Java

Module 5

Exception Handling

Author
Srinivas Dande

My JAVA LEARNING CENTER

5.1 Introduction to Exception Handling

- An **exception** is an event that occurs during the execution of a program and it interrupts the normal flow of program execution.
- Some common problems which may cause exception:
  - Creating array object with negative size.
  - Accessing index of array which is not available.
  - Dividing an integer value with zero.
  - Invoking instance members with null reference.
  - Recursive method invocation without conditional check.

    etc.

Q) What will happen if you are compiling and executing the following code?

Lab627.java

```
class Lab627{
public static void main(String args[]){
System.out.println("Main Begin");
String str=args[0];
int a=Integer.parseInt(str);
int x=10/a;
System.out.println("x = "+x);
System.out.println("Main End");
}
}
```

Execute Lab As:

Case 1: java Lab627

Case 2: java Lab627 JLC

Case 3: java Lab627 0

Case 4: java Lab627 10

Case 1:

java Lab627 (When value is not provided from Command Line)

```
Main Started
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
```

Case 2:

java Lab627 JLC (When String is provided as Command Line Argument)

Main Started
Exception in thread "main" java.lang.NumberFormatException: For input string: "JLC"

Case 3:

java Lab627 0 (When 0 is provided as Command Line Argument)

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

Case 4:

java Lab627 10 (When 10 is provided as Command Line Argument)

Main Started
Result is: 1
Main End

What Happens When you run the Program

A) java Hello 0

B) java Hello 10

i. JVM will be Initialized
ii. JVM Creates and Starts the Main Thread.
iii. Main Thread will do the following tasks
    1) Collects the Command Line Arguments
    2) Creates the String array with CLA
    3) Calls main() method by passing String array as Parameter.

        Hello.main(str);

Now Control will be transfered from main Thread to main() method

Control will come back to main thread from main() methid in two ways.

A) When problem comes in main() method

B) When main() method is executed successfully.

A) When Control returns abnormally with Problem then following Tasks will happen.

A1) Checks Problem

A2) Identifies the Exception class realated to Problem

A3) Creates the Object of Identified Exception class.

ArithmeticExcetpion ae=new ArithmeticExcetpion()

A4) Throws the Exception Object

throw ae

A5) If main() method is catching the Exception thrown then controll will be return to main() method. so that main() method execution will continue.

A6) If main() method is not catching the Exception thrown then main Thread only handles the Exception by displaying Error message.

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

at Hello.main(Hello.java:6)

A7) main Thread will be terminated.

B) When main() method is executed successfully then main Thread will be terminated.

iv. JVM shutdown process will be initialized.

5.2 Problem Types

Problem occurred while executing the statement can be divided into two types.

1. Error
2. Exception

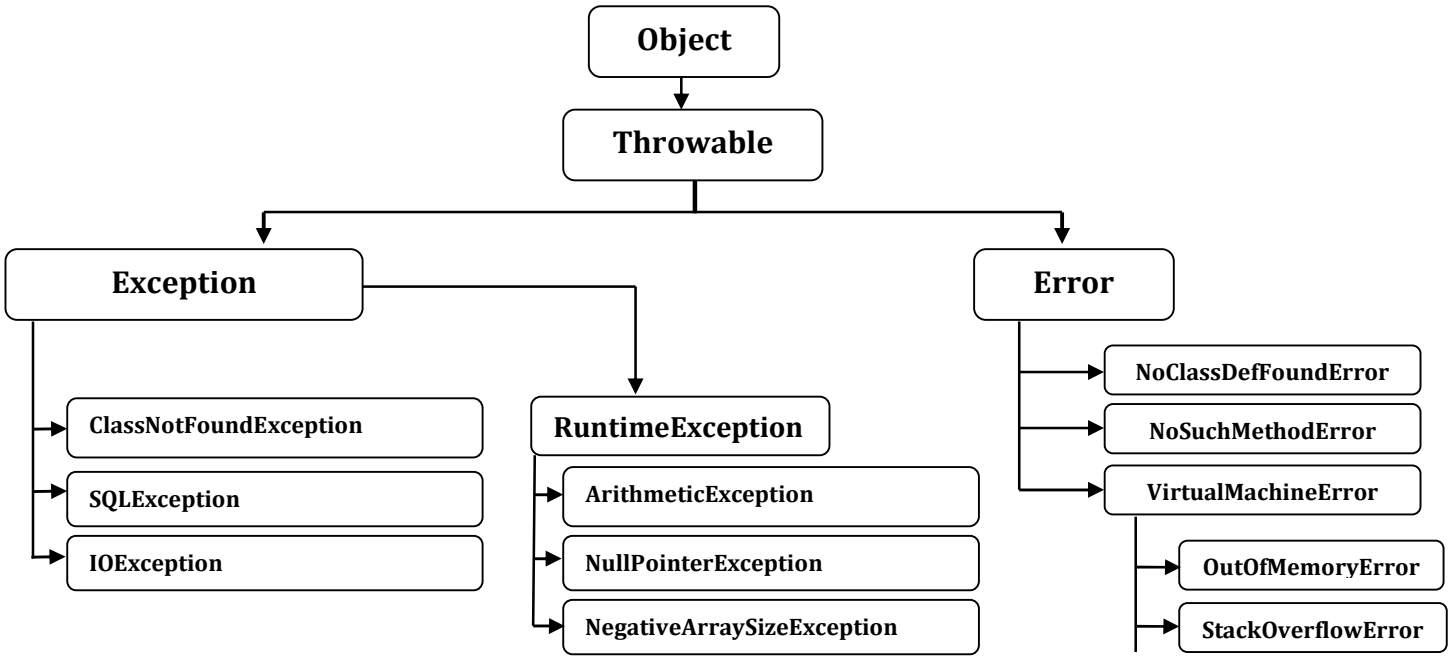
5.2.1 Error

- It is a type of problem that will not be handled and the execution of the application will be terminated.
- Depending on the reason, various classes are defined for various errors and all the error classes are subclasses of **java.lang.Error** class.
- Following are some sub classes of java.lang.Error class
    - NoClassDefFoundError
    - NoSuchMethodError
    - VirtualMachineError
    - OutOfMemoryError
    - StackOverflowError

      etc

5.2.2 Exception

- It is a type of event that can be handled and the other statements of the application can be executed successfully.
- Depending on the reason, various classes are defined for various exceptions and all the exception classes are subclasses of **java.lang.Exception** class.



