SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

NAME: Ayush Vinod Upadhyay
ROLL NO: I025
SAP ID: 60003220131
BRANCH: Information Technology
BATCH: 1

EXPERIMENT NO. 06

CO/LO:

CO1- Modify the behaviour of methods, classes, and interfaces at runtime.

AIM / OBJECTIVE:

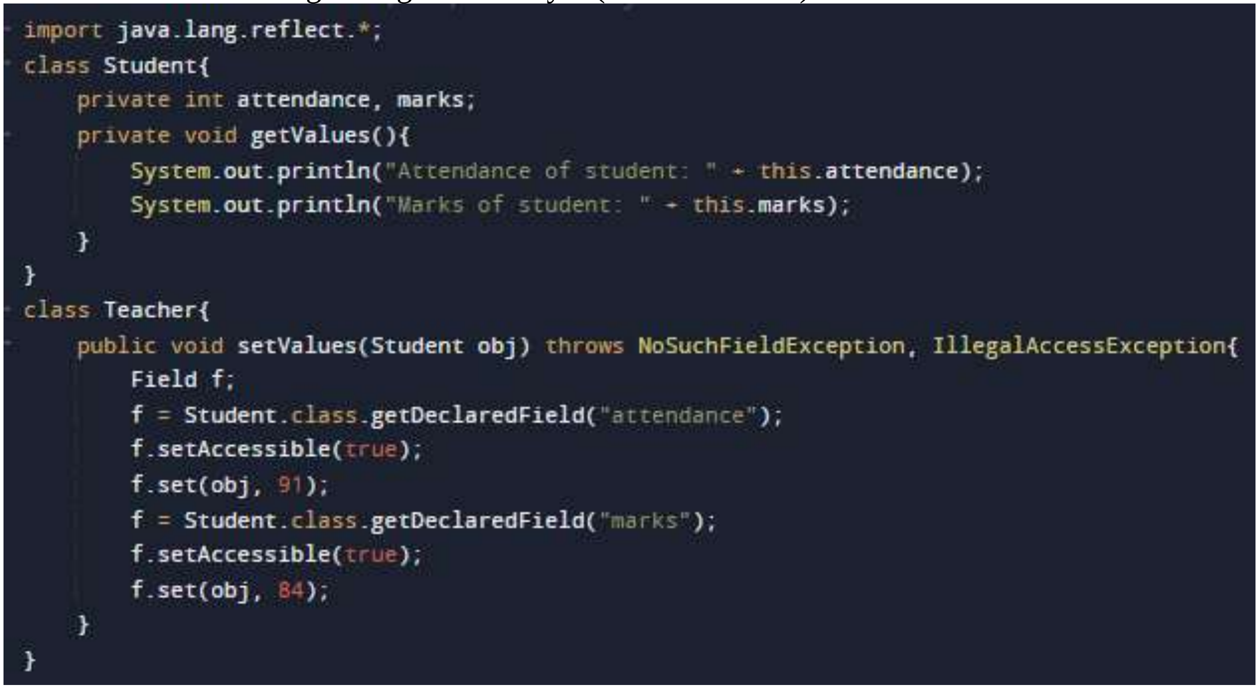
Use reflection API to examine or modify the behavior of methods, classes, and interfaces at runtime.

PROBLEM STATEMENTS:

Create a class student with private members attendance and marks. Create a class teacher who sets the values for marks and attendance. Finally create a class parent who creates a reflection of methods to know the values of marks and attendance of the student.

