



# SDE Readiness Training

Empowering Tomorrow's Innovators



# Module I

*Java Software Development:  
Effective Problem Solving*

# Exception Handling

Learning Level :

DATE : 03.07.2025



## Exception handling

### Realtime Scenario

- Whenever we **travel in Aircraft**, **Airhostess** used to give **demonstration of steps** that the passengers have to take in case of **emergency**.



# Realtime Scenario

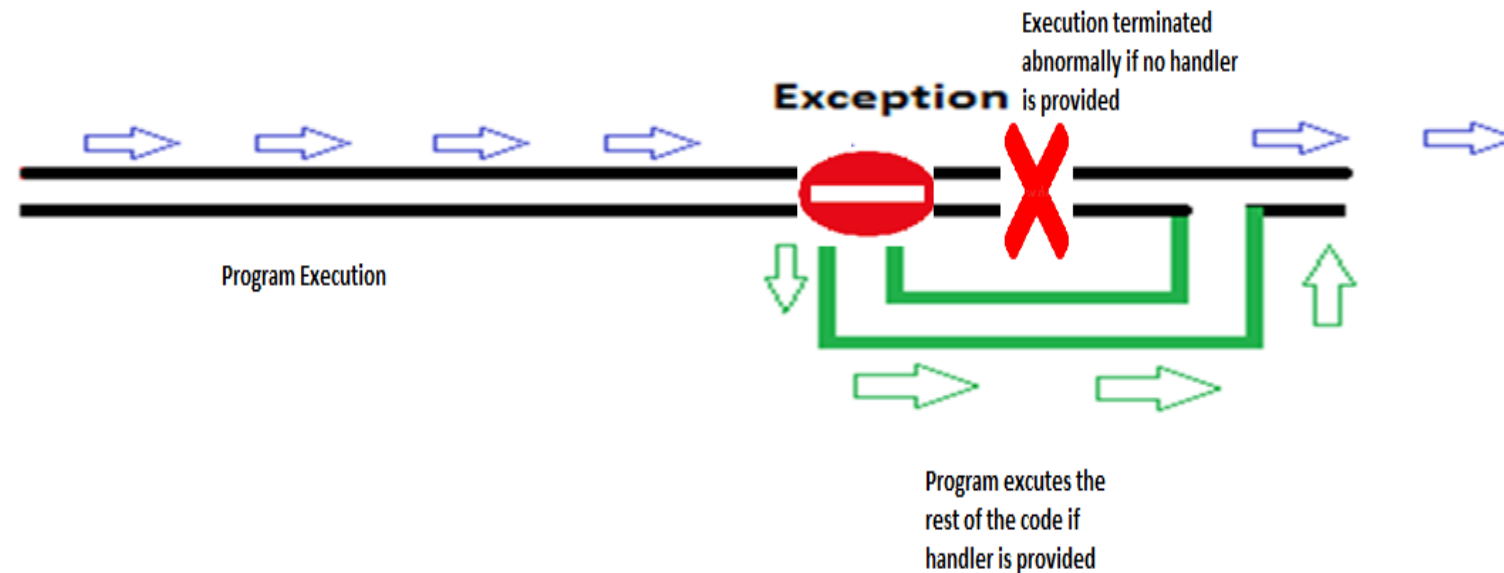
### Why these demonstration steps are important?

- The reason is that we have to be **aware of how to handle a situation** in case of any emergency while you are flying.
- This scenario explains how you have to think in advance the many possibilities of mishaps that can happen and the **preventive measures to save**.

## Exception handling

### Exception handling: Need

- Similarly, when we write the programs as a part of any application, we have to visualize the challenges that can **disrupt the normal flow of execution**. If so, how to overcome these.
- In Java, **Exception handling mechanism** helps the programmer to have a comfortable seat when such situation occurs.



# What is an Exception?

- An exception is an **unexpected event**, which occurs during the execution of a program i.e., at **run time that disrupts the normal flow** of the program's instructions.
- Below are some **situations** when an exception could occur
  - Performing an illegal arithmetic operation.
  - Inserting/accessing an array beyond its boundary.
  - Accessing a file that does not exist
  - etc.

