Exception

Exceptions are one of the most important concepts dealt with in the Java programming language to handle the runtime errors in an efficient manner.

An exception can cause a program to break down half way through before it runs the entire code. It is undesirable and to counter it we have exception handling introduced. Few examples of exceptions are ClassNotFoundException, IOException, SQLException, etc.

If there are certain X number of statements to be executed and an error occurs before completion of this number it can be fatal for the program.

Types of Exception

There are three types of exception i.e

1. Checked Exception: Checked exceptions are checked at the compile – time. Few examples are IOException, SQLException. These classes inherit directly Throwable class except Runtime and errors.
2. Unchecked Exception: Unchecked exceptions are check at Runtime and not at compile-time. They inherit RuntimeException. Examples are ArithmeticException, NullPointerException, etc
3. Error: When an error occurs it tends to be irrecoverable. Examples are OutOfMemoryError etc.

A simple example to illustrate this would be

Example:

```
public class Test{
  public static void main(String args[]){
   try{
      int data=100/0;
   }catch(ArithmeticException e){
```

```
System.out.println("Invalid operation! Cannot divide by 0.");
}   finally{
System.out.println("Finally executes code in any case.");     }
  }
}
```

In this example by observing we can note that when an exception in this case ArithmeticException occurs it will lead to the program terminating so to avoid it we are using try and catch block. In the try block we will the run the lines given and if there's an exception instead of terminating or breaking the flow of code the catch block catches the exception and an appropriate message can be sent to the user as a feedback. The finally block is used to execute the statements like closing the file writer, etc. It is always executed when there is an exception or not.

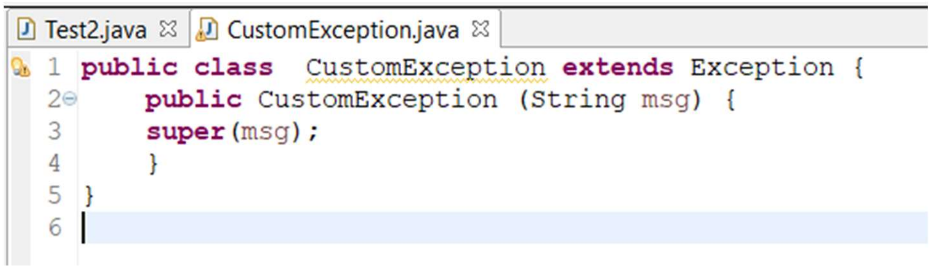
Java Custom Exception

Java Custom Exception is one of the many features that Java provides to its developers to create their own exceptions. They can customize them and so also called as User-Defined Exception.

There are many types of Exceptions already available in Java. But at times there you want to work with exceptions which are more suited to the program and so they are preferred.