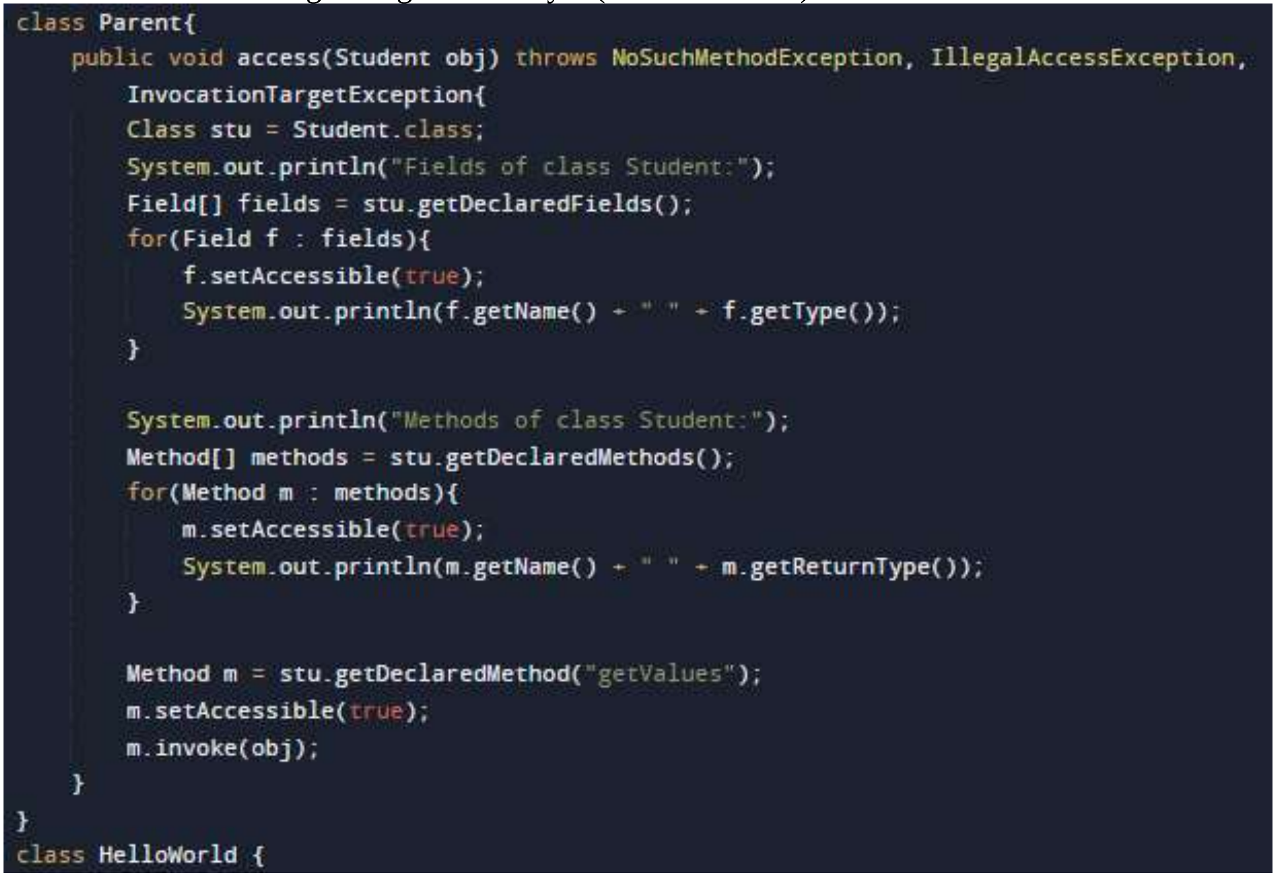
Code:

```
// Online Java Compiler
// Use this editor to write, compile and run your Java code online
import java.lang.reflect.*;
class Student{
    private int attendance, marks;
    private void getValues(){
        System.out.println("Attendance of student: " + this.attendance);
        System.out.println("Marks of student: " + this.marks);
    }
}
class Teacher{
    public void setValues(Student obj) throws NoSuchFieldException, IllegalAccessException{
        Field f;
        f = Student.class.getDeclaredField("attendance");
        f.setAccessible(true);
        f.set(obj, 91);
        f = Student.class.getDeclaredField("marks");
        f.setAccessible(true);
        f.set(obj, 84);
    }
}
class Parent{
    public void access(Student obj) throws NoSuchMethodException, IllegalAccessException,
InvocationTargetException{
        Class stu = Student.class;
        System.out.println("Fields of class Student:");
        Field[] fields = stu.getDeclaredFields();
        for(Field f : fields){
            f.setAccessible(true);
            System.out.println(f.getName() + " " + f.getType());
        }

        System.out.println("Methods of class Student:");
        Method[] methods = stu.getDeclaredMethods();
        for(Method m : methods){
            m.setAccessible(true);
            System.out.println(m.getName() + " " + m.getReturnType());
        }
        Method m = stu.getDeclaredMethod("getValues");
        m.setAccessible(true);
        m.invoke(obj);
    }
}
class HelloWorld {
    public static void main(String[] args) throws NoSuchFieldException, NoSuchMethodException, IllegalAccessException, InvocationTargetException{
        //System.out.println("Hello, World!");
        Student s1 = new Student();
        Teacher t1 = new Teacher();
        t1.setValues(s1);
        Parent p = new Parent();
        p.access(s1);
    }
}
```

