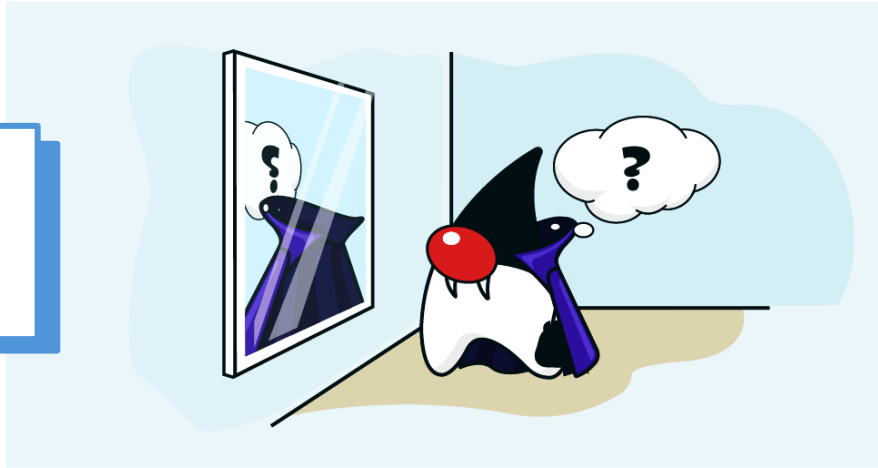
Reflection in Java

AP Fall-1404 Dr. Mojtaba Vahidi Asl
by Danial Yaghooti

What is Reflection?

Reflection is a feature in Java that allows a program to inspect and modify the structure and behavior of classes, methods, and fields at runtime.It enables accessing metadata and interacting with objects dynamically.

Reflection is exposed to developers as a set of standard Java APIs (Application Programming Interfaces) found primarily in the java.lang and java.lang.reflect packages.

