Day 30

Java throws keyword

The **Java throws keyword** is 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 the normal flow of the program can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers' fault that he is not checking the code before it being used.

Syntax of Java throws

```
return_type method_name() throws exception_class_name{
//method code
}
```

Which exception should be declared?

Ans: Checked exception only, because:

- **unchecked exception:** under our control so we can correct our code.
- **error:** beyond our control. For example, we are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

Java throws Example

Let's see the example of Java throws clause which describes that checked exceptions can be propagated by throws keyword.

Testthrows1.java

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

Rule: If we are calling a method that declares an exception, we must either caught or declare the exception.

There are two cases:

1. **Case 1:** We have caught the exception i.e. we have handled the exception using try/catch block.
2. **Case 2:** We have declared the exception i.e. specified throws keyword with the method.

Case 1: Handle Exception Using try-catch block

In case we handle the exception, the code will be executed fine whether exception occurs during the program or not.

Testthrows2.java

```
  import java.io.*;
  class M{
   void method()throws IOException{
    throw new IOException("device error");
   }
  }
  public class Testthrows2{
     public static void main(String args[]){
      try{
       M m=new M();
       m.method();
      }catch(Exception e){System.out.println("exception handled");}

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

Case 2: Declare Exception

- In case we declare the exception, if exception does not occur, the code will be executed fine.
- In case we declare the exception and the exception occurs, it will be thrown at runtime because **throws** does not handle the exception.

Let's see examples for both the scenario.

A) If exception does not occur

Testthrows3.java

```
import java.io.*;
class M{
 void method()throws IOException{
  System.out.println("device operation performed");
 }
}
class Testthrows3{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
     m.method();

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

B) If exception occurs

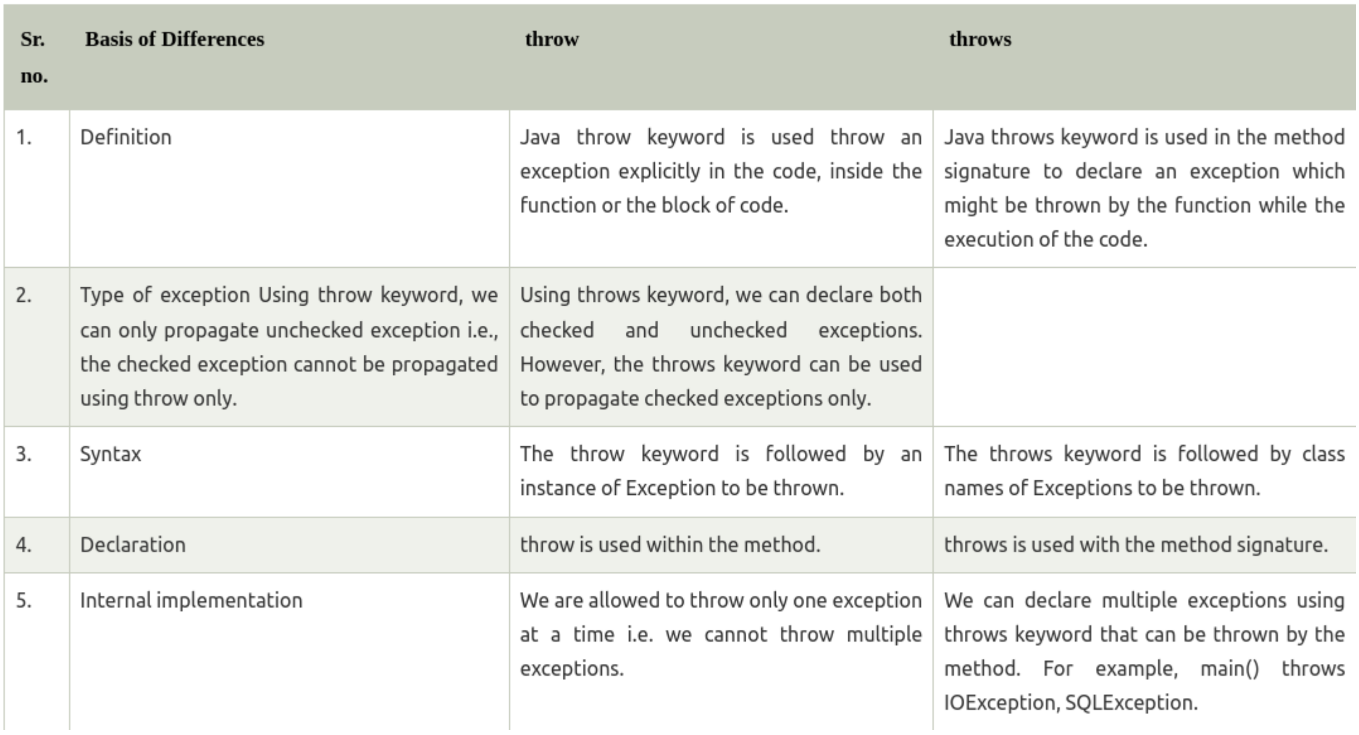
Testthrows4.java

```
import java.io.*;
class M{
 void method()throws IOException{
  throw new IOException("device error");
 }
}
class Testthrows4{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
     m.method();

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

Difference between throw and throws in Java

The throw and throws is the concept of exception handling where the throw keyword throw the exception explicitly from a method or a block of code whereas the throws keyword is used in signature of the method.

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:



Sr. no.	Basis of Differences	throw	throws
1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.	
3.	Syntax	The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

