Exception

Exception

- Saat kita membuat aplikasi, kita tidak akan terhindar dengan yang namanya error
- Di Java, error direpresentasikan dengan istilah exception, dan semua direpresentasikan dalam bentuk class exception
- Kita bisa menggunakan class exception sendiri, atau menggunakan yang sudah disediakan oleh Java
- Jika kita ingin membuat exception, maka kita harus membuat class yang extends class Throwable atau turunan-turunannya

Kode : Membuat Class Exception

```
public class ValidationException extends Throwable {

  private String message;

  public ValidationException(String message) {
    this.message = message;
  }

  public String getMessage() {
    return message;
  }
}
```

Membuat Exception

- Exception biasanya terjadi di method, ketika kita membuat exception di sebuah method, maka method tersebut harus ditandai dengan kata kunci thrown diikuti dengan class exception nya.
- Jika method tersebut bisa menimbulkan lebih dari satu jenis exception, kita bisa menambah lebih dari satu class exception
- Di dalam kode program kita, untuk membuat exception kita cukup menggunakan kata kunci throw, diikuti dengan object exception nya

Kode : Membuat Exception

```
public class ValidationUtil {

    public static void validate(LoginRequest loginRequest) throws ValidationException {
        if (loginRequest.username() == null) {
            throw new ValidationException("Username tidak boleh null");
        } else if (loginRequest.username().isBlank()) {
            throw new ValidationException("Username tidak boleh kosong");
        } else if (loginRequest.password() == null) {
            throw new ValidationException("Password tidak boleh null");
        } else if (loginRequest.password().isBlank()) {
            throw new ValidationException("Password tidak boleh kosong");
        }
```

Try Catch

- Saat kita memanggil sebuah function yang bisa menyebabkan exception, maka kita wajib menggunakan try-catch expression di Java
- Ini gunanya agar kita bisa menangkap exception yang terjadi, karena jika tidak ditangkap, lalu terjadi exception, maka secara otomatis program kita akan berhenti
- Cara menggunakan try-catch expression di java sangat mudah, di block try, kita tinggal panggil method yang bisa menyebabkan exception, dan di block catch, kita bisa melakukan sesuatu jiga terjadi exception

Kode : Try Catch

```
    LoginRequest loginRequest = new LoginRequest( username: null,  password: null);

    try {
      ValidationUtil.validate(loginRequest);
    } catch (ValidationException e) {
      System.out.println("Terjadi Error Dengan Pesan : " + e.getMessage());
    }

  }

}
```

Kode : Multiple Try Catch (1)

```
    LoginRequest loginRequest = new LoginRequest( username: null,  password: null);

    try {
        ValidationUtil.validate(loginRequest);
    } catch (ValidationException e) {
        System.out.println("Terjadi Error Dengan Pesan : " + e.getMessage());
    } catch (NullPointerException e) {
        System.out.println("Terjadi Error Dengan Pesan : " + e.getMessage());
    }

}
```

Kode : Multiple Try Catch (2)

```
    LoginRequest loginRequest = new LoginRequest( username: null,  password: null);

    try {
      ValidationUtil.validate(loginRequest);
    } catch (ValidationException | NullPointerException e) {
      System.out.println("Terjadi Error Dengan Pesan : " + e.getMessage());
    }

  }

}
```

Finally Keyword

- Dalam try-catch, kita bisa menambahkan block finally
- Block finally ini adalah block dimana akan selalu dieksekusi baik terjadi exception ataupun tidak
- Ini sangat cocok ketika kita ingin melakukan sesuatu, tidak peduli sukses ataupun gagal, misal di block try kita ingin membaca file, di block catch kita akan tangkap jika terjadi error, dan di block finally error ataupun sukses membaca file, kita wajib menutup koneksi ke file tersebut, biar tidak menggantung di memory

Kode : Finally

```
    LoginRequest loginRequest = new LoginRequest( username: null,  password: null);

    try {
        ValidationUtil.validate(loginRequest);
    } catch (ValidationException | NullPointerException e) {
        System.out.println("Terjadi Error Dengan Pesan : " + e.getMessage());
    } finally {
        System.out.println("Error Gak Error, Tetap Di Panggil");
    }

}
```

