COCSIT

EXCEPTION HANDLING

Exception Handling

Agenda

1. Introduction to Exception Handling
2. Exception Handling Keywords

Introduction to Exception Handling




What is an Exception?

- An exception is an event that occurs **during the execution of a program** that disrupts the normal flow of instructions
- The ability of a program to intercept run-time errors, take corrective measures and continue execution is referred to as exception handling

- There are various situations when an exception could occur:
  - Attempting to access a file that does not exist
  - Inserting an element into an array at a position that is not in its bounds
  - Performing some mathematical operation that is not permitted
  - Declaring an array using negative values

What is an Exception? (Contd.).

- The following exceptions have to be addressed by the programmer.
- They are frequently encountered in java programs.

  - NPE -- NullPointerException
  - NFE -- NumberFormatException
  - AIOOBE -- ArrayIndexOutOfBoundsException
  - SIOOBE -- StringIndexOutOfBoundsException
  - AE -- ArithmeticException

- Can You list under which situation, these exceptions occur?
- Can You say, **How to avoid them** ?
- Can You list some exceptions ?

Uncaught Exceptions

```
class  Demo {
  public static void main(String args[]) {
    int x = 0;
    int y = 50/x;
    System.out.println("y  = " +y);
  }
}
```

Will compile, but when you execute it, displays:

java.lang.ArithmeticException: / by zero

at Demo.main(Demo.java:4)

At What Line Exception occurred?
Can you see?

Exception Handling Techniques

- There are several built-in exception classes that are used to handle the very fundamental errors that may occur in your programs

- You can create your own exceptions also by extending the **Exception** class
- These are called **User-defined Exceptions.**
- **Can you say some example situations, where you will create User-defined Exceptions?**

Handling Runtime Exceptions

- Whenever an exception occurs in a program, an object representing that exception is created and thrown in the method in which the exception occurred
- Either you can handle the exception, or ignore it
- In the latter case, the exception is handled by the Java run-time-system and the program terminates
- Handling the exceptions will avoid **abnormal program termination!**

Exception Handling Keywords

Exception Handling Keywords

Java's exception handling is managed using the following keywords: **try, catch, throw, throws and finally.**

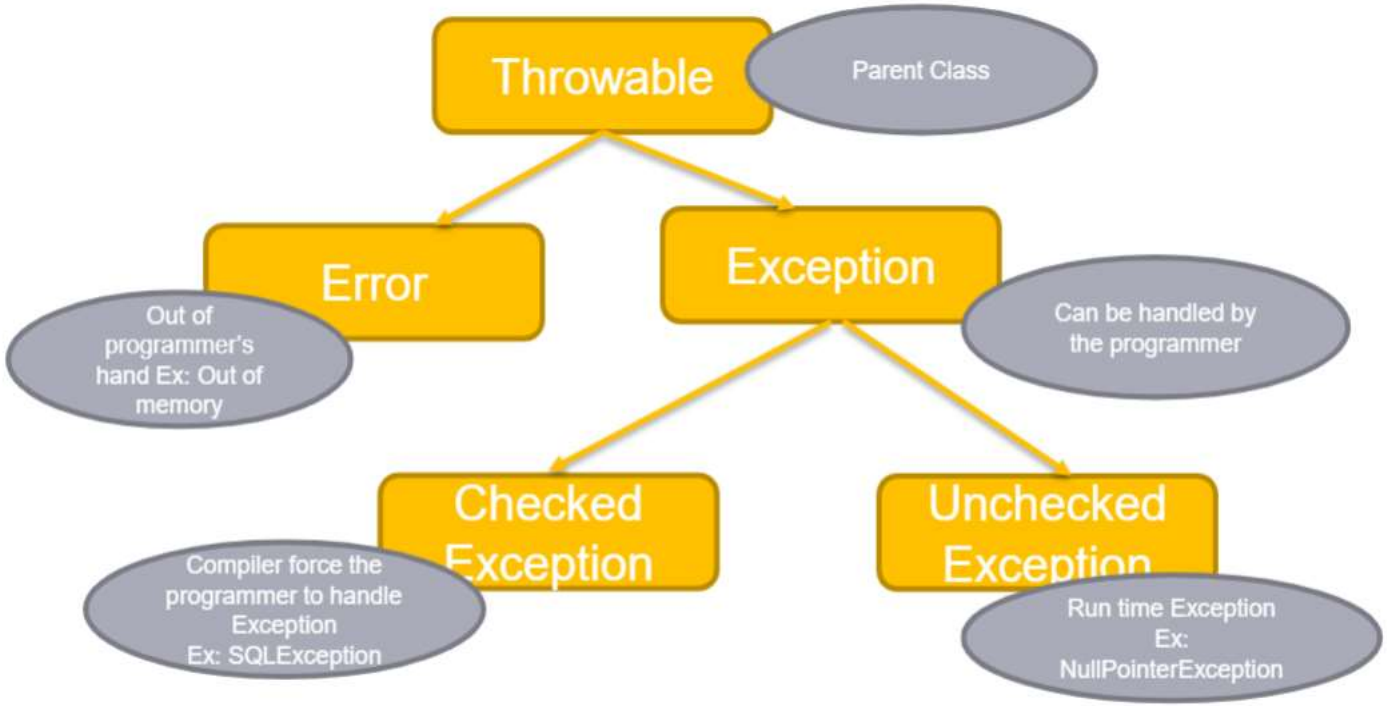
```
try {
   // code comes here
   }
catch(TypeofException  obj) {
    //handle the exception
     }
finally  {
              //code to be executed before the program ends
   }
```

