Exception Handling

---

1. try with resource.
2. try with multi-catch block.
3. Rules of Overriding associated with Exception.

4.instanceof vs isInstanceOf(Object obj)

5.How to create a userdefined package and in realtime project how it is used?

1.7 version Enhancements
==========

1. try with resource
2. try with multicatch block

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

```
eg:: 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 apporach
==========

1. Compulsorily the programmer is required to close all opened resources which increases the complexity of the program 2. Compulsorily we should write finally block explicitly, which increases the length of the code and reviews readablity. To Overcome this problem SUN MS introduced try with resources in "1.7" version of jdk.

try with resources
==========

In this apporach, 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 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 seperated with ;  eg#1.  try(R1;R2;R3;){
   //use the resources
   }

2. All resources are said to be AutoCloseable resources iff the class implements aninterface called "java.lang.AutoCloseable"  either directly or indirectly
   eg:: java.io package classes, java.sql.package classes

```
public interface java.lang.AutoCloseable {   public
abstract void close() throws java.lang.Exception;
}
```

Note: which ever class has implemented this interface those classes objects are refered as "resources".

3. All resource reference by default are treated as implicitly final and hence we can't perform reassignment with in 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. untill 1.6 version try should compulsorily be followed by either catch or finally, but from 1.7 version we can take only take try with resources without cath 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.

6. try with resource nesting is also possible.try(R1){ try(R2){ try(R3){

```
                    }
               }
          }
```

MultiCatchBlock
==========

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

logic
==== try{
....

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

To overcome this problem SUNMS has introduced "Multi catch block" concept in 1.7 version try{ ....

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

In multicatch block,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:: try{

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

Output:CompileTime Error

throw =>handle the exception using catch block and throw it back the exception object to the caller.
throws => method signature and commonly used if the exception is "CheckedException".

Rules of Overriding when exception is involved
==========

While Overriding if the child class method throws any checked exception compulsorily the parent class method should throw the same checked exception or its parent otherwise we will get Compile Time Error. There are no restrictions on UncheckedException.

eg#1.

```
class Parent{ public void
      methodOne();
}
class Child extends Parent{ public void
      methodOne() throws Exception{}
}
error: methodOne() in Child cannot override methodOne() in Parent
public void methodOne() throws Exception{}
overridden method does not throw Exception
```

Rules w.r.t Overriding
==========
parent: public void methodOne() throws Exception{}
child : public void methodOne()
output: valid

parent: public void methodOne(){} child : public
void methodOne() throws Exception{} output:
invalid

parent: public void methodOne()throws Exception{}
child : public void methodOne()throws Exception{} output:
valid

parent: public void methodOne()throws IOException{}
child : public void methodOne()throws IOException{} output:
valid

parent: public void methodOne()throws IOException{} child : public void
methodOne()throws FileNotFoundException,EOFException{}
output: valid

parent: public void methodOne()throws IOException{} child : public void
methodOne()throws FileNotFoundException,InterruptedException{}
output: invalid

parent: public void methodOne()throws IOException{} child : public void
methodOne()throws FileNotFoundException,ArithmeticException{}
output: valid

parent: public void methodOne() child
: public void methodOne()throws
ArithmeticException,NullPointerException,RuntimeException{}
output: valid

parent: public void methodOne()throws IOException{}
child : public void methodOne()throws Exception{}
output: invalid parent: public void
methodOne()throws Throwable{}
child : public void methodOne()throws IOException{} output:
valid

instanceof
=========
1. We can use the instanceof operator to check whether the given an object is particular type or not. r instanceof X r => reference X => class/interfaceName eg:

```
     ArrayList al =new Arraylist();//inbuilt object where we can keep any type
of  other objects al.add(new Student());//0th position al.add(new
Cricketer());//1st position al.add(new Customer());//2nd position

      Object o=l.get(0); // l is an arraylist object if(o
      instanceof Student) {
                  Student s=(Student)o ;
                  //perform student specific operation
      }
      elseif(o instanceof Customer) {
                  Customer c=(Customer)o;
                  //perform Customer specific operations
      }

eg#2.
      Thread t = new Thread( );
      System.out.println(t instanceof Thread);//true
      System.out.println(t instanceof Object);//true
      System.out.println(t instanceof Runnable); //true

Ex : public class Thread extends Object implements
Runnable { }


=> To use instanceof operator compulsory there should be some relation between
argument types       (either child to parent Or parent to child Or same type)
Otherwise we will get compile time error saying inconvertible types.

eg: String s= new String("sachin");
System.out.println(s instanceof Thread);//CE

      Thread t=new Thread( );
      System.out.println(t instanceof String);//CE

=> Whenever we are checking the parent object is child type or not by using
instanceof operator that we get false.
      Object o=new Object( );
      System.out.println(o instanceof String ); //false

      Object o=new String("ashok");
      System.out.println(o instanceof String); //true