## Exception handling

### Example

```
class SimpleArithmetic {  
    public static void main(String args[]){  
        int result=75/0; //exception occur  
        System.out.println("Arithmetic Operation result: "+ result);  
        ....  
    }  
}
```

#### Output:

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



## Exception handling

# Exception Types

- There are **three categories** of exceptions
  - Checked Exceptions
  - Unchecked Exceptions
  - Errors

## Exception handling

# Checked Exceptions

- It is an exception that occurs at the **compile time**, these are also called as compile-time exceptions.
- These exceptions **cannot** simply be **ignored** at the time of **compilation**, the programmer should take care of (handle) these exceptions.

### Examples:

- IOException
- ClassNotFoundException
- SQLException
- SocketException,
- etc.,

# Unchecked Exceptions

- An unchecked exception is an exception that occurs at **the time of execution**. These are also called as **Runtime Exceptions**.
- These include programming **bugs**, such as **logic errors** or improper use of an API. Runtime exceptions are ignored at the time of compilation.

### Examples:

- ArithmeticException
- NullPointerException,
- ArrayIndexOutOfBoundsException
- etc.,

# Errors

- Errors are **abnormal conditions** that happen in case of severe failures, these are **not handled by the Java programs**.
- Errors are generated to indicate errors generated by the runtime environment.

### Examples:

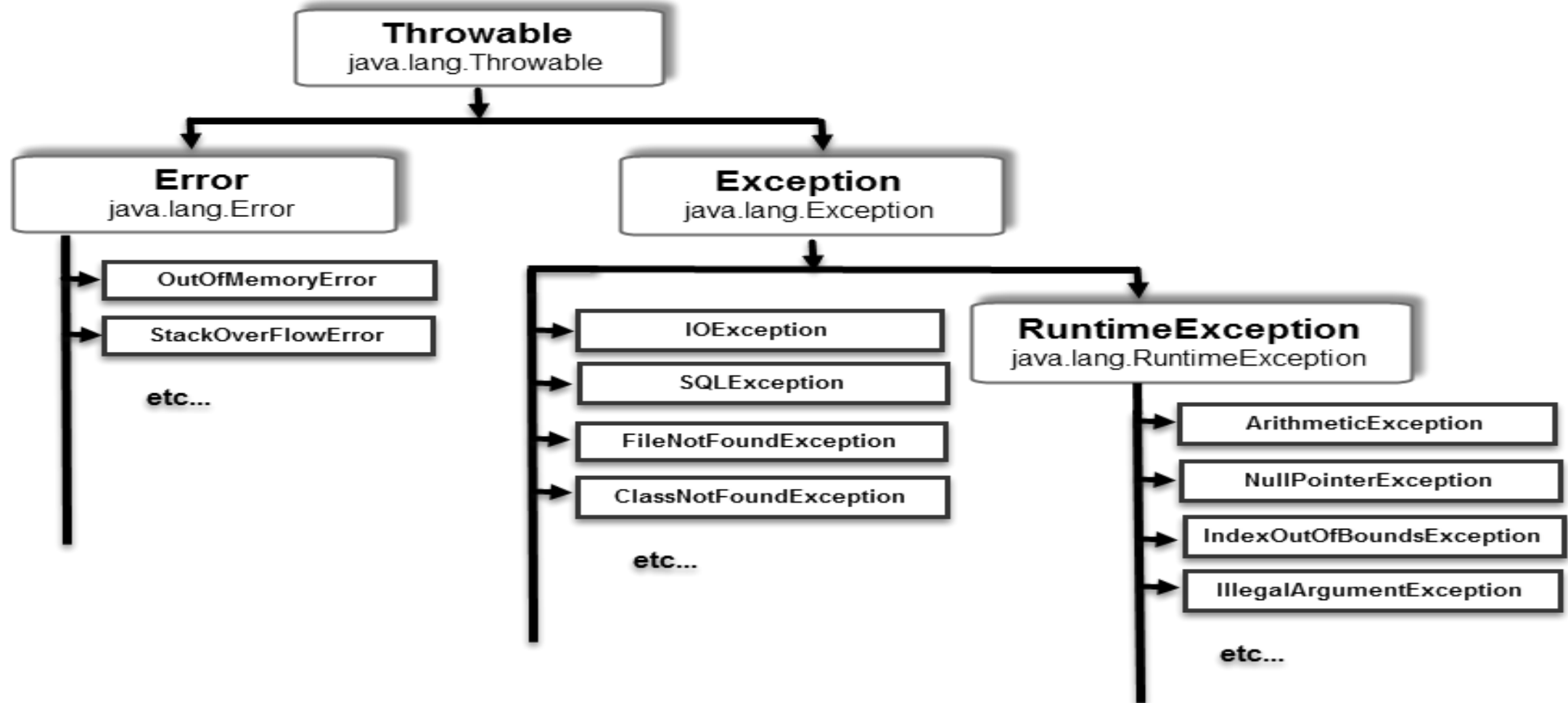
- OutOfMemoryError
- VirtualMachineError
- StackOverflowError
- etc.,

