Day 24

Task 3: Reflection API

Use reflection to inspect a class's methods, fields, and constructors, and modify the access level of a private field, setting its value during runtime.

A)

To use reflection in Java to inspect a class's methods, fields, and constructors, and to modify the access level of a private field to set its value during runtime, follow the steps below.

Java code:

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

   public Person() {
      this.name = "Unknown";
      this.age = 0;
      this.salary = 0.0;
   }

   public Person(String name, int age, double salary) {
      this.name = name;
      this.age = age;
      this.salary = salary;
   }

   private void displayPrivateInfo() {
      System.out.println("Private Info: Name - " + name + ", Age - " + age + ", Salary - " + salary);
   }

   public void displayPublicInfo() {
      System.out.println("Public Info: Name - " + name + ", Age - " + age);
   }
}
```

Using Reflection

Now, let's write code to inspect the Person class and modify the private field salary.

Java code:

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
```

```
public class ReflectionPersonExample {
   public static void main(String[] args) {
      try {
         // Load the Person class
         Class<?> personClass = Class.forName("Person");

         // Inspect constructors
         System.out.println("Constructors:");
         Constructor<?>[] constructors = personClass.getDeclaredConstructors();
         for (Constructor<?> constructor : constructors) {
            System.out.println(constructor);
         }

         // Inspect fields
         System.out.println("\nFields:");
         Field[] fields = personClass.getDeclaredFields();
         for (Field field : fields) {
            System.out.println(field);
         }

         // Inspect methods
         System.out.println("\nMethods:");
         Method[] methods = personClass.getDeclaredMethods();
         for (Method method : methods) {
            System.out.println(method);
         }

         // Create an instance of Person
         Constructor<?> constructor = personClass.getDeclaredConstructor(String.class, int.class,
double.class);
         Object personInstance = constructor.newInstance("John Doe", 30, 50000.0);

         // Access and modify the private field 'salary'
         Field salaryField = personClass.getDeclaredField("salary");
         salaryField.setAccessible(true); // Bypass the private modifier
         System.out.println("\nOriginal salary value: " + salaryField.get(personInstance));

         // Modify the private field
         salaryField.set(personInstance, 60000.0);
         System.out.println("Modified salary value: " + salaryField.get(personInstance));

         // Call a private method
         Method privateMethod = personClass.getDeclaredMethod("displayPrivateInfo");
         privateMethod.setAccessible(true); // Bypass the private modifier
         privateMethod.invoke(personInstance);

      } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

Explanation:

1. Loading the Class:

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

This loads the Person class dynamically.

2. Inspecting Constructors:

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

This retrieves all constructors (public and private) of the class.

3. Inspecting Fields:

Field[] fields = personClass.getDeclaredFields();

This retrieves all fields (public and private) of the class.

4. Inspecting Methods:

Method[] methods = personClass.getDeclaredMethods();

This retrieves all methods (public and private) of the class.

5. Creating an Instance:

Constructor<?> constructor = personClass.getDeclaredConstructor(String.class, int.class, double.class);

Object personInstance = constructor.newInstance("John Doe", 30, 50000.0);

This creates a new instance of the Person class using the parameterized constructor.

6. Modifying a Private Field:

Field salaryField = personClass.getDeclaredField("salary");

salaryField.setAccessible(true);

salaryField.set(personInstance, 60000.0);

This makes the private field salary accessible, reads its value, modifies it, and prints the new value.

7. Calling a Private Method:

```
Method privateMethod = personClass.getDeclaredMethod("displayPrivateInfo");

privateMethod.setAccessible(true);

privateMethod.invoke(personInstance);
```

This makes the private method displayPrivateInfo accessible and invokes it.

This example demonstrates how to use Java reflection to inspect and manipulate another class's details at runtime.