5.3 Handling Exception in Program

- When JVM handles the exception using default exception handler then program execution will be terminated abnormally after displaying the message.
- When you want the normal termination of the program after executing all required statements then you have to handle the execptions.
- You can use the following two keywords to handle the execptions:
  - **try**
  - **catch**

try and catch blocks

- Try block should be followed by zero or more catch blocks.
- When you expect the problems with any Java statements then place those statements inside the try block.
- When exception is raised with try block statement then control will be transferred to the corresponding catch block.
- Catch block should contain the statement to handle the exception raised in the try block statements.

Syntax 1:

```
try{
        // Statements
}catch(<throwableType> <refVar>){
        // Statements
}
```

Syntax 2:

```
try{
        // Statements
}catch(<throwableType1> <refVar>){
        // Statements
}catch(<throwableType2> <refVar>){
        // Statements
}
```

Lab628.java

```
class Lab628{
public static void main(String args[]){
System.out.println("Main Begin");
try{
String str=args[0];
int a=Integer.parseInt(str);
int x=10/a;
System.out.println("x = "+x);
}catch(Exception ex){
System.out.println("Hey, Provide Correct
Input");
}
System.out.println("Main End");
}
}
```

Lab629.java

```
class Lab629{
public static void main(String args[]){
System.out.println("Main Begin");
 try{
String str=args[0];
int a=Integer.parseInt(str);
int x=10/a;
System.out.println("x = "+x);
}catch(ArrayIndexOutOfBoundsException ex){
System.out.println("Hey, Provide Input");
}
catch(NumberFormatException ex){
System.out.println("Hey, Provide Number Only");
}
catch(ArithmeticException ex){
System.out.println("Hey, Provide Non-Zero
Number");
}
System.out.println("Main End");
}
}
```

Lab630.java

```
class Lab630{
public static void main(String args[]){
System.out.println("Main Begin");
 try{
String str=args[0];
int a=Integer.parseInt(str);
int x=10/a;
System.out.println("x = "+x);
}catch(ArrayIndexOutOfBoundsException ex){
System.out.println("Hey, Provide Input");
}
catch(NumberFormatException ex){
System.out.println("Hey, Provide Number
Only");
}
System.out.println("Main End");
}
}
```

Lab631.java

```
class Lab631{
public static void main(String args[]){
System.out.println("Main Begin");
 try{
String str=args[0];
int a=Integer.parseInt(str);
int x=10/a;
System.out.println("x = "+x);
}catch(Exception ex){
System.out.println("Hey, Hello ooooooo");
}
catch(ArrayIndexOutOfBoundsException ex){
System.out.println("Hey, Provide Input");
}
catch(NumberFormatException ex){
System.out.println("Hey, Provide Number Only");
}
System.out.println("Main End");
} }
```

Lab632.java

```
class Lab632{
public static void main(String args[]){
System.out.println("Main Begin");
 try{
String str=args[0];
int a=Integer.parseInt(str);
int x=10/a;
System.out.println("x = "+x);
}
catch(ArrayIndexOutOfBoundsException ex){
System.out.println("Hey, Provide Input");
}
catch(NumberFormatException ex){
System.out.println("Hey, Provide Number
Only");
}
catch(Exception ex){
System.out.println("Hey, Hello ooooooo");
}
System.out.println("Main End");
}
}
```

Lab633.java

```
class Lab633{
public static void main(String args[]){
System.out.println("Main Begin");
try{
String str=args[0];
System.out.println(str);
}
catch(String str){
System.out.println("Hey, Hello ooooooo");
}

System.out.println("Main End");
}
}
```

Lab634.java

```
class Lab634{
public static void main(String args[]){
System.out.println("Main Begin");

try{
String str=args[0];
System.out.println(str);
}

System.out.println("Main End");
}
}
```

Lab635.java

```
class Lab635{
public static void main(String args[]){
System.out.println("Main Begin");

try{
int x=10/0;
System.out.println(x);
}
catch(NumberFormatException ex){
System.out.println("Hey, Provide Number Only");
}

System.out.println("Main End");
}
}
```

Lab636.java

```
class Lab636{
public static void main(String args[]){
System.out.println("Main Begin");
try{
int x=10/0;
System.out.println(x);
}
catch(ArithmeticException ex){
System.out.println("Hey, Dont Divide with
Zero");
}
System.out.println("Main End");
}
}
```

Lab637.java

```
class Lab637{
public static void main(String args[]){
System.out.println("Main Begin");
try{
int x=10/0;
System.out.println(x);
}
System.out.println("Hello Guys!!!");
catch(ArithmeticException ex){
System.out.println("Hey, DOnt Divide with Zero");
}
System.out.println("Main End");
}
}
```

Lab638.java

```
class Lab638{
public static void main(String args[]){
System.out.println("Main Begin");

try{
int x=10/0;
System.out.println(x);
}
catch(ArithmeticException ex){
System.out.println("Hey, DOnt Divide with
Zero");
}
System.out.println("Hello Guys!!!");
catch(Exception ex){
System.out.println("Hey, Hellooooooooo");
}

System.out.println("Main End");
}
}
```

Lab639.java

```
class Lab639{
public static void main(String args[]){
System.out.println("Main Begin");

try{
int x=10/0;
System.out.println(x);
}
catch(ArithmeticException ex){
System.out.println("Hey, DOnt Divide with Zero");
}
catch(ArithmeticException ex){
System.out.println("Hey, Hellooooooooo");
}

System.out.println("Main End");
}
}
```

Lab640.java

```
class Lab640{
public static void main(String args[]){
System.out.println("Main Begin");

String str="";
try{
str=args[0];
}catch(ArrayIndexOutOfBoundsException ex){
System.out.println("Hey, Provide Input");
}

int a=0;
try{
a=Integer.parseInt(str);
}
catch(NumberFormatException ex){
System.out.println("Hey, Provide Number
Only");
}

try{
int x=10/a;
System.out.println("x = "+x);
}
catch(ArithmeticException ex){
System.out.println("Hey, Provide Non-Zero
Number");
}

System.out.println("Main End");
}
}
```

Lab640.java

```
class Lab640{
public static void main(String args[]){
System.out.println("Main Begin");

String str="";
try{
str=args[0];
}catch(ArrayIndexOutOfBoundsException ex){
System.out.println("Hey, Provide Input");
}

int a=0;
try{
a=Integer.parseInt(str);
}
catch(NumberFormatException ex){
System.out.println("Hey, Provide Number Only");
}

try{
int x=10/a;
System.out.println("x = "+x);
}
catch(ArithmeticException ex){
System.out.println("Hey, Provide Non-Zero
Number");
}

System.out.println("Main End");
}
}
```

5.4 Catching Multiple Exception Types

- It is new features from Java 7.
- Using this, single catch block can handle more than one type of exceptions.
- It can reduce code duplication.
- When you are writing multiple exception in one catch block then
  - Multiple exceptions should be unique
  - Multiple exceptions should not have any Inheritance relationship.