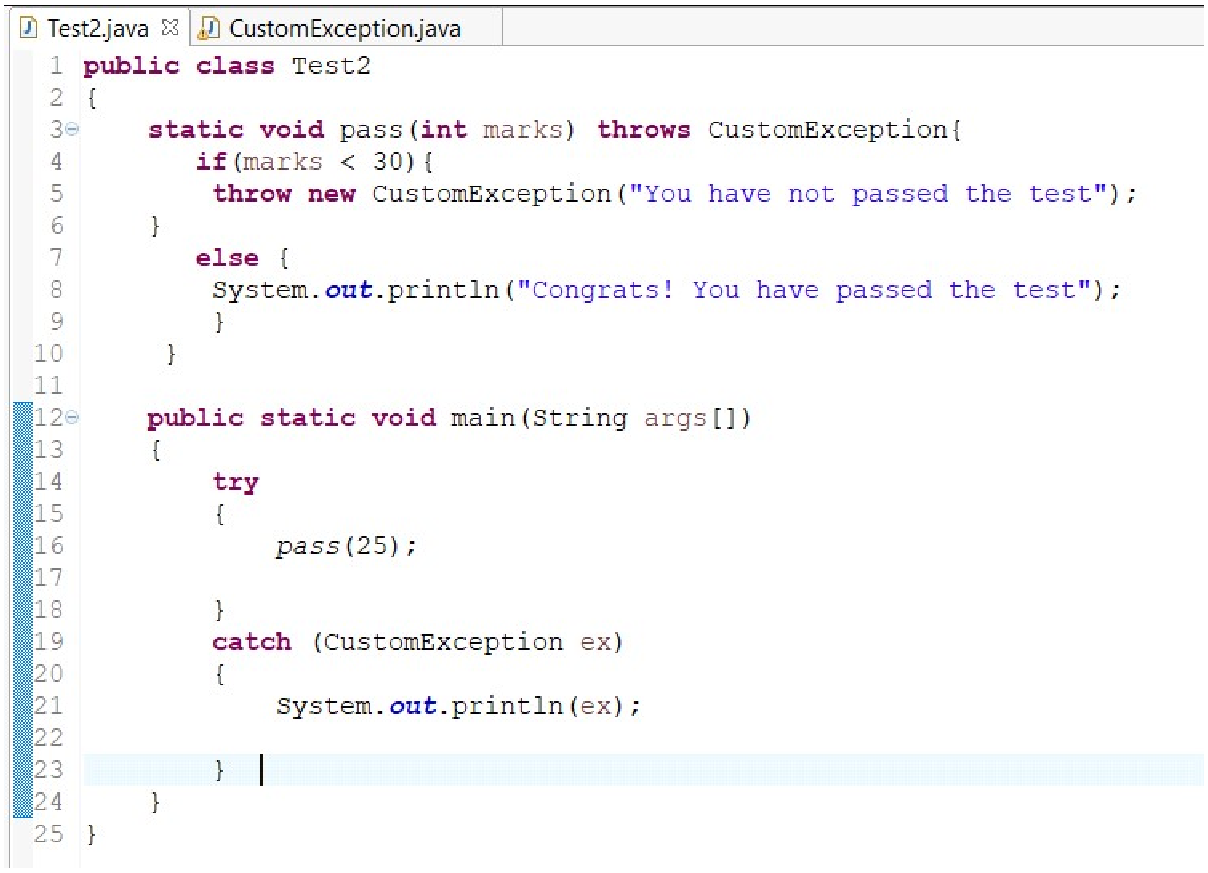
To create a custom exception it's necessary to extend the Exception class.

Example:



```
public class  CustomException extends Exception {
    public CustomException (String msg) {
    super(msg);
    }
}
```

This is the CustomException class create and it extends Exception with string parameter passed to the constructor using super() method.