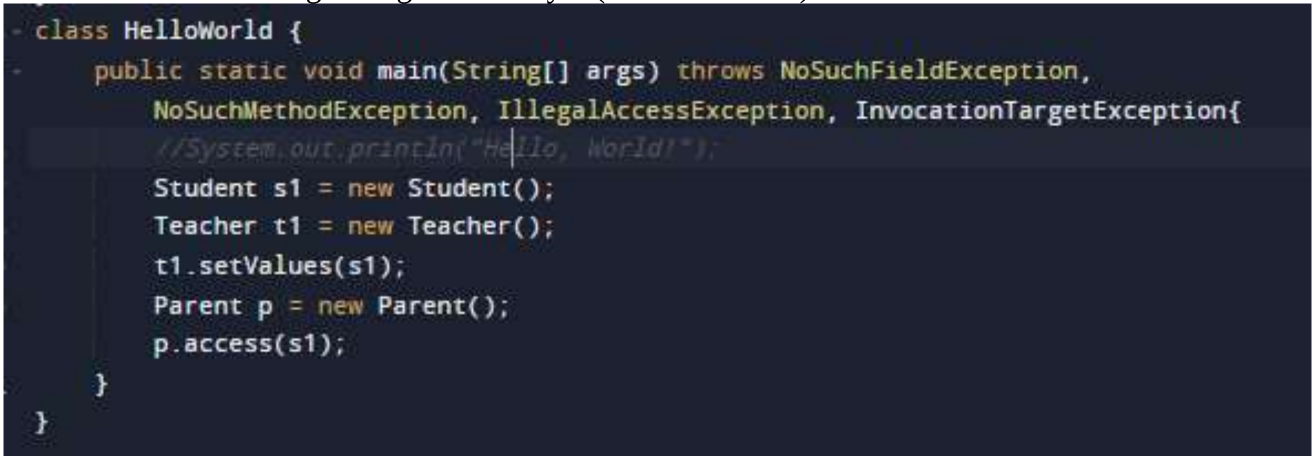


```java
import java.lang.reflect.*;
class Student{
    private int attendance, marks;
    private void getValues(){
        System.out.println("Attendance of student: " + this.attendance);
        System.out.println("Marks of student: " + this.marks);
    }
}
class Teacher{
    public void setValues(Student obj) throws NoSuchFieldException, IllegalAccessException{
        Field f;
        f = Student.class.getDeclaredField("attendance");
        f.setAccessible(true);
        f.set(obj, 91);
        f = Student.class.getDeclaredField("marks");
        f.setAccessible(true);
        f.set(obj, 84);
    }
}
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

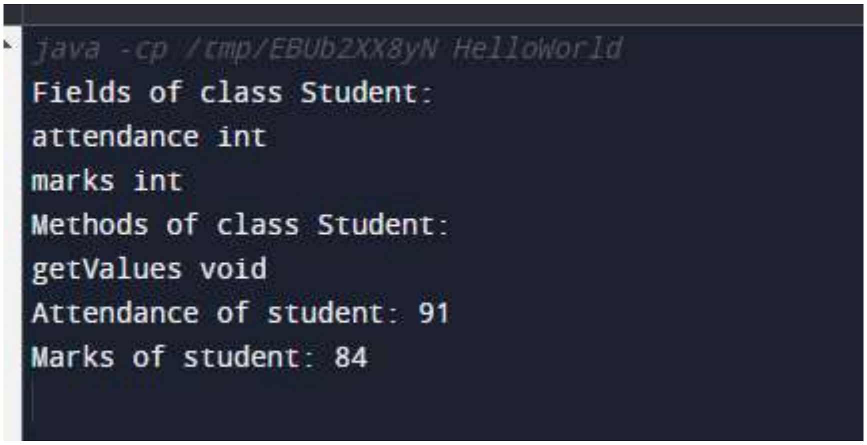


```
class Parent{
    public void access(Student obj) throws NoSuchMethodException, IllegalAccessException,
        InvocationTargetException{
        Class stu = Student.class;
        System.out.println("Fields of class Student:");
        Field[] fields = stu.getDeclaredFields();
        for(Field f : fields){
            f.setAccessible(true);
            System.out.println(f.getName() + " " + f.getType());
        }

        System.out.println("Methods of class Student:");
        Method[] methods = stu.getDeclaredMethods();
        for(Method m : methods){
            m.setAccessible(true);
            System.out.println(m.getName() + " " + m.getReturnType());
        }

        Method m = stu.getDeclaredMethod("getValues");
        m.setAccessible(true);
        m.invoke(obj);
    }
}
class HelloWorld {
```



```
class HelloWorld {
    public static void main(String[] args) throws NoSuchFieldException,
        NoSuchMethodException, IllegalAccessException, InvocationTargetException{
        //System.out.println("Hello, World!");
        Student s1 = new Student();
        Teacher t1 = new Teacher();
        t1.setValues(s1);
        Parent p = new Parent();
        p.access(s1);
    }
}
```

output :



```
java -cp /tmp/EBUb2XX8yN HelloWorld
Fields of class Student:
attendance int
marks int
Methods of class Student:
getValues void
Attendance of student: 91
Marks of student: 84
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

OBSERVATION:

Explain the reflection API and its uses.

API Reflection allows a program to examine or "reflect" on its own structure while it's running. Imagine your code is a house, and reflection is a tool that lets you look at the blueprints of the house and figure out things like the number of rooms, their sizes, and what they're used for while you're already inside the house.

Uses of Reflection:

- **Inspecting Structures**: It helps to see what classes, methods, and fields exist in your code at runtime.
- **Dynamic Loading**: Allows loading classes, methods, and objects at runtime, which can be handy for plugins or extensions.
- **Configuration or Metadata**: Useful for reading configuration files or annotations to change how code behaves.
- **Debugging and Testing**: Allows tools to understand and manipulate code during debugging or testing processes.

However, while reflection is powerful, it can also make code less readable, harder to maintain, and might impact performance negatively. So, it's essential to use it carefully and only when needed.

CONCLUSION:

In this experiment I learnt about the reflection API and how it is useful to access, get information and manipulate classes, methods, interfaces and fields at runtime.