Lab641.java

```
class Lab641{
public static void main(String args[]){
System.out.println("Main Begin");
try{
String str=args[0];
int a=Integer.parseInt(str);
int x=10/a;
System.out.println("x = "+x);
}catch(ArrayIndexOutOfBoundsException |
NumberFormatException |
ArithmeticException ex){
System.out.println("Hey, Hello Huys");
}
System.out.println("Main End");
}
}
```

Lab642.java

```
class Lab642{
public static void main(String args[]){
System.out.println("Main Begin");
try{
String str=args[0];
int a=Integer.parseInt(str);
int x=10/a;
System.out.println("x = "+x);
}catch(ArrayIndexOutOfBoundsException |
NumberFormatException ex){
System.out.println("Hey, Hello Huys");
}
catch(ArithmeticException ex){
System.out.println("Hey, Dont Provide Zero");
}
System.out.println("Main End");
}
}
```

Lab643.java

```
class Lab643{
public static void main(String args[]){
System.out.println("Main Begin");
try{
int x=10/0;
System.out.println("x = "+x);
}catch(ArithmeticException |
ArithmeticException ex){
System.out.println("Hey, Dont Divide with
Zero");
}
System.out.println("Main End");
}
}
```

Lab644.java

```
class Lab644{
public static void main(String args[]){
System.out.println("Main Begin");
try{
int x=10/0;
System.out.println("x = "+x);
}catch(Exception | ArithmeticException ex){
System.out.println("Hey, Dont Divide with Zero");
}
System.out.println("Main End");
}
}
```

Lab645.java

```
class Lab645{
public static void main(String args[]){
System.out.println("Main Begin");

try{
int x=10/0;
System.out.println("x = "+x);
}catch(ArithmeticException | Exception ex){
System.out.println("Hey, Dont Divide with
Zero");
}

System.out.println("Main End");
}
}
```

5.5 finally Block

- To ensure that all the statements of a program gets executed we are handling the exeception by writing try and catch block.
- There are some cases in which statemenets after the catch block will not be executed:
  - When you have return statement inside the try or catch block.
  - When exeception occurs in try block and matching catch block is not available.
  - When exeception occurs in catch block.

  etc
- When you want to execute statements after the catch block always without fail then you have to place those statements inside the finally block.

Syntax 1:

```
try{
        // Statements
}catch(<throwableType1> <refVar>){
        // Statements
}
```

Syntax 2:

```
try{
        // Statements
}finally{
        // Statements
        }
```

Syntax 3:

```
try{
        // Statements
}catch(<throwableType1> <refVar>){
        // Statements
}finally{
        // Statements
}
```

Syntax 4:

```
try{
        // Statements
}catch(<throwableType1> <refVar>){
        // Statements
} catch(<throwableType2> <refVar>){
        // Statements
}
finally{
        // Statements
}
```

Lab646.java

```
class Lab646{
public static void main(String args[]){
System.out.println("Main Begin");

try{
int x=10/0;
System.out.println("x = "+x);
}catch(NumberFormatException ex){
System.out.println("Hey, Dont Divide with
Zero");
}

System.out.println("Hello Guys");      //IMP
System.out.println("OK Ok");    //IMP

System.out.println("Main End");
}
}
```

Lab647.java

```
class Lab647{
public static void main(String args[]){
System.out.println("Main Begin");

try{
int x=10/0;
System.out.println("x = "+x);
}catch(NumberFormatException ex){
System.out.println("Hey, Dont Divide with Zero");
}
finally{
System.out.println("Hello Guys");      //IMP
System.out.println("OK Ok");    //IMP
}

System.out.println("Main End");
}
}
```

Lab648.java

```
class Lab648{
public static void main(String args[]){
System.out.println("Main Begin");

try{
int x=10/2;
System.out.println("x = "+x);
return;
}catch(NumberFormatException ex){
System.out.println("Hey, Dont Divide with
Zero");
}

System.out.println("Hello Guys");      //IMP
System.out.println("OK Ok");    //IMP

System.out.println("Main End");
}
}
```

Lab649.java

```
class Lab649{
public static void main(String args[]){
System.out.println("Main Begin");

try{
int x=10/2;
System.out.println("x = "+x);
return;
}catch(NumberFormatException ex){
System.out.println("Hey, Dont Divide with Zero");
}
finally{
System.out.println("Hello Guys");      //IMP
System.out.println("OK Ok");    //IMP
}

System.out.println("Main End");
}
}
```

Lab650.java

```
class Hello{
int show(int a){
int x=0;
try{
System.out.println("Try Begin");
x=10/a;
System.out.println("x = "+x);
System.out.println("Try End");
return x;
}catch(ArithmeticException ex){
x=20;
System.out.println("Hey, Dont Divide with
Zero");
return x;
}
finally{
System.out.println("Hello Guys");
System.out.println("OK Ok");
}
}
}
```

```
class Lab650{
public static void main(String args[]){
System.out.println("Main Begin");

Hello h= new Hello();
int a=2;
//int a=0;
int result =h.show(a);
System.out.println("result = "+result);

System.out.println("Main End");
}
}
```

Lab652.java

```
class Hello{
void show(int a){
try{
System.out.println("Try Begin");
int x=10/a;
System.out.println("x = "+x);
System.out.println("Try End");
System.exit(0);
}catch(ArithmeticException ex){
System.out.println("Hey, Dont Divide with
Zero");
}
finally{
System.out.println("Hello Guys");      //IMP
System.out.println("OK Ok");    //IMP
}
}
}

class Lab652{
public static void main(String args[]){
System.out.println("Main Begin");

Hello h= new Hello();
int a=2;
h.show(a);

System.out.println("Main End");
}
}
```

Lab653.java

```
class Hello{
void show(int a){
try{
System.out.println("Try Begin");
int x=10/a;
System.out.println("x = "+x);
System.out.println("Try End");
}catch(ArithmeticException ex){
System.out.println("Hey, Dont Divide with Zero");
System.exit(0);
}
finally{
System.out.println("Hello Guys");      //IMP
System.out.println("OK Ok");    //IMP
}
}
}

class Lab653{
public static void main(String args[]){
System.out.println("Main Begin");

Hello h= new Hello();
int a=0;
h.show(a);

System.out.println("Main End");
}
}
```

5.6 Nested try-catch-finally Blocks

- You can write try-catch-finally inside try block,catch block and finally block.