Exception Handling - Types

Types of Exception

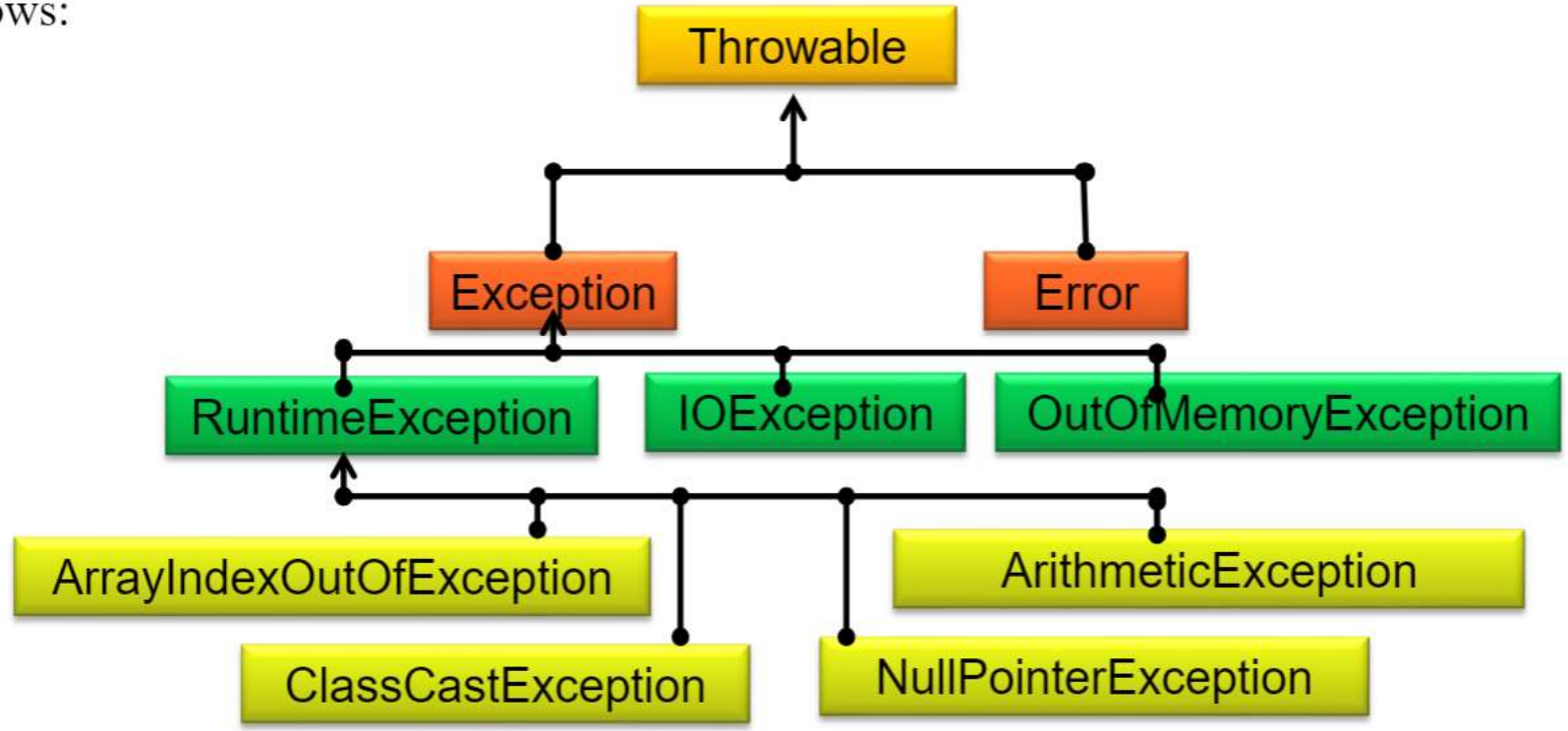



Exception in a Nutshell



Exception Types

Exceptions are implemented in Java through a number of classes. The exception hierarchy is as follows:



Checked and Unchecked Exceptions




Checked Exception

- A checked exception must be handled using a try or catch or at least declared to be thrown using throws clause.
- Non compliance of this rule results in a compilation error