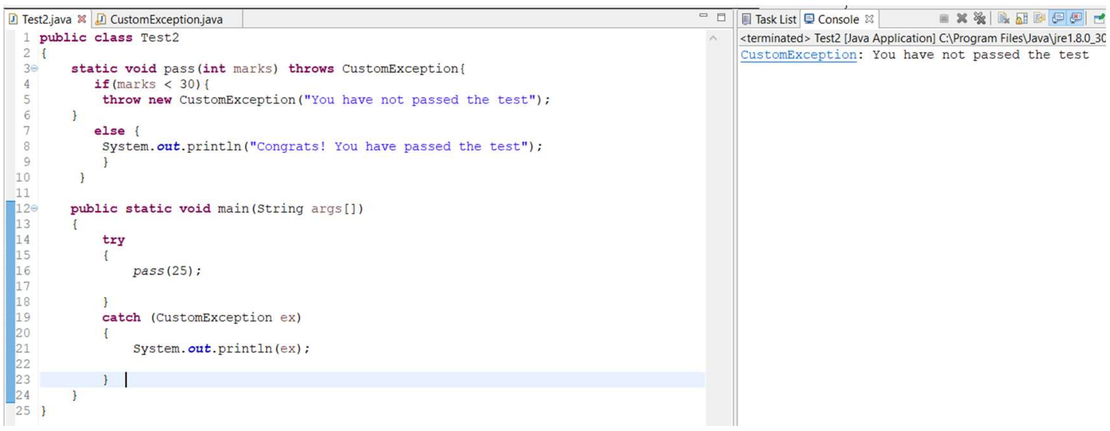


Test2.java CustomException.java

```java
public class Test2
{
    static void pass(int marks) throws CustomException{
       if(marks < 30){
        throw new CustomException("You have not passed the test");
    }
       else {
        System.out.println("Congrats! You have passed the test");
        }
     }

    public static void main(String args[])
    {
        try
        {
            pass(25);

        }
        catch (CustomException ex)
        {
            System.out.println(ex);

        }
    }
}
```

Here, we have create a Test2 class and in it there's a static void function 'pass' with int marks as a parameter and the throws keyword along with CustomException. This function is used to check if a student has passed the test or not. If the student fails the test we throw an exception by using throw new keywords and the message passed in CustomException. Else we print a line in console congratulating them.

The try and catch block is used to do this in main function and a parameter is passed through it. In first case 25, because it's less than 30 which is the criteria mentioned in the function so it throw an exception and the catch block will take over and execute the statement in it.

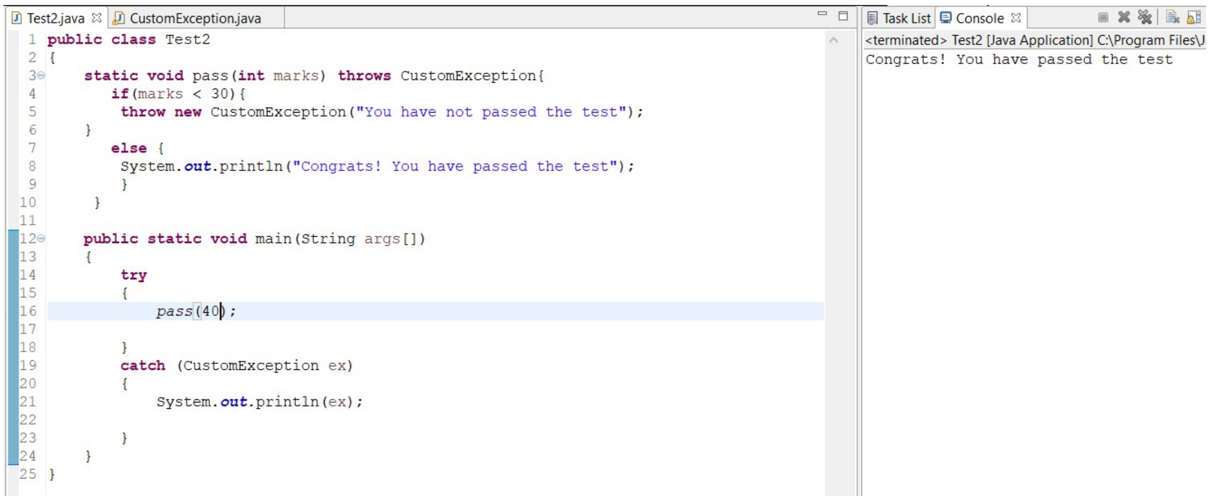


Test2.java CustomException.java

```java
public class Test2
{
    static void pass(int marks) throws CustomException{
       if(marks < 30){
        throw new CustomException("You have not passed the test");
    }
       else {
        System.out.println("Congrats! You have passed the test");
        }
     }

    public static void main(String args[])
    {
        try
        {
            pass(25);

        }
        catch (CustomException ex)
        {
            System.out.println(ex);

        }
    }
}
```

Task List Console

```
<terminated> Test2 [Java Application] C:\Program Files\Java\jre1.8.0_30
CustomException: You have not passed the test
```



Test2.java | CustomException.java

```java
public class Test2
{
    static void pass(int marks) throws CustomException{
        if(marks < 30){
         throw new CustomException("You have not passed the test");
    }
        else {
         System.out.println("Congrats! You have passed the test");
         }
     }

    public static void main(String args[])
    {
        try
        {
            pass(40);

        }
        catch (CustomException ex)
        {
            System.out.println(ex);

        }
    }
}
```

