Reflection

Reflection is a process of examining or modifying the runtime behavior of a class at run time.

java.lang.Class class

- provides methods to get the metadata of a class at run time.
- provides methods to examine and change the run time behavior of a class.

Fetch the object of Class:

- forName() method of Class class. forName() cannot be used for primitive types. This is only used when we know the class name

```
package Reflection;

class LoadDynamically{
    public void load(){
        System.out.println("LoadDynamically.....");
    }
}

class Testing{
    public static void main(String []args) throws ClassNotFoundException{
      (Class.forName("LoadDynamically")).getName();
    }
}
```

- getClass() method of Object class. Returns the instance of Class class. It can be used with primitives. Additionally should be put into used when know the type of the class.

```
package Reflection;

class LoadDynamically{
    public void load(){
        System.out.println("LoadDynamically.....");
    }
}

class Testing{
```

```
    public static void main(String []args) throws ClassNotFoundException{
      LoadDynamically obj = new LoadDynamically();
      System.out.println(obj.getClass().getName());
    }
}
```

- .class

```
package Reflection;

class LoadDynamically{
    public void load(){
        System.out.println("LoadDynamically.....");
    }
}


class classExample {
      public static void main(String []args){
            System.out.println((Boolean.class).getName());
            System.out.println((LoadDynamically.class).getName());
      }
}
```

OUTPUT:
java.lang.Boolean // PackageName.className
Reflection.LoadDynamically

newInstance() method

The newInstance() method is available for Class class and Constructor class.

The newInstance() method of Class class takes zero-argument constructor. newInstance() method of Constructor class can invoke any number of arguments. So Constructor class is preferred over Class class.

```
package Reflection;

class LoadDynamicallyTest{
    public void load(){
```

```
        System.out.println("LoadDynamically.....");
    }
}

class newInstanceExample {
      public static void main(String []args) throws InstantiationException,
IllegalAccessException, ClassNotFoundException{
            ((Reflection.LoadDynamicallyTest)
(Class.forName("Reflection.LoadDynamicallyTest")).newInstance()).load();
//PackageName.className
      }
}
```

Display the information about the fields,constructors and methods present in a class file

```
package Reflection;
import java.lang.reflect.Field;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.util.Scanner;

class disassemblesClass {
      public static void main(String []args) throws ClassNotFoundException{
            Scanner input = new Scanner(System.in);
            Class obj = Class.forName(input.nextLine());
            System.out.println("Class : "+obj + " DeclaredFields..");
            Field[] field = obj.getDeclaredFields();
            for(int index=0; index<field.length; index++){
                  System.out.println(field[index]);
            }
            System.out.println("Class : "+obj + " DeclaredConstructors..");
            Constructor[] cons = obj.getDeclaredConstructors();
            for(int index=0; index<cons.length; index++){
                  System.out.println(cons[index]);
            }
            System.out.println("Class : "+obj + " DeclaredMethods..");
            Method[] method = obj.getDeclaredMethods();
            for(int index=0; index<method.length; index++){
                  System.out.println(method[index]);
            }
      }
}
OUTPUT:

      java.lang.Object
```

```
      Class : class java.lang.Object DeclaredFields..
      Class : class java.lang.Object DeclaredConstructors..
      public java.lang.Object()
      Class : class java.lang.Object DeclaredMethods..
      protected void java.lang.Object.finalize() throws java.lang.Throwable
      public final void java.lang.Object.wait(long,int) throws
java.lang.InterruptedException
      public final native void java.lang.Object.wait(long) throws
java.lang.InterruptedException
      public final void java.lang.Object.wait() throws
java.lang.InterruptedException
      public boolean java.lang.Object.equals(java.lang.Object)
      public java.lang.String java.lang.Object.toString()
      public native int java.lang.Object.hashCode()
      public final native java.lang.Class java.lang.Object.getClass()
      protected native java.lang.Object java.lang.Object.clone() throws
java.lang.CloneNotSupportedException
      private static native void java.lang.Object.registerNatives()
      public final native void java.lang.Object.notify()
      public final native void java.lang.Object.notifyAll()
```