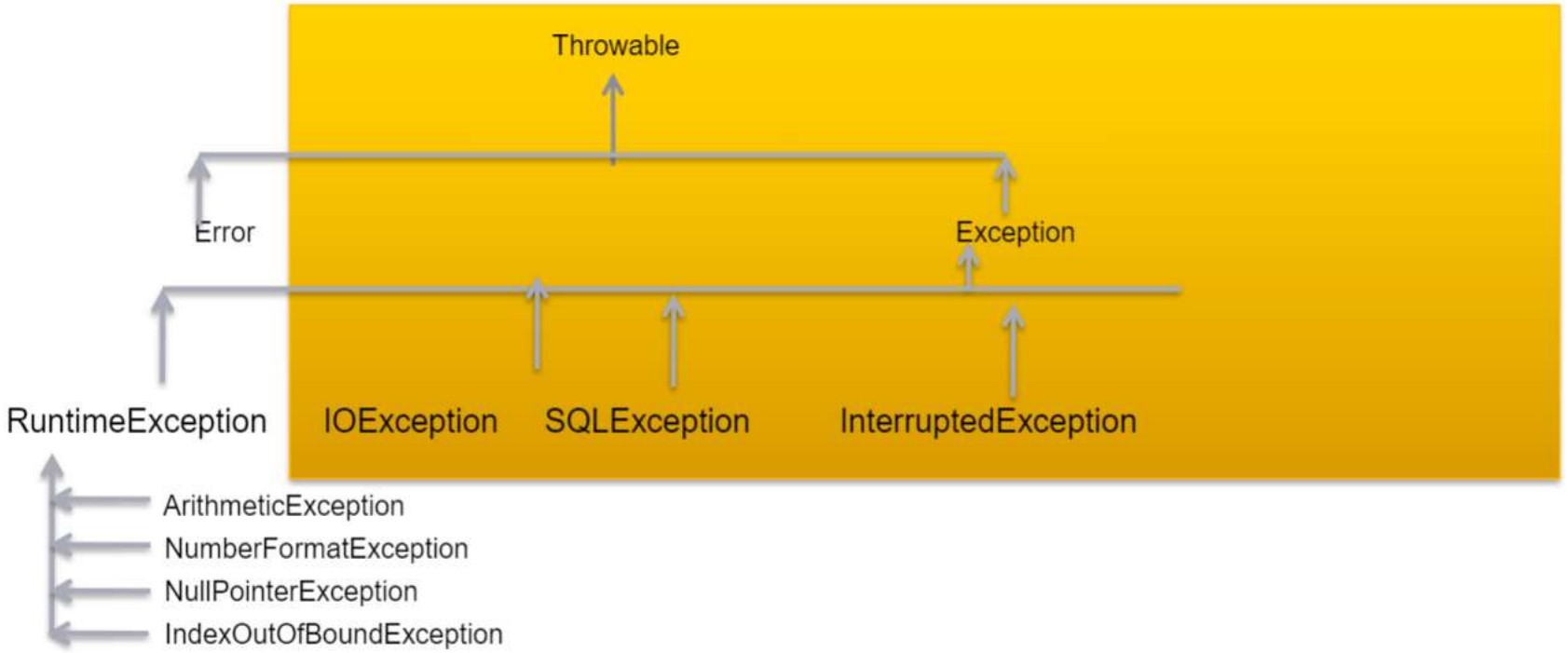
Ex: FileNotFoundException

- If you try to open a file using

```
FileInputStream fx = new FileInputStream("A1.txt");
```

- During execution, the system will