Runtime Exception

Jenis Exception

Secara garis besar, di Java, exception dibagi menjadi 3 jenis

- Checked Exception, yaitu exception yang wajib di try catch, seperti yang sudah kita bahas sebelumnya
- Runtime Exception, dan
- Error (yang akan kita bahas di materi selanjutnya)

Runtime Exception

- Runtime exception adalah jenis exception yang tidak wajib di tangkap menggunakan try catch
- Kompiler Java tidak akan protes walaupun kita tidak menggunakan try catch ketika kita memanggil method yang bisa menyebabkan runtime exception
- Untuk membuat class runtime exception, kita wajib mengextends class RuntimeException
- Ada banyak di Java yang merupakan runtime exception, seperti NullPointerException, IllegalArgumentException, dan lain-lain

Kode : Membuat Runtime Exception

```java
package eko.belajarjava.error;

public class BlankException extends RuntimeException{

  public BlankException(String message) {
    super(message);
  }
}
```

Kode : Method Dengan Runtime Exception

```
@ public static void validateRuntime(LoginRequest loginRequest) {
    if (loginRequest.username() == null) {
      throw new BlankException("Username tidak boleh null");
    } else if (loginRequest.username().isBlank()) {
      throw new BlankException("Username tidak boleh kosong");
    } else if (loginRequest.password() == null) {
      throw new BlankException("Password tidak boleh null");
    } else if (loginRequest.password().isBlank()) {
      throw new BlankException("Password tidak boleh kosong");
    }
}
```

Kode : Tanpa Try Catch

```java
public class ValidationApp {
  public static void main(String[] args) {

    LoginRequest loginRequest = new LoginRequest( username: null,  password: null);
    ValidationUtil.validateRuntime(loginRequest);

  }

}
```

Perlu Diperhatikan

- Walaupun runtime exception tidak wajib di try-catch, tapi ada baiknya kita tetap melakukan try-catch
- Karena jika terjadi runtime exception, yang ditakutkan adalah program kita berhenti

Error

Error

- Error adalah jenis exception yang terakhir
- Error adalah sebuah class di Java, yang tidak direkomendasikan untuk di try-catch
- Biasanya error terjadi ketika ada masalah serius, dan sangat tidak direkomendasikan untuk di try catch
- Artinya, direkomendasikan untuk mematikan aplikasi
- Contoh, misal jika diawal aplikasi kita tidak bisa terkoneksi ke database, direkomendasikan untuk membuat exception jenis Error, dan menghentikan aplikasi

Kode : Membuat Error

```
package eko.belajarjava.error;

public class DatabaseError extends Error{

  public DatabaseError(String message) {
    super(message);
  }
}
```

Kode : Terjadi Error

```
public class DatabaseApp {
  public static void main(String[] args) {
    connectDatabase( username: "admin",  password: null);
  }

  public static void connectDatabase(String username, String password) {
    if (username == null || password == null) {
      throw new DatabaseError("Tidak bisa konek ke database");
    }
  }
}
```

StackTraceElement Class

StackTraceElement Class

- Throwable memiliki method yang bernama getStackTrace(), dimana menghasilkan Array dari StackTraceElement object
- StackTraceElement berisikan informasi tentang, class, file bahkan baris lokasi terjadinya error
- Class StackTraceElement ini sangat penting untuk menelusuri lokasi kejadian error yang terjadi
- Cara yang paling mudah, kita bisa memanggil method printStackTrace() class Throwable, untuk memprint ke console detail error StackTraceElement nya

Kode : StackTraceElement

```
public static void main(String[] args) {
    try {
        String[] names = {
            "Eko", "Kurniawan", "Khannedy"
        };
        System.out.println(names[100]);
    }catch (Throwable throwable){
        StackTraceElement[] stackTraces = throwable.getStackTrace();

        throwable.printStackTrace();
    }
}
```

