accenture
Latvia ATC

Test Automation Engineering Fundamentals: Java

Module 8: Exceptions and Assertions

Module Objectives

accenture
Latvia ATC

- At the end of this module, you will be able to:
  - Explain the concept of Exceptions and Assertions.
  - Explain the usage of Exceptions and Assertions.
  - Manage exceptions using try-catch-finally.
  - Create customized exceptions and exception conditions.
  - Use assertion statements to improve code quality.

Exceptions

- Exception is:
  - An event during program execution that prevents the program from continuing normally.
  - An error condition that changes the normal flow of control in a program.
  - A signal that some unexpected condition has occurred in the program.
  - Classified as Checked, Unchecked, and Errors.

Handling Exceptions

- The exception handling mechanism is built around the **throw-and-catch paradigm:**
  - ➢ **‘to throw’ means an exception has occurred.**
  - ➢ **‘to catch’ means to deal with, or handle an exception.**
- If an exception is not caught, it is **propagated** to the call stack until a handler is found.

Using try-catch-finally Blocks

```java
try {
   /*
    * some codes to test here
    */
} catch (SQLException sx) {
   /*
    * handle Exception1 here
    */
} catch (IOException ix) {
   /*
    * handle Exception2 here
    */
} catch (Exception ex) {
   /*
    * handle Exception3 here
    */
} finally {
   /*
    * always execute codes here
    */
}
```

Try block encloses the context where a possible exception can be thrown

Each **Catch() block** is an exception handler and can appear several times

An optional **Finally block** is always executed before exiting the **Try** statement**.**

Refer to the TryCatchFinallySample.java sample code.

Using try-catch-finally Blocks (cont.)