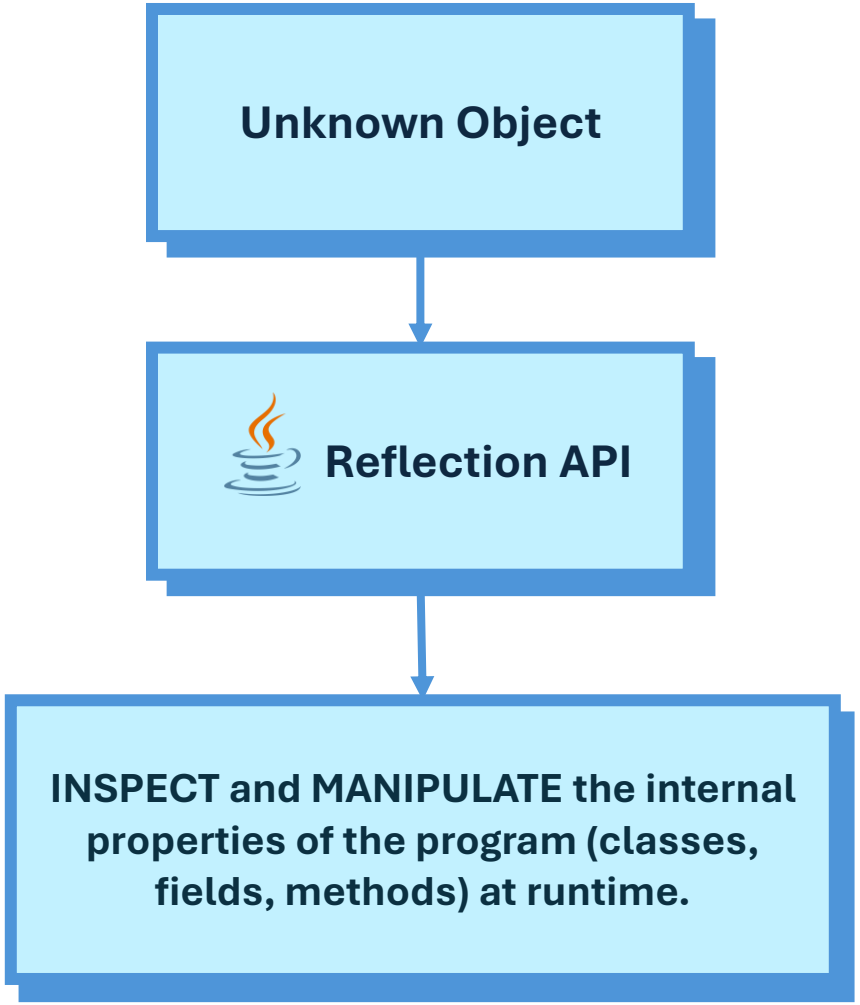


💡 Think of Reflection as a program looking at itself in the mirror and having the ability to change its own appearance *while* it's running.

Features

- Reflection gives us information about the class to which an object belongs and also the methods of that class that can be executed by using the object.

- Through reflection, we can invoke methods at runtime irrespective of the access specifier used with them.



Obtaining the `Class` Object

You cannot use Reflection without first obtaining the Class object for the target class.

There are three common ways to get Class:

1.Using Class.forName()(fully qualified class name):

```
Class<?> personClass = Class.forName("com.example.Person");
```

2.Using .class syntax (if the class is available at compile time):

```
Class<Person> personClass = Person.class;
```

3. Using .getClass() on an object instance:

```
Person person = new Person("Adam");
Class<? extends Person> personClass = person.getClass();
```

```
public class Person {
    private String name;
    public int age;

    public Person(String name){
        this(name, 18);
    }
    public Person(String name,int age){
          this.name = name;
          this.age = age;
    }
    public String getName(){
        return "Hi I`m " + name;
    }
}
```

Our Target Class

Listing Class Methods

The Class object provides methods to get information about the class's methods and fields such as:

```
SampleClass.getMethods(); //Returns Array of Methods
```

Example:

```
Class<?> PersonClass = Person.class;

// 1. getMethods(): Returns all PUBLIC methods (including inherited ones)
Method[] publicMethods = serviceClass.getMethods();

System.out.println("Public Methods:");
for (Method method : publicMethods) {
System.out.println(" > " + method.getName());
}

// Output includes: getName, wait, equals, toString, etc. (inherited)
```

Examining Fields

We can inspect the fields (instance variables) of a class in two ways:

```
1. SampleClass.getFields(); //Returns Only PUBLIC Fields
```

```
2. SampleClass.getDeclaredFields(); //Returns ALL declared fields
```

Example:

```
Field[] fields = personClass.getDeclaredFields();

for (Field field : fields)
{
    System.out.println("Field: " + field.getName() + ", Type: " + field.getType());
}

// Output: Field: name, Type: class java.lang.String
//         Field: age, Type: int
```

Examining Fields

Access the value of a field from an object:

```
Person p1 = new Person(“Negar", 21);

Field nameField = personClass.getDeclaredField("name");

nameField.setAccessible(true);

String nameValue = (String) nameField.get(p1);
System.out.println("Name value: " + nameValue);

// Output: Name value: Negar
```

⚠ To access PRIVATE field’s value, it is necessary to enable your accessibility by using: `.setAccessible(true);`

Since Java 9, setAccessible(true) is not guaranteed due to stricter security rules. In Java 16+, this code will cause your program to crash unless the module explicitly allows access.

`Field.get();` returns **Object** type so we needed to down-casting into **String** type.

Modifying Field Values

To change field values at Runtime, Use the `Field.set();`

```
2. field.set(TargetObject, TargetFieldType);
```

Example:

```
Person p = new Person("Shokooh", 31);
System.out.println("Before: " + p.getName());
// Output: Before: Shokooh

Field nameField = personClass.getDeclaredField("name");

nameField.setAccessible(true);
nameField.set(p, "Diana");

System.out.println("After: " + p.getName());
// Output: After: Diana
```

Dynamic Method Invocation

The Method object allows you to execute a method at runtime using the invoke() method.

