# OBJECT ORIENTED PROGRAMMING (OOP)

## ERRORS!

In software industrial programming most of the programs contain bugs. Bigger the program greater number of bugs it contains.

The following are mainly errors or bugs that occurred in any program:

- **Logical error:**
  In this error the logic implemented is incorrect. This error occurs due to less concentration of programmer or poor knowledge of programmer regarding program.
- **Compilation error:**
  This error is occurred due to use of wrong idiom, function or structure. This error is shown at compilation time of program.
- **Run Time error:**
  This error occurred at the run time. This error occurs when program crashes during run time..

## EXCEPTIONS!

An exception is a situation, which occurred by the runtime error. In other words, an exception is a runtime error.

**Exceptions are different**, however. You can't eliminate exceptional circumstances; you can only prepare for them. Your users will run out of memory from time to time, and the only question is what you will do. Your choices are limited to these:

- Crash the program.
- Inform the user and exit gracefully.
- Inform the user and allow the user to try to recover and continue.
- Take corrective action and continue without disturbing the user.

## EXCEPTION HANDLING IN JAVA

statement

statement

statement

exception ……an exception occurred, then JVM will handle it and will exit the prog.

statement

statement

statement

## EXCEPTION HANDLING IN JAVA (CONT..)

For handling exceptions, there are **2 possible approaches**

**1. JVM**

If an exception is not handled explicitly, then JVM takes the responsibility of handling the exception.

Once the exception is handled, JVM will halt the program and no more execution of code will take place

## EXCEPTION HANDLING IN JAVA (CONT..)

**1. JVM - Example:**

```
import java.util.*;

class Main {
   public static void main (String[] args) {
      System.out.println(5/0);
      System.out.println("End of program!");
      }
}
```