throw a FileNotFoundException, if the file A1.txt is not located, which may be beyond the control of a programmer

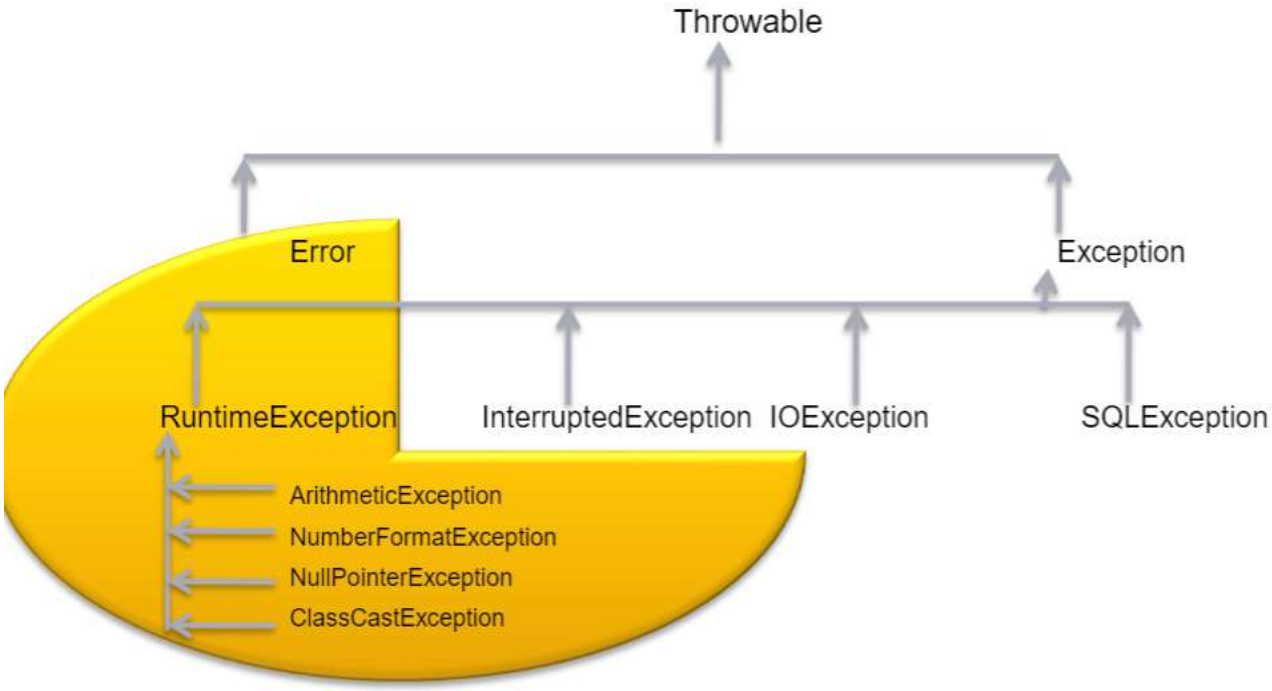
Checked Exceptions (marked in yellow color)



