Reflection

In this chapter we will learn about reflection API provided by Java. What is reflection, where reflection can be used and what are the classes and methods that are available in reflection API

Introduction to Reflection

Java Reflection is the process of analyzing and modifying all the capabilities of a class at runtime. Reflection API in Java is used to manipulate class and its members which include fields, methods, constructor, etc. at runtime.

One advantage of reflection API in Java is, it can manipulate private members of the class too.

The java.lang.reflect package provides many classes to implement reflection java.Methods of the java.lang.Class class is used to gather the complete metadata of a particular class.

Reflection is an "Application Programming Interface" (API) provided by Java.

The Reflection API is mainly used in:

- IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
- Debugger
- Test Tools etc.
- It is used, when your application has third-party libraries and when you want to know about the classes and methods available.

Thus a Reflection can be defined as a "technique of inspecting and modifying the runtime behavior of an unknown object at run time. An object can be a class, a field, or a method."

Class class

There is a class in Java named Class that keeps all the information about objects and classes at runtime. The object of Class can be used to perform reflection. In order to reflect a Java class, we first need to create an object of Class.

And, using the object we can call various methods to get information about methods, fields, and constructors present in a class. In short methods of the java.lang.Class class is used to gather the complete metadata of a particular class.

Here exists three ways to create objects of Class:

1. **Using forName() method**

```
// create object of Class
```

Class class

There is a class in Java named Class that keeps all the information about objects and classes at runtime. The object of Class can be used to perform reflection. In order to reflect a Java class, we first need to create an object of Class.

And, using the object we can call various methods to get information about methods, fields, and constructors present in a class. In short methods of the java.lang.Class class is used to gather the complete metadata of a particular class.

Here exists three ways to create objects of Class:

1. **Using forName() method**

   ```
   // create object of Class
   // to reflect the String class
   Class a = Class.forName("java.lang.String");
   ```

   Here, the forName() method takes the name of the class to be reflected as its argument. This class name should be fully qualified name of the class. This method throws checked exception as ClassNotFoundException

2. **Using getClass() method**

   ```
   // create an object of String class
   String s1 = new String();
   // create an object of Class// to reflect String
   Class b = s1.getClass();
   ```

   Here, we are using the object of the String class to create an object of Class.

3. **Using .class extension**

   ```
   // create an object of Class// to reflect the Dog class
   Class c = Dog.class;
   ```

   Now that we know how we can create objects of the Class. We can use this object to get information about the corresponding class at runtime.

Few methods avaialble in Class class :

Sr No	Return type	Method name	Description
1	Class<? super T	getSuperclass()	This method returns the Class which represents the superclass of the entity represented by this Class.
2	T	newInstance()	This method creates a new instance of the class represented by this Class object.

3. **Using .class extension**

// create an object of Class// to reflect the Dog class

Class c = Dog.class;

Now that we know how we can create objects of the Class. We can use this object to get information about the corresponding class at runtime.

Few methods avaialble in Class class :

Sr No	Return type	Method name	Description
1	Class<? super T	getSuperclass()	This method returns the Class which represents the superclass of the entity represented by this Class.
2	T	newInstance()	This method creates a new instance of the class represented by this Class object.
3	Package	getPackage()	It simply gets the package for this class.
4	int	getModifiers()	It returns the Java language modifiers for this class or interface, encoded in an integer.
5	Constructor[]	getConstructors()	Returns an array containing Constructor objects reflecting all the public constructors of the class represented by this Class object.
6	Method[]	getMethods()	It returns an array containing Method objects reflecting all the public methods of the class or interface represented by this Class object. It also includes the declared methods of the class or interface and those inherited from super classes and super interfaces
7	Class<?>[]	getInterfaces()	It determines the interfaces implemented by the interface or class represented by this object.
8	Field[]	getFields()	It returns an array containing Field objects reflecting all the accessible public fields of the interface or class represented by this Class object.

Creating Instance Dynamically

The idea is to use interfaces and Java reflection to allow your application to instantiate certain classes based on runtime configuration, instead of hard-coding instantiation of those classes.

Creating Instance Dynamically

The idea is to use interfaces and Java reflection to allow your application to instantiate certain classes based on runtime configuration, instead of hard-coding instantiation of those classes.

```
try
{
   String className= "com.cc.ABC";
   Class c = Class.forName(className);
   Object o = c.newInstance();
   com.cc.ABC a = (com.cc.ABC) o;
}
catch (Exception e)
{
  System.out.println(e.getMessage());
}
```

Dynamic Method Invocation

In Java you can invoke any method by its string name dynamically using reflection API.

java.lang.reflect API provides powerful reflection mechanism which can load classes by its name even if classes are not available at compile time, Can get all methods including private and public from class and allow you to invoke any method dynamically using reflection.

java.lang.reflect package have a class called Method which represent method Reflectively and Method.invoke() is used to call any Java method dynamically using reflection.

Method.invoke() takes an object whose method has to call and list of parameters to be passed to method and throws InvocationTargetException if called method throws any Exception.

```
public class ReflectMethodinvokeExample1  {
   private static void process(String str) {
      System.out.println("processing " + str);
   }
   public static void main(String... args) throws NoSuchMethodException,  InvocationTargetException,
IllegalAccessException {
      Method m = ReflectMethodinvokeExample1.class.getDeclaredMethod("process", String.class);
      Object rv = m.invoke(null, "test");
      System.out.println(rv);
```

Dynamic Method Invocation

In Java you can invoke any method by its string name dynamically using reflection API.

java.lang.reflect API provides powerful reflection mechanism which can load classes by its name even if classes are not available at compile time, Can get all methods including private and public from class and allow you to invoke any method dynamically using reflection.

java.lang.reflect package have a class called Method which represent method Reflectively and Method.invoke() is used to call any Java method dynamically using reflection.

Method.invoke() takes an object whose method has to call and list of parameters to be passed to method and throws InvocationTargetException if called method throws any Exception.

```
public class ReflectMethodinvokeExample1  {
   private static void process(String str) {
      System.out.println("processing " + str);
   }
   public static void main(String... args) throws NoSuchMethodException,  InvocationTargetException,
IllegalAccessException {
      Method m = ReflectMethodinvokeExample1.class.getDeclaredMethod("process", String.class);
      Object rv = m.invoke(null, "test");
      System.out.println(rv);
   }
}
```

Assignments

1. Accept fully qualified name from the user. Show all the constructors and methods in that class. And also show information about the class like package, super class, implemented interfaces, whether it is final or abstract. Show the message if class name is invalid.
2. Accept fully qualified name of the class from the user. Choose any constructor and create the instance dynamically and display the object.
3. Accept fully qualified name of the class from the user. Accept the method to be invoked on the instance of the class and display the outcome of the method.