**Runtime Error:**

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Main.main(File.java:5)
```

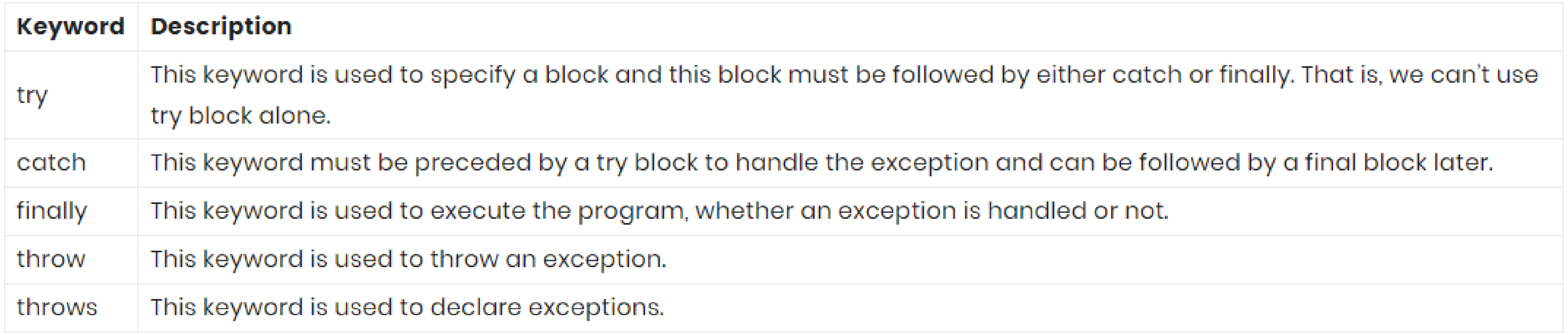
## EXCEPTION HANDLING IN JAVA (CONT..)

**2. Developer**

Developers can explicitly write the implementation for handling the exception. Once an exception is handled, the normal execution of code will continue.

**Preferable:** handle exceptions to ensure your code gets executed normally.

## JAVA EXCEPTION KEYWORDS

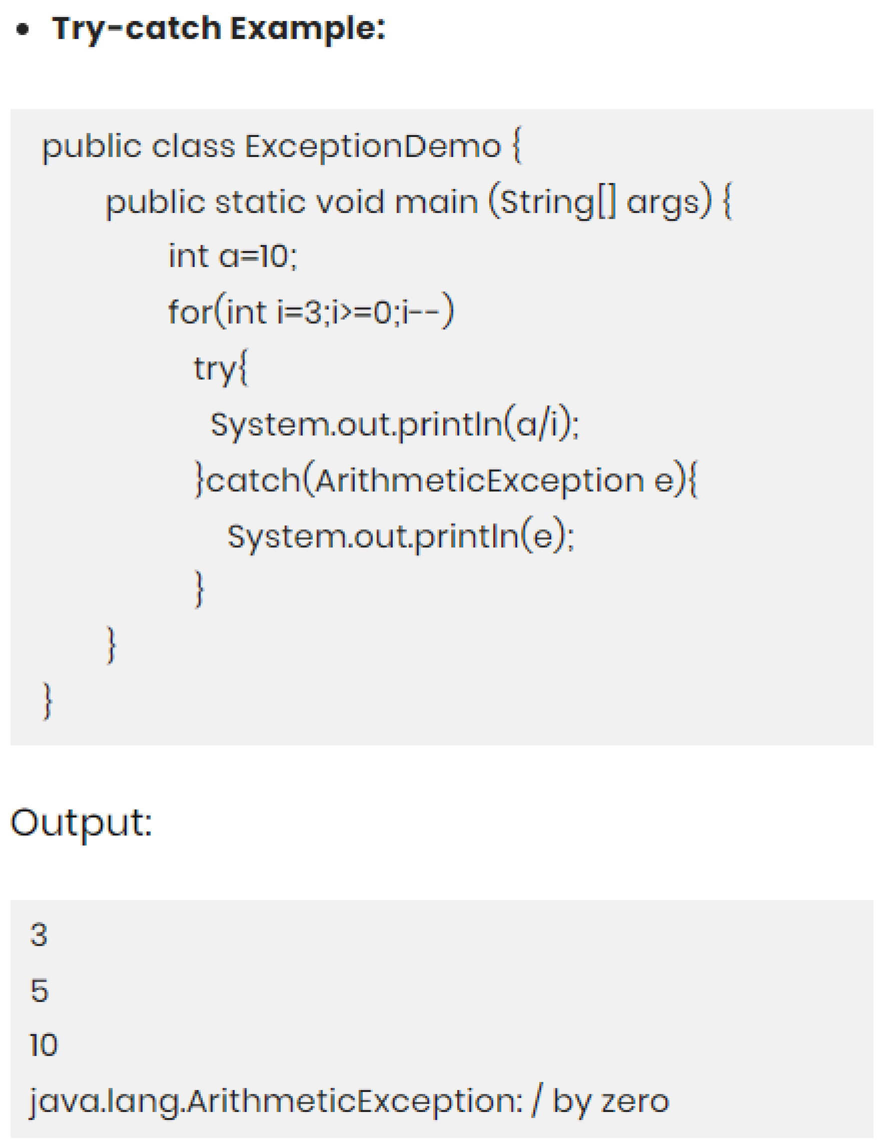


| Keyword | Description |
|---|---|
| try | This keyword is used to specify a block and this block must be followed by either catch or finally. That is, we can't use try block alone. |
| catch | This keyword must be preceded by a try block to handle the exception and can be followed by a final block later. |
| finally | This keyword is used to execute the program, whether an exception is handled or not. |
| throw | This keyword is used to throw an exception. |
| throws | This keyword is used to declare exceptions. |

# JAVA TRY-CATCH BLOCK

Try-catch syntax:

