Exception handling

Exception

- An exception can occur for many different reasons, below given are some scenarios where exception occurs.
  - A user has entered invalid data.
  - A file that needs to be opened cannot be found.
  - A network connection has been lost in the middle of communications or the JVM has run out of memory.

Checked exceptions

- A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions.

Checked exceptions

```
import java.io.File;
import java.io.FileReader;

public class FilenotFound_Demo {

   public static void main(String args[]){
      File file=new File("E://file.txt");
      FileReader fr = new FileReader(file);
   }

}
```

If you try to compile the above program you will get exceptions as shown below.

```
C:\>javac FilenotFound_Demo.java
FilenotFound_Demo.java:8: error: unreported exception FileNotFoundException; must be
caught or declared to be thrown
      FileReader fr = new FileReader(file);
                      ^
1 error
```

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.

Unchecked exceptions

```
public class Unchecked_Demo {

   public static void main(String args[]){
      int num[]={1,2,3,4};
      System.out.println(num[5]);
   }

}
```

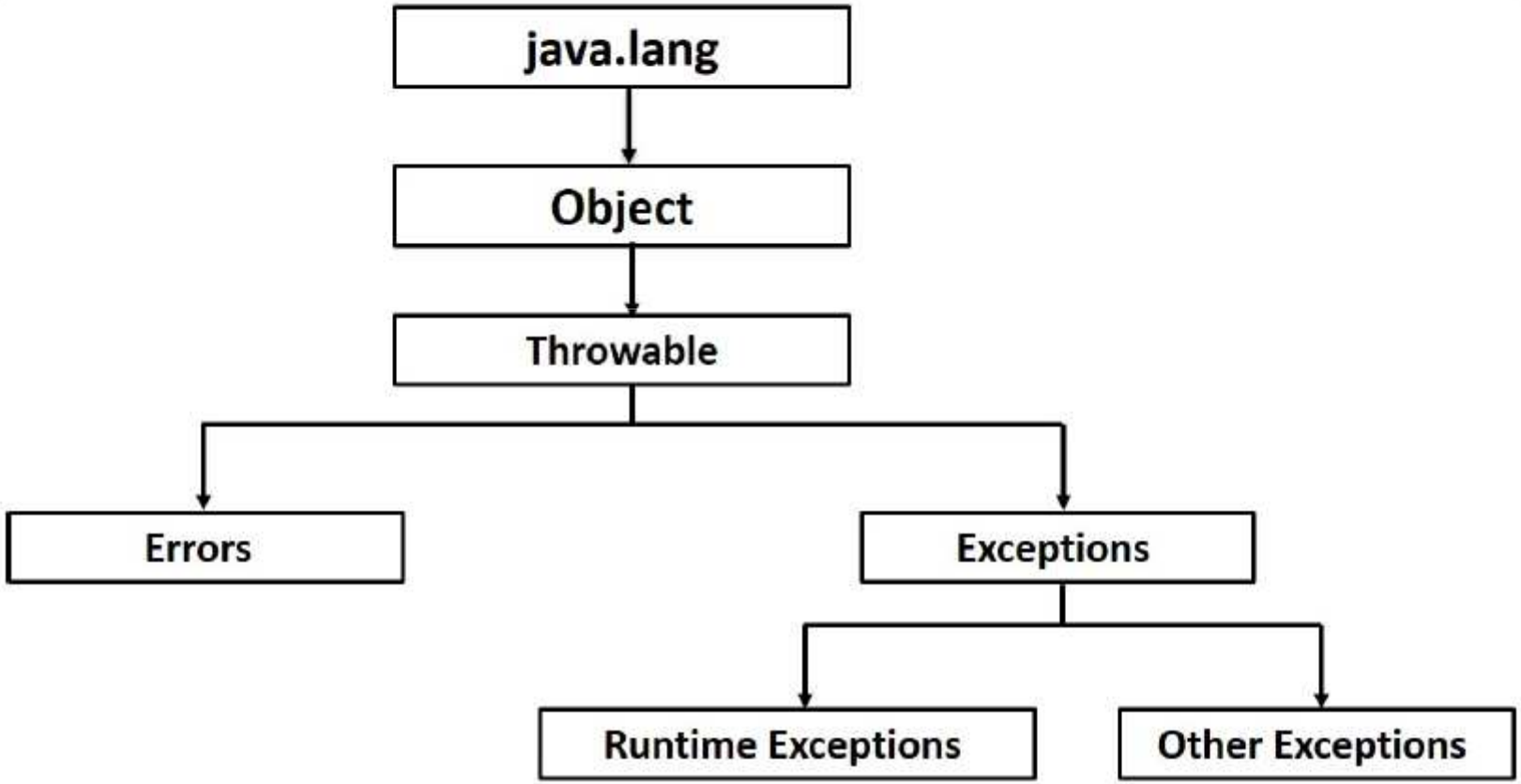
If you compile and execute the above program you will get exception as shown below.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
 at Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)
```

Errors

- These are not exceptions at all, but problems that arise beyond the control of the user or the programmer.
- Errors are typically ignored in your code because you can rarely do anything about an error.
- For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception hierarchy



Catching exception

- A method catches an exception using a combination of the **try and catch keywords.**
- **A try/catch** block is placed around the code that might generate an exception.

Example

```
// File Name : ExcepTest.java
import java.io.*;
public class ExcepTest{

   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");
   }
}
```

This would produce the following result:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

Multiple Catch Block

- If an exception occurs in the protected code, the exception is thrown to the first catch block in the list.
- If the data type of the exception thrown matches ExceptionType1, it gets caught there.
- If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Multiple Catch Block – Syntax & Example

```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}
```

```
public class TestMultipleCatchBlock{
 public static void main(String args[]){
  try{
   int a[]=new int[5];
   a[5]=30/0;
  }
  catch(ArithmeticException e){System.out.println("task1 is completed");}

  catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2
   completed");}
  catch(Exception e){System.out.println("common task completed");}

  System.out.println("rest of the code...");
 }
}
```

Catching multiple type of exceptions

```
catch (IOException|FileNotFoundException ex) {
    logger.log(ex);
    throw ex;
```

The throws/throw Keywords

- If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.
- You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw keyword.**
- Try to understand the difference between throws and throw keywords, throws is used to postpone the handling of a checked exception and throw is used to invoke an exception explicitly.

Throw vs throws

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.

Example - throw

```
public class TestThrow1{
  static void validate(int age){
    if(age<18)
     throw new ArithmeticException("not valid");
    else
     System.out.println("welcome to vote");
  }
  public static void main(String args[]){
    validate(13);
    System.out.println("rest of the code...");
 }
}
```

Output:

Exception in thread main java.lang.ArithmeticException: not valid

Example - throws

```
import java.io.IOException;
class Testthrows1{
  void m()throws IOException{
    throw new IOException("device error");//checked exception
  }
  void n()throws IOException{
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Output:
exception handled
normal flow...

Finally

- The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.
- Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

```
public class ExcepTest{

   public static void main(String args[]){
      int a[] = new int[2];
      try{
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      finally{
         a[0] = 6;
         System.out.println("First element value: " +a[0]);
         System.out.println("The finally statement is executed");
      }
   }
}
```

This would produce the following result:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```