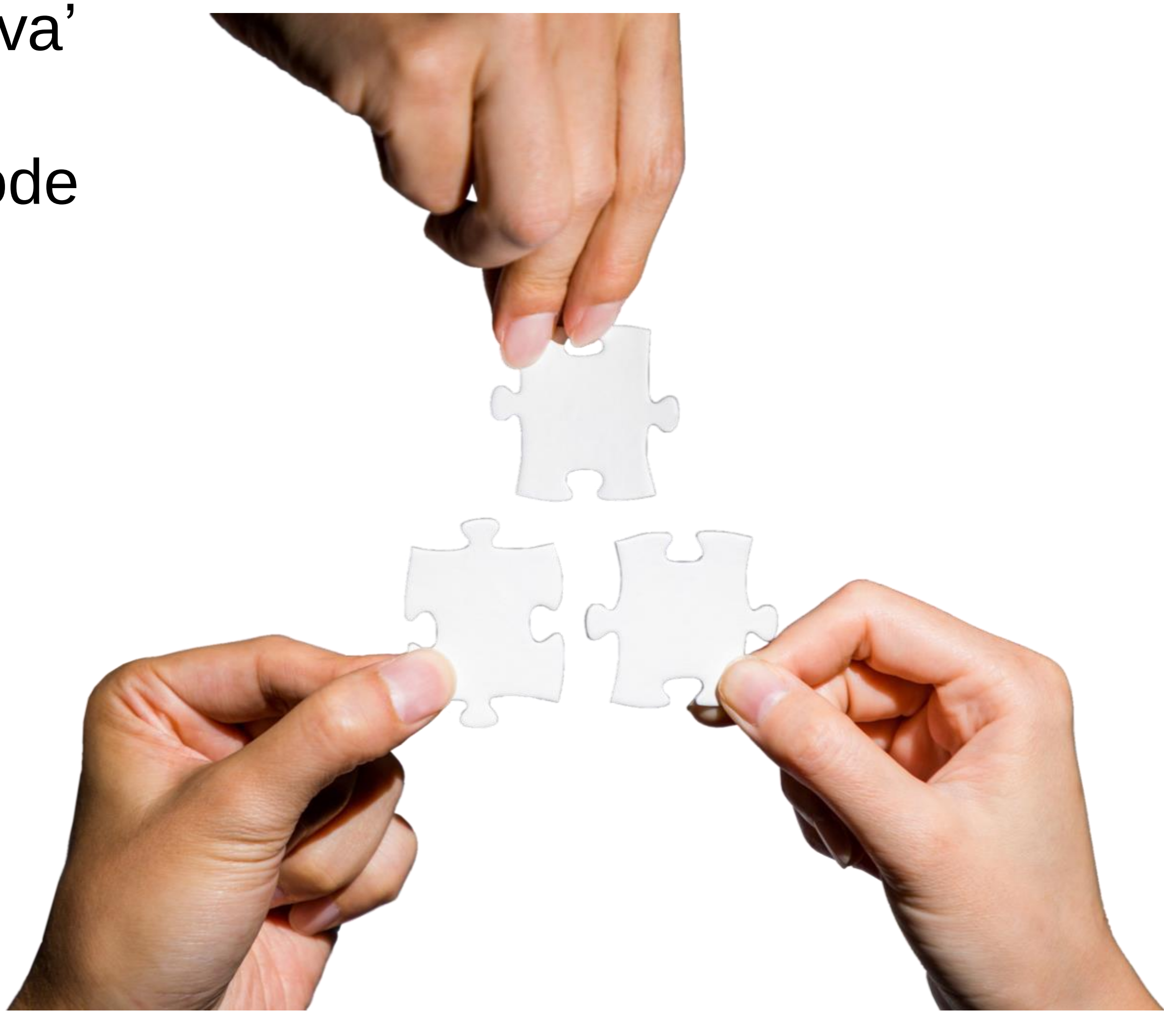
accenture
Latvia **ATC**

- Isolate code that might throw an exception in the **try** block.
- For each individual **catch()** block, you write code that is to be executed if an exception of that particular type occurs in the **try** block.
- In the **finally** block, you write code that will be run whether or not an error has occurred. This is optional.

