UNIT I - JAVA FUNDAMENTALS

- Java Data types
- Class – Object
- I / O Streams
- File Handling concepts
- Threads
- Applets
- Swing Framework
- Reflection

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

Java Reflection

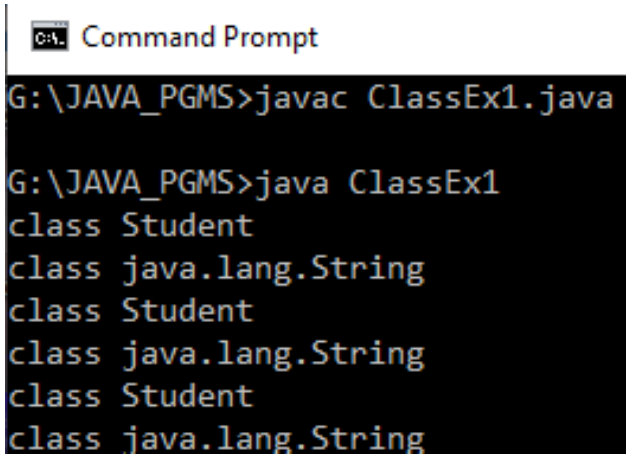
- **Reflection means ability of a software to analyze itself**.
- In Java, Reflection API provides facility to analyze and change runtime behaviour of a class at runtime.
- For example, using reflection at the runtime we can determine what method, field, constructor or modifiers a class supports.
- One of the advantage of reflection API is, we can manipulate private members of the class too.
- The **java.lang.Class** class provides methods that are used to get metadata and manipulate the run time behaviour of a class.
- The **java.lang** and**java.lang.reflect** packages provide many classes for reflection and get metadata of a particular class.

java.lang.Class class

- Before we learn about reflection in Java, we need to know about a Java class named Class.
- There is **a class in Java named Class that keeps all the information about objects and classes at runtime.**
- The **object of Class describes the properties of a particular class.**
- **This object is used to perform reflection.**
- We can create objects of Class by,
  - **using getClass() Method →** This method uses the object of a particular class to create a new object of Class
  - **using forName() Method →**This method takes a string argument (name of a class) and returns an object of Class. The returned object refers to the class specified by the string.
  - **using .class**

create objects of Class – 3 methods

```
class Student
{
        int stuID;
        String name;
}
class ClassEx1
{
        public static void main(String[] args) throws ClassNotFoundException
        {
                Student stu = new Student();
                Class obj1 = stu.getClass();
                System.out.println(obj1);
                obj1 = "Welcome".getClass();
                System.out.println(obj1);
                Class obj2 = Class.forName("Student");
                System.out.println(obj2);
                obj2 = Class.forName("java.lang.String");
                System.out.println(obj2);
                Class obj3 = Student.class;
                System.out.println(obj3);
                obj3 = String.class;
                System.out.println(obj3);
        }
}
```



java.lang.Class class Cont'd

- **Once the objects of Class are created, we can use these objects to perform reflection.**
- The java.lang.Class class performs mainly two tasks:
  - provides methods to get the metadata of a class at run time.
  - provides methods to examine and change the run time behaviour of a class.

java.lang.reflect

- The java.lang.reflect package provides many classes to implement reflection java.
- This package provides classes that can be used for manipulating class members. For example,
  - **Method class** - provides information about methods in a class
  - **Field class** - provides information about fields in a class
  - **Constructor class** - provides information about constructors in a class
- **Reflection of a Field**
- We can inspect and modify different fields of a class using various methods provided by the Field class.
  - **getFields() -** returns all public fields from the class and its superclass
  - **getDeclaredFields()** - returns all the fields of the class
  - **getModifier()** - returns the modifier of fields in integer form
  - **set(classObject, value)** - set the value of a field with the specified value
  - **get(classObject)** - get the value of a field
  - **setAccessible(boolean)** - make the private field accessible
  - **Note:** If we know the name of a field, we can use
  - **getField("fieldName")** - returns the public field having name **fieldName** from the class.
  - **getDeclaredField("fieldName")** - returns the field having name **fieldName** from the class.

