1. **Reflection**
2. **Reflection API**
3. **How Reflection API Useful to Javac.**
4. **How Reflection API useful to java.**
5. **Class //Placed in separate file.**
6. **Java.lang.reflect**

**1.Reflection:-** The process of analyzing and modifying internal properties/members of class dynamically /at runtime is called “reflection”.

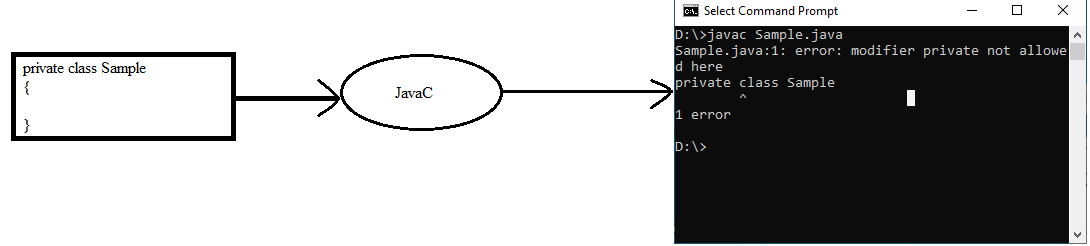
**2. Reflection API:-** Sun Micro System / Oracle corporation has given set of developed classes to analyze and modify internal proeries of class at run time. The collection of these developed classes is called “Reclection API”.

This reflection API is used in

1. Frame Works.(Ex:spring, Hibernate)
2. IDE’s (Eclipse,NetBeans,…etc)
3. Testing Tools.
4. Javac, Java Tools.
5. It is used, when your application has third-party libraries and when you want to know about the classes and methods available.

**3.How Reflection API Is useful to Javac?**

A.



First ,

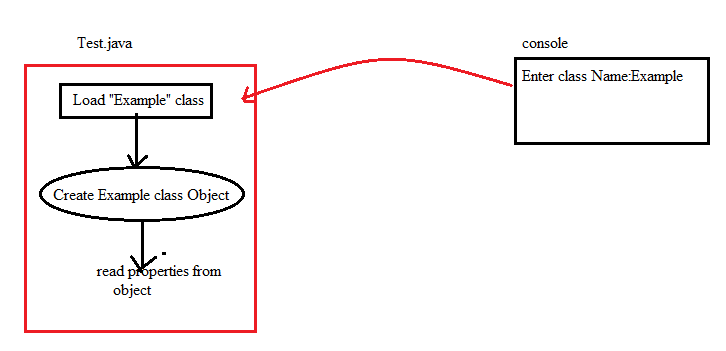
Step1. The Javac read complete information of Sample class using “**Reflection API”**.

Step2. Javac Determines whether sample class is as per the class rules or not using Read information.

Step3. Based on determination, Javac either generates Byte code or display error message.

**4.How Reflection API is useful to java**?

A.



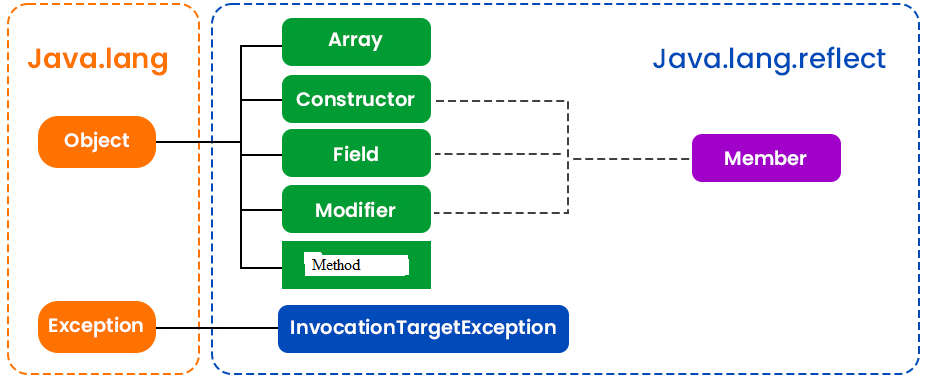
The test.java reads class name from key board at run time. At the time of execution, Example class to be loaded and Example class object to be created.

To do this, JVM needs complete information of “Example class”. This information is gathered by JVM using the **Reflection API**.

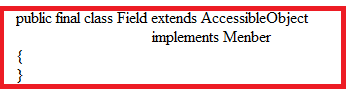
Note:- The Reflection API package name is “reflection”. It is sub package. This sub package is in java.lang. package.

**Java.lang.Reflect**

**6.Java.lang.Reflect:-**

****

**6.1.Field:-** This class provides information about single field of class or interface.



Methods:

1.getXXX:- This method returns value of static or non-static XXX field.