```
try{
}
catch(Exception e){
}
```



- **Try-catch Example:**

```
public class ExceptionDemo {
    public static void main (String[] args) {
        int a=10;
        for(int i=3;i>=0;i--)
          try{
            System.out.println(a/i);
          }catch(ArithmeticException e){
              System.out.println(e);
          }
    }
}
```

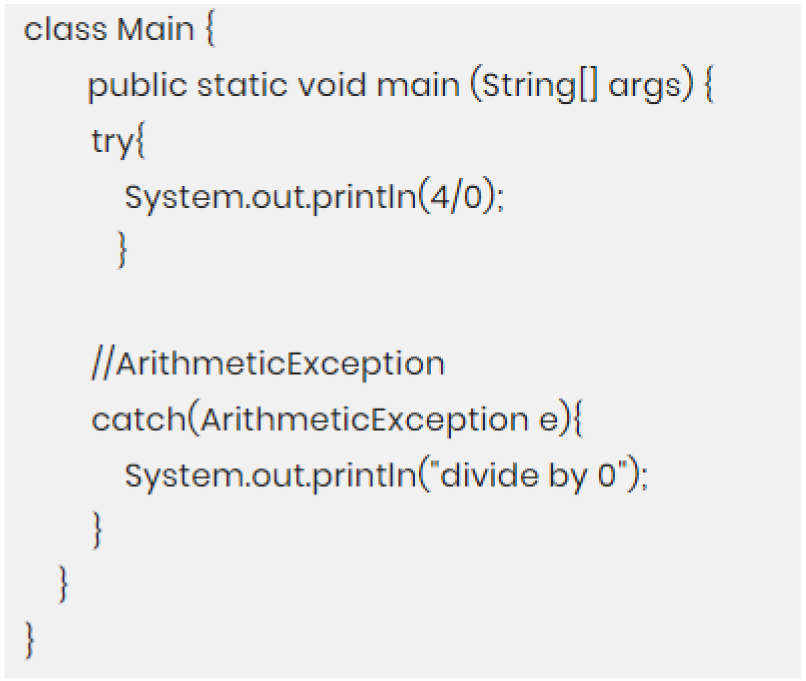
Output:

```
3
5
10
java.lang.ArithmeticException: / by zero
```

# OBJECTS IN CATCH BLOCK

1. Exact Exception

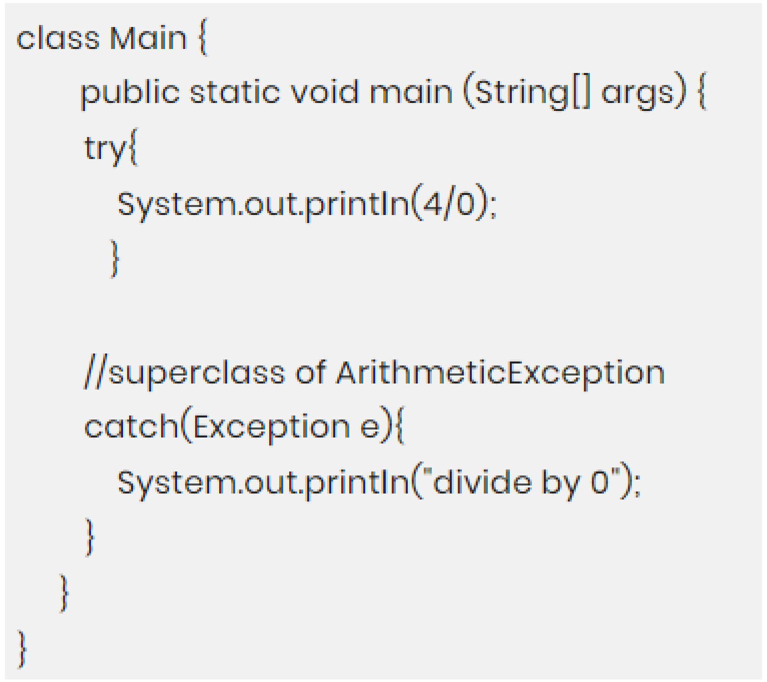
Exact Exception



```
class Main {
      public static void main (String[] args) {
      try{
         System.out.println(4/0);
        }

