Throws Keyword

Definition:

The Java *throws* keyword is used to declare the exception information that may occur during the program execution. It gives information about the exception to the programmer.

Syntax:

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

Need for *throws* keyword:

In a program, if there is a chance of raising an exception then the compiler always warns us about it and compulsorily we should handle that checked exception. Otherwise we will get a compile time error saying an unreported exception must be caught or declared to be thrown.

To prevent this compile time error we can handle the exception in two ways:

- By using try catch
- By using *throws* keyword

Usage of *throws* keyword:

1. *throws* keyword is required only for checked exceptions and usage of *throws* keyword for unchecked exception is meaningless. Because
    a. Unchecked exceptions are under programmers control. Programmer should correct his code.
    b. Errors are beyond the control of the programmers. Eg: Stackoverflow errors or Virtual Machine Errors
2. *throws* keyword is required only to convince compiler and usage of *throws* keyword does not prevent abnormal termination of program.

3. By the help of *throws* keyword we can provide information to the caller of the method about the exception.
4. We can rethrow the exception. Used to propagate checked exceptions.

Important points to remember:

1. *throws* is mainly used to throw checked exceptions.
2. If you are calling a method that declares an exception, you must either catch or declare the exception.

Examples:

1. When exception is not handled:

```
class Main
{
    public static void main(String[] args)
    {
        Thread.sleep(10000);//Interrupted exception occurs and it is not handled causing abnormal termination
        System.out.println("Hello World");
    }
}
```

OUTPUT

error: unreported exception InterruptedException; must be caught or declared to be thrown

Explanation:

Exceptions should be handled by using either *try-catch* block or *throws* keyword. Here exception is not handled so the program terminates abruptly.

2. Exception handled using *throws* keyword:

```
class Main
```

```
{
    //Exception handled using throws keyword
    public static void main(String[] args)throws InterruptedException
    {
        Thread.sleep(10000);//Interrupted exception occurs
        System.out.println("Hello World");
    }
}
```

OUTPUT

Hello World

Explanation:

InterruptedException is handled by using *throws* keyword. So the program runs smoothly without abrupt termination.
When an exception is handled, the program runs normally even if the exception occurs or not.

3. Exception is declared but exception not occurred:

```
import java.io.*;

class M{
    void method()throws IOException
    {
    //Exception does not occur
        System.out.println("device operation performed");
    }
}
class Main{
   public static void main(String args[])throws IOException
   {
//declare exception
     M m=new M();
     m.method();
     System.out.println("normal flow...");
  }
}
```

OUTPUT

device operation performed

normal flow...

Explanation:

Exception is handled by using *throws* keyword. But there is no exception. The program runs normally.

4. Exception is declared but exception not occurred:

```
import java.io.*;

class M{
    void method()throws IOException
    {
        throw new IOException("device error");
    }
}
class Main{
   public static void main(String args[])throws IOException
   {
    //declare exception
       M m=new M();
       m.method();
       System.out.println("normal flow...");
  }
}
```

OUTPUT

normal flow. . .

Explanation:

IO Exception occurs in the program. But the program does not terminate abnormally because *throws* keyword is used to handle the exception.

5. Propagation of checked exception:

```
import java.io.IOException;
```

```
class Test
{
    void m()throws IOException
    {
        throw new IOException("device error");//checked exception is
thrown
    }
    void n()throws IOException
    {
        m();//calling method m
        //exception is received from method m and returns to called method
p
    }
    void p()
    {
        try
        {
            n();//calling method n
            //exception is received from method n
        }
        catch(Exception e)
        {
            System.out.println("exception handled");//handling exception
        }
    }
    public static void main(String args[])
    {
        Test obj=new Test();
        obj.p();//calling method p
        System.out.println("normal flow...");
    }
}
```

OUTPUT

exception handled
normal flow...

Explanation:

In the main function, method p is called. Method p calls another method n. Method n calls another method m. m throws an exception to the method n(which calls method m). n receives the exception and throws to the method p. Method p has a try-catch block to handle exceptions. Thus, the program runs normally.