Try with Resources Enahancements

Need of Try with Resources:

Until 1.6 version it is highly recommended to write finally block to close all resources which are open as part of try block.

```
BufferedReader br=null;
try
{
   br=new BufferedReader(new FileReader("abc.txt"));
   //use br based on our requirements
}
catch(IOException e)
{
   // handling code
}
finally
{
   if(br != null)
     br.close();
}
```

Problems in this Approach:

1. Compulsory programmer is required to close all opened resources which increases the complexity of the programming
2. Compulsory we should write finally block explicitly which increases length of the code and reduces readability.

To overcome these problems Sun People introduced "try with resources" in 1.7 version.

Try with Resoruces:

The main advantage of "try with resources" is the resources which are opened as part of try block will be closed automatically Once the control reaches end of the try block either normally or abnormally and hence we are not required to close explicitly so that the complexity of programming will be reduced. It is not required to write finally block explicitly and hence length of the code will be reduced and readability will be improved.

```
try(BufferedReader br=new BufferedReader(new FileReader("abc.txt")))
{
   use be based on our requirement, br will be  closed automatically , Onec control reaches
 end of try either normally or abnormally and we are not required to close explicitly
```

```
}
catch(IOException e)
{
   // handling code
}
```

Conclusions:

1. We can declare any no of resources but all these resources should be seperated with ;(semicolon)

```
try(R1 ; R2 ; R3)
{
   -------------
}
```

2. All resources should be AutoCloseable resources. A resource is said to be auto closable if and only if the corresponding class implements the java.lang.AutoCloseable interface either directly or indirectly.

All database related, network related and file io related resources already implemented AutoCloseable interface. Being a programmer we should aware this point and we are not required to do anything extra.

3. AutoCloseable interface introduced in Java 1.7 Version and it contains only one method: close()

4. All resource reference variables should be final or effectively final and hence we can't perform reassignment within the try block.

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

CE : Can't reassign a value to final variable br

5. Untill 1.6 version try should be followed by either catch or finally but in 1.7 version we can take only try with resource without catch or finally

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

6.The main advantage of "try with resources" is finally block will become dummy because we are not required to close resources of explicitly.

Problems with JDK 7 Try with Resources:

1. The resource reference variables which are created outside of try block cannot be used direclty in try with resources.

```
BufferedReader br=new BufferedReader(new FileReader("abc.txt"))
try(br)
{
   // Risky code
}
```

This syntax is invalid in until java 1.8V.

We should create the resource in try block primary list or we should declared with new reference variable in try block. i.e Resource reference variable should be local to try block.

Solution-1: Creation of Resource in try block primary list

```
try(BufferedReader br=new BufferedReader(new FileReader("abc.txt")))
{
   // It is valid syntax in java 1.8V
}
```

Solution-2: Assign resource with new reference variable

```
BufferedReader br=new BufferedReader(new FileReader("abc.txt"))
try(BufferedReader br1=br)
{
   // It is valid syntax in java 1.8V
}
```

But from JDK 9 onwards we can use the resource reference variables which are created outside of try block directly in try block resources list.i.e The resource reference variables need not be local to try block.

```
BufferedReader br=new BufferedReader(new FileReader("abc.txt"))
try(br)
{
   // It is valid in JDK 9 but in valid until JDK 1.8V
}
```

But make sure resource(br) should be either final or effectively final. Effectively final means we should not perform reassignment.

This enhancement reduces length of the code and increases readability.

```
MyResource r1 = new MyResource();
MyResource r2 = new MyResource();
MyResource r3 = new MyResource();
try(r1,r2,r3)
{
}
```

Demo Program:

```
class  MyResource implements AutoCloseable
{
    MyResource()
    {
        System.out.println("Resource Creation...");
    }
    public void doProcess()
    {
        System.out.println("Resource Processing...");

    }
    public void close()
    {
        System.out.println("Resource Closing...");
    }
}
class Test
{
    public static void preJDK7()
    {
        MyResource r=null;
        try
        {
            r=new MyResource();
            r.doProcess();
        }
        catch (Exception e)
        {
            System.out.println("Handling:"+e);
        }
        finally
        {
            try
            {
```

```java
                if (r!=null)
                {
                    r.close();
                }
            }
            catch (Exception e)
            {
                System.out.println("Handling:"+e);
            }
        }
    }
    public static void JDK7()
    {
        try(MyResource r=new MyResource())
        {
            r.doProcess();
        }
        catch(Exception e)
        {
            System.out.println("Handling:"+e);
        }
    }
    public static void JDK9()
    {
        MyResource r= new MyResource();
        try(r)
        {
            r.doProcess();
        }
        catch(Exception e)
        {
            System.out.println("Handling:"+e);
        }
    }
    public static void multipleJDK9()
    {
        MyResource r1= new MyResource();
        MyResource r2= new MyResource();
        MyResource r3= new MyResource();
        MyResource r4= new MyResource();
        try(r1;r2;r3;r4)
        {
            r1.doProcess();
            r2.doProcess();
            r3.doProcess();
            r4.doProcess();
        }
        catch(Exception e)
        {
            System.out.println("Handling:"+e);
        }
    }
    public static void main(String[] args)
    {
        System.out.println("Program Execution With PreJDK7");
        preJDK7();

        System.out.println("Program Execution With JDK7");
        JDK7();

        System.out.println("Program Execution With JDK9");
        JDK9();
        System.out.println("Program Execution Multiple Resources With JDK9");
        multipleJDK9();
    }
}
```

Output:

```
Program Execution With PreJDK7
Resource Creation...
Resource Processing...
Resource Closing...
Program Execution With JDK7
Resource Creation...
Resource Processing...
Resource Closing...
Program Execution With JDK9
Resource Creation...
Resource Processing...
Resource Closing...
Program Execution Multiple Resources With JDK9
Resource Creation...
Resource Creation...
Resource Creation...
Resource Creation...
Resource Processing...
Resource Processing...
Resource Processing...
Resource Processing...
Resource Closing...
Resource Closing...
Resource Closing...
Resource Closing...
```