Sample Code:

```
public class Test{
	public static void main(String[] args) {
		System.out.println("Stmt 1");
		try {
			System.out.println("Stmt 2");
			try {
				System.out.println("Stmt 3");
			} catch (Exception e) {
				System.out.println("Stmt 4");
			} finally {
				System.out.println("Stmt 5");
			}
			System.out.println("Stmt 6");
		} catch (Exception e) {
			System.out.println("Stmt 7");
			try {
				System.out.println("Stmt 8");
			} catch (Exception e1) {
				System.out.println("Stmt 9");
			} finally {
				System.out.println("Stmt 10");
			}
			System.out.println("Stmt 11");
		} finally {
			System.out.println("Stmt 12");
			try {
				System.out.println("Stmt 13");
			} catch (Exception e) {
				System.out.println("Stmt 14");
			} finally {
				System.out.println("Stmt 15");
			}
			System.out.println("Stmt 16");
		}
		System.out.println("Stmt 17");
	}
}
```

5.7 Methods from Throwable class

Methods from Throwable class	
Methods	Description
public void String getMessage();	Access the message available with Exception
public Throwable getCause();	Access the reason of the exception if available
public void printStackTrace()	Print the stack trace to the default output device (CONSOLE)
public void printStackTrace(PrintStream)	Print the stack trace to the specified PrintStream (File / Network etc)
public void printStackTrace(PrintWriter)	Print the stack trace to the specified PrintWriter (File / Network etc)

Lab654.java

```
class Lab654{
public static void main(String args[]){
System.out.println("Main Begin");
try{
Hello h= new Hello();
h.show();
}catch(Exception ex){
System.out.println("1. "+ex);
System.out.println("2. "+ex.getMessage());
System.out.println("3. Stack Trace------->");
ex.printStackTrace();
}
System.out.println("Main End");
}
}

class Hello{
void show(){
System.out.println("Show Begin");
new A().m1();
System.out.println("Show End");
}
}
```

```
class A{
void m1(){
System.out.println("m1 Begin");
new B().m2();
System.out.println("m1 End");
}
}

class B{
void m2(){
System.out.println("m2 Begin");
new C().m3();
System.out.println("m2 End");
}
}

class C{
void m3(){
System.out.println("m3 Begin");
  int x=10/0;
System.out.println("m3 End");
}
}
```

5.8 Types of Exceptions

- Depending on whether the compiler is verifying the exception at compile time or not, exceptions are divided into two types.
    1) Checked Exceptions
    2) Unchecked Exceptions

5.8.1 Checked Exceptions

- These are also called as Compile time exceptions.
- All the subclasses of java.lang.Exception except java.lang.RuntimeException and its subclasses are checked exceptions.
- If you have any Java statement that may cause any exception and that exception is verified by the compiler by forcing you to report the exception then those exceptions are called as checked exceptions.
- Checked exceptions must be reported in any one of the following forms
    - Handling the exceptions with try-catch.
    - Propagating the exception using throws.
- If you are not reporting checked exception then you will get the following error message at compile time:
    - unreported exception <ExceptionName>; must be caught or declared to be thrown

Lab655.java

```
class HelloException extends Exception{
}
class Hello{
void show() throws HelloException {
HelloException  ex = new HelloException();
throw ex;
}
}

class Lab655{
public static void main(String args[]){
System.out.println("Main Begin");
Hello h=new Hello();
h.show();
System.out.println("Main End");
}
}
```

5.8.2 Unchecked Exceptions

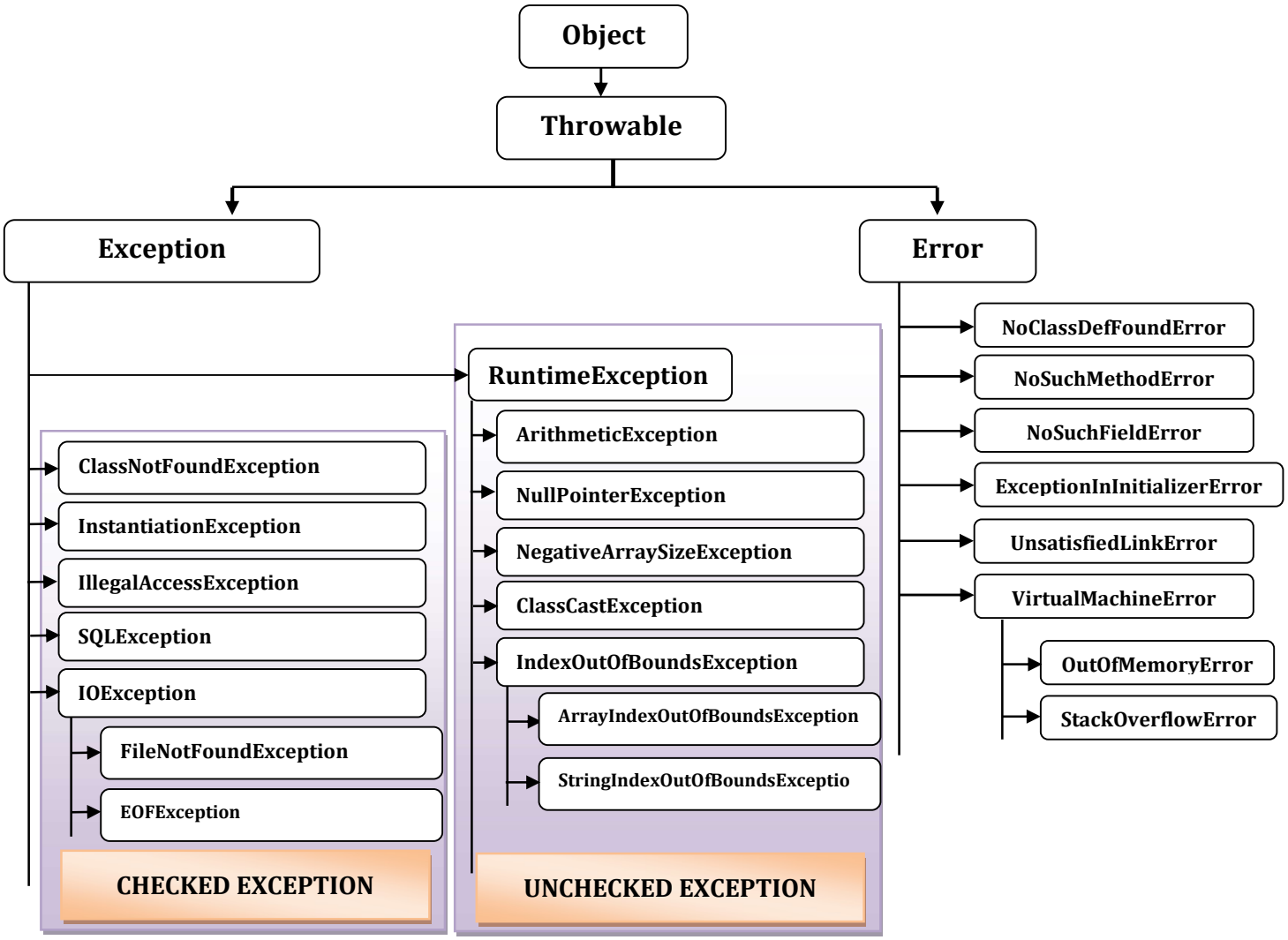
- These are also called as Runtime time exceptions.
- java.lang.RuntimeException and its subclasses are unchecked exceptions.
- If you have any Java statement that may cause any exception and that exception is not verified by the compiler at compile time then those exceptions are called as unchecked exceptions.
- In the case of unchecked exception compiler is not responsible to verify whether you are reporting about exception or not i.e Unchecked exceptions may or may not be reported.