## Exception handling

# Exception handling technique

- There are **several built-in classes** that are used to handle the fundamental errors that may occur in your programs.
- Also, we can create our **own exceptions** based on the application needs by **extending Exception class**.

## Exception Hierarchy



## Exception handling

# Exception Hierarchy

- All exception and errors types are subclasses of class **Throwable**, which is the base class of the hierarchy. There are **three** main types of **Throwable**:

## Error

- Typically, an unrecoverable external error

## RuntimeException

- Typically caused by a programming mistake

## Exception

- Recoverable error

### Note:

- In Java exceptions under **Error** and **RuntimeException** classes are **unchecked** exceptions, **everything else** under throwable is **checked**.

## Exception handling

### Example without handler

```
// Java program to demonstrate how exception is thrown.
class ThrowsExecp{
    public static void main(String args[]){
        String str = null;
        System.out.println(str.length());
        . . .
    }
}
```

#### **Output:**

Exception in thread "main"  
java.lang.NullPointerException at  
ThrowsExecp.main(File.java:8)

In the above example, program will be terminated abruptly and the rest of codes will not get executed due to exception



## Exception handling in Java

# Exception handling Mechanism

Below are the **steps** to handle the exceptions

**Step 1:** Identify the problem (**Hit** the Exception)

**Step 2:** Notify that an error has occurred (**Throw** the Exception)

**Step 3:** Recieve the error notification (**Catch** the Exception)

**Step 4:** Take corrective actions (**Handle** the Exception)

### Note:

First two steps are taken care by **Try** block and rest of the steps taken care by **Catch** block.

## Exception handling

# How JVM handle an Exception?

- Whenever inside a method, if an exception has occurred, the method creates an Object known as **Exception Object** and hands it off to the run-time system(JVM).
- The exception object contains **name and description of the exception**, and current state of the program where exception has occurred.
- Creating the Exception Object and handling it to the run-time system is called **throwing an Exception**.
- There could be a list of methods that were called to obtain the method in which the exception took place.
- This list of the methods is called as **Call Stack**.

# Procedure for Exception Handling

- The run-time system searches the call stack to find the method that contains block of code that can handle the occurred exception. The block of the code is called **Exception handler**.
- The run-time system starts searching from the method in which exception occurred, proceeds through **call stack in the reverse order** in which methods were called.
- If it finds **appropriate handler** (the type of the exception object thrown matches the type of the exception object it can handle) then it passes the occurred exception to it.
- If couldn't have found the appropriate handler then run-time system handover the Exception Object to **default exception handler** , which is part of **run-time system**. This handler prints the exception information and terminates program **abnormally**.

## Exception handling

# Exception handling: Keywords

| Keyword | Description  |
|---------|--|
| try     | to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.  |
| catch   | to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.             |
| finally | The to execute the important code of the program. It is executed whether an exception is handled or not.   |
| throw   | to throw an exception.   |
| throws  | to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |

## Exception handling

# Try-catch block

```
try{  
    ....  
    -- code which may cause an exception  
    ....  
}  
  
catch(ExceptionType e){  
    ...  
    -- Code to handle the exception  
    ...  
}
```

## Exception handling

### Example with handler: Example #1

```
// Java program to demonstrate exception handling.
class ThrowsExcep{
    public static void main(String args[]){
        try{
            String str = null;
            System.out.println(str.length());
        }
        catch(NullPointerException e){
            System.out.println(e);
        }
        System.out.println("rest of the code");//rest of the code
    }
}
```

## Exception handling

### Example with handler : Example #2

```
// Java program to demonstrate exception handling.

public class ExcepDemo{
    public static void main(String args[]) {
        try {
            int a[] = new int[2];
            System.out.println("Access element three : " + a[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown : " + e);
        }
        System.out.println("Out of the block");
    }
}
```

## Printing Exception Messages

- 3 **different ways** to print Exception messages in Java
- **java.lang.Throwable.printStackTrace()** method :we can print the name(e.g. java.lang.ArithmeticException), description(e.g. / by zero) of an exception separated by colon and stack trace (where that exception has occurred) in the next line.