java.lang.reflect

- **Reflection of Java Methods**
- Like fields, we can inspect different methods of a class using various methods provided by the Method class.
  - **getMethods()** - returns all public methods of the class and its superclass
  - **getDeclaredMethod()** - returns all methods of the class
  - **getName()** - returns the name of methods
  - **getModifiers()** - returns the access modifier of methods in integer form
  - **getReturnType()** - returns the return type of methods
- **Reflection of Constructor**
- We can also inspect different constructors of a class using various methods provided by the Constructor class.
  - **getConstructors()** - returns all public constructors of a class and superclass of the class
  - **getDeclaredConstructor()** - returns all the constructors
  - **getName()** - returns the name of constructors
  - **getModifiers()** - returns the access modifier of constructors in integer form
  - **getParameterCount()** - returns the number of parameters of constructors

Commonly used Methods in Reflection

- **getClass()→**It returns the instance of Class class. It should be used if you know the type. Moreover, it can be used with primitives.
- **public String getName()→**returns the class name
- **public Class getSuperclass()→**returns the superclass class reference.
- **public Field[] getDeclaredFields()throws SecurityException →** returns the total number of fields of this class
- **public Method[] getDeclaredMethods()throws SecurityException →** returns the total number of methods of this class.
- **public Constructor[] getDeclaredConstructors()throws SecurityException →** returns the total number of constructors of this class
- **public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException →** returns the method class instance.
- **public boolean isInterface():** determines if the specified Class object represents an interface type.
- **public boolean isArray():** determines if this Class object represents an array class.
- **public boolean isPrimitive():** determines if the specified Class object represents a primitive type.

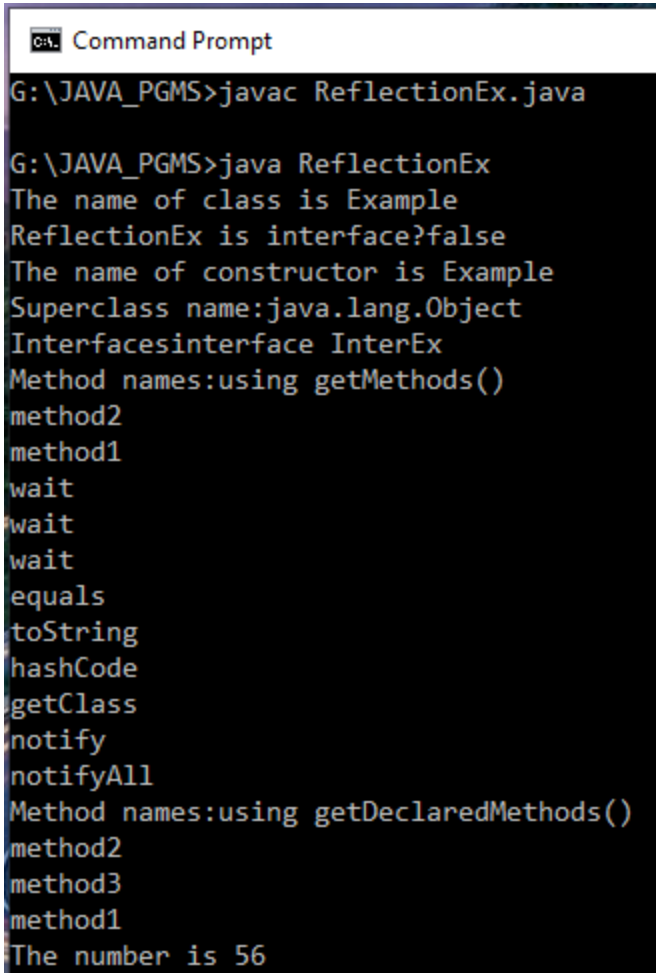
```
// A simple Java program to demonstrate the use of reflection
/*import java.lang.reflect.Method;
import java.lang.reflect.Field;
import java.lang.reflect.Constructor;*/

import java.lang.Class;
import java.lang.reflect.*;
//interface created
 interface InterEx
 {

 }

// class whose object is to be created
class Example implements InterEx
{
    // creating a private field
    private String str;
     // creating a public constructor
    public Example()
   {
                str = "welcome";
   }
```