      //ArithmeticException
      catch(ArithmeticException e){
         System.out.println("divide by 0");
      }
   }
}
```

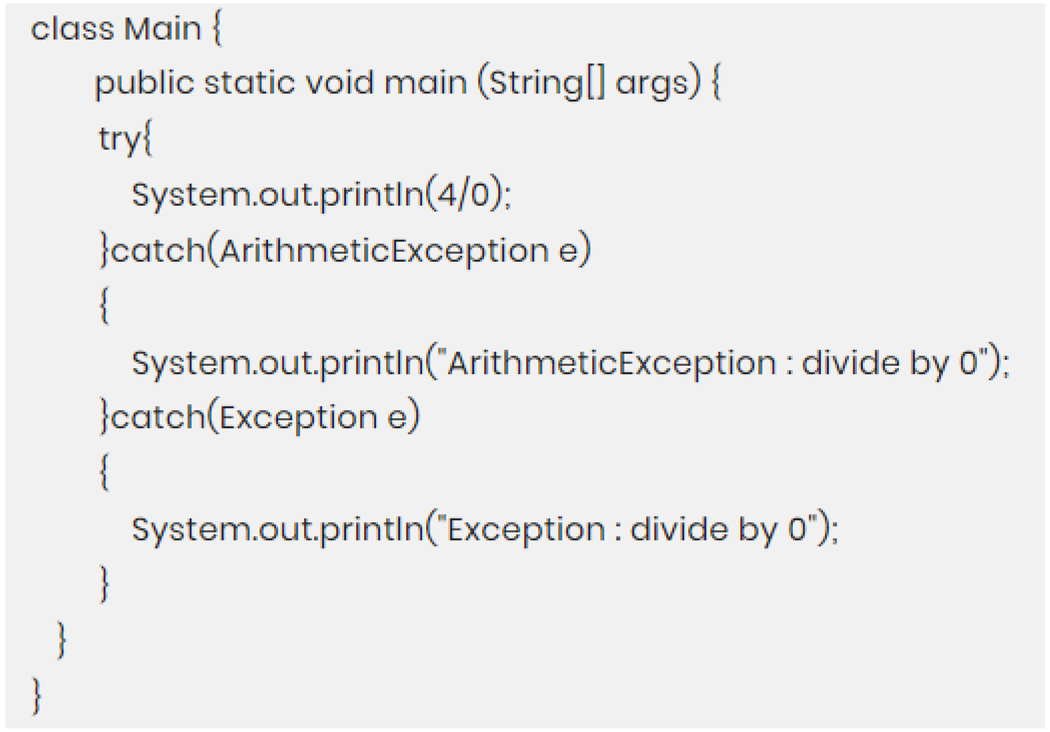
2. Superclass of



```
class Main {
      public static void main (String[] args) {
      try{
         System.out.println(4/0);
        }

      //superclass of ArithmeticException
      catch(Exception e){
         System.out.println("divide by 0");
      }
   }
}
```

## JAVA MULTIPLE CATCH BLOCK



```
class Main {
     public static void main (String[] args) {
     try{
        System.out.println(4/0);
     }catch(ArithmeticException e)
     {
        System.out.println("ArithmeticException : divide by 0");
     }catch(Exception e)
     {
        System.out.println("Exception : divide by 0");
     }
  }
}
```

## JAVA NESTED TRY

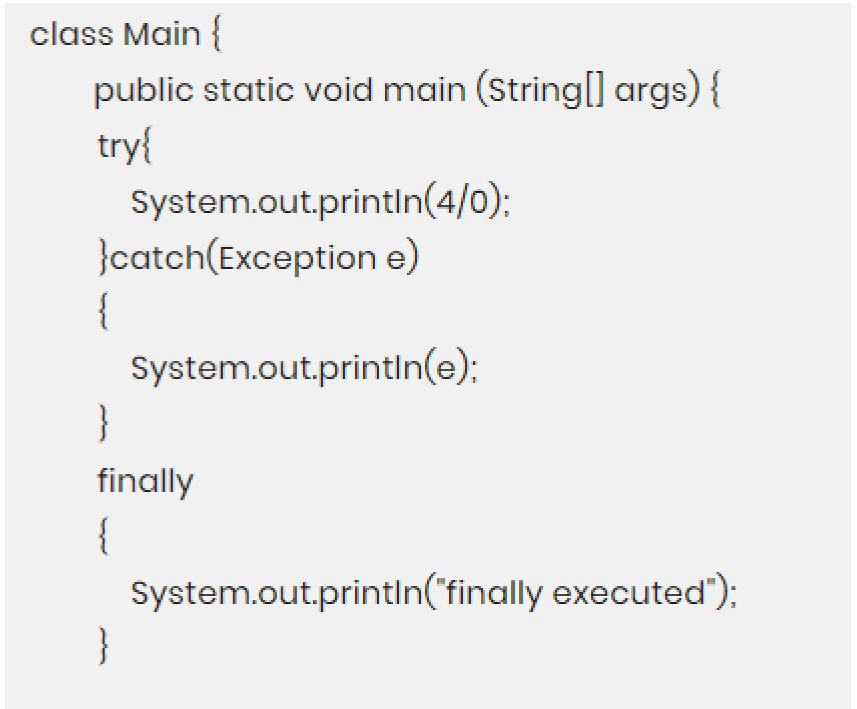
```
class Main {
    public static void main (String[] args) {
    try{
        try{
          int[] a={1,2,3};
          System.out.println(a[3]);
        }
  catch(ArrayIndexOutOfBoundsException e)
        {
          System.out.println("Out of bounds");
        }
       System.out.println(4/0);
    }
   catch(ArithmeticException e)
   {
      System.out.println("ArithmeticException : divide by 0");
   }
   }
 }
```

**Output:**

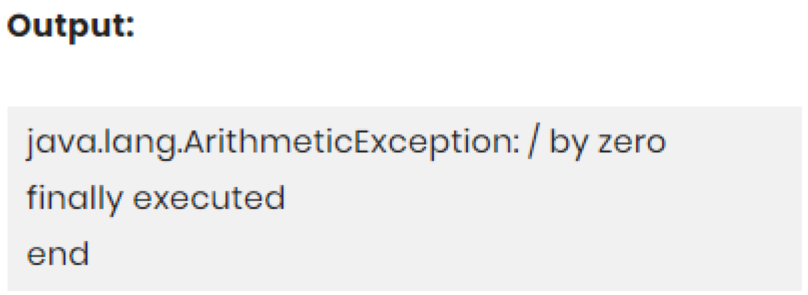
```
Out of bounds
ArithmeticException: Divide by 0
```

## JAVA FINALLY BLOCK

Contains code that must be executed no matter if an exception is thrown or not. It contains code of file release, closing connections, etc.