Syntax:

Public xxx getXXX(User defined class object)

Where

Xxx is byte,short, int,long,char,float,double or boolean.

2.setXXX:- This method set value in field.

Syntax:

Public void setXXX(Userdefined class object, xxx value);

Where

Xxx is byte,short,int,long,char,float,double or boolean.

3.getName:- This method returns name of field.

Syntax:

Public String getName();

4.toString:- This method returns String which describes the Field. The format is the access modifiers for the field, if any, followed by the field type, followed by a space, followed by the fully-qualified name of the class declaring the field, followed by a period, followed by the name of the field.

Syntax:

Public String getName();

5.equals:-This method compares this Field against the specified object. Returns true if the objects are the same. Two Field objects are the same if they were declared by the same class and have the same name and type.

Syntax:

Public boolean equals(userdefined object);

6.getModifiers:- This method returns the no.of modifier of single field.

Syntax:

Public int getModifiers();

7. getDeclaredAnnotations:-

Example:

import java.lang.Class;

import java.lang.reflect.\*;

class Example{

public int a;

private float b;

public Example(){}

private Example(int x){}

public void display(){}

private void display1(){}

}

class Sample{

public static void main(String args[]){

Example e1=new Example();

Class c1=e1.getClass();

Field []f=c1.getDeclaredFields();

try{

System.out.println("Field Name:"+f[0].getName());

f[0].setInt(e1,10);

System.out.println("Field Value:"+f[0].getInt(e1));

System.out.println("To Stirng:"+f[0].toString());

System.out.println("No.of.Modifiers:"+ f[0].getModifiers());

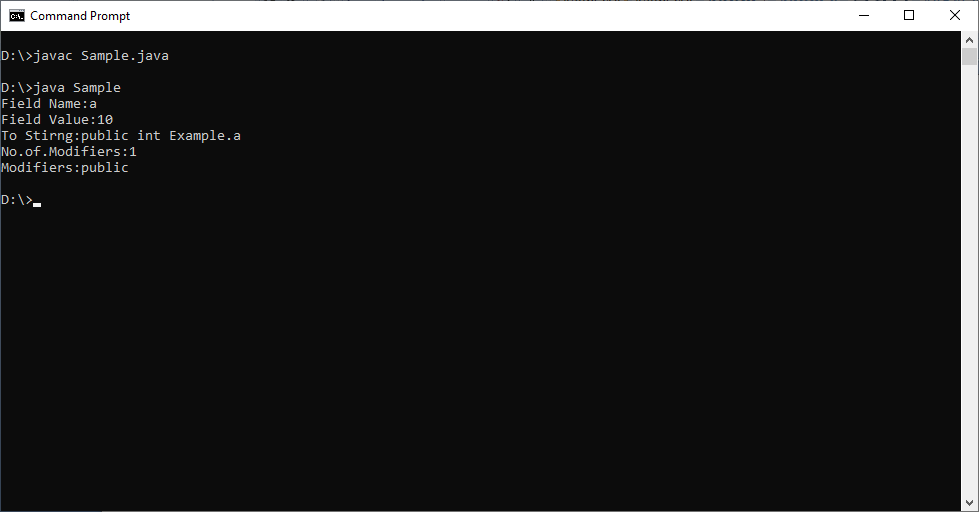
System.out.println("Modifiers:"+Modifier.toString(f[0].getModifiers()));

}

catch(Exception e){}

}

}



**6.2.Method:-** This method provides information of single method of class.



**a.getName:-** This method returns name of Method.

Syntax:

Public String getName()

**b.getReturnType:-** This method returns Class object that represents the formal return type of the method represented by this Method object.

Syntax:

Public Class getReturnType()

**c.getModifiers:-** This method returns the Java language modifiers for the method represented by this Method object, as an integer. The Modifier class should be used to decode the modifiers.

Syntax:

Public int getModifiers()

**d.getParameterTypes:-** This method returns an array of Class objects that represent the formal parameter types, in declaration order, of the method represented by this Method object. Returns an array of length 0 if the underlying method takes no parameters.

Syntax:

Public Class[] getParameterTypes();

**e.invoke:-**This method invokes the underlying method represented by this Method object, on the specified object with the specified parameters. Individual parameters are automatically unwrapped to match primitive formal parameters, and both primitive and reference parameters are subject to method invocation conversions as necessary.

Syntax:

Public Object invoke(user defined class object, Object []args)throws IllegalArgumentException,IllegalAccessException.

**h.toString():-** It returns string which describes a method.

Syntax:

Public String toString();

**i.equals:**-This  method compares this Method against the specified object. Returns true if the objects are the same. Two Method objects are the same if they were declared by the same class and have the same formal parameter types.