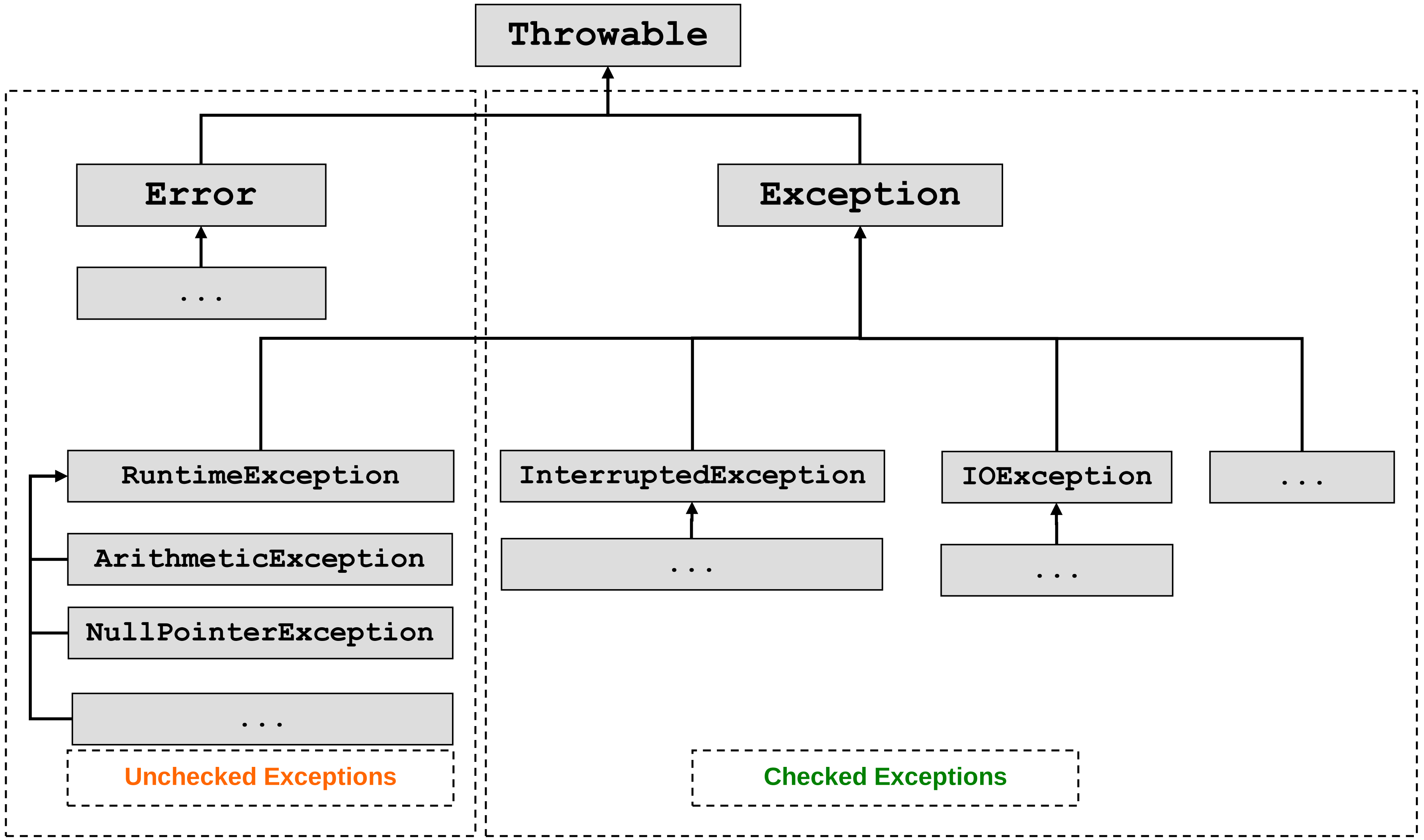
i Refer to the TryCatchFinallySample.java sample code.

Activity 1 – TryCatchFinally

- In this activity, you will:
  - Open the file ‘TryCatchFinallyActivity.java’ in the package sef.module8.activity.
  - Read the instructions and create the code to complete this program.



Exception Class Hierarchy



Un-Checked Exceptions

accenture
Latvia ATC

- The Un-checked Exceptions represent errors usually **caused by incorrect program code or logic** such as invalid parameters passed to a method.
- They are a subclass of the **RuntimeException** class.
- The application is not required to handle these exceptions as these should be recovered by correcting program code.
- **Examples**: IllegalArgumentException, NumberFormatException.

i Refer to the ArrayExceptionSample.java and FormatExeptionSample.java sample code.

Activity 2 – Arithmetic Exception

accenture
Latvia ATC

- In this activity, you will:
  - Open the file ‘ArithmeticExceptionActivity.java’ in the package sef.module8.activity.
  - Read the instructions and create the code to complete this program.

Checked Exceptions

accenture
Latvia **ATC**

- The Checked Exceptions represent errors **caused by factors outside of the application code.**
- They are subclasses of the **Exception** class excluding the RuntimeException class.
- The application is required to handle these exceptional scenarios through try-catch constructs.
- **Examples**: IOException, SQLException