Lab656.java

```
class HaiException extends RuntimeException{
}

class Hai{
void show() throws HaiException {
HaiException ex = new HaiException();
throw ex;
}
}

class Lab656{
public static void main(String args[]){
System.out.println("Main Begin");
Hai h=new Hai();
h.show();
System.out.println("Main End");
}
}
```


Object
Throwable
Exception
Error
RuntimeException
ClassNotFoundException
InstantiationException
IllegalAccessException
SQLException
IOException
FileNotFoundException
EOFException
CHECKED EXCEPTION
ArithmeticException
NullPointerException
NegativeArraySizeException
ClassCastException
IndexOutOfBoundsException
ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsExceptio
UNCHECKED EXCEPTION
NoClassDefFoundError
NoSuchMethodError
NoSuchFieldError
ExceptionInInitializerError
UnsatisfiedLinkError
VirtualMachineError
OutOfMemoryError
StackOverflowError

- **To verify that exception is checked or unchecked**
  - If Compiler will force to handle
  - By verifying the Hierarchy of the class
  - By Using the instanceof with RuntimeException class

Lab657.java

```
class HaiException extends RuntimeException{ }
class HelloException extends Exception{ }

class Lab657{
public static void main(String args[]){
Exception ex1=new HaiException();
Exception ex2= new HelloException();

System.out.println(ex1 instanceof RuntimeException);
System.out.println(ex2 instanceof RuntimeException);
}
}
```

Lab658.java

```
class Lab658{
public static void main(String args[]){

try{
int x=10/0;
}catch(NullPointerException ex){
System.out.println("Hey, Some Problem");
}

}
}
```

Lab659.java

```
import java.sql.*;

class Lab659{
public static void main(String args[]){

try{
int x=10/0;
}catch(SQLException ex){
System.out.println("Hey, Some Problem");
}
}  }
```

Lab660.java

```
class Lab660{
public static void main(String args[]){
try{
//
}catch(NullPointerException ex){
System.out.println("Hey, Some Problem");
}
}
}
```

Lab661.java

```
import java.io.*;
class Lab661{
public static void main(String args[]){
try{
//
}catch(IOException ex){
System.out.println("Hey, Some Problem");
}
} }
```

5.9 throw keyword

- throw is a keyword.
- It is used to throw the exceptions explicitly.
- You can throw any checked or unchecked exceptions.
- You can throw any Built-in or user-defined exceptions.

Syntax

```
throw <throwableTypeObjectRef>
```

Ex:

```
throw new NullPointerException();
throw new InvalidAccountNumberException();
throw new StudentNotFoundException ("JLC-099");
etc
```

5.10 throws keyword

- throws is a keyword.
- throws is used to propagate the exceptions to the caller method by specifying at method level.
- You can define any checked or unchecked exceptions at method level.
- You can define any Built-in or user defined exceptions at method level.
- When the exception is unchecked exception then throws keyword is optional, but for the checked exception throws keyword is mandatory.

Lab662.java

```
import java.io.*;
class C{
void m3() throws IOException  {
System.out.println("m3 Begin");
if(1==1)
throw new IOException();
System.out.println("m3 End");
}
}
class B{
void m2() throws IOException {
System.out.println("m2 Begin");
new C().m3();
System.out.println("m2 End");
}
}
class A{
void m1() throws IOException {
System.out.println("m1 Begin");
new B().m2();
System.out.println("m1 End");
}
}
class Hello{
void show(){
System.out.println("Show Begin");
try{
new A().m1();
}catch(Exception ex){
System.out.println("1. "+ex);
}
System.out.println("Show End");
}
}
class Lab662{
public static void main(String args[]){
System.out.println("Main Begin");
Hello h= new Hello();
h.show();
System.out.println("Main End");
}
}
```

Lab663.java

```
class C{
void m3()  {
System.out.println("m3 Begin");
if(1==1)
throw new ArithmeticException();
System.out.println("m3 End");
}
}
class B{
void m2()  {
System.out.println("m2 Begin");
new C().m3();
System.out.println("m2 End");
}
}
class A{
void m1()  {
System.out.println("m1 Begin");
new B().m2();
System.out.println("m1 End");
}
}
class Hello{
void show(){
System.out.println("Show Begin");
try{
new A().m1();
}catch(Exception ex){
System.out.println("1. "+ex);
}
System.out.println("Show End");
}
}
class Lab663{
public static void main(String args[]){
System.out.println("Main Begin");
Hello h= new Hello();
h.show();
System.out.println("Main End");
}
}
```

5.11 Types of Exceptions

1) Built-in Exceptions
2) User-defined Exceptions

5.11.1 Built-in Exceptions

- The exceptions which are already defined, implemented and provided in Java or other technologies are called as Built-in Exceptions.
  Ex:
  ArithmeticException
  SQLException
  IOException
  ServletException etc

5.11.2 User-defined Exceptions

- The exceptions which are defined and implemented by Application developer as per project requirements are called as User-defined Exceptions.
  Ex:
  StudentNotFoundException
  InvalidAccountNoException
  InsufficientFundsException etc

Steps to define User-define Exceptions

- Write a Java class by extending either java.lang.Exception or java.lang.RuntimeException.
  - If you are extending java.lang.Exception then your custom exception will be checked exception.
  - If you are extending java.lang.RuntimeException then the custom exception will be unchecked exception.

- If you want to store some data related to the exception then
  - You define instance variables.
  - You define constructors.