```
class Main {
      public static void main (String[] args) {
      try{
         System.out.println(4/0);
      }catch(Exception e)
      {
         System.out.println(e);
      }
      finally
      {
         System.out.println("finally executed");
      }
```



**Output:**

```
java.lang.ArithmeticException: / by zero
finally executed
end
```

## JAVA THROW KEYWORD

It is a keyword that is used to explicitly throw an exception.

We can use throw where according to our logic an exception should occur.

## EXAMPLE

```
public class ExceptionDemo {
    static void canVote(int age){
        if(age<18)
      try{
         throw new Exception();
      }catch(Exception e){
         System.out.println("you are not an adult!");
      }
        else
          System.out.println("you can vote!");
    }
    public static void main (String[] args) {
        canVote(20);
        canVote(10);
    }
}
```

**Output:**

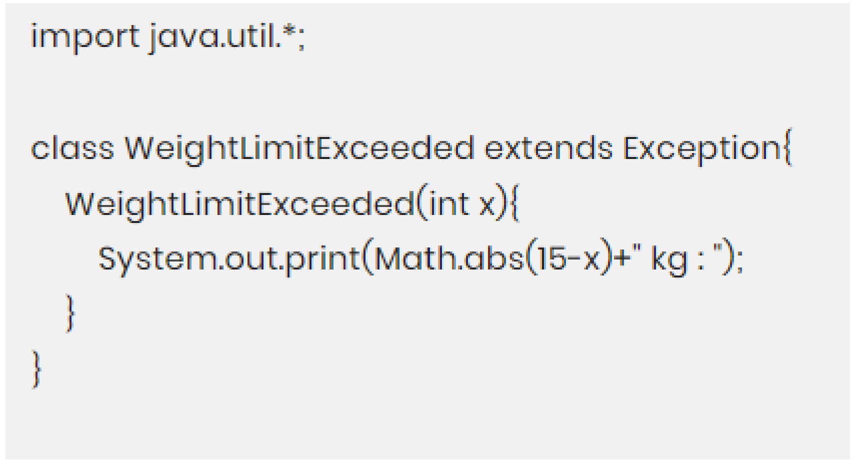
```
you can vote!
you are not an adult!
```

