EXCEPTION HANDLING

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can define our own set of conditions or rules and throw an exception explicitly using throw keyword. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions. For example, we can throw ArithmeticException when we divide number by 5, or any other numbers, what we need to do is just set the condition and throw any exception using throw keyword.

throw new exception_class("error message");

```
throw Instance
```

But this exception i.e, *Instance* must be of type **Throwable** or a subclass of **Throwable**. For example Exception is a sub-class of Throwable and user defined exceptions typically extend Exception class.

throw new ArithmeticException("dividing a number by 5 is not allowed in this program");

The flow of execution of the program stops immediately after the throw statement is executed and the nearest enclosing **try** block is checked to see if it has a **catch** statement that matches the type of exception. If it finds a match, control is transferred to that statement otherwise next enclosing **try** block is checked and so on. If no matching **catch** is found then the default exception handler will halt the program.

Example of throw keyword

Lets say we have a requirement where we need to only register the students when their age is less than 12 and weight is less than 40, if any of the condition is not met then the user should get an ArithmeticException with the warning message “Student is not eligible for registration”. We have implemented the logic by placing the code in the method that checks student eligibility if the entered student age and weight doesn’t met the criteria then we throw the exception using throw keyword.

```
/* In this program we are checking the Student age
 * if the student age<12 and weight <40 then our program
 * should return that the student is not eligible for registration.
 */
public class ThrowExample {
   static void checkEligibilty(int stuage, int stuweight){
      if(stuage<12 && stuweight<40) {
         throw new ArithmeticException("Student is not eligible for registration");
```

```
    }
    else {
      System.out.println("Student Entry is Valid!!");
    }
  }

  public static void main(String args[]){
    System.out.println("Welcome to the Registration process!!");
    checkEligibilty(10, 39);
    System.out.println("Have a nice day..");
 }
}
```

Output:

```
Welcome to the Registration process!!Exception in thread "main"
java.lang.ArithmeticException: Student is not eligible for registration
```

```
// Java program that demonstrates the use of throw
class ThrowExcep
{
	static void fun()
	{
		try
		{
			throw new NullPointerException("demo");
		}
		catch(NullPointerException e)
		{
			System.out.println("Caught inside fun().");
			throw e; // rethrowing the exception
		}
	}

	public static void main(String args[])
	{
		try
		{
			fun();
		}
		catch(NullPointerException e)
		{
			System.out.println("Caught in main.");
		}
	}
}
```

Throws clause in java

As we know that there are two types of exception checked and unchecked. Checked exception (compile time) force you to handle them, if you don't handle them then the program will not compile.
On the other hand unchecked exception (Runtime) doesn't get checked during compilation. Throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block. **Throws keyword** is used for handling checked exceptions . By using throws we can declare multiple exceptions in one go.

Syntax:

type method_name(parameters) throws exception_list
exception_list is a comma separated list of all the
exceptions which a method might throw.

In a program, if there is a chance of rising an exception then compiler always warns us about it and compulsorily we should handle that checked exception, Otherwise we will get compile time error saying **unreported exception XXX must be caught or declared to be thrown**. To prevent this compile time error we can handle the exception in two ways:

1. By using try catch
2. By using **throws** keyword

We can use throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then caller method is responsible to handle that exception.

What is the need of having throws keyword when you can handle exception using try-catch?

We already know we can handle exceptions using try-catch block.
The throws does the same thing that try-catch does but there are some cases where you would prefer throws over try-catch. For example:
Lets say we have a method myMethod() that has statements that can throw either ArithmeticException or NullPointerException, in this case you can use try-catch as shown below:

```
public void myMethod()
{
 try {
  // Statements that might throw an exception
 }
 catch (ArithmeticException e) {
  // Exception handling statements
 }
 catch (NullPointerException e) {
  // Exception handling statements
 }
}
```

But suppose you have several such methods that can cause exceptions, in that case it would be tedious to write these try-catch for each method. The code will become unnecessary long and will be less-readable.