```
public class TestThrow {
    //defining a method
    public static void checkNum(int num) {
        if (num < 1) {
            throw new ArithmeticException("\nNumber is negative, cannot calculate square");
        }
        else {
            System.out.println("Square of " + num + " is " + (num*num));
        }
    }
    //main method
    public static void main(String[] args) {
            TestThrow obj = new TestThrow();
            obj.checkNum(-3);
            System.out.println("Rest of the code..");
    }
}
```

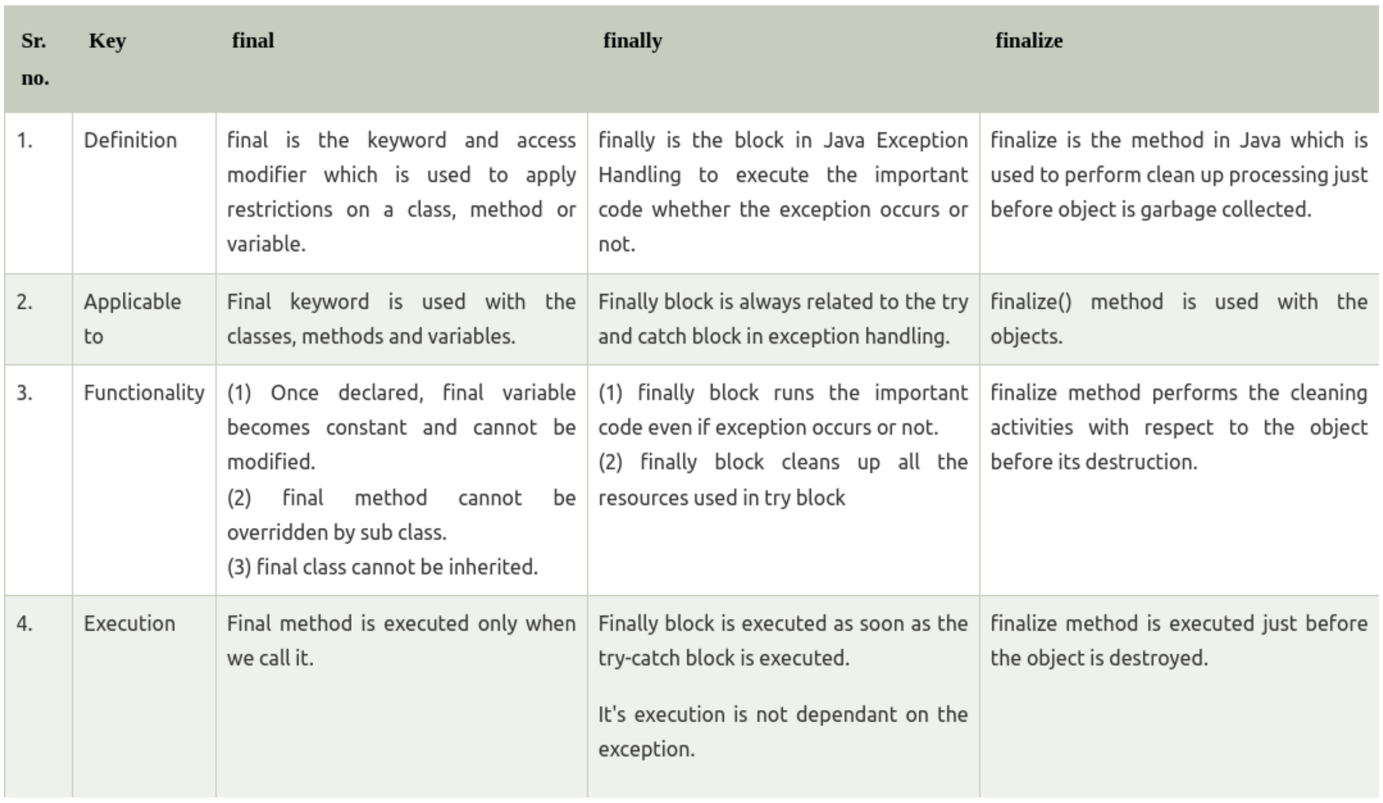
```
public class TestThrows {
    //defining a method
    public static int divideNum(int m, int n) throws ArithmeticException {
        int div = m / n;
        return div;
    }
    //main method
    public static void main(String[] args) {
        TestThrows obj = new TestThrows();
        try {
            System.out.println(obj.divideNum(45, 0));
        }
        catch (ArithmeticException e){
            System.out.println("\nNumber cannot be divided by 0");
        }

