public static void main(String [] arg) throws Salah{
                Salah s = new Salah(“Salah disengaja ha ha ”); ..ha..”);
                int i = 0;
                if (i==0)
                        throw s;
        }
}
```