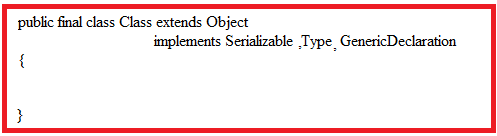
**CLASS**

**Class:-**

* It is class of “java.lang” package.
* The object of java.lang.Class represents class/enum/interface and holds the meta data.
* We can’t create object to java.lang.Class Because It does not has public constructor and It has only private 0-param constructor.
* This class provides a methods to get the metadata of class at run time.



We get “Class object” of any user defined class in 3 ways:

1. **getClass:-** It is method of “Object” class. By default , This method is inherited in user defined class from “Object class” Because “Object” is super class to any user defined class in java. This method returns “Class” object of current user defined class.

Example:

Class Sample{

Public static void main(String args[])

{

Sample s1=new Sample();

Class c1=s1.getClass();

}

}

**b.class:-** It is final variable in “Object” class. . By default , This variable is inherited in user defined class from “Object class” Because “Object” is super class to any user defined class in java.

After creating the Object of user defined class, JVM immediately creates “Class “ object to User defined class. JVM immeidately place the “Class” object reference in **class** variable of user defined class.

Example:

Class Sample{

Public static void main(Stirng arg[])

{

Sample s1=new Sample();

Class c1=s1.class;

}

}

**c.forName:-** It is static method of Class class. It returns the Class object associated with class or interface with specified string name.

Syntax:

Public Class<?> forName( String classname);

5.1.Methods:-

**a.isInterface:-** This method determines if the specified Class object represents an [interface](https://www.geeksforgeeks.org/interfaces-in-java/) type.

Syntax:

Public boolean isInterface()

**b.isPrimitive**:- This method determines if the specified class object represents primitive type.

Syntax:

Public boolean isPrimitive()

**c.isArray:-** This method determines if the specified class object represents array type.

Syntax:

Public boolean isArray()

**d.isLocalClass:-** This method returns true if and only if the this class is a local class. A local class is a class that is declared locally within a block of Java code, rather than as a member of a class.

Syntax:

Public boolean isLocalClass()

**e.isMemberClass:-** This method returns true if and only if the this class is a member class. A member class is a class that is declared as non-static member of container class.

Syntax:

Public boolean isMembeClass();

**f.isEnum:-** This method returns true if and only if this class was declared as an enum in the source code.

Syntax:

Public boolean isEnum();

**g.isAnnotation:-** This method determines if this Class object represents an annotation type. Note that if this method returns true, isInterface() method will also return true, as all annotation types are also interfaces.

Syntax:

Public Boolean isAnnotation();

**h.getName:-** This method returns Name of class,interface,array, as string.

Syntax:

Public String getName();

**i.getConstructors:-** This method returns constructor array. It contains all public constructors references.

Syntax:

Public Constructor<?>[]getConstructors() throws SecurityException

**j.getDeclaredConstructors:-** The getDeclaredConstructors() method of java Class class returns an array of constructor objects representing all the constructors(both public and private) defined in this class object. If no constructor is declared, then it returns an array of length 0.

Syntax:

Public Constructor<?>[]getDeclaredConstructors() throws SecurityException

**k.getFields:-** The getFields() method of java Class returns an array containing field objects representing all the accessible public fields of the class or interface represented by this Class object otherwise if no field is present then it returns an array of length 0.

Syntax:

Public Field<?>[]getFields() throws SecurityException

**l.getmethods():-**This method returns an array of Method objects reflecting all the accessible public methods of the class or interface and those inherited from superclasses and super interfaces represented by this Class object.

Syntax:

Public Method<?>[]getMethods() throws SecurityException

**m. getField:-** This method returns a Field object that reflects the specified public member field of the class or interface represented by this Class object.

Syntax;

Public Field getField(String fieldname) throws SecurityException,NoSuchFieldException

**n.getMethod:-** This method returns a method object that reflects the specified public member method of class or interface represented by this class object.

Syntax:

Public Method getMethod(String methodname, Class parameterTypes …) throws SecurityException, NoSuchMethodException

**0.getDeclaredMethods:**- This method returns an array of Method objects reflecting all the accessible public and private methods of the class or this Class object.