Unchecked Exceptions

- An unchecked exception is an exception, which could have been avoided by the programmer
- The class **RuntimeException** and all its subclasses are categorized as Unchecked Exceptions
- If there is any chance of an unchecked exception occuring in the code, it is ignored during compilation

Unchecked Exception (marked in yellow color)



Exception Handling – Try-Catch Block

Agenda

1. Try-Catch Block
2. Multiple Catch Block
3. Nested Try Block

Try-Catch Block




Try-Catch Block

- Any part of the code that can generate an error should be put in the **try** block
- Any error should be handled in the **catch** block defined by the **catch** clause
- This block is also called the **catch block**, or the **exception handler**
- The corrective action to handle the exception should be put in the **catch** block

How to Handle exceptions?

```
class ExceptDemo{
  public static void main(String args[]){
    int x, a;
    try{
        x = 0;
        a = 22 / x;
        System.out.println("This will be bypassed.");
    }
    catch (ArithmeticException e){
        System.out.println("Division by zero.");
    }
    System.out.println("After catch statement.");
  }
}
```

Quiz

- What will be the result, if we try to compile and execute the following code as java Ex1 Wipro Bangalore

```
Class Ex1 {
    public static void main(String[] xyz){
        for(int i=0;i<=args.length;i++)
          System.out.println(args[i]);
    }
}
```

Compile but throws exception during runtime!
Why this exception is thrown?

Multiple Catch Block




Multiple Catch Statements

- A single block of code can raise more than one exception
- You can specify two or more **catch** clauses, each catching a different type of execution
- 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

Multiple Catch Statements (Contd.).

```
class MultiCatch{
  public static void main(String args[]){
    try{
        int l = args.length;
        System.out.println("l = " +l);
        int b = 42 / l;
        int arr[] = { 1 };
        arr[22] = 99;
    }
    catch(ArithmeticException e){
        System.out.println("Divide by 0: "+ e);
    }
```

Multiple Catch Statements (Contd.).

```
    catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Array index oob: "+e);
    }
    System.out.println("After try/catch blocks.");
  }
}
```

Quiz

- What will be the result, if we try to compile and execute the following code as java Ex2 100

```
class Ex2 {
    public static void main(String[] args) {
      try {
        int i= Integer.parseInt(args[0]);
        System.out.println(i);
      }
        System.out.println("Wipro");          // is there any problem?
      catch(NumberFormatException e) {
        System.out.println(e);
      }
    }
}
```

It will throw compilation Error

Nested Try Block




Nested try Statements

- The **try** statement can be nested
- If an inner **try** statement does not have a **catch** handler for a particular exception, the outer block's catch handler will handle the exception
- This continues until one of the **catch** statement succeeds, or until all of the nested **try** statements are exhausted
- If no catch statement matches, then the Java runtime system will handle the exception

Syntax

```
try
{
     statement 1;
     statement 2;
     try
     {
          statement 1;
          statement 2;
     }
     catch(Exception e)
     {
     }
}
catch(Exception e)
{
}
```

Example for nested try

```
class Nested_Try{
 public static void main(String args[]){
  try{
          try{
             System.out.println("Arithmetic Division");
             int b =39/0;
           }catch(ArithmeticException e){
                System.out.println(e);
           }
           try{
             int a[]=new int[5];
             System.out.println("Accessing Array Elements");
              a[5]=4;
           } catch(ArrayIndexOutOfBoundsException e)    {
                System.out.println(e);
           }
              System.out.println ("Inside Parent try");
           } catch(Exception e) {
              System.out.println("Exception caught");
           }
      System.out.println("Outside Parent try");
 }
 }
```

Quiz

1. Debug the code

```
public class Tester {
public static void main(String[] args) {
     try {
          System.out.println("A");
     }
     catch (Exception e)
     {
          System.out.println("B");
     }
     catch (ArithmeticException a)      {
          System.out.println("C");
     }
}
}
```