**e.printStackTrace();**

- **toString()** method :we can print only name and description of an exception

**System.out.println(e.toString()); (or) System.out.println(e);**

- **java.lang.Throwable.getMessage()** method : we can print only description of an exception.

**System.out.println(e.getMessage());**



## Exception handling

### Multiple catch statements

- A single block of code can raise more than one exception
- You can specify two or more **catch clauses**, each can catch different type of exception
- When an exception is thrown, each **catch statement is inspected in order**, and the first one whose type matches that of the exception is executed
- After one **catch statement executes**, the others are bypassed, and execution continues after the **try/catch block**
- It is mandatory to handle the exceptions according to their inheritance hierarchy.

## Exception handling

# Multiple catch statements

```
try {  
    // block of code to monitor for errors  
    // the code you think can raise an exception  
}  
catch (ExceptionType1 exOb) {  
    // exception handler for ExceptionType1  
}
```

## Exception handling

# Multiple catch statements

```
catch (ExceptionType2 exOb) {  
    // exception handler for ExceptionType2  
}  
  
catch (ExceptionType3 exOb) {  
    // exception handler for ExceptionType3  
}
```

## Exception handling

### Multiple catch block: Example #1

```
// Java program to demonstrate multiple catch block.  
public class ExceptionDemo{  
    public static void main(String args[]){  
        try{  
            int a[]=new int[5];  
            a[5]=30/0;  
        }  
        catch(ArithmeticException e){  
            System.out.println(e);  
        }  
    }  
}
```

## Exception handling

### Multiple catch block: Example #1

```
        catch(ArrayIndexOutOfBoundsException e) {  
            System.out.println(e);  
        }  
        catch(Exception e) {  
            System.out.println(e);  
        }  
        System.out.println("rest of the code...");  
    }  
}
```

## Exception handling

### Multiple catch block: Example #2

```
// Java program to demonstrate multiple catch block.
public class ExceptionDemo{
    public static void main(String args[]){
        try{
            int a[]=new int[5];
            a[5]=30/0;
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

## Exception handling

### Multiple catch block: Example #2

```
        catch(ArithmeticException e){  
            System.out.println(e);  
        }  
        catch(ArrayIndexOutOfBoundsException e) {  
            System.out.println(e);  
        }  
        System.out.println("rest of the code...");  
    }  
}
```

**Output:** Error

#### **Note:**

- At a time only one Exception is occurred and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general i.e. catch for `ArithmeticException` must come before catch for `Exception`.

## Exception handling

### throw keyword

- The **throw** keyword in **Java** is used to explicitly **throw** an exception from a method or any block of code.
- We can throw either checked or unchecked exception by throw keyword.
- The throw keyword is mainly used to throw **user defined exception**.

#### Syntax:

```
throw exception;
```



## Exception handling

### throw keyword: Example #1

```
// Java program to demonstrate the use of throw keyword.
public class ExceptionDemo{
    static void validate(int num){
        if(num<0)
            throw new ArithmeticException("Invalid value");
        else
            System.out.println("Valid to proceed");
    }
}
```

## Exception handling

### throw keyword: Example #1

```
public static void main(String args[]){  
    try{  
        validate(-10);  
    }  
    catch(Exception e){  
        System.out.println("Error:"+e);  
    }  
    System.out.println("rest of the code...");  
}
```

## Exception handling

### throw keyword: Example #2

```
// Java program to demonstrate the use of throw keyword.
class ExceptionDemo{
    static void fun() {
        try{
            throw new NullPointerException("Demo");
        }
        catch(NullPointerException e) {
            System.out.println("Caught inside fun().");
            throw e; // rethrowing the exception
        }
    }
}
```

## Exception handling

### throw keyword: Example #2