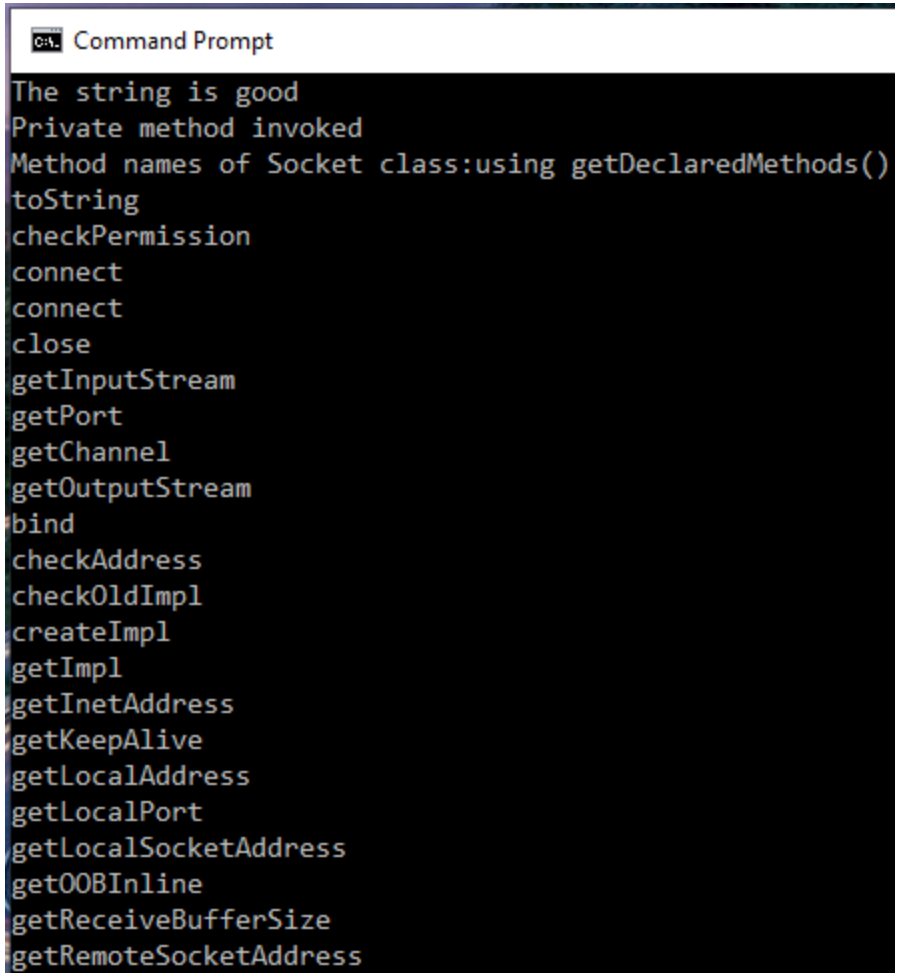
```
// Creating a public method with no arguments
    public void method1()
   {
       System.out.println("The string is " + str);
    }
     // Creating a public method with int as argument
    public void method2(int n)
   {
       System.out.println("The number is " + n);
    }
     // creating a private method
    private void method3()
   {
       System.out.println("Private method invoked");
    }
}
class ReflectionEx
{
    public static void main(String args[]) throws Exception
    {
       // Creating object whose property is to be checked
       Example obj = new Example();
        // Creating class object from the object using getclass method
       //Class cls = obj.getClass();
       Class<?> cls = obj.getClass();
```