Syntax:

Public Method<?>[]getDeclaredMethods() throws SecurityException

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();

System.out.println(c1.isInterface());

System.out.println(c1.isEnum());

System.out.println(c1.isLocalClass());

System.out.println(c1.isPrimitive());

System.out.println(c1.isArray());

System.out.println(c1.getName());

Method[] m=c1.getDeclaredMethods();

System.out.println("============================Only Declared Methods in Example Class ===============");

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

System.out.println(m[i]);

}

m=c1.getMethods();

System.out.println("============================Only Declared Methods in Example Class &Its Super class Methods ===============");

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

System.out.println(m[i]);

}

Field [] f=c1.getDeclaredFields();

System.out.println("================== only Declared Fields in Example class ================");

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

System.out.println(f[i]);

}

f=c1.getFields();

System.out.println("================== only Declared Fields in Example class & Its super class fields================");

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

System.out.println(f[i]);

}

Constructor c[]=c1.getConstructors();

System.out.println("================== only Declared public constructors in Example class =============");

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

System.out.println(c[i]);

}

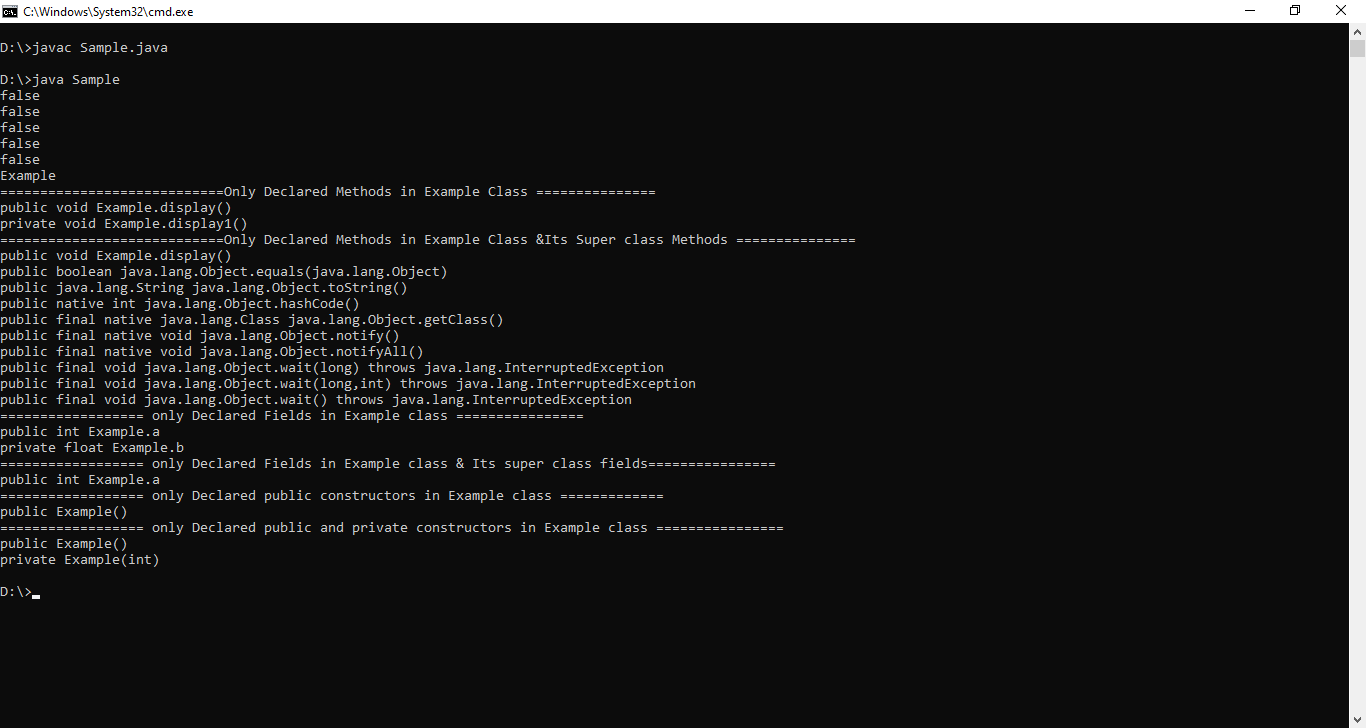
c=c1.getDeclaredConstructors();

System.out.println("================== only Declared public and private constructors in Example class ================");

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

System.out.println(c[i]);

}}}



Rought Memory Diagram of above program-