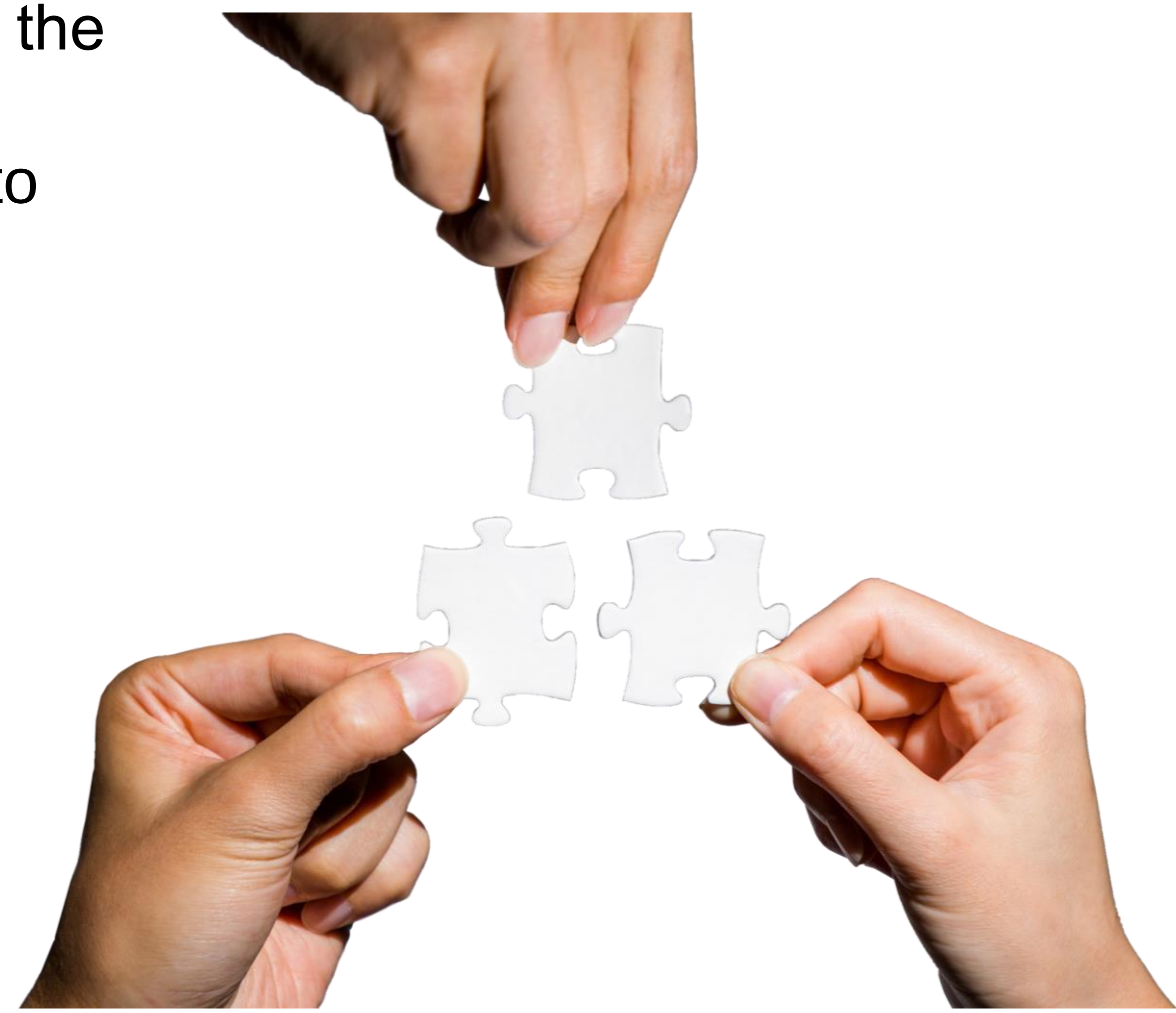
i Refer to CheckedExceptionSample.java sample code.

Activity 3 – SQL Exception

In this activity, you will:

Open the file ‘SQLExceptionActivity.java’ in the package sef.module8.activity.

Read the instructions and create the code to complete this program.



Errors

accenture
Latvia ATC

- Errors represent critical errors that should not occur and that the application is not expected to recover from.
- Errors are typically generated from mistakes in program logic or design and should be handled through correction of design or code.
- **Examples**: OutOfMemoryError, StackOverFlowError

i Refer to ErrorSample.java sample code.