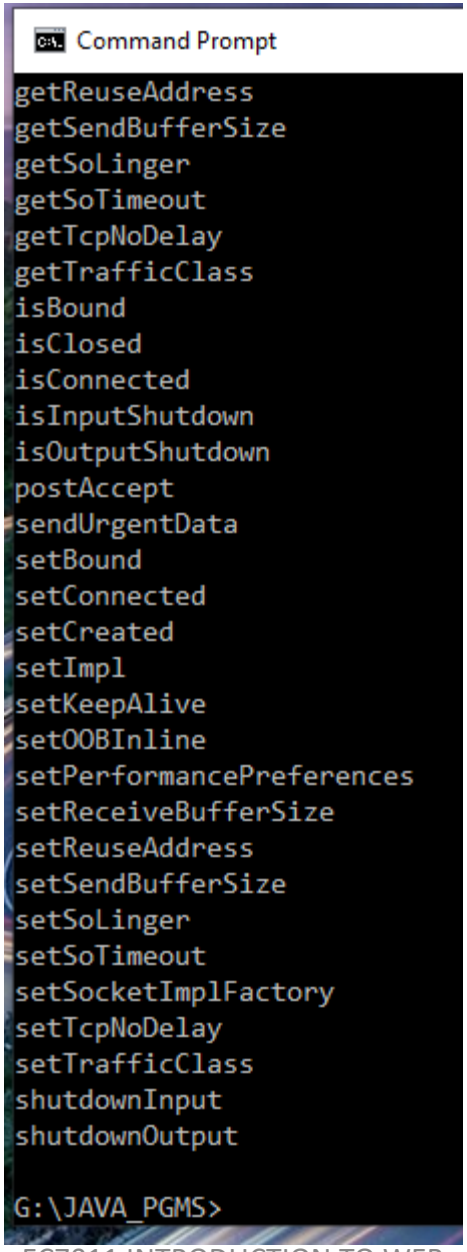
```
System.out.println("The name of class is " + cls.getName());
System.out.println("ReflectionEx is interface?"+cls.isInterface());

// Getting the constructor of the class through the object of the class
Constructor constructor = cls.getConstructor();
System.out.println("The name of constructor is " + constructor.getName());

//Getting superclass name
System.out.println("Superclass name:"+cls.getSuperclass().getName());

//Getting Interfaces
Class inter[]=cls.getInterfaces();
for(int j=0;j<inter.length;j++)
            System.out.println("Interfaces"+inter[j]);

// Getting methods of the class through the object of the class by using getMethods
//It does not show private method, but show  methods of superclass
 Method methods1[] = cls.getMethods();

// Printing method names
      System.out.println("Method names:using getMethods()");
      for(int i=0;i<methods1.length;i++)
         System.out.println(methods1[i].getName());
```

```
// Getting methods of the class through the object of the class by using getDeclaredMethods
      Method methods2[] = cls.getDeclaredMethods(); // show all  methods within the class
 // Printing method names
     System.out.println("Method names:using getDeclaredMethods()");
      for(int i=0;i<methods2.length;i++)
         System.out.println(methods2[i].getName());
  // creates object of desired method by providing the method name and parameter class as
  //arguments to the getDeclaredMethod
      Method methodcall1 = cls.getDeclaredMethod("method2",int.class);

   // invokes the method at runtime
      methodcall1.invoke(obj, 56);
      // creates object of the desired field by providing the name of field as argument to the
      // getDeclaredField method
       Field field=cls.getDeclaredField("str");

      // allows the object to access the field irrespective
      // of the access specifier used with the field
      field.setAccessible(true);

      // takes object and the new value to be assigned
      // to the field as arguments
      field.set(obj, "good”);
```

```
// Creates object of desired method by providing the
 // method name as argument to the getDeclaredMethod
 Method methodcall2 = cls.getDeclaredMethod("method1");
  // invokes the method at runtime
 methodcall2.invoke(obj);
  // Creates object of the desired method by providing
 // the name of method as argument to the
 // getDeclaredMethod method
 Method methodcall3 = cls.getDeclaredMethod("method3");
  // allows the object to access the method irrespective
 // of the access specifier used with the method
 methodcall3.setAccessible(true);
  // invokes the method at runtime
 methodcall3.invoke(obj);
//Getting details about in built classes
 java.net.Socket obj2=new java.net.Socket();
 Class cls2=obj2.getClass();
 Method methods3[] = cls2.getDeclaredMethods(); // show all  methods within the class
 // Printing method names
 System.out.println("Method names of Socket class:using getDeclaredMethods()");
 for(int i=0;i<methods3.length;i++)
    System.out.println(methods3[i].getName());
}
}
```