Like Field object you can access all methods including PRIVATE ones by using:

```
Method[] method = sampleClass.getDeclaredMethods(); //Returns Array of All Methods
```

Example1:

```
public class Calculator {
        public int sum(int a, int b) { return a + b; }
}

Calculator calc = new Calculator();
Class<?> calcClass = Calculator.class;

// 1. Get the Method object for 'sum’
Method sumMethod = calcClass.getDeclaredMethod("sum", int.class, int.class);

// 2. Invoke the method:
int result = (int) sumMethod.invoke(calc, 10, 5);
System.out.println("Result: " + result);
// Output: Result: 15
```

Dynamic Method Invocation

We used .getDeclaredMethod(); to access one specific method. In general for this purpose we should use following structure:

```
sampleClass.getDeclaredMethod("MethodName", ParameterNames);
```

Such as:

```
Method method = studentClass.getDeclaredMethod("updateInfo",
    String.class, int.class, boolean.class
);
```

Or pass null parameter if it has no input argument.

 *studentClass* is type of **Class** object.

⚠ This actions throws .NoSuchMethodException so you need to use try-catch block or another way to handle it.

Dynamic Instance Creation

Reflection allows frameworks to create objects even when the class is only known as a string at runtime. Use the following method to get all public constructors:

```
sampleClass.getConstructors(); //Returns array of PUBLIC constructors
```

Example:

```
Class<Person> personClass = Person.class;

Constructor<?>[] publicConstructors = personClass.getConstructors();

System.out.println("Public Constructors:");
for (Constructor<?> constructor : publicConstructors) {
    System.out.println(" - " + constructor);
}

//out:
        - public Person(java.lang.String,int)
        - public Person(java.lang.String)
```

Target Class

```
public class Person {
   private String name;
   private int age;
   private String email;

// Public constructors
    public Person() {}
    public Person(String name) {
        this.name = name;    }
    public Person(String name, int age) {
        this.name = name;
        this.age = age;    }

 // Private constructor
        private Person(String name, String email) {
        this.name = name;
        this.email = email;
    }
}
```

Dynamic Instance Creation

Now we are going to get all constructors(Including PRIVATE ones) by using:

```
sampleClass.getDeclaredConstructors(); //Returns array of All constructors
```

Example:

```
Class<Person> personClass = Person.class;

Constructor<?>[] allConstructors = personClass. getDeclaredConstructors();

System.out.println("ALL Constructors:");
for (Constructor<?> constructor : allConstructors) {
    System.out.println(" - " + constructor);
}

//out:
        - public Person(java.lang.String,int)
        - public Person(java.lang.String)
        - public Person(java.lang.String,int)
        - private Person(java.lang.String,java.lang.String)
```

Dynamic Instance Creation

To get specific constructors:

```
sampleClass.getDeclaredConstructor(ParamitersType); //Returns Specific Constructor
```

Example:

```
// Get specific constructor with exact parameter types
try {

Constructor<Person> specificConstructor =
personClass.getDeclaredConstructor(String.class, int.class);
System.out.println("Specific Constructor: " + specificConstructor);

// Create instance using the constructor
Person person = specificConstructor.newInstance("PariNaz", 25);
System.out.println("Created: " + person.getName() + ", " + person.getAge());

    } catch (NoSuchMethodException e) {
    System.out.println("Constructor not found!");
}
```

Dynamic Instance Creation

You can also Find Required Fields for Each Constructor by using:

```
Class<?>[] paramTypes = constructor.getParameterTypes();
```

```
Constructor<?>[] constructors = personClass.getDeclaredConstructors();
Constructor<?> C1 = constructors[2]; //Person(String name, int age);
Class<?>[] paramTypes = C1.getParameterTypes();

System.out.println("\nConstructor: " + constructor.getName());
System.out.println("Parameter count: " + paramTypes.length);

for (int i = 0; i < paramTypes.length; i++) {
        System.out.println("  Param " + i + ": " + paramTypes[i].getSimpleName());
    }

//out:
        Constructor: Person
        Parameter count: 2
        Param 0: String
        Param 1: int
```

Find more about Reflection

Thank You for Your Time ()[_])