

# EXCEPTION HANDLING

---

# Learning Outcomes

---

Pada akhir pertemuan ini, diharapkan mahasiswa akan mampu:

**Menjelaskan pengertian exception handling**

**Menerapkan penulisan program dengan exception handling**

# Outline Materi

---

Definisi *Exception Handling*

*try and catch*

*finally*

# Exception Handling

---

3 jenis error:

- *Syntax errors (compile errors)* → melanggar aturan sintaks bahasa pemrograman, ditemukan saat kompilasi oleh kompiler
- *Logic errors (bug)* → kesalahan logika, menghasilkan output/performa yang menyimpang
- *Runtime errors* → operasi yang salah saat eksekusi program, program berakhir

Runtime errors : *exception*

Exception menyebabkan program *terminate* (berakhir)

Contoh:

Nasabah A mentransfer uang ke rekening nasabah B, saat rekening A berkurang dan rekening B belum bertambah, terjadi exception dan program terminate. Nasabah A kehilangan uang.

# Exception Handling

---

Menangkap/penanganan *runtime errors (exception handling)*

Menggunakan *try and catch*

Jenis kesalahan yang umum terjadi:

- Inputan yang salah
- Aritmetika (pembagian dengan nol)
- Melewati batas array yang dipesan
- Object yang belum diinisialisasi

Jika kesalahan tidak ditangani/ditangkap (*catch*), maka kesalahan akan diteruskan ke penanganan berikutnya

Kesalahan yang tidak ditangani akan menyebabkan program berakhir

# Exception Handling

---

```
1  import java.util.Scanner;
2
3  public class ExceptionDemo
4  {
5      public static void main(String[] args)
6      {
7          Scanner input = new Scanner(System.in);
8          int bilangan;
9
10         System.out.print("Masukan bilangan : ");
11         bilangan = input.nextInt();
12         System.out.print("Bilangan yang dimasukan adalah "+bilangan);
13     }
14 }
15
```

```
Masukan bilangan : 3.7
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:840)
    at java.util.Scanner.next(Scanner.java:1461)
    at java.util.Scanner.nextInt(Scanner.java:2091)
    at java.util.Scanner.nextInt(Scanner.java:2050)
    at ExceptionDemo.main(ExceptionDemo.java:11)
```

# Exception Handling

---

Pesan errors:

```
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:840)
    at java.util.Scanner.next(Scanner.java:1461)
    at java.util.Scanner.nextInt(Scanner.java:2091)
    at java.util.Scanner.nextInt(Scanner.java:2050)
    at ExceptionDemo.main(ExceptionDemo.java:11)
```

ExceptionDemo.main(ExceptionDemo.java:11) ← kesalahan ada di baris 11

Program berakhir (*terminate*) dan statement berikutnya tidak dijalankan

# Exception Handling

---

```
import java.util.Scanner;

public class HandleExceptionDemo
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        int bilangan;

        System.out.print("Masukan bilangan : ");

        try
        {
            bilangan = input.nextInt();
            System.out.print("Bilangan yang dimasukan adalah "+bilangan);
        }
        catch(Exception e)
        {
            System.out.println("Inputan anda salah");
        }

        System.out.println("Terima kasih");
    }
}
```

```
Masukan bilangan : 3.7
Inputan anda salah
Terima kasih
```



# *Exception Handling*

---

Statement yang bisa menyebabkan *exception* berada pada lingkup **try**

*Exception* ditangkap ada lingkup **catch**

*Statement* pada lingkup **catch** merupakan operasi yang dilakukan jika terjadi *exception*

*Exception* ditangkap pada **catch**(*Exception e*)

Setelah **catch**, maka program kembali normal

*Statement* berikutnya akan berjalan normal

# Exception Handling

---

```
import java.io.*;
import java.util.*;

public class ExceptionHandling
{
    public static void main(String[] args)
    {
        int hasil;
        char [] kata = {'a', 'b', 'c'};

        System.out.println("Contoh penanganan kesalahan pembagian dengan nol");
        try
        {
            hasil = 5/0;
        }
        catch(ArithmeticException ae)
        {
            System.out.println("Masuk ke ArithmeticException dengan pesan:");
            System.out.println(""+ae+"\n");
        }
    }
}
```

```
Contoh penanganan kesalahan pembagian dengan nol
Masuk ke ArithmeticException dengan pesan:
java.lang.ArithmeticException: / by zero
```

