c) At the time of method overriding if the super class method does not reporting or throwing checked exception then the overridden method of sub class not allowed to throw checked exception otherwise it will generate compilation error but overridden method can throw Unchecked Exception.

```
package com.ravi.method_related_rule;

import java.io.EOFException;
import java.io.FileNotFoundException;
import java.io.IOException;

class Super
{
    public void show()
    {
        System.out.println("Super class method not throwing checked Exception");
    }
}
class Sub extends Super
{
    @Override
    public void show()  //throws EOFException
    {
        System.out.println("Sub class method should not throw checked Exception");
    }
}
public class MethodOverridingWithChecked
{
    public static void main(String[] args)
    {
        // TODO Auto-generated method stub

    }
}
```

d) If the super class method declare with throws keyword to throw a checked exception, then at the time of method overriding, sub class method may or may not use throws keyword.
If the Overridden method is also using throws keyword to throw checked exception then it must be either same exception class or sub class, it should not be super class as well as we can't add more exceptions in the overridden method.

```
package com.ravi.method_related_rule;

import java.io.FileNotFoundException;
import java.io.IOException;

class Base
{
    public void show() throws FileNotFoundException
    {
        System.out.println("Super class method ");
    }
}
class Derived extends Base
{
    public void show() //throws IOException
    {
        System.out.println("Sub class method ");
    }
}
public class MethodOverridingWithThrows
{
    public static void main(String[] args)
    {
      System.out.println("Overridden method may or may not throw checked exception but if it is
throwing then must be same or sub class");
    }

}
```

---

e) Just like return keyword, we can't use throw keyword inside static and non static block to throw an exception object because all initializers must be executed normally.

We can use throw keyword in the protection of try-catch so the code will be executed normally.

```
//Program

public class ExceptionDemo
{
    {
        try
        {
            throw new ArithmeticException();
        }
        catch (ArithmeticException e)
        {
            System.out.println(e);
        }
    }

    public static void main(String[] args)
    {
      new ExceptionDemo();
    }
}
```

---

f) If we call any method and if the method throws java.lang.Exception OR java.lang.Throwable then handling is compulsory at caller Method otherwise it will generate compilation error because Exception and Throwable both are representing checked and Unchecked.

```
package com.ravi.method_related_rule;

public class CheckedExceptionCalling
{
    public static void main(String[] args)
    {
        try
        {
          m1();
          m2();
        }
        catch(Exception e)
        {
            System.out.println("Exception type handled");
        }
        catch(Throwable e)
        {
            System.out.println("Throwable type handled");
        }
    }

    public static void m1() throws Exception
    {
        System.out.println("Exception Type");

    }

    public static void m2() throws Throwable
    {
        throw new Throwable();
    }

}
```

Nested try block :

Nested try block :

If we write a try block inside another try block then it is called Nested try block.

```
try  //Outer try
{
 statement1;
   try  //Inner try
   {
     statement2;
   }
   catch(Exception e) //Inner catch
   {
   }
}
catch(Exception e) //Outer Catch
{
}
```

The execution of inner try block depends upon outer try block that means if we have an exception in the Outer try block then inner try block will not be executed.

```
package com.ravi.method_related_rule;

public class NestedTryBlock
{
    public static void main(String[] args)
    {
        try
        {
            String str = "nit";
            System.out.println(str.toUpperCase());

            try
            {
                int no = Integer.parseInt(str);
                System.out.println("Number is :"+no);
            }
            catch(NumberFormatException e)
            {
                System.err.println("Number is not in a proper format");
            }
        }
        catch(NullPointerException e)
        {
          System.err.println("str is pointing to null");
        }

    }

}
```

Writing try-catch inside catch block :

We can write try-catch inside catch block but this try-catch block will be exceuted if the catch block will executed that means if we have an exception in the try block.
We can also write try-catch inside finally block.

```
package com.ravi.basic;

import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;

public class TryWithCatchInsideCatch
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        try(sc)
        {
            System.out.print("Enter your Roll number :");
            int roll = sc.nextInt();
            System.out.println("Your Roll is :"+roll);

        }
        catch(InputMismatchException e)
        {
            System.err.println("Provide Valid input!!");

            try
            {
                System.out.println(10/0);
            }
            catch(ArithmeticException e1)
            {
              System.err.println("Divide by zero problem");
            }

        }
        finally
        {
            System.out.println("Finally block");

            try
            {
                Object []arr = new String[3];

                arr[0] = "Scott";
                arr[1] = "Smith";
                arr[2] = 90;  //java.lang.ArrayStoreException
                System.out.println(Arrays.toString(arr));
            }
            catch(ArrayStoreException e)
            {
                System.err.println("Invalid data inside array");
            }

        }
    }

}
```

---

try-catch with return statement

If we write try-catch block inside a method and that method is returning some value then we should write return statement in both the places i.e inside the try block as well as inside the catch block.

We can also write return statement inside the finally block only (Not recommended), if the finally block is present. After this return statement we cannot write any kind of statement. (Unrechable)

Always finally block return statement having more priority then try-catch return statement because finally block return statement
override try - catch return statement so compiler will generate
warning.

```
package com.ravi.advanced;

public class ReturnDemo1
{
    public static void main(String[] args)
    {
        System.out.println(m1());

    }

    public static int m1()
    {
        try
        {

            System.out.println("Inside Try");
            return 10;
        }
        catch(Exception e)
        {
            System.out.println("Inside Catch");
            return 20;
        }

        //System.out.println("Unreachable line");

    }
}
```

```
package com.ravi.advanced;

public class ReturnDemo2
{
    public static void main(String[] args)
    {
        System.out.println(m1());
    }

    public static int m1()
    {
        try
        {
            System.out.println("Inside Try");
            return 10/0;
        }
        catch(Exception e)
        {
            System.out.println("Inside Catch");
            return 20;
        }
    }

}
```

```
package com.ravi.advanced;

public class ReturnDemo3 {

    public static void main(String[] args)
    {
        System.out.println(m1());
    }

    @SuppressWarnings("finally")
    public static int m1()
    {
        try
        {
            System.out.println("Inside Try");
            return 10;
        }
        catch(Exception e)
        {
            System.out.println("Inside Catch");
            return 20;
        }

        finally
        {
            System.out.println("Inside finally");
            return 30;
        }
    }

}
```

Initialization of a local variable in try and catch :

A local variable must be initialized inside try block as well as catch block OR at the time of declaration.

If we initialize inside the try block only then from catch block we cannot access local variable value,
Here initialization is compulsory inside catch block.

```
package com.ravi.basic;

public class VariableInitialization
{
    public static void main(String[] args)
    {
       int x;
       try
       {
           System.out.println("Inside Try");
           x = 100;
           System.out.println("x value is :"+x);
       }
       catch(Exception e)
       {
           x = 200;
           System.out.println("x value is :"+x);
        }

    }

}
```