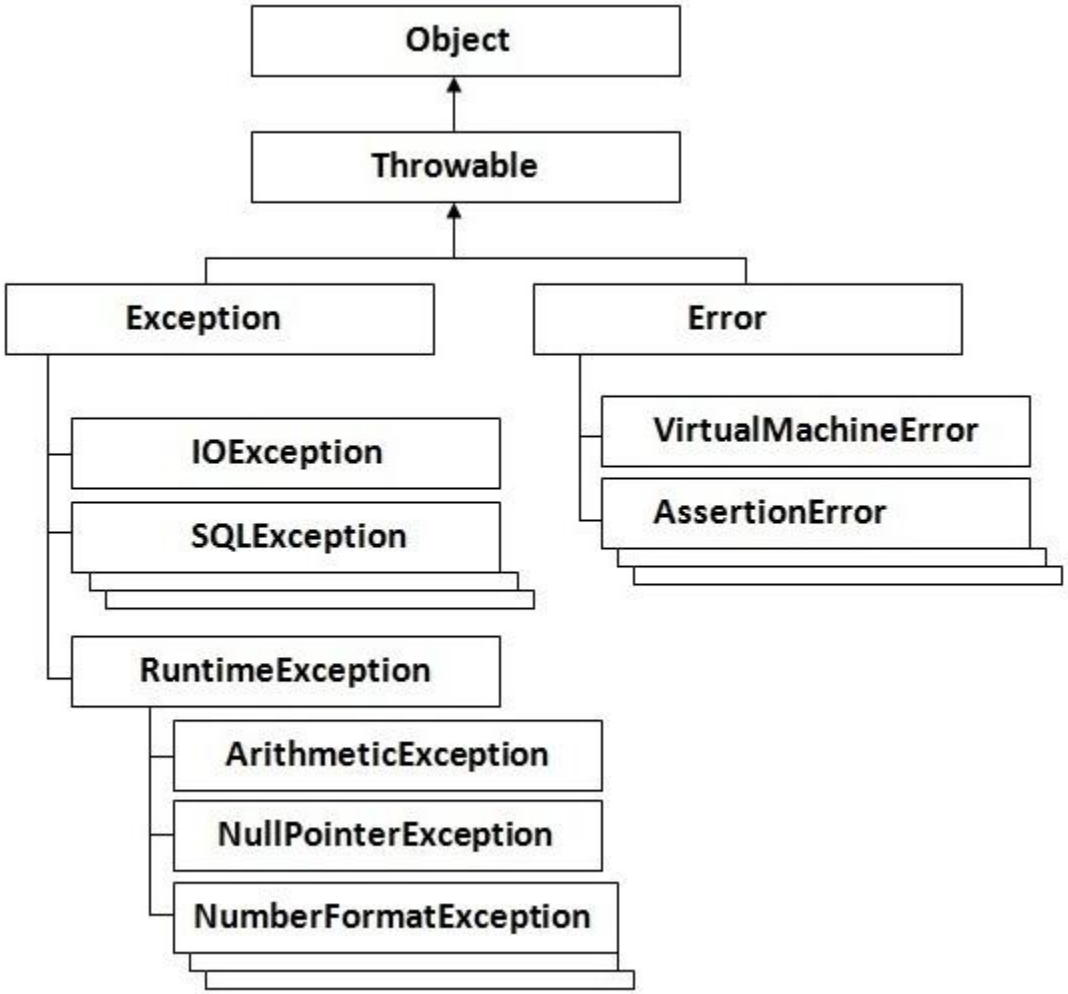
Exception Handling in Java

- Amal Prasad

What is exception handling

- The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.
- **Advantage of Exception Handling**
  - to maintain the normal flow of the application

Hierarchy of Java Exception



Amal Prasad

Types of Exception

- Checked Exception
  - extend Throwable class
  - Checked exceptions are checked at compile-time
  - e.g.IOException, SQLException etc.
- Unchecked Exception
  - The classes that extend RuntimeException are known as unchecked exceptions
  - e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.
  - Unchecked exceptions are checked at runtime.
- Error
  - Error is irrecoverable
  - e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Common Scenarios

- int a=50/0; //ArithmeticException
- String s=null;

  System.out.println(s.length()); //NullPointerException
- String s="abc";

  int i=Integer.parseInt(s);//NumberFormatException
- int a[]=new int[5];

  a[10]=50; //ArrayIndexOutOfBoundsException

Exception Handling Keywords

- try
- catch
- finally
- throw
- throws

try-catch

- Used to enclose the code that might throw an exception
- Must be used within the method
- Must be followed by either catch or finally block
- You can use multiple catch block with a single try

```
public class Testtrycatch2{
  public static void main(String args[]){
   try{
      int data=50/0;
   }catch(ArithmeticException e){System.out.println(e);}
   System.out.println("rest of the code...");
}
}
```

Catch multiple exceptions

- **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**

```
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...");
 }
}
```

Amal Prasad

Nested try block

- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error

```
class Excep6{
 public static void main(String args[]){
  try{
    try{
     System.out.println("going to divide");
     int b =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");
 }
}
```

finally block

- A block that is used *to execute important code* such as closing connection, stream etc
- Always executed whether exception is handled or not
- Must be followed by try or catch block
- Used to put "cleanup" code such as closing a file, closing connection etc
- **For each try block there can be zero or more catch blocks, but only one finally block**

finally block…

```
public class TestFinallyBlock2{
  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...");
  }
}
```

throw exception

- Used to explicitly throw an exception

```
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...");
  }
}
```

Exception Propagation

- An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.

Exception Propagation…

```
class TestExceptionPropagation1{
  void m(){
    int data=50/0;
  }
  void n(){
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   TestExceptionPropagation1 obj=new TestExceptionPropagation1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Java throws keyword

- Used to declare an exception
- It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained
- 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...");
  }
}
```

Amal Prasad

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.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Amal Prasad

final, finally and finalize

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

Amal Prasad

Custom Exception

- Creating your own Exception is known as custom exception or user-defined exception

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}
class TestCustomException1{

   static void validate(int age)throws InvalidAgeException{
     if(age<18)
      throw new InvalidAgeException("not valid");
     else
      System.out.println("welcome to vote");
   }

   public static void main(String args[]){
      try{
      validate(13);
      }catch(Exception m){System.out.println("Exception occured: "+m);}

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

Thanks…

Amal Prasad