One way to overcome this problem is by using throws like this: declare the exceptions in the method signature using throws and handle the exceptions where you are calling this method by using try-catch.
Another advantage of using this approach is that you will be forced to handle the exception when you call this method, all the exceptions that are declared using throws, must be handled where you are calling this method else you will get compilation error.

```
public void myMethod() throws ArithmeticException, NullPointerException
```

```
{
  // Statements that might throw an exception
}

public static void main(String args[]) {
  try {
    myMethod();
  }
  catch (ArithmeticException e) {
    // Exception handling statements
  }
  catch (NullPointerException e) {
    // Exception handling statements
  }
}
```

Example of throws Keyword

In this example the method myMethod() is throwing two **checked exceptions** so we have declared these exceptions in the method signature using **throws** Keyword. If we do not declare these exceptions then the program will throw a compilation error.

```
import java.io.*;
class ThrowExample {
  void myMethod(int num)throws IOException, ClassNotFoundException{
     if(num==1)
        throw new IOException("IOException Occurred");
     else
        throw new ClassNotFoundException("ClassNotFoundException");
  }
}

public class Example1{
  public static void main(String args[]){
   try{
     ThrowExample obj=new ThrowExample();
     obj.myMethod(1);
   }catch(Exception ex){
     System.out.println(ex);
    }
  }
}
```

Output:

```
java.io.IOException: IOException Occurred
```

```
// Java program to demonstrate working of throws
class ThrowsExecp
{
	static void fun() throws IllegalAccessException
	{
		System.out.println("Inside fun(). ");
		throw new IllegalAccessException("demo");
	}
	public static void main(String args[])
	{
		try
		{ fun(); }

		catch(IllegalAccessException e)
		{ System.out.println("caught in main."); }
	}
}
```

Throw vs Throws

1. **Throws clause** is used to declare an exception, which means it works similar to the try-catch block. On the other hand **throw** keyword is used to throw an exception explicitly.

2. If we see syntax wise then **throw** is followed by an instance of Exception class and **throws** is followed by exception class names.
For example:

```
throw new ArithmeticException("Arithmetic Exception");
```

and

```
throws ArithmeticException;
```

3. Throw keyword is used in the method body to throw an exception, while throws is used in method signature to declare the exceptions that can occur in the statements present in the method.

For example:

Throw:

```
...
void myMethod() {
  try {
    //throwing arithmetic exception using throw
    throw new ArithmeticException("Something went wrong!!");
  }
  catch (Exception exp) {
    System.out.println("Error: "+exp.getMessage());
  }
}
...
```

Throws:

```
...
//Declaring arithmetic exception using throws
void sample() throws ArithmeticException{
  //Statements
}
...
```

4. You can throw one exception at a time but you can handle multiple exceptions by declaring them using throws keyword.
For example:

Throw:

```
void myMethod() {
  //Throwing single exception using throw
  throw new ArithmeticException("An integer should not be divided by zero!!");
}
..
```

Throws:

```
//Declaring multiple exceptions using throws
void myMethod() throws ArithmeticException, NullPointerException{
  //Statements where exception might occur
}
```
These were the main **differences between throw and throws in Java**. Lets see complete examples of throw and throws keywords.

Throw Example

```
public class Example1{
  void checkAge(int age){
        if(age<18)
          throw new ArithmeticException("Not Eligible for voting");
        else
          System.out.println("Eligible for voting");
  }
  public static void main(String args[]){
        Example1 obj = new Example1();
        obj.checkAge(13);
        System.out.println("End Of Program");
  }
}
```
Output:

```
Exception in thread "main" java.lang.ArithmeticException:
Not Eligible for voting
at Example1.checkAge(Example1.java:4)
at Example1.main(Example1.java:10)
```

Throws Example

```
public class Example1{
  int division(int a, int b) throws ArithmeticException{
        int t = a/b;
        return t;
  }
  public static void main(String args[]){
        Example1 obj = new Example1();
        try{
          System.out.println(obj.division(15,0));
        }
        catch(ArithmeticException e){
          System.out.println("You shouldn't divide number by zero");
        }
  }
```

```
}
```

Output:

```
You shouldn't divide number by zero
```