Specifying Exceptions

accenture
Latvia **ATC**

- Exceptions can also be handled by propagating them up the call stack instead of handling them in the current method.
- A method can declare that one of its statements might throw an Exception and that it is leaving to whoever is calling the method to handle it.

```
<method signature> throws <Exception1>,<Exception2>

public void connectToDB (String query)throws SQLException,IOException {
      //code here
}
```

- Any statement that might generate a checked exception that is declared by the method is considered 'handled' and does not need a try-catch block.

i Refer to SpecifyExceptionSample.java sample code.

Handling Exception Through Declaration

- Code can be told to explicitly throw an Exception (Checked or Unchecked).
- Exceptions are represented as Java objects and can be created just like any other object, and then ‘thrown’ using the throw keyword.

```
Example:
public void setAge(int age){
   if(age < 0 ){
            //create an instance and throw at the same time
throw new IllegalArgumentException(“parameter age cannot be less than 0”);
   }
}
```

Refer to the ExceptionDeclarationSample.java sample code.

Customizing Exceptions

- Exceptions in the standard API may not be sufficient to cover the scenarios needed by the application.
- A customized exception can be declared by **sub-classing the Exception class.**
- The customized exception should contain appropriate data and behavior in order to assist in properly identifying and correcting the problem.

Refer to the CustomException.java and CustomExceptionSample.java sample code.

Activity 4 – Custom Exception

- In this activity, you will:
  - Open the files ‘CustomExceptionActivity.java’ and ‘CustomExceptionActivityTest.java’ in the package sef.module8.activity.
  - Read the instructions and create the code to complete this program.

Assertion Statements

- An **assertion** is a programming language construct that checks whether a specified expression is true.
- The assertions are used to assist the programmer in improving code quality. Verification done using assertions are not a part of the actual code logic.
- Assertions can be used to:
  - ➢ Validate pre-conditions before entering a section of code.
  - ➢ Validate post-conditions after executing a section of code.
  - ➢ Validating class invariants whenever the state of the object is modified.

Using Assert Statements

accenture
Latvia ATC

- Assertions can be inserted anywhere in code using the following syntax:
  - ➢ assert <boolean expression>
  - ➢ assert<boolean expression> : <String expression>

- If the boolean expression is *false* then the statement will throw an **AssertionError** and will display the String expression (if specified)

i Refer to the AssertSample.java sample code.

Assertion Sample Code

accenture
Latvia ATC

```
import java.util.Scanner

   public class AssertTest
    {
        public static void main( String args[] )
        {
            Scanner input = new Scanner( System.in );
            System.out.print( "Enter a number between 0 and 10: " );
            int number = input.nextInt();

            // assert that the absolute value is between 1-10
            assert ( number > 0 && number <= 10 ) : "bad number: " + number;

            System.out.printf( "You entered %d\n", number );
        } // end main

    } // end class AssertTest
```

OUTPUT 1	OUTPUT 1
Enter a number between 0 and 10:5.	Enter a number between 0 and 10:50
You entered 5	Exception in thread “main” java.lang.AssertionError: bad number: 50 at AssertTest.main(AssertTest.java:15)

Enabling/Disabling Assertions

accenture
Latvia ATC

- To enable assertions at runtime, use the following commands:
  - java enableassertion <java class file> OR
  - java –ea <java class file>
  - E.g., java –ea AssertionMain
- To disable assertions at runtime, use the following commands:
  - java disableassertion <java class file> OR
  - java –da <java class file>
  - E.g., java –da AssertionMain
- To enable assertions at Runtime (in Eclipse), use the following commands:
  - Right-click on the file and select Run As >Run Configurations
  - Click on the Arguments tab
  - In the VM arguments text box, enter -ea

Activity 5 – Exception Sequence

- In this activity, you will:
  - Open the file ‘ExceptionSequenceActivity.java’ in the package sef.module8.activity.
  - Read the instructions and create the code to complete this program.

Questions and Comments

accenture
Latvia ATC

What questions or comments do you have?