Throws Clause




Using throws

- Sometimes, a method is capable of causing an exception that it does not handle
- Then, it must specify this behavior so that callers of the method can guard themselves against that exception
- While declaring such methods, you have to specify what type of exception it may throw by using the **throws** keyword
- A **throws** clause specifies a comma-separated list of exception types that a method might throw:
  - type method-name(parameter list) throws exception-list

Using throws (Contd.).

```
class ThrowsDemo{
  static void throwOne(){
    System.out.println("Inside throwOne.");
    throw new FileNotFoundException();
  }
  public static void main(String args[]){
    throwOne();
  }
}
```

What happens when this code is compiled ?

Compilation Error……why?

Implementing throws

```
import java.io.*;
class ThrowsDemo{
  static void throwOne() throws FileNotFoundException{
    System.out.println("Inside throwOne.");
    throw new FileNotFoundException();
  }
  public static void main(String args[]) {
    try{
        throwOne();
    }
    catch (FileNotFoundException e){
        System.out.println("Caught " + e);
    }
  }
}
```

Rule governing overriding method with throws

- The overriding method must NOT throw checked exceptions that are new or broader than those declared by the overridden method

For eg : A method that declares(throws) an SQLException cannot be overriden by a method that declares an IOException, Exception or any other exception unless it is a subclass of SQLException

- In other words, if a method declares to throw a given exception, the overriding method in a subclass can only declare to throw the same exception or its subclass

- This rule does not apply for unchecked exceptions

Quiz

- What will be the result, if we try to compile the following code (FileNotFoundException is a subclass of IOException)

```
import java.io.*;
class Super {
  void m1() throws FileNotFoundException {
    FileInputStream fx = new FileInputStream("Super.txt");
  }
}
class Sub extends Super {
  void m1() throws IOException {
    FileInputStream fx = new FileInputStream("Sub.txt");
  }
}
```

Yes, it will throw compilation Error

Quiz (Contd.).

- What will be the result, if we try to compile the following code (FileNotFoundException is a subclass of IOException)

```
import java.io.*;
class Super {
  void m1() throws IOException {
    FileInputStream fx = new FileInputStream("Super.txt");
  }
}
class Sub extends Super {
  void m1() throws FileNotFoundException {
    FileInputStream fx = new FileInputStream("Sub.txt");
  }
}0
```

No Error!
Compilation successful

Quiz (Contd.).

- What will be the result, if we try to compile the following code

```
class Super {
  void m1() throws ArithmeticException {
    int x = 100, y=0;
        int z=x/y;
        System.out.println(z);
  }
}
class Sub extends Super {
  void m1() throws NumberFormatException {
    System.out.println("Wipro");
  }
}
```

No Error!
Compilation successful

Quiz (Contd.).

- What will be the result, if we try to compile the following code
- (FileNotFoundException & SQLException are not related hierarchically)

```
import java.io.*;
import java.sql.*;
class Super {
  void m1() throws FileNotFoundException {
    FileInputStream fx = new FileInputStream("Super.txt");
  }
}
class Sub extends Super {
  void m1() throws SQLException {
    FileInputStream fx = new FileInputStream("Sub.txt");
  }
}
```

It will throw compilation Error. Why?

Quiz (Contd.).

What will be the result, if we try to compile and execute the following code

```
import java.io.*;
class Plane {
    public Plane() throws IOException, RuntimeException {
    System.out.println("Plane");
    }
}
class Jet extends Plane {    // empty class
}
public class Tester {
    public static void main(String args[]) throws IOException {
    new Plane();
    }
}
```

It will throw compilation Error.
Why ?

Throw Clause




Using throw

- System-generated exceptions are thrown automatically
- At times you may want to throw the exceptions explicitly which can be done using the **throw** keyword
- The exception-handler is also in the same block
- The general form of throw is:
  - throw **ThrowableInstance**
- Here, **ThrowableInstance** must be an object of type **Throwable**, or a subclass of **Throwable**

Using throw (Contd.).

```
class ThrowDemo{
  public static void main(String args[]) {
    try {
         int age=Integer.parseInt(args[0]);
         if(age < 18)
             throw new ArithmeticException();
        else
             if(age >=60)
                 throw new ArithmeticException("Employee is retired");
    }
  catch(ArithmeticException e) {
    e.printStackTrace();
  }
  System.out.println("After Catch");
    }
}
```

What is ArithmeticException ?