- You can override the following method from java.lang.Throwable class.
  - public String getMessage()

- You can override the following method from java.lang.Object class.
  - public String toString()

Ex:

1) User-defined Checked Exception

```
public class StudentNotFoundException extends Exception {
        String sid;
        StudentNotFoundException (String sid){
        this.sid = sid;
        }

        //Override getMessage() method
        //Override toString() method

}
```

2) User-defined Un-Checked Exception

```
public class StudentNotFoundException extends RuntimeException {
        String sid;
        StudentNotFoundException (String sid){
        this.sid = sid;
        }

        //Override getMessage() method
        //Override toString() method

}
```

Lab664.java

```
class InsufficientFundsException extends RuntimeException{

public String toString(){
return  "InsufficientFundsException : "+ getMessage();
}
public String getMessage() {
return  "Sorry, No Funds";
}
}

class InvalidAccountNumberException extends Exception{

int accno;
InvalidAccountNumberException(){ }
InvalidAccountNumberException(int accno){
this.accno=accno;
}
public String toString(){
String str = "InvalidAccountNumberException : "+ getMessage();
return str;
}
public String getMessage() {
String msg = accno + " is Invalid Account Number";
return msg;
}
}

class AccountService{

double bal = 50000;
int accno = 150;

double getBalance(int accno) throws InvalidAccountNumberException {
if(accno==150){
return bal;
}else{
throw new InvalidAccountNumberException(accno);
}
}
```

```
double deposit(int accno,double amount) throws InvalidAccountNumberException {
if(accno==150){
bal = bal + amount;
return bal;
}else{
throw new InvalidAccountNumberException(accno);
}
}
double withdraw(int accno, double amount) throws InvalidAccountNumberException
{
if(accno==150){
   if(amount<=bal){
      bal = bal - amount;
     return bal;
    }else{
        throw new InsufficientFundsException();
   }
}else{
throw new InvalidAccountNumberException(accno);
}
}  }
class Lab664{
public static void main(String args[]){
AccountService as = new AccountService();
System.out.println("--1.getBalance()-------");
          try{
          double bal = as.getBalance(150);
          System.out.println(bal);
          }catch(Exception ex){
          System.out.println(ex.getMessage());
          System.out.println(ex);
          }

System.out.println("--2.getBalance()-------");
          try{
          double bal = as.getBalance(105);
          System.out.println(bal);
          }catch(Exception ex){
          System.out.println(ex.getMessage());
          System.out.println(ex);
          }
```

```
System.out.println("--3.deposit()-------");
        try{
        double bal = as.deposit(150,9000);
        System.out.println(bal);
        }catch(Exception ex){
        System.out.println(ex.getMessage());
        System.out.println(ex);
        }

System.out.println("--4.deposit()-------");
        try{
        double bal = as.deposit(105,9000);
        System.out.println(bal);
        }catch(Exception ex){
        System.out.println(ex.getMessage());
        System.out.println(ex);
        }

System.out.println("--5.withdraw()-------");
        try{
        double bal = as.withdraw(150,29000);
        System.out.println(bal);
        }catch(Exception ex){
        System.out.println(ex.getMessage());
        System.out.println(ex);
        }

System.out.println("--6.withdraw()-------");
        try{
        double bal = as.withdraw(150,99000);
        System.out.println(bal);
        }catch(Exception ex){
        System.out.println(ex.getMessage());
        System.out.println(ex);
        }
}
}
```

5.12 Revision of method overriding

1) **When super class method is not specified with method level exception then sub class method can do the following:**
   a. Subclass method may not throw any method level exception.
   b. Subclass method can throw the unchecked exception.
   c. Subclass method can't throw the checked exception.

2) **When super class method is specified with some method level unchecked exception then sub class method can do the following :**
   a. Subclass method can ignore that method level exception.
   b. Subclass method can throw the same exception.
   c. Subclass method can throw any other unchecked exception.
   d. Subclass method cannot throw any checked exception.

3) **When super class method is specified with some method level checked exception then sub class method can do the following :**
   a. Subclass method can ignore that method level exception.
   b. Subclass method can throw the same exception.
   c. Subclass method can throw any unchecked exception.
   d. Subclass method can throw the exception which is sub class to super class method exception.
   e. Subclass method cannot throw the exception which is super class to super class method exception.
   f. Subclass method cannot throw the exception which is non subclass of super class method exception.

Lab665.java

```
class Hai{
public void m1() {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{
public void m1() {  //A1
System.out.println("Hello-m1()");
}
}
```

```
class Lab665{
public static void main(String args[]){
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab666.java

```
class Hai{
public void m1() {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //A2
public void m1() throws
ArithmeticException {
System.out.println("Hello-m1()");
}
}
class Lab666{
public static void main(String args[]){
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab667.java

```
import java.sql.*;

class Hai{
public void m1() {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //A3
public void m1() throws SQLException {
System.out.println("Hello-m1()");
}
}
class Lab667{
public static void main(String args[]){
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab668.java

```
class Hai{
public void m1() throws
ArithmeticException{
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //B1
public void m1()  {
System.out.println("Hello-m1()");
}
}
class Lab668{
public static void main(String args[]){
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab669.java

```
class Hai{
public void m1() throws
ArithmeticException{
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //B2
public void m1()  throws
ArithmeticException {
System.out.println("Hello-m1()");
}
}
class Lab669{
public static void main(String args[]){
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab670.java

```
class Hai{
public void m1() throws
ArithmeticException{
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //B3
public void m1()  throws
NullPointerException{
System.out.println("Hello-m1()");
}
}
class Lab670{
public static void main(String args[]){
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab671.java

```
import java.sql.*;

class Hai{
public void m1() throws
ArithmeticException{
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //B4
public void m1()  throws SQLException {
System.out.println("Hello-m1()");
}
}
class Lab671{
public static void main(String args[]){
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab672.java

```
import java.io.*;

class Hai{
public void m1() throws IOException {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //C1
public void m1() {
System.out.println("Hello-m1()");
}
}
class Lab672{
public static void main(String args[]){
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab673.java

```
import java.io.*;

class Hai{
public void m1() throws IOException {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //C2
public void m1()  throws IOException {
System.out.println("Hello-m1()");
}
}
class Lab673{
public static void main(String args[]) throws
IOException {
Hello  hello =new Hello();
hello.m1();
} }
```

Lab674.java

```
import java.io.*;
class Hai{
public void m1() throws IOException {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //C3
public void m1()  throws
NullPointerException {
System.out.println("Hello-m1()");
}
}
class Lab674{
public static void main(String args[]) throws
IOException {
Hello  hello =new Hello();
hello.m1();
} }
```

Lab675.java

```
import java.io.*;
class Hai{
public void m1() throws IOException {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //C4
public void m1()  throws
FileNotFoundException {
System.out.println("Hello-m1()");
}
}
class Lab675{
public static void main(String args[]) throws
IOException {
Hello  hello =new Hello();
hello.m1();
} }
```

Lab676.java

```
import java.io.*;

class Hai{
public void m1() throws IOException {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //C5
public void m1()  throws Exception {
System.out.println("Hello-m1()");
}
}
class Lab676{
public static void main(String args[]) throws
IOException {
Hello  hello =new Hello();
hello.m1();
}
}
```

Lab677.java

```
import java.io.*;
import java.sql.*;

class Hai{
public void m1() throws IOException {
System.out.println("Hai-m1()");
}
}
class Hello extends Hai{ //C6
public void m1()  throws SQLException {
System.out.println("Hello-m1()");
}
}
class Lab677{
public static void main(String args[]) throws
IOException {
Hello  hello =new Hello();
hello.m1();
} }
```

5.13 try-with-resources Statement

- Upto Java 6, when you are initializing resources to establish the connection between Java program and IO Device or DB Engine etc then you must have to close the resources explicitly using finally blocks.
- From Java 7, new feature added called try-with resources which helps to close or clean the resources automatically.