Command Prompt
G:\JAVA_PGMS>javac ReflectionEx.java
G:\JAVA_PGMS>java ReflectionEx
The name of class is Example
ReflectionEx is interface?false
The name of constructor is Example
Superclass name:java.lang.Object
Interfacesinterface InterEx
Method names:using getMethods()
method2
method1
wait
wait
wait
equals
toString
hashCode
getClass
notify
notifyAll
Method names:using getDeclaredMethods()
method2
method3
method1
The number is 56


Command Prompt
The string is good
Private method invoked
Method names of Socket class:using getDeclaredMethods()
toString
checkPermission
connect
connect
close
getInputStream
getPort
getChannel
getOutputStream
bind
checkAddress
checkOldImpl
createImpl
getImpl
getInetAddress
getKeepAlive
getLocalAddress
getLocalPort
getLocalSocketAddress
getOOBInline
getReceiveBufferSize
getRemoteSocketAddress


Command Prompt
getReuseAddress
getSendBufferSize
getSoLinger
getSoTimeout
getTcpNoDelay
getTrafficClass
isBound
isClosed
isConnected
isInputShutdown
isOutputShutdown
postAccept
sendUrgentData
setBound
setConnected
setCreated
setImpl
setKeepAlive
setOOBInline
setPerformancePreferences
setReceiveBufferSize
setReuseAddress
setSendBufferSize
setSoLinger
setSoTimeout
setSocketImplFactory
setTcpNoDelay
setTrafficClass
shutdownInput
shutdownOutput
G:\JAVA_PGMS>

Important observations

- We can invoke a method through reflection if we know its name and parameter types.
- We use below two methods for this purpose,

  1.**getDeclaredMethod() :** To create an object of method to be invoked. The syntax for this method is **Class.getDeclaredMethod(name, parametertype)**

  name- the name of method whose object is to be created

  parametertype - parameter is an array of Class objects

  2.invoke() : To invoke a method of the class at runtime we use following method–

  Method.invoke(Object, parameter)

  – If the method of the class doesn't accepts any parameter then null is passed as argument.
  – Through reflection **we can access the private variables and methods** of a class with the help of its class object and invoke the method by using the object as discussed above. We use below two methods for this purpose.
- **Class.getDeclaredField(FieldName) :** Used to get the private field. Returns an object of type Field for specified field name.
  Field.setAccessible(true) : Allows to access the field irrespective of the access modifier used with the field.

Advantages of Using Reflection:

- **Extensibility Features:** An application may make use of external, user-defined classes by creating instances of extensibility objects using their fully-qualified names.
- **Debugging and testing tools**: Debuggers use the property of reflection to examine private members on classes.

Drawbacks

- **Performance Overhead:** Reflective operations have slower performance than their non-reflective counterparts, and should be avoided in sections of code which are called frequently in performance-sensitive applications.
- **Exposure of Internals:** Reflective code breaks abstractions and therefore may change behavior with upgrades of the platform.

Application of Reflection API

- The Reflection API is mainly used in:
  - IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
  - Debugger
  - Test Tools etc.