Example:

Public Boolean equals(Another Method object reference);

Example:

import java.lang.Class;

import java.lang.reflect.\*;

class Example{

public Example(){}

private Example(int x){}

public static void display(int x, float y){

System.out.println("this method has been invoked");

}

}

class Sample{

public static void main(String args[]){

Example e1=new Example();

Class c1=e1.getClass();

Method []m=c1.getDeclaredMethods();

try{

System.out.println("Method Name:"+m[0].getName());

System.out.println("To Stirng:"+m[0].toString());

System.out.println("No.of.Modifiers:"+ m[0].getModifiers());

System.out.println("Modifiers:"+Modifier.toString(m[0].getModifiers()));

m[0].invoke(e1,2,4.5f);

Class c2[]=m[0].getParameterTypes();

System.out.println("Total no.of parameters:"+ c2.length);

System.out.print("Parameters are: ");

for(int i=0;i<c2.length;i++)

{

System.out.print(c2[i].getName()+ " ");

}

System.out.println();

Class c3=m[0].getReturnType();

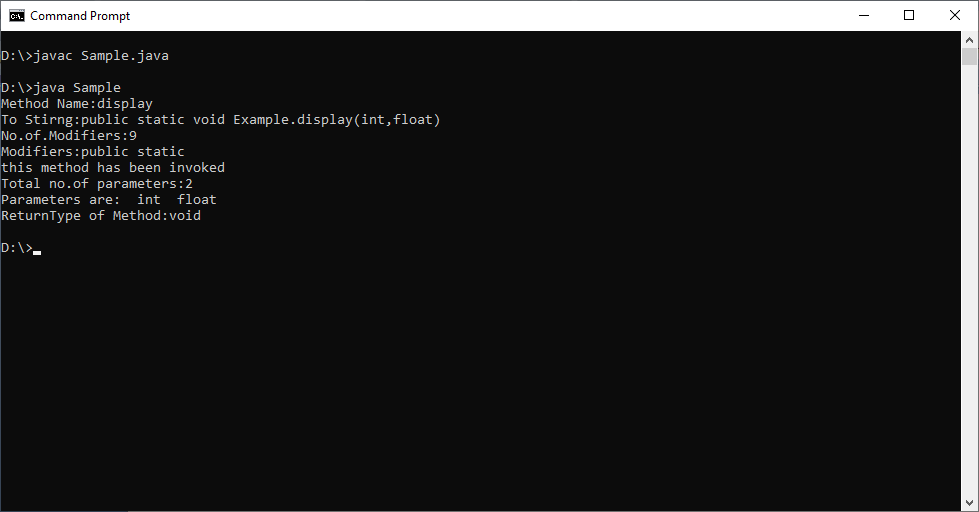
System.out.println("ReturnType of Method:"+c3.getName());

}

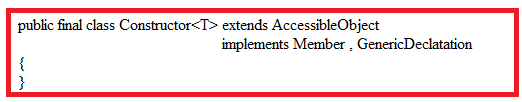
catch(Exception e){}

}

}



**6.3.Constructors:-** This class provides information of single constructor for class.



**Methods:**

**1.**getname

2.getParameterTypes

3.getModifiers

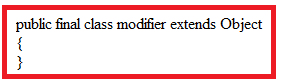
4.toString

5.equals

6.getExceptionTypes.

7.getDeclaredAnnotations.

**6.4.Modifier:-**This class provides information about class,member and method access specifier. These modifiers are represented as int values.



Fields:-

1. static int FINAL
2. static int PUBLIC
3. static int PRIVATE
4. static int PROTECTED
5. static int STRICT
6. static int STATIC
7. static int NATIVE
8. static int ABSTRACT
9. static int SYNCHRONIZED
10. static int TRANSIENT
11. static int VOLTAILE

Methods:

**a.isXXX**:- This method returns true, if integer argument includes XXX modifier,false otherwise.

Syntax:

Public static boolean isXXX(int val);

Where XXX is any one of modifiers.

**b.classModifier**:-This method return an int value OR-ing together the source language modifiers that can be applied to a class.

Syntax:

Public static int classModifiers();

**c.FieldModifiers**:- This method return an int value OR-ing together the source language modifiers that can be applied to a filed.

Syntax:

Public static int fieldModifiers();

**d.methodModifiers**:- This method return an int value OR-ing together the source language modifiers that can be applied to Method.

Syntax:

Public static int methodModifiers();

**e.constructorModifiers**:- This method return an int value OR-ing together the source language modifiers that can be applied to Constructor.

Syntax:

Public static int constructorModifiers();

**f.interfaceModifiers**:- This method return an int value OR-ing together the source language modifers that can be applied to interface.

Syntax:

Public static int interfaceModifiers();

Example:-