When to provide try-catch or declare the method as throws for Checked Exception :-

try-catch

We should provide try-catch if we want to handle the exception in the method where checked exception is encountered, as well as if we want to provide user-defined messages to the client.

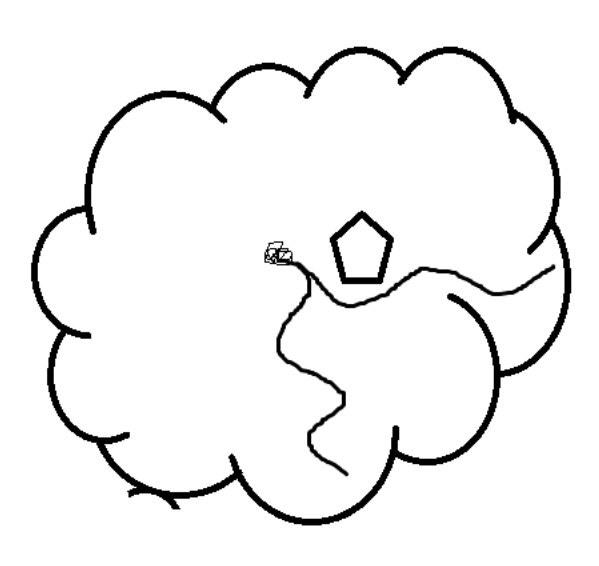
throws :
----
throws keyword describes that the method might throw an Exception, It also might not. It is used only at the end of a method declaration to indicate what exceptions it supports OR what type of Exception it might throw which will be handled by JVM OR caller method.

Note :- It is always better to use try catch so we can provide appropriate user defined messages to our client.

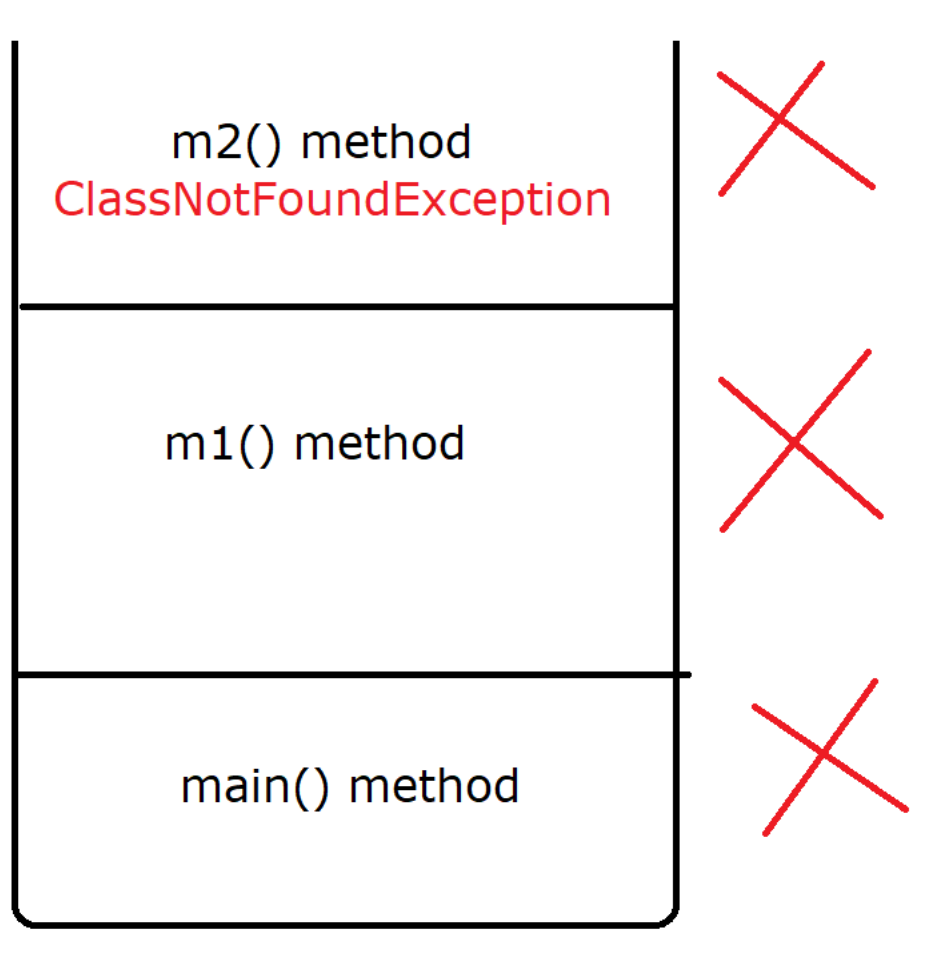
```java
package com.ravi.exception;

class Test
{
    static
    {
        System.out.println("Static Block");
    }
}

public class TryCatchORThrows
{
    public static void main(String[] args)
    {
         System.out.println("Main method started");
         m1();
         System.out.println("Main method Ended");
    }

    public static void m1()
    {
        System.out.println("M1 method started");
        try
        {
            m2();
        }
        catch(ClassNotFoundException e)
        {
            System.out.println("Handled By M1 method");
        }
        System.out.println("M1 method Ended");
    }

    public static void m2() throws ClassNotFoundException
    {
       Class.forName("Test");
    }
}
```

* In case of Checked exception, a method can declare throws (not interested to handle) but It is important that any of the caller method must handle the exception, If all the caller methods (including main) write throws keyword then default excption handler of JVM will handle the exception and terminate the program in the middle (Abnormal termination).