What is e.printStackTrace()?

Match the following :

Match the content of Column A with the most appropriate content from column B :

Column A	Column B
a) An exception is	a) Used to throw an exception explicitly
b) Throwable	b) Caused by Dividing an integer by zero
c) ArithmeticException is	c) An event that can disrupt the normal flow of instructions
d) Catch Block	d) This class is at the top of exception hierarchy
e) “throw” is	e) Exception Handler

Quiz

Debug the code & Tell Whats the output ?

```
import java.io.*;
class ThrowTester {

    void method() throws IOException {
    throw new IOException("Method Error");
    }

    public static void main(String args[]) {
        ThrowTester m = new ThrowTester();
        m.method();
        System.out.println("main ends...");
    }

}
```

COCSIT

Finally Clause




wipro

Using finally

- When an exception occurs, the execution of the program takes a non-linear path, and could bypass certain statements
- A program establishes a connection with a database, and an exception occurs
- The program terminates, but the connection is still open
- To close the connection, **finally** block should be used
- The finally block is guaranteed to execute in all circumstances

Using finally (Contd.).

```
import java.io.*;
class FinallyDemo{
static void funcA() throws FileNotFoundException
{
    try{
        System.out.println("inside funcA( )");
        throw new FileNotFoundException( );
    }
    finally{
        System.out.println
        ("inside finally of funA( )");
  }
}
```

Using finally (Contd.).

```
    static void funcB(){
      try{
          System.out.println("inside funcB( )");
      }
      finally{
          System.out.println
          ("inside finally of funB( )");
      }
    }
}
```

Using finally (Contd.).

```
static void funcC() {
  try{
      System.out.println("inside funcC( )");
  }
  finally{
      System.out.println
      ("inside finally of funcC( )");
  }
}
```

Using finally (Contd.).

```
    public static void main(String args[]){
      try{
          funcA();
      }
      catch (Exception e){
          System.out.println("Exception caught");
      }
      funcB( );
      funcC( );
    }
}
```

Significance of printStackTrace() method

- We can use the *printStackTrace()* method to print the program's execution stack
- This method is used for debugging

Example on printStackTrace() method

```
import java.io.*;
class PrintStackExample {
  public static void main(String args[]) {
    try {
        m1();
    }
    catch(IOException e) {
        e.printStackTrace();
    }
  }
```

contd..

Example on printStackTrace() method

```
    static void m1() throws IOException {
      m2();
    }
    static void m2() throws IOException {
      m3();
    }
    static void m3() throws IOException{
      throw new IOException();
    }
}
```

Expected Output

java.io.IOException
    at PrintStackExample.m3(PrintStackExample.java:24)
    at PrintStackExample.m2(PrintStackExample.java:20)
    at PrintStackExample.m1(PrintStackExample.java:16)
    at PrintStackExample.main(PrintStackExample.java:5)

Quiz

What will be the result, if we try to compile and execute the following code as java Ex2 A

```
class Ex2 {
    public static void main(String[] args) {
       try {
         int i= Integer.parseInt(args[0]);
         System.out.println(i);
       }
       catch(NumberFormatException e) {
         System.out.println(e);
       }
       System.out.println("Exception Caught");
       finally { }
  }
}
```

It will throw compilation Error

Quiz(Contd.).

What will be the result, if we try to compile and execute the following code

```
public class Tester {
static void method() {
        throw new Exception();
}
public static void main(String[] args) {
        try {
        method();
        } catch (Throwable e) {
                try {
                throw new Exception();
                } catch (Exception ex) {
                System.out.print("exception");
                } finally {
                System.out.print("finally");
                }
        }
}}
```

It will throw compilation Error;
Why ?
How to remove the same?

Thank You