=>  For any class or interface X null instanceof X is always returns false
      System.out.println(null instanceof  X); //false

public class Test { public static void
      main(String[] args) {
            Object t = new Thread();
            System.out.println(t instanceof Object);//true
            System.out.println(t instanceof Thread);//true
            System.out.println(t instanceof Runnable);//true
            System.out.println(t instanceof String);//false
            System.out.println(null instanceof Object);//false
      }
}
```

isInstance()
==========
Difference between instanceof and isInstance( ) :

instanceof ========= instanceof an operator which can be used to check whether the given object is
particular type or not We know at the type at beginning it is available.

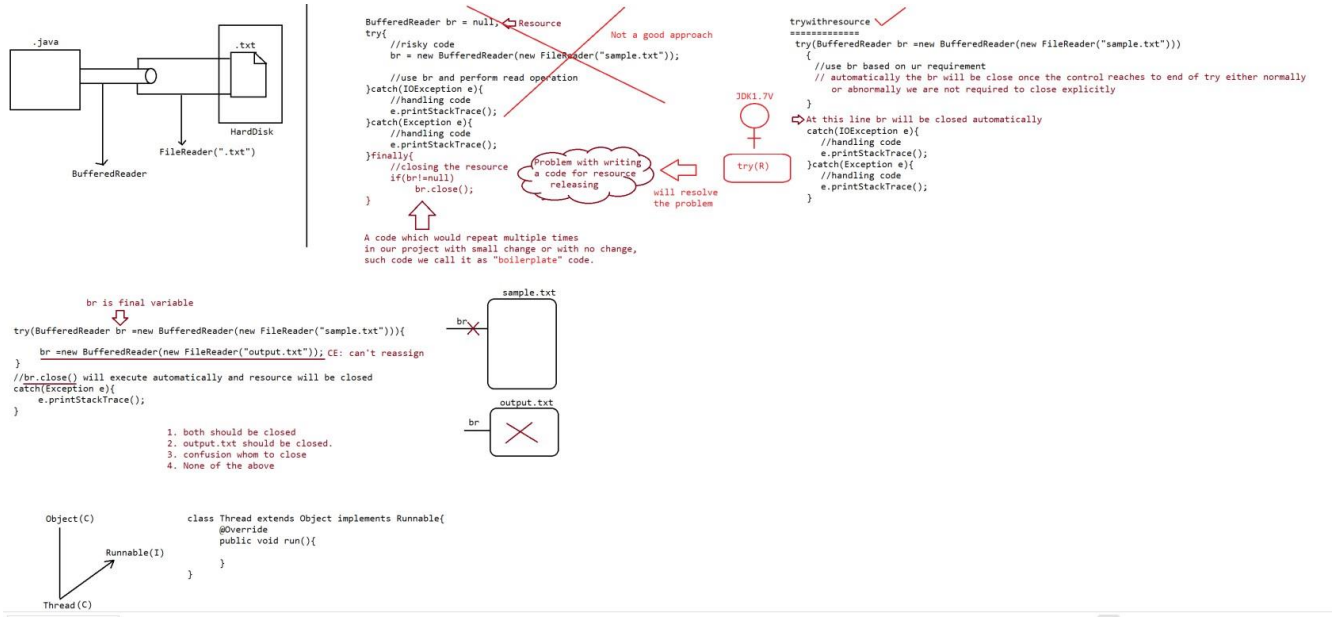
```
eg:  String s = new String("sachin");
      System.out.println(s instanceof Object );//true
      //If we know the type at the beginning only.
```

isInstance( ) isInstance( ) is a method , present in class Class , we can use isInstance() method to checked whether the given object is  particular type or not We don't know at the type at beginning it is available Dynamically at Runtime.

```
class Test { public static void main(String[]
      args) {
                  Test t = new Test( ) ;

      System.out.println(Class.forName(args[0]).isInstance(t));////arg[0] --- We don't know the type at beginning
      }
}
java Test Test //true
java Test String //false java
Test Object //true
```

pictorial
representation:



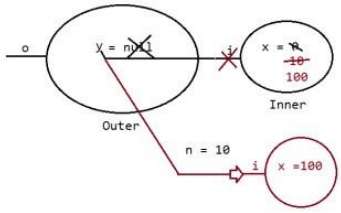
```
class Inner {
     private int x;
     public void setX( int x ){ this.x = x; }
     public int getX(){ return x;}
}
```

```
class Outer {
     private Inner y;
     public void setY( Inner y ){ this.y = y; }
     public Inner getY() { return y; }
}
```

```
Outer o = new Outer();
Inner i = new Inner();
int n = 10;
i.setX(n);
o.setY(i);
```

answer: produce the output 100?

```
// insert code here 29
System.out.println(o.getY().getX());
```




```
A. n = 100;
B. i.setX( 100 );
C. o.getY().setX( 100 );
D. i = new Inner(); i.setX( 100 );
E. o.setY( i ); i = new Inner(); i.setX( 100 );
F. i = new Inner(); i.setX( 100 ); o.setY( i );
```

```
public class Hello {
     String title;
     int value;
     public Hello() {
          title += " World";
     }
     public Hello(int value) {
          this.value = value;
          title = "Hello";
          Hello();
     }
}
```

constructor can't be called explicitly
it results in "CE".

```
Hello c = new Hello(5);
System.out.println(c.title);
```

<<<<<<<<<<<<<<<<<<<<<<<<< End Of Exception Handling>>>>>>>>>>>>>>>>>>>>>>>>