# Exception Handling

---

```
System.out.println("Contoh penanganan kesalahan melebihi batas index array");
try
{
    kata[100]='a' ;
}
catch(ArithmeticException ae)
{
    System.out.println("Masuk ke ArithmeticException dengan pesan:");
    System.out.println(""+ae);
}
catch(ArrayIndexOutOfBoundsException ie)
{
    System.out.println("Masuk ke ArrayIndexOutOfBoundsException dengan pesan:");
    System.out.println(""+ie);
}
catch(Exception ex)
{
    System.out.println("Masuk ke Exception dengan pesan:");
    System.out.println(""+ex);
}
}
```

```
Contoh penanganan kesalahan melebihi batas index array
Masuk ke ArrayIndexOutOfBoundsException dengan pesan:
java.lang.ArrayIndexOutOfBoundsException: 100
```

# Exception Handling

---

Pada:

```
try
{
    kata[100]='a';
}
```

- Maka hasil *exception* merupakan **ArrayIndexOutOfBoundsException**, sehingga tidak akan masuk ke **ArithmeticException**
- Jika **ArrayIndexOutOfBoundsException** tidak dideklarasikan, maka akan masuk ke **Exception**
- **Exception** merupakan jenis umum → letakkan

# Exception Handling

---

Mencetak pesan kesalahan:

```
catch(Exception e)
```

```
{
```

```
    ...
```

```
}
```

- `System.out.println(e);`
- `e.printStackTrace();`

Hasil output:

```
System.out.println(e); java.util.InputMismatchException
```

```
e.printStackTrace(); java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:840)
    at java.util.Scanner.next(Scanner.java:1461)
    at java.util.Scanner.nextInt(Scanner.java:2091)
    at java.util.Scanner.nextInt(Scanner.java:2050)
    at HandleExceptionDemo.main(HandleExceptionDemo.java:14)
```

# Advanced Learning

---

*Statement* dapat dieksekusi walaupun terjadi *exception*

Keyword: **finally**

Sintaks:

```
try
{
    tryStatements;
}
catch(Exception e)
{
    handling ex;
}
finally
{
    finalStatements;
}
```

---

Statements pada **finally** akan dieksekusi:

- Tidak terjadi *exception*
- Terjadi *exception* pada *statements* **try**, dan *exception* di-**catch**
- Terjadi *exception* pada *statements* **try**, dan *exception* tidak di-**catch**

**finally** dapat dideklarasikan tanpa **catch**

Deklarasi **try** perlu disertakan dengan **catch** atau **finally**

Contoh:

```
try
{
    ...
}
finally
{
    ...
}
```

---

*try and catch* sebaiknya untuk penanganan kesalahan yang tidak diharapkan  
Jangan gunakan jika dapat ditangani manual

Contoh:

```
if(pembagi==0)
    System.out.println("Pembagi tidak boleh nol");
else
    hasil = bilangan / pembagi;
```

*Akan lebih baik daripada:*

```
try
{
    hasil = bilangan / pembagi;
}
catch(Exception e)
{
    System.out.println("Kesalahan pembagian");
}
```




---

### Kerugian *try and catch*:

- Memerlukan waktu lebih saat dieksekusi
- Memerlukan memori yang lebih banyak
- Memerlukan pencarian handler

### Keuntungan *try and catch*:

- Menangkap kesalahan yang kompleks
  - Mudah untuk dibaca dan dimodifikasi
- 

# Referensi

---

## Exception Handling:

- <http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html>
- <http://www.javabeginner.com/java-exceptions.htm>
- <http://rotterdam.ics.uci.edu/info/ExceptionHandler.htm>