Syntax:

```
try(<Resources Declaration and Initialization>)
{
…
}catch(<throwableType> refVar){
…
}
```

Before Java 7

```
Connection con = null;
Statement st = null;
try{
con = DM.getConnection(…);
st = con.createStatement();
…
}catch(SQLException e){
…
}finally{
if(st!=null)
        st.close();
if(con!=null)
        con.close();
}
```

From Java 7

```
try(
Connection con =DM.getConnection("");
Statement st = con.createStatement();
) {
…
}catch(SQLException e){
…
}
```

- A resource is as an object that must be closed after the program execution is completed.
- The try-with-resources statement ensures that each resource is closed at the end of the statement. Any object that implements java.lang.AutoCloseable can be used as a resource.
- When a class is implementing java.lang.AutoCloseable interface then only those class objects can be used with try-with resources for auto cleanup.
- With try-with resources try block can be used without catch and finally blocks also.

SUMMARY

1) When there is no exception in try block then catch block will not be executed.

2) When exception is occurred in try block then matching catch block will be executed by skiping the remaining statements of try block.

3) When you have try block with multiple catch block then follow the following rules:
   a. The exception type in the multiple catch blocks must be unique.
   b. If the exception type doesn't have inheritance relation then the order of the exception in catch block can be anything.
   c. If the exception type has some inheritance relation then the order of the exception must be first sub type then super type.
   d. If you are placing parent class catch exception first then all the exceptions will be caught by parent class exception only, so there is no chance of executing child class catch block.
   e. There should not be any statements between try block and catch block.
   f. There should not be any statements between two catch blocks.
   g. There should not be any statements between try - catch or finally blocks.

4) When you are writing try with multiple catch blocks then only one matching catch block will be executed if available and program will be terminated normally.

5) When matcing catch block is not available then error will be handled by JVM and program will be terminated abnormally.

6) Exception is a type of communication between caller and called method to instruct about the problem and to send some message to the caller method.

7) Unchecked exception can be caught without rising in try block.

```
try {
        int x = 10 / 0; //Raising ArithmeticException
} catch (NullPointerException ex)  { }
```

8) Checked exception cannot be caught without rising in try block.

```
try {
        int x = 10 / 0;
} catch (ClassNotFoundException ex) { }
```

9) If exception is not thrown from the try block then also finally block will be executed.

10) If exception is thrown from the try block and handled in catch block then finally block will be executed after the catch block

11) If try block is throwing the exception that is not handled in catch block then before returning the control to the caller method finally block will be executed.

12) If any return statement available in the try or catch block then before returning the control the finally block will be executed.

13) Normally the finally block will be used to close the resources like database connection, I/O connection, network connection etc.

14) If the JVM is terminated before executing the finally block then the finally block will not be executed.

15) You can terminate the JVM explicitly by using the following statement

```
System.exit(0);
```

16) finally block is mainly used to write resource cleanup code.

17) When you are writing resource cleanup code inside try block then there is no guarantee of executing all the statements of try block.

18) When you are writing resource cleanup code inside catch block then if there is no problem in try block then catch block will not be executed.

19)So you must write resource cleanup code inside finally block.

20)You can write the resource cleanup code inside finalize () method also but it is not recommandable because you cannot expect exact behaviour of gc ().

21)Following three statements are valid.

throw new Exception();
throw new Error();
throw new Throwable();
 	
throw new Object(); //Not Ok
	catch(Exception){}
	catch(Error){}
	catch(Throwable){}
	
	catch(Object){} //Not Ok
		void m1() throws Exception
		void m1() throws Error
		void m1() throws Throwable
		
		void m1() throws Object //Not Ok

Error Message: No exception of type Object can be thrown; an exception type must be a subclass of Throwable.

22) Difference between Exception and Error

a) Exception is a type of problem that can be handled and you can continue your program execution.
b) Exception occurs because of programming mistake.
c) Error is a type of problem that should not be handled and your program should be terminated.
d) Error occurs due to lack of resources.
e) Exception and Error are always occurs at runtime only.
f) Error coming at compile time is called as syntectical error and it is entirly different from error coming at runtime.

Some Errors:

1) NoSuchMethodError:main

(if main() method is not found upto jdk1.6.)

2) NoClassDefFoundError
(if class name is wrong while executing.)

3) StackOverflowError
In recursive method call, if memory is not available in stack.)

4) OutOfMemoryError
(While creating the object, if there is no memory in heap.)
int arr[]=new int[54453453];

5) UnsupportedClassVersionError
(If JVM version is lower then compiler version.)

6) ExceptionInInitializerError
(if exception occurs at static variable or static block while loading the class.)

```
static int p;
static   {p=Integer.parseInt("SRI");}
}
```

Some Exceptions:

1) ClassCastException
(If you try to convert superclass object into subclass object whitout having the ref. of subclass.)

```
Object ob1 = new Object();
String st1=(String)ob1;
```

2) IllegalStateException

```
Thread t= new Thread();
t.start();
t.start();//Here this err will occur
```

3) IllegalArgumentException

```
t.setPriority(30);
```

Thread priority must be b/w 1-10.

Assignment # 18

Q1) What is Exception?

Q2) Write the usage of following?

a. try block
b. catch block
c. finally block
d. throw keyword
e. throws keyword

Q3) Can I define multiple catch blocks with one try block?

Q4) Can I define multiple finally block with one try block?

Q5) Can I define any other statements between try & catch, try & finally,catch and finally?

Q6) Can I define finally block after try block but before catch block?

Q7) Can I define try block without catch block?

Q8) Can I define try block without catch block and finally block?

Q9) Can I define multiple try-catch blocks in one method?

Q10) Can I define throw statement immediately after return statement ?

Q11) Can I define return statement immediately after throw statement ?

Q12) What is difference between checked and unchecked exception?

Q13) What is difference between throw and throws keyword?

Q14) What is custom exception? Write the steps to define custom exception.

Q15) Can I handle the error?

Q16) What is the difference between exception and error?