Exception Propagation :



Digarm :




```java
package com.ravi.exception;

class Sample
{
    static
    {
        System.out.println("Static Block");
    }
}

public class ExceptionPropagationDemoWithChecked
{
    public static void main(String[] args) throws
ClassNotFoundException
    {
      System.out.println("Main method started");
       m1();
      System.out.println("Main method Ended");

    }

    public static void m1()  throws
ClassNotFoundException
    {
        System.out.println("M1 method started");
        m2();
        System.out.println("M1 method Ended");
    }

    public static void m2() throws
ClassNotFoundException
    {
        System.out.println("m2 method started");

        Class.forName("Sample");

        System.out.println("M2 method Ended");
    }
}
```

In this program any of the caller method is not interested to handle the ClassNotFoundException so JVM will terminate all the method from Stack Frame and finally default exception handler will handle the exception to provide error meesage and program will be terminated **abnormally**

Exception Propagation Defination : [Propagation of Exception Object from Callee to Caller]

Whenever we call a method and if the the callee method contains any kind of exception (checked OR Unchecked) and if callee method doesn't contain any kind of exception handling mechanism (try-catch OR throws) then JVM will propagate the exception object to caller method for handling purpose. This is called Exception Propagation.

If the caller method also does not contain any exception handling mechanism then JVM will terminate the method from the stack frame hence the remaining part of the method(m1 method) will not be executed even if we handle the exception in another caller method like main.

If any of the the caller method does not contain any exception handling mechanism then exception will be handled by JVM, JVM has default exception handler which will provide the exception message and terminates the program abnormally.

```java
package com.ravi.exception;

class Sample
{
    static
    {
        System.out.println("Static Block");
    }
}

public class ExceptionPropagationDemoWithChecked
{
    public static void main(String[] args)
    {
      System.out.println("Main method started");
       try
       {
           m1();
       }
       catch(ClassNotFoundException e)
       {
           System.out.println("Handled by main method");
       }
      System.out.println("Main method Ended");

    }

    public static void m1()  throws ClassNotFoundException
    {
        System.out.println("M1 method started");
        m2();
        System.out.println("M1 method Ended");
    }

    public static void m2() throws ClassNotFoundException
    {
        System.out.println("m2 method started");

        Class.forName("Sample");

        System.out.println("M2 method Ended");
    }
}
```

```java
package com.ravi.exception;

public class ExceptionPropagationDemoWithUnchecked
{
      public static void main(String[] args)
      {
        System.out.println("Main method started");
        m1();
        System.out.println("Main method Ended");
      }

      public static void m1()
      {
          System.out.println("M1 method started");
          try
          {
              m2();
          }
          catch(ArithmeticException e)
          {
              System.out.println("Handled By m1 method");
          }
          System.out.println("M1 method Ended");
      }

      public static void m2()
      {
          System.out.println("m2 method started");
          System.out.println(10/0);
          System.out.println("M2 method Ended");
      }
   }
```

Note : It is strongly recommended that while working with checked exception, method uses throws keyword to skip from the current situation and try to send the exception object to the called method then any of the caller method must handle the exception.

Some important rules regarding the checked Exception :
----
a) If the try block does not throw any checked exception then in the corresponding catch block we can't handle checked exception.It will generate compilation error i.e "exception never thrown from the corresponding try statement"

Example :-

```java
public class Test
{
   public static void main(String[] args)
   {
      try
      {
            //try block is not throwing checked exception
         //i.e. InterruptedException
      }
      catch (InterruptedException e) //error
      {
      }
   }
}
```

Note :- The above rule is not applicable for Unchecked Exception

```java
              try
      {

      }
      catch(ArithmeticException e)  //Valid
      {
         e.printStackTrace();
      }
```
----
b) If the try block does not throw any exception then in the corresponding catch block we can write Exception OR Throwable because both are the super classes for all types of Exception whether it is checked or unchecked.

```java
package com.ravi.method_related_rule;

public class CatchingWithSuperClass
{
    public static void main(String[] args)
    {
        try
        {

        }
        catch(Throwable e)  //Exception and Throwable both are allowed
        {
           e.printStackTrace();
        }
    }
}
```