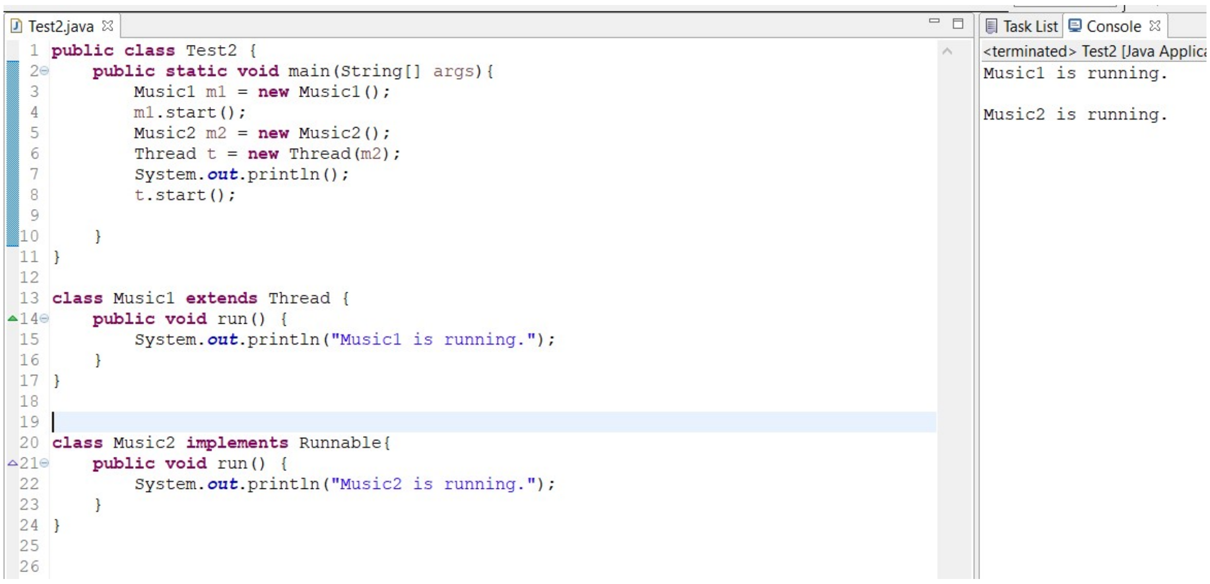
Task List | Console

```
<terminated> Test2 [Java Application] C:\Program Files\J
Congrats! You have passed the test
```

In case 2, we try 40 as the parameter we get a congratulatory message.

Thread

Thread allows a program to perform multiple things at the same time. This can be performed by using the Thread Class provided. It contains constructors and methods to carry out the process. The object classes are extended by the Thread class and they implement the Runnable interface.



Test2.java

```java
public class Test2 {
    public static void main(String[] args){
        Music1 m1 = new Music1();
        m1.start();
        Music2 m2 = new Music2();
        Thread t = new Thread(m2);
        System.out.println();
        t.start();

    }
}

class Music1 extends Thread {
    public void run() {
        System.out.println("Music1 is running.");
    }
}


class Music2 implements Runnable{
    public void run() {
        System.out.println("Music2 is running.");
    }
}
```

Task List | Console

```
<terminated> Test2 [Java Applica
Music1 is running.

Music2 is running.
```

There are two cases to consider here the first one is of class Music1 where we have extended Thread and added a method in it. When we try to start the thread we get an output after creating an instance m1 in the main function. But in the second where we have implemented Runnable to class Music2 there we can’t just create an instance of m2 and use the start method. But, we are supposed to create a Thread instance t and pass m2 as the parameter after which we can use the method start as t.start() as shown to get the desired output. In case to the class is not considered a thread so it’s not eligible to perform start and we have to follow the steps.