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

```
public class TestThrowAndThrows
{
    // defining a user-defined method
    // which throws ArithmeticException
    static void method() throws
ArithmeticException
        {
            System.out.println("Inside the
method()");
            throw new
ArithmeticException("throwing
ArithmeticException");
        }
        //main method
        public static void main(String args[])
        {
            try
            {
                method();
            }
            catch(ArithmeticException e)
            {
                System.out.println("caught in main()
method");
            }
        }
    }
```

Difference between final, finally and finalize

The final, finally, and finalize are keywords in Java that are used in exception handling. Each of these keywords has a different functionality. The basic difference between final, finally and finalize is that the **final** is an access modifier, **finally** is the block in Exception Handling and **finalize** is the method of object class.

Along with this, there are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:



Sr. no.	Key	final	finally	finalize
1.	Definition	final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
2.	Applicable to	Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
3.	Functionality	(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
4.	Execution	Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependant on the exception.	finalize method is executed just before the object is destroyed.

Java final Example

Let's consider the following example where we declare final variable age. Once declared it cannot be modified.

FinalExampleTest.java

```
public class FinalExampleTest {
    //declaring final variable
    final int age = 18;
    void display() {

    // reassigning value to age variable
    // gives compile time error
    age = 55;
    }

    public static void main(String[] args) {

    FinalExampleTest obj = new FinalExampleTest();
    // gives compile time error
    obj.display();
    }
}
```

In the above example, we have declared a variable final. Similarly, we can declare the methods and classes final using the final keyword.

Java finally Example

Let's see the below example where the Java code throws an exception and the catch block handles that exception. Later the finally block is executed after the try-catch block. Further, the rest of the code is also executed normally.

```java
public class FinallyExample {
      public static void main(String args[]){
      try {
        System.out.println("Inside try block");
      // below code throws divide by zero exception
       int data=25/0;
       System.out.println(data);
      }
      // handles the Arithmetic Exception / Divide by zero exception
      catch (ArithmeticException e){
        System.out.println("Exception handled");
        System.out.println(e);
      }
      // executes regardless of exception occurred or not
      finally {
        System.out.println("finally block is always executed");
      }
      System.out.println("rest of the code...");
      }
    }
```

```java
    public class FinalizeExample {
         public static void main(String[] args)
        {
            FinalizeExample obj = new FinalizeExample();
            // printing the hashcode
            System.out.println("Hashcode is: " + obj.hashCode());
            obj = null;
            // calling the garbage collector using gc()
            System.gc();
            System.out.println("End of the garbage collection");
        }
       // defining the finalize method
        protected void finalize()
        {
            System.out.println("Called the finalize() method");
        }
    }
```