```
public static void main(String args[]) {  
    try {  
        fun();  
    }  
    catch(NullPointerException e) {  
        System.out.println("Caught in main.");  
    }  
}
```

## Exception handling

### throws keyword

- throws is a keyword in Java which is used in the **signature of method** to indicate that this method might throw one of the listed type exceptions.
- The caller to these methods has to handle the exception using a try-catch block.
- It provides information to the caller of the method about the exception.
- To handle the exception when you call this method, all the exceptions that are declared using throws, must be handled where you are calling this method else you will get a compilation error.

#### Syntax:

```
type method_name(parameters) throws exception_list
```

## Exception handling

### throws keyword: Example #1

```
// Java program to demonstrate the use of throws keyword.
class ExceptionDemo{
    static void fun() throws IllegalAccessException {
        System.out.println("Inside fun(). ");
        throw new IllegalAccessException("demo");
    }
    public static void main(String args[]) {
        try{
            fun();
        }
        catch(IllegalAccessException e) {
            System.out.println("caught in main.");
        }
    }
}
```

## Exception handling

### throws keyword: Example #2

```
// Java program to demonstrate the use of throws keyword.  
import java.io.*;  
class ThrowExample {  
    void myMethod(int num)throws IOException, ClassNotFoundException{  
        if(num==1)  
            throw new IOException("IOException Occurred");  
        else  
            throw new ClassNotFoundException("ClassNotFoundException");  
    }  
}
```

## Exception handling

### throws keyword: Example #2

```
public class ExceptionDemo{  
    public static void main(String args[]){  
        try{  
            ThrowExample obj=new ThrowExample();  
            obj.myMethod(1);  
        }  
        catch(Exception ex){  
            System.out.println(ex);  
        }  
    }  
}
```



## Exception handling

### finally block

- finally block is a block that is used to execute **important code** such as closing connection, stream etc.
- It is **guaranteed to be executed** whether the exception is handled or not.
- It follows try or catch block.

**Note:** If you don't handle exception, before terminating the program, JVM executes finally block(if any).

## Exception handling

### finally block: Example #1

```
// Java program to demonstrate finally block.
class ExceptionDemo{
    public static void main(String args[]) {
        try {
            int data=25/5;
            System.out.println(data);
        }
        catch(NullPointerException e) {
            System.out.println(e);
        }
        finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code..."); } }
```

## Exception handling

### finally, keyword: Example #2

```
// Java program to demonstrate finally block.
class ExceptionDemo{
    public static void main(String args[]) {
        try {
            int data=25/0;
            System.out.println(data);
        }
        catch(NullPointerException e) {
            System.out.println(e);
        }
        finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code..."); } }
```

## Exception handling

### finally, keyword: Example #3

```
// Java program to demonstrate finally block.
class ExceptionDemo{
    public static void main(String args[]) {
        try {
            int data=25/0;
            System.out.println(data);
        }
        catch(ArithmeticException e) {
            System.out.println(e);
        }
        finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code..."); } }
```

## Exception handling

# User defined Exceptions

- Java provides extensive set of in-built exceptions
- But there may be cases where we may have to define our own exceptions which are application specific
- We can customize the exception according to our need and purpose.
- While creating user defined exceptions, the following aspects have to be taken care :
  - The user defined exception class should extend from the Exception class and its Subclass.
  - If we want to display meaningful information about the exception, we should override the toString() method

## Exception handling

# User defined Exceptions: Example #1

```
// Java program to demonstrate user defined exception.
class InvalidAgeException extends Exception {
    InvalidAgeException(String s) {
        // Call constructor of parent Exception
        super(s);
    }
}

class ExceptionDemo{
    static void validate(int age)throws InvalidAgeException {
        if(age<18)
            throw new InvalidAgeException("not eligible");
        else
            System.out.println("Eligible");
    }
}
```

## Exception handling

# User defined Exceptions: Example #1

```
public static void main(String args[]) {  
    try {  
        validate(13);  
    }  
    catch(Exception m) {  
        System.out.println("Exception occurred: "+m);  
    }  
  
    System.out.println("rest of the code...");  
}  
}
```

## Exception handling

# User defined Exceptions: Example #2

