Rules associated with Exception handling

- Whenever we are writing try block compulsorily we should write either catch block or finally try without catch and finally is invalid.
- Whenever we are writing a catch block, compulsorily try block is required.
- Whenever we are writing a finally block, compulsorily try block is required.
- try catch and finally order is important.
- With in try catch finally blocks, we can take try catch finally.
- For try catch finally blocks curly braces are mandatory.

1.7 version Enhancements

- try with resource
- try with multi catch block

until jdk1.6, it is compulsorily required to write a finally block to close all the resources which are open as a part of try block.

Example:

```
BufferReader br=null
      try{
  br=new BufferedReader(new FileReader("abc.txt"));
      }catch(IOException ie){
          ie.printStackTrace();
      }finally{
  try{
    if(br≠null){
  br.close();
    }
          }catch(IOException ie){
  ie.printStackTrace();
   }
}
```

Problems in the approach

- Compulsorily the programmer is required to close all opened resources which increases the complexity of the program
- Compulsorily we should write finally block explicitly, which increases the length of the code and reviews readability.
- To Overcome this problem SUN MS introduced try with resources in "1.7" version of jdk.

try with resources

- In this approach, the resources which are opened as a part of try block will be closed automatically once the control reaches to the end of
- try block normally or abnormally,so it is not required to close explicitly so the complexity of the program would be reduced.
- It is not required to write a finally block explicitly,so length of the code would be reduced and readability is improved.

```
try(BufferedReader br=new BufferedReader(new FileReader("abc.txt")){
 //use br and perform the necessary operation
        //once the control reaches the end of try automatically br will be closed
}catch(IOException ie){
     //handling code
}
```

Rules of using try with resource

1. we can declare any no of resources, but all these resources should be separated with ;
eg#1.

```
 try(R1;R2;R3;){
         //use the resources
   }
```

2. All resources are said to be AutoCloseable resources iff the class implements an interface called "java.lang.AutoCloseable" either directly or indirectly

eg:: `java.io package classes, java.sql.package classes`

3. All resource references by default are treated as implicitly final and hence we can't perform reassignment within the try block.

```
try(BufferedReader br=new BufferedReader(new FileWriter("abc.txt")){
  br=new BufferedReader(new FileWriter("abc.txt"));
}
```

output:: CE: can't reassign a value

4. Until the 1.6 version try should compulsorily be followed by either catch or finally, but from 1.7 version we can only take try with resources without catch or finally.

```
try(R){
  //valid
}
```

5. Advantage of try with resources concept is finally block will become dummy because we are not required to close resources explicitly.

MultiCatchBlock

- Till jdk1.6, even though we have multiple exceptions having the same handling code we have to write a seperate catch block for every exception, it increases the length of the code and reviews readability.

Eg#1

```
try{
   ....
   ....
   ....
   ....
}catch(ArithmeticException ae){
 ae.printStackTrace();
}catch(NullPointerException ne){
 ne.printStackTrace();
}catch(ClassCastException ce){
 System.out.println(ce.getMessage());
}catch(IOException ie){
 System.out.println(ie.getMessage());
}
```

To overcome this problem SUMS has introduced "Multi catch block" concept in 1.7 version

```
try{
   ....
   ....
   ....
   ....
}catch(ArithmeticException |NullPointerException e){
 e.printStackTrace();
}catch(ClassCastException |IOException e){
 e.printStackTrace();
}
```

- In multi catch blocks,there should not be any relation b/w exception types(either child to parent or parent to child or same type) it would result in compile time error.

Eg#1

```
try{

     }catch( ArithmeticException | Exception e){
  e.printStackTrace();
}
```

output:: CompileTime Error