Q17) What is the reason of followoing excepiton and errors:

a. NoSuchMethodError:main
b. NoClassDefFoundError
c. StackOverflowError
d. OutOfMemoryError
e. UnsupportedClassVersionError

f. ExceptionInInitializerError

g. ClassCastException

h. IllegalStateException

i. IllegalArgumentException

Q18) What are the new features in exception handling added from Java 7?

Q19) Explain Catching Multiple Exceptions using Single catch block.

Q20) What is the use of try with Resource

Practice Test # 18

Q. No	Questions	Options	Answer

1	public class Test {
	public static void main(String[] args) {
	
	try {
	int res = Integer.parseInt(("JLC"));
	System.out.println("result :" + res);
	} catch (NumberFormatException e) {
	int x = 10 / 0;
	System.out.println("NFEx");
	} catch (ArithmeticException e) {
	System.out.println("AEx");
	}
	}
	}
		A) NFEx
		AEx
		B) NFEx
		C) AEx
		D) Runtime Exception	

2	public class Test {
	public static void main(String[] args) {
	
	try {
	// ...
	} catch (NullPointerException |
	ArithmeticException | Exception e){
	System.out.println(" Ex ");
	}
	}
	}
		A) Ex
		B) No Output
		C) Compilation Error
		D) Runtime Exception	

3	public class Test {
	public static void main(String[] args) {
	try {
	// ...
	} catch (NullPointerException |
	ArithmeticException e) {
	System.out.println(" B1 ");
	} catch (Exception e) {
	System.out.println(" B2 ");
	}
	}
	}
		A) B1
		B2
		B) B1
		C) B2
		D) Compilation Error
		E) Runtime Exception	

4	public class Test {
	public static void main(String[] args) {
	Abc ob = new Abc();
	int a = ob.show();
	System.out.println(a);
	}
	}
	class Abc {
	int show() {
	throw new ArithmeticException();
	return 0;
	}
	}
		A) 0
		B) ArithmeticException
		C) NullPointerException
		D) Compilation Error	

5	public class Test {
	public static void main(String[] args) {
	Abc ob = new Abc();
	long a = ob.getPhone("");
	System.out.println(a);
	}
	}
	class Abc {
	long getPhone(String sid) {
	if (sid != null)
	return 9972365983L;
	else
	return 9880979999L;
	throw new NullPointerException();
	}
	}
		A) 9972365983
		B) 9880979999
		C) ArithmeticException
		D) NullPointerException
		E) Compilation Error	

6	public class Test {
	public static void main(String[] args) {
	Abc ob = new Abc();
	int a = ob.show();
	System.out.println(a);
	}
	}
	class Abc {
	int show() {
	throw new ArithmeticException();
	}
	}
		A) 0
		B) ArithmeticException
		C) NullPointerException
		D) Compilation Error	

7	public class Test {
	public static void main(String[] args) {
	Abc ob = new Abc();
	long a = ob.getPhone("");
	System.out.println(a);
	}
	}
	
	class Abc {
	long getPhone(String sid) {
	if (sid != null)
	return 9972365983L;
	else
	throw new NullPointerException();
	}
	}
		A) 9972365983
		B) 9880979999
		C) ArithmeticException
		D) NullPointerException
		E) Compilation Error	

8	public class Test {
	public static void main(String[] args) {
	System.out.println("St1");
	throw new NumberFormatException();
	System.out.println("St2");
	}
	}
		A) St1
		St2
		B) St1
		C) St1
		NumberFormatException
		D) NumberFormatException
		E) Compilation Error	

9	public class Test {
	public static void main(String[] args) {
	System.out.println("St1");
	throw new MyException();
	}
	}
	class MyException { }
		A) St1
		B) St1
		MyException
		C) MyException
		D) NullPointerException
		E) Compilation Error	

10	public class Test {
	public static void main(String[] args) {
	System.out.println("St1");
	throw new MyException();
	}
	}
	class MyException extends
	RuntimeException{}
		A) St1
		B) St1
		MyException
		C) MyException
		D) Compilation Error	

11	public class Test {
	public static void main(String[] args) {
	System.out.println("St1");
	MyException ex = null;
	throw ex;
	}
	}
	class MyException extends
	RuntimeException{}
		A) St1
		B) St1
		MyException
		C) MyException
		D) NullPointerException
		E) Compilation Error	

12	public class Test {
	public static void main(String[] args) {
	int x;
	try {
	x = 99;
	System.*out*.println(x);
	} catch (Exception e) {
	}
	}
	}
		A) 99
		B) 0
		C) RuntimeException
		D) Compilation Error	

13	public class Test {
	public static void main(String[] args) {
	int x;
	try {
	x = 99;
	} catch (Exception e) { }
	System.out.println(x);
	}
	}
		A) 99
		B) 0
		C) RuntimeException
		D) Compilation Error	

14	public class Test {
	public static void main(String[] args) {
	int x;
	try {
	x = 99;
	} catch (Exception e) {
	x = 0;
	}
	System.*out*.println(x);
	}
	}
		A) 99
		B) 0
		C) RuntimeException
		D) Compilation Error	

15	public class Test {
	public static void main(String[] args) {
	int x;
	try {
	} finally {
	x = 99;
	}
	System.*out*.println(x);
	}
	}
		A) 99
		B) 0
		C) RuntimeException
		D) Compilation Error	

16	public class Test {
	public static void main(String[] args) {
	try {
	return;
	} finally {
	System.out.println("JLC");
	}
	}
	}
		A) JLC
		B) 0
		C) RuntimeException
		D) Compilation Error	

17	public class Test {
	public static void main(String[] args) {
	try {
	int a = 0;
	int b = 5 / a;
	} catch (Exception e) {
	System.out.println("Ex");
	} catch (ArithmeticException ae) {
	System.out.println("AEx");
	}
	}
	}
		A) Ex
		B) AEx
		C) RuntimeException
		D) Compilation Error	

18	public class Test {
	public static void main(String[] args) {
	try (MyRes res = new MyRes();) {}
	}
	}
	class MyRes implements AutoCloseable
	{
	public void close() {
	System.out.println("CLOSE");
	}
	}
		A) CLOSE
		B) No Output
		C) RuntimeException
		D) Compilation Error	

19	public class Test {
	public static void main(String[] args) {
	Hello h = new Hello();
	h.process();
	System.out.println("JLC");
	}
	}
	class Hello{
	public int process() {
	return 10;
	}
	}
		A) JLC
		B) 10
		C) RuntimeException
		D) Compilation Error	

20	public class Test {
	public static void main(String[] args) {
	Hello h = new Hello();
	h.process();
	System.out.println("JLC");
	}
	}
	class Hello {
	public int process() {
	throw new NullPointerException();
	}
	}
		A) JLC
		B) 10
		C) RuntimeException
		D) Compilation Error	