```
// Java program to demonstrate user defined exception.
class InvalidProductException extends Exception{
    public InvalidProductException(String s) {
        // Call constructor of parent Exception
        super(s);
    }
}

public class ExceptionDemo {
    void productCheck(int weight) throws InvalidProductException{
        if(weight<100){
            throw new InvalidProductException("Product Invalid");
        }
    }
}
```



## Exception handling

# User defined Exceptions: Example #2

```
public static void main(String args[]) {  
    ExceptionDemo obj = new ExceptionDemo();  
    try {  
        obj.productCheck(60);  
    }  
    catch (InvalidProductException ex) {  
        System.out.println("Caught the exception");  
        System.out.println(ex.getMessage());  
    }  
}
```

## Assertions

- It is used for testing the correctness of any assumptions that have been made in the program.
- An assert statement is used to declare an expected boolean condition in a program.
- When the program is running with assertions enabled, the condition is checked at runtime. If the condition is false, the Java runtime system **throws** an **AssertionError**.
- It is mainly used for testing purposes during development.

### Syntax:

***assert*** *expression*; (or)

***assert*** *expression1* : *expression2*;

## Exception handling

# Assertions

- By default, assertions are disabled. We need to run the code as given.

### **Enabling Assertions**

- `java -ea Test` (or) `java -enableassertions Test`

Here, Test is the file name.

### **Disabling Assertions**

- `java -da Test` (or) `java -disableassertions Test`

Here, Test is the file name.

## Exception handling

# Assertions: Example #1

```
// Java program to demonstrate syntax of assertion
class AssertionDemo{
    public static void main( String args[] ) {
        int value = 55;
        assert value >= 60:"Wrong Data";
        System.out.println("value is "+value);
    }
}
```

## Assertions: Example #2

```
// Java program to demonstrate syntax of assertion
import java.util.Scanner;
class AssertionDemo{
    public static void main( String args[] ){
        Scanner scanner = new Scanner( System.in );
        System.out.print("Enter ur age ");
        int value = scanner.nextInt();
        assert value>=18:" You are not Eligible for Vote";
        System.out.println("Your age: "+value);
    }
}
```

## Assertions: Example #3

```
// Java program to demonstrate syntax of assertion
import java.util.Scanner;
class AssertionDemo{
    public void Test(int a){
        assert a>=18:" You are not Eligible for Vote";
        System.out.println("Your age: "+a);
    }
    public static void main( String args[] ) {
        AssertionDemo obj=new AssertionDemo();
        Scanner scanner = new Scanner( System.in );
        System.out.print("Enter your age ");
        int value = scanner.nextInt();
        obj.Test(value);
    }
}
```

# Assertions

### Where to use Assertions

- Arguments to private methods.
- Use assertion in the default case of the Switch statement
- Conditional cases.
- Conditions at the beginning of any method.

### Where not to use Assertions

- Should not be used to replace error messages.
- Do *not* use assertions for argument checking in public methods
- Avoid catching the assertion-related exceptions.
- Should not be used on command line arguments.
- Avoid the use of evaluating more than one condition in an assertion

## Quiz



1.Exception is a class/interface/abstract class/other?

a) Class

b) Interface

c) Abstract Class

d) None of the above

a) Class



## Quiz



2. Exception is found in which package in java

a) java.lang

b) java.util

c) java.io

d) java.awt.

e) None of the above

a) java.lang

### Quiz



3. Which is valid about java.lang.Exceptions?

a) The class Exception and all its subclasses that are not also subclasses of RuntimeException are checked exceptions

b) The class Error and all its subclasses are unchecked exceptions

c) The class RuntimeException and all its subclasses are unchecked exceptions

d) All the above

d) All the above

## Exception handling

### Quiz



4. Which of these is valid code snippet in exception handling in java?

a) `catch{ }`

b) `finally{ }`

c) `try{ }`  
`catch(Exception e){ }`  
`finally{ }`

d) `try{ }.`

c) `try{ }`  
`catch(Exception e){ }`  
`finally{ }`

## Exception Handling

### Quiz



**5. Which is invalid statement in Exception handling in java?**

a) finally block can't throw exception

b) try block can throw exception

c) catch block can throw exception

d) finally block can throw exception.

e) None of the above

a) finally block can't throw exception

”

An investment in  
knowledge pays the  
best interest.

**- Benjamin Franklin**

# THANK YOU