Exception Handling in Java

Bca 4th semester
Ashmita Mahanty
Assistant professor
Departement of IT

Exception handling

Successful termination/graceful termination- termination after executing program successfully

***Abnormal termination*:** termination in middle of program execution

What is exception?

- ➔ Unexpected and unwanted situation in the program execution is called exception.
- ➔ Exception will disturb normal flow of the program execution.
- ➔ When exception occurred program will be terminated abnormally.
  Note- as a programmer we are responsible for programs graceful termination.
- ➔ To achieve graceful termination we need to handle the exceptions occurred while program executing.
- ➔ The process of handling exception is called as exception handling.
- ➔ The main aim of exception handling to achive graceful termination of the program\
- ➔ In java we have so many predefined exceptions.
- ➔ Eg-
  AirthematicExecption
  NullPointerException
  FileNotFouneException
  SqlExecption

Exception Hierarchy

- ➔ In this hierarchy Throwable is the root class
  1. Throwable
     1.1 Exception
        1.1.1 Checked Exception
        1.1.2 Unchecked exception
     1.2 Error

Q) what is the difference between exceptions and Error?

→ *Exceptions can be handled where as Errors can't be handled.*

Exception Types-

Exceptions are divided into 2 types-

1) Checked Exception:: will be identified at compile time (occurs at run time)
   Eg- SQLException, FileNotFoundException
2) UnChecked Exceptions :: will occur at Run time (compiler can't identify these exceptions)
   Eg- NullPointerException, ArithematicException etc..

```java
public class DemoException {
    public static void main(String[] args) {
        System.out.println("main() method started...");
        String s=null;
        System.out.println(s.length());
        System.out.println("main() method ended...");
    }
```

```
}
C:\Users\user\Desktop\My_java>javac DemoException.java

C:\Users\user\Desktop\My_java>java DemoException
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"String.length()" because "<local1>" is null
        at DemoException.main(DemoException.java:4)

C:\Users\user\Desktop\My_java>
```

Exception handling

➔ Java provided 5 keyword to handle the exceptions

1. try
2. catch
3. finally
4. throws
5. throw

try

➔ it is used to keep risky code or we can say that in this we can keep critical statements.

syntax:

```
try{
        //stmts
}
```

Note- we can't write only try block.

try- catch ---- > valid combination

try- finally----- > valid combination

try- catch-finally----- > valid combination

only try block ---- > invalid

catch

➔ catch block is used to catch the exception which occurred in try block.

➔ To write catch block, try block is mandatory.

➔ If exception is occurred in try block then only catch block will execute otherwise it will not execute.

Syntax:

```
try{
        //logic
```

```
}
catch(Exception e){
        //logic to catch exception info
}
```

```
public class DemoException {
    public static void main(String[] args) {
        System.out.println("main() method started...");
        try{
            System.out.println("try block started");
```

// try with multiple catch blocks

note- order should be child to parent

```
public class DemoException {
    public static void main(String[] args) {
        System.out.println("main() method started...");
        try{
            System.out.println("try block started");
            String s=null;
            System.out.println(s.length());
            System.out.println("try block eneded");
        }
        catch(ArithmeticException e)
        {
            e.printStackTrace();
        }
        catch(NullPointerException e)
        {
            e.printStackTrace();
        }
        catch(Exception e){
            System.out.println("in catch block");
            e.printStackTrace();
        }

        System.out.println("main() method ended...");
    }
}
```

// below scenario will give error

```
public class DemoException {
    public static void main(String[] args) {
        System.out.println("main() method started...");
        try{
            System.out.println("try block started");
            String s=null;
            System.out.println(s.length());
            System.out.println("try block eneded");
        }
```

```
        catch(Exception e){
            System.out.println("in catch block");
            e.printStackTrace();
        }
        catch(ArithmeticException e)
        {
            e.printStackTrace();
        }
        catch(NullPointerException e)
        {
            e.printStackTrace();
        }


        System.out.println("main() method ended...");
    }
}
```

printStackTrace()-

what exception we got print the information of that exception.

Finally block

- ➔ It is used to perform resource cleanup activities
  Eg- file close, db conn close etc..
- ➔ Finally block will execute always (irrespective of the exception)

try with finally: valid combination
try- catch: valid combination
try-catch-finally: valid combination
catch- finally: invalid combination
Only finally: invalid combination

//in below program finally block is executed even after exception is occurring

```
public class DemoException {
    public static void main(String[] args) {
        System.out.println("main() method started...");
        try{
            System.out.println("try block started");
            String s=null;
            System.out.println(s.length());
            System.out.println("try block eneded");
        }

        catch(Exception e){
            System.out.println("in catch block");
            e.printStackTrace();
        }
        finally{
            System.out.println("in finally block");
        }
```

```
        System.out.println("main() method ended...");
    }
}
main() method started...
try block started
java.lang.NullPointerException: Cannot invoke "String.length()" because
"<local1>" is null
    at DemoException.main(DemoException.java:7)
in catch block
in finally block
main() method ended..
```

