Exception handling
==========
1. try catch and finally
2. throws(best suited for checkedException)
3. throw(best sutied for uncheckedException and customException)

Syntax
=====
```
  try{
      //risky code
  }catch(XXXXX e){
      //handling code
  }finally{
      //resource releasing code
  }
```

In realtime application,we use many resources where all the resource should be closed inside finally block.
resource => In File operations we use FileReader,FileWriter,BufferedReader,BufferedWriter
In JDBC Operations we use Connection,Statement,PreapredStatement,CallableStatement,....

Realtime coding
==========
```
 //declaration of resources
  try{
      //risky code
      use the resource
  }catch(XXXXX e){
      //handling code
  }finally{
      //resource releasing code
  }
```

JDK1.6V for developers
==========
```
eg: BufferedReader br=null;
      FileReader fr =null;
      try{
            fr  = new FileReader("sample.txt");
            br = new BuffereReader();
      }catch(IOException e){
            e.printStackTrace();
      }finally{
            if(br!=null)
                  br.close();
            if(fr!=null)
                  fr.close();
      }
```

boilerplate -> the code which is repeated in multiple modules of project with no change or with small change.
whenever boiler plate code comes into pitcutre, we always try to avoid it by using
a. using JDK software higher version(jdk1.0,jdk1.2,..... jdk18)
b. using 3rd party API's

In JDK1.7 version they made few enhancement in the Excpetion handling area
==========
1. try with resource
2. try with multicatch block

try with resource
==========
Syntax: try(R){
                        use resource as per ur application requirement
                        if exception occurs or not occurs and if it is handled or not handled
                        still Resources will be closed once the control comes out of try block
            }catch(XXXX e){

            }

eg: without using try with resource
    ==========
      BufferedReader br=null;
    FileReader fr =null;
    try{
          fr = new FileReader("sample.txt");
          br = new BuffereReader();
    }catch(IOException e){
          e.printStackTrace();
    }finally{
          if(br!=null)
                br.close();
          if(fr!=null)
                fr.close();
    }

eg: try with resource
       try(BufferedReader br= new BufferedReader(new FileReader("sample.txt"))){
                //use the resource
       }catch(IOException e){
                e.printStatckTrace();
       }

Advantage of try with Resource
==========
  1. The main advantage of try with resource is the resources which are a part of try block gets close automatically.
    once the control comes of out try block automatically that resource will be closed.
    while try block is getting executed
                a. exception occured and handled
                b. exception occured and not handled
     In both these cases also jvm will close the resource automatically, if we use resource with "try with resource".

  2. Using try with resource increases readabilty and reduces redundant code in our application.

Conclusions
==========

1. we can declare any no of resources ,but all the resoures should be seperated with ; symbol.

```
try(R1;R2;R3;……){

}catch(XXXXX e){

}
```

2.From JDK1.7 for Resource Releasing logic Requirement specification they had comewith an interface called "AutoCloseable" which is added in "java.lang" package.

```
interface AutoCloseable{
      public abstract void close() throws Exception;
}
public class BufferedReader implements AutoCloseable{
            @Override
            public void close(){
                  //logic of closing.
            }
}
try(BufferedReader br=new BufferedReader(new FileReader("sample.txt"))){
            //logic of using br
}
  catch(IOException e){
            //handling logic
}
```

Note:

```
try(String name =new String("sachin")){
                  //using name object
     }catch(Exception e){}
       output: Compile Time Error
```

All java.io classes and java.sql classes has implemented AutoCloseable interface.

3. All resources reference variable are been made as final automatically when they are used, so we can't re-assign the reference of the Resource Variable.

```
      CompileTime Error
      ==============
try(BufferedReader br=new BufferedReader(new FileReader("sachin.txt"))){
            br=new BufferedReader(new FileReader("kohli.txt")));
}catch(IOException e){}
```

4. Before JDK1.6

```
try{

}catch(XXXX e){

}finally{

}
```

After JDK1.7, do we need finaly block ?

Ans. no

finally block becomes dummy if we use "try with Resource".

5. JDK1.6V

```
try{
```

```
      }finally{

      }
      a. if exception does not occur => normal termination/smoothfull termination
still finally executes.
      b. if exception occurs => abnormal termination still finally executes.

     JDK1.7
      try(R){

      }
      it is possible to write only try also from JDK1.7 version ,but that try
should be associated with Resource.


JDK1.5 features
===========
1.Wrapper classes
2.Var-Args
```