## JAVA CUSTOM EXCEPTION

You can create your own exception and give implementation as to how it should behave. Your exception will behave like a child's class of Exception.

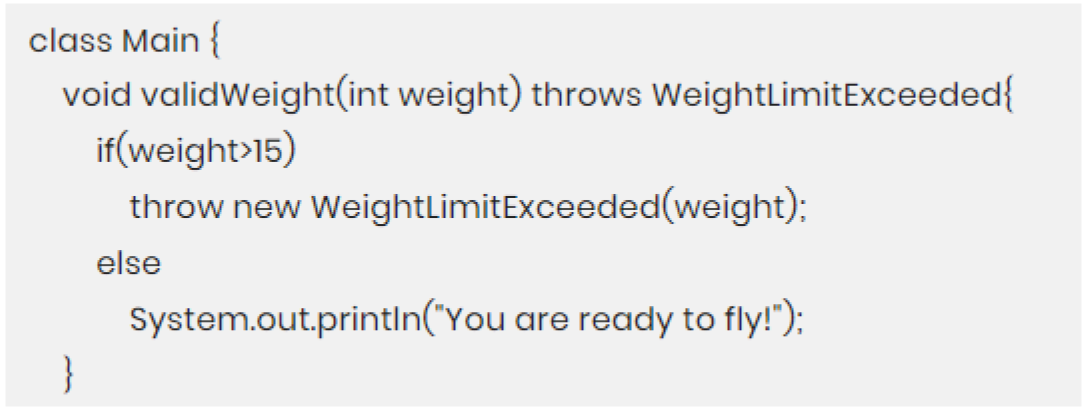
**class YourException extends Exception{}**

## EXAMPLE



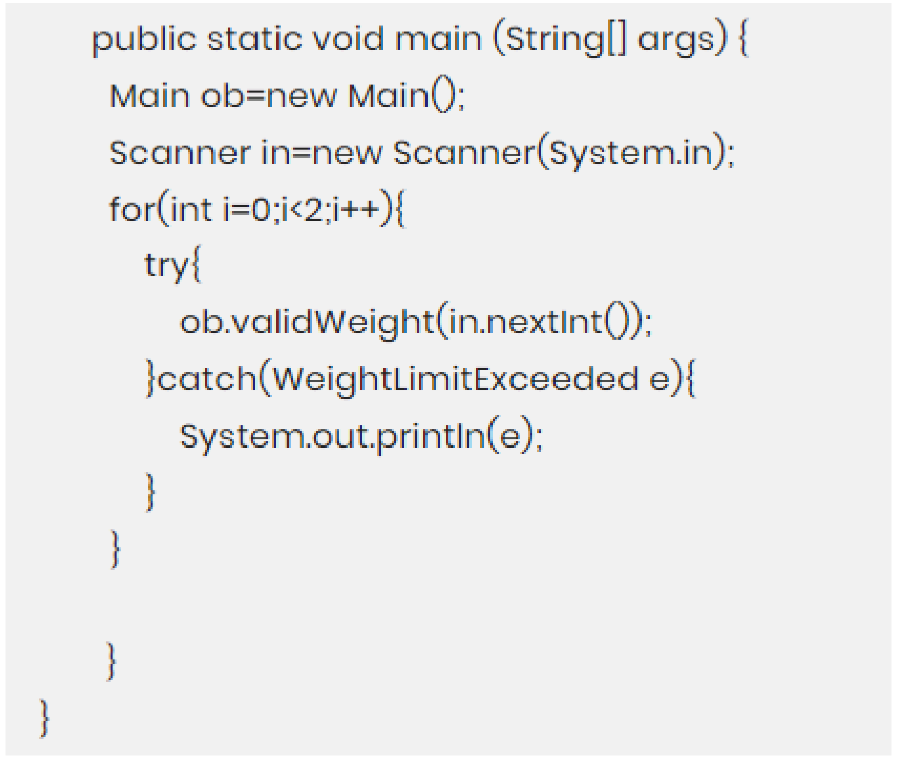
```
import java.util.*;

class WeightLimitExceeded extends Exception{
    WeightLimitExceeded(int x){
        System.out.print(Math.abs(15-x)+" kg : ");
    }
}
```



```
class Main {
    void validWeight(int weight) throws WeightLimitExceeded{
        if(weight>15)
            throw new WeightLimitExceeded(weight);
        else
            System.out.println("You are ready to fly!");
    }
```

# EXAMPLE(CONT..)



```
      public static void main (String[] args) {
        Main ob=new Main();
        Scanner in=new Scanner(System.in);
        for(int i=0;i<2;i++){
          try{
            ob.validWeight(in.nextInt());
          }catch(WeightLimitExceeded e){
            System.out.println(e);
          }
        }


      }
  }
```