//in below program finally block is executed but exception is not occuring

```
public class DemoException {
    public static void main(String[] args) {
        System.out.println("main() method started...");
        try{
            System.out.println("try block started");
            String s="ashmita";
            System.out.println(s.length());
            System.out.println("try block eneded");
        }

        catch(Exception e){
            System.out.println("in catch block");
            e.printStackTrace();
        }
        finally{
            System.out.println("in finally block");
        }


        System.out.println("main() method ended...");
    }
}
main() method started...
try block started
7
try block eneded
in finally block
main() method ended...
```

Q) what is the difference between final, finalize() and finally?

final: it is a keyword which is used to declare final variables, final methods and final classes

finalize(): it is a predefined method available in object class, and it will be called by garbage collector before removing unused objects from heap area.

finally: it is a block we will use to execute some clean up activities in exception handling.

throws:

→it is used to handover checked exceptions to caller method/jvm

note: we can ignore checked exceptions using throws keyword.

→The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

```
import java.io.FileNotFoundException;
import java.io.FileReader;

public class DemoException {
    public static void main(String[] args) throws FileNotFoundException{
        FileReader fr=new FileReader("abc.txt");
    }
}
```

throw:

- Throw keyword is used to create the exception
- This keyword is used to explicitly throw an exception.

Syntax:

throw new Exception(“msg”);

```
import java.io.FileNotFoundException;
import java.io.FileReader;

public class DemoException {
    int id;
    String print(int id) throws Exception{

            if (id==100)
            {
                return "rani";
            }
            else if(id==200)
            {
                return "raja";
            }
            else
            {
                throw new Exception("invalid input");
            }

        }


    public static void main(String[] args) throws Exception {
        DemoException obj=new DemoException();
```

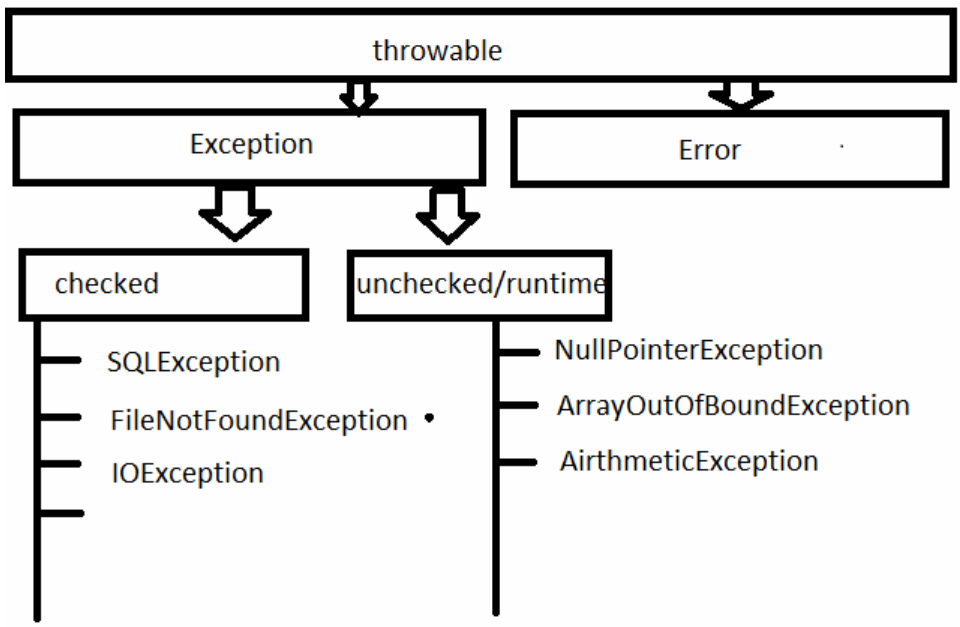
```
        obj.print(100);
        obj.print(101);
    }
}
Exception in thread "main" java.lang.Exception: invalid input
    at DemoException.print(DemoException.java:18)
    at DemoException.main(DemoException.java:27)
```

Example with Custom Exception

You can also create your own custom exceptions by extending the Exception class:

```
class MyCustomException extends Exception {
   public MyCustomException(String message) {
      super(message);
   }
}
public class CustomExceptionExample {
   public static void main(String[] args) {
      try {
         validateAge(15);
      } catch (MyCustomException e) {
         System.out.println("Caught custom exception: " + e.getMessage());
      }
   }
   static void validateAge(int age) throws MyCustomException {
      if (age < 18) {
         throw new MyCustomException("Age less than 18 is not allowed");
      }
   }
}
```

Exception Hierarchy



Error example in java

```
public class DemoException {
    int id;
   void m1()
   {
      m2();
   }
   void m2()
   {
        m1();
   }


    public static void main(String[] args) throws Exception {
        DemoException obj=new DemoException();
        obj.m1();
    }
}
Exception in thread "main" java.lang.StackOverflowError
    at DemoException.m2(DemoException.java:10)
    at DemoException.m1(DemoException.java:6)
    at DemoException.m2(DemoException.java:10)
    at DemoException.m1(DemoException.java:6)
    at DemoException.m2(DemoException.java:10)
    at DemoException.m1(DemoException.java:6)
    at DemoException.m2(DemoException.java:10)
```