Kode : Multiple StackTraceElement

```
public static void sampleError() throws Throwable {
    try {
        String[] names = {
            "Eko", "Kurniawan", "Khannedy"
        };
        System.out.println(names[100]);
    } catch (Throwable throwable) {
        throw new Throwable(throwable);
    }
}
}
```

Try with Resource

Try with Resource

- Di Java 7, terdapat fitur baru yang bernama try with resource
- Try with resource adalah sebuah mekanisme agar kita lebih mudah menggunakan resource (yang wajib di close) dalam block try
- Jika kita ingin menggunakan fitur ini, kita wajib menggunakan interface AutoCloseable

Kode : Manual Close Resource

```
BufferedReader reader = null;
try {
  reader = new BufferedReader(
      new FileReader( fileName: "sample")
  );
  while (true) {
    String text = reader.readLine();
    if (text == null) {
      break;
    }
    System.out.println(text);
  }
  }
} catch (IOException exception) {
  exception.printStackTrace();
} finally {
  if (reader != null) {
    try {
      reader.close();
    } catch (IOException exception) {
      exception.printStackTrace();
    }
  }
}
```

Kode : Try with Resource

```
try (BufferedReader reader = new BufferedReader(new FileReader( fileName: "sample"))) {
  while (true) {
    String text = reader.readLine();
    if (text == null) {
      break;
    }
    System.out.println(text);
  }
} catch (IOException exception) {
  exception.printStackTrace();
}
```

Annotation

Annotation

- Annotation adalah menambahkan metadata ke kode program yang kita buat
- Tidak semua orang membutuhkan Annotation, biasanya Annotation digunakan saat kita ingin membuat library / framework
- Annotation sendiri bisa diakses menggunakan Reflection, yang akan kita bahas nanti
- Untuk membuat annotation, kita bisa menggunakan kata kunci @interface
- Annotation hanya bisa memiliki method dengan tipe data sederhana, dan bisa memiliki default value

Attribute Annotation

Attribute	Keterangan
@Target	Memberitahu annotation bisa digunakan di mana? Class, method, field, dan lain-lain
@Retention	Memberitahu annotation apakah disimpan di hasil kompilasi, dan apakah bisa dibaca oleh reflection?

Kode : Membuat Annotation

```java
import java.lang.annotation.*;

@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = {ElementType.TYPE})
public @interface Fancy {

  String name();

  String[] tags() default {};

}
```

Kode : Menggunakan Annotation

```
package eko.belajarjava.annotation;

@Fancy(name = "Eko", tags = {"app", "java"})
public class Application {

}
```

Predefined Annotation

Java juga sudah memiliki annotation bawaan, seperti :

- @Override, untuk menandai bahwa method yang meng-override method parent class nya
- @Deprecated, untuk menandai bahwa method tersebut tidak direkomendasikan lagi untuk digunakan
- @FunctionalInterface, untuk menandai bahwa class tersebut bisa dibuat sebagai lambda expression
- dan lain-lain

Reflection

Reflection

- Reflection adalah kemampuan melihat struktur aplikasi kita pada saat berjalan
- Reflection biasanya sangat berguna saat kita ingin membuat library ataupun framework, sehingga bisa meng-otomatisasi pekerjaan
- Untuk mengakses reflection class dari sebuah object, kita bisa menggunakan method getClass() atau NamaClass.class

Kode : Annotation di Field

```
public class CreateUserRequest {

  @NotBlank
  private String username;

  @NotBlank
  private String password;

  public String getUsername() {
    return username;
  }
```

Kode : Validasi Menggunakan Reflection

```
public static void validateRequest(CreateUserRequest request) throws IllegalAccessException, Vali
  Field[] fields = request.getClass().getDeclaredFields();
  for (Field field : fields) {
    if (field.getAnnotation(NotBlank.class) != null) {
      field.setAccessible(true);
      String value = (String) field.get(request);
      if (value == null || value.isBlank()) {
        throw new ValidationException("Field " + field.getName() + " is blank");
      }
    }
  }
}
```