Lab2

March 20, 2025

1 Lab 2

Deadline: **Week 3** in your respective lab session

1.0.1 Name: Kiril Radev

1.0.2 Student ID: 241005831

1.1 Question 1 [1 mark]

Write a class `BankAccount`, with instance variables `accountNumber` and `balance`. The `balance` by default should be set to `0`. Apart from appropriate accessor methods and a constructor, you need to implement two instance methods: `deposit` and `withdraw`.

`deposit` - should take any number as an argument, check if it is a valid number (greater than `0`), and add it to the `balance` if it is a valid number. If successful, return `true`; otherwise, return `false`.

`withdraw` - should take any number as an argument, check if it is a valid number (greater than `0` and less than the `balance`) and deduct it from the balance if it is a valid number. If successful, return `true`; otherwise, return `false`.

Lastly define class `Main` with the main method to test your code.

The main objective of this exercise is to use an appropriate access modifier to encapsulate the data.

Write your answer below:

[5]:

```java
public class BankAccount
{
        String accountNumber;
        double balance;

        //constructor
        public BankAccount(String newAccountNumber)
        {
                this.accountNumber = newAccountNumber;
                this.balance = 0;
        }
        //accessor methods
        public String getAccountNumber()
        {
                return accountNumber;
        }
        public double getBalance()
        {
                return balance;
        }

        public boolean deposit(double depositAmount)
        {
                //deposit validation
                if(depositAmount <= 0)
                {
                        return false;
                }

                balance += depositAmount;
                return true;
        }

        public boolean withdraw(double withdrawAmount)
        {
                //withdraw validation
                if(withdrawAmount < 0 || withdrawAmount > balance)
                {
                        return false;
                }

                balance -= withdrawAmount;
                return true;
        }
}
```

[6]:

```java
public class Main1 {
    public static void main(String[] args) {
        BankAccount ba = new BankAccount("123456789");

        System.out.println(ba.getAccountNumber());
        System.out.println(ba.getBalance());

        double depositAmount = 100;
        System.out.println(ba.deposit(depositAmount));        // true

        double invalidDepositAmount = -100;
        System.out.println(ba.deposit(invalidDepositAmount)); // false
        double withdrawAmount = 50;
        System.out.println(ba.withdraw(withdrawAmount));      // true

        double tooHighWithdrawal = 200;
        System.out.println(ba.withdraw(tooHighWithdrawal));   // false

        double tooLowWithdrawal = -200;
        System.out.println(ba.withdraw(tooLowWithdrawal));    // false

        System.out.println(ba.getAccountNumber());
        System.out.println(ba.getBalance());
    }
}
```

Run your program:

[7]:

```
Main1.main(null)
```

```
123456789
0.0
true
false
true
false
false
123456789
50.0
```

1.2 Question 2 [1 mark]

Write a class `Student` with 2 instance variables, `name` and `id`. It also contains a constructor which initialises `name` and `id` to the values passed as an argument.

Implement a class method (i.e. a static method) `checkDuplicates` inside the `Student` class, which takes an array of `Student` elements as an argument and checks whether there are two identical students in the array. If yes, it should return `true` and `false` otherwise.

Lastly, define class `Main2` with the main method and test your code. Test at least one array with a duplicate and one without duplicates.

Write your answer below:

[8]:

```
public class Student
{
        String name;
        int id;
        public Student(String new_name, int new_id)
        {
                this.name = new_name;
                this.id = new_id;
        }

        static boolean checkDuplicates(Student[] s)
        {
                //linear search
                for(int i = 0; i < s.length; i++)
                {
                        for(int j = i + 1; j < s.length; j++)
                        {
                                //check both name and id
                                if(s[i].name.equals(s[j].name) && s[i].id ==␣
 ↪s[j].id)
                                {
                                        return true;
                                }
                        }
                }

                return false;
    }
}
```

[9]:

```
public class Main2 {
    public static void main(String[] args) {

        Student[] studentsArrayWithDuplicate = {
                new Student("Alice", 1),
                new Student("Bob", 2),
                new Student("Charlie", 3),
                new Student("Alice", 1)
        };

        Student[] studentsArrayWithoutDuplicate = {
                new Student("Alice", 1),
                new Student("Bob", 2),
                new Student("Charlie", 3),
                new Student("David", 4)
        };

        System.out.println(Student.checkDuplicates(studentsArrayWithDuplicate));
        System.out.println(Student.
 ↪checkDuplicates(studentsArrayWithoutDuplicate));
    }
}
```

Run your program:

```
[10]: Main2.main(null);
```

```
true
false
```

1.3 Question 3 [1 mark]

Write a method `sortStudents` which, given an array of `Student` elements, sorts it using the Bubble Sort algorithm by `name` in alphabetical order. If more than one student has the same `name`, sort it by `id` in ascending order. You can assume that each `id` is unique.

Test your code!

Write your answer below:

```
[11]: //following the lexicological order format
public static void sortStudents(Student[] s)
{
        int SIZE = s.length - 1;

        for(int j = 0; j < SIZE; j++)
        {
                for(int position = 0; position < SIZE; position++)
                {
                        //assign variables
                        String name1 = s[position].name.toLowerCase();
                        String name2 = s[position + 1].name.toLowerCase();
                        int id1 =  s[position].id;
                        int id2 =  s[position + 1].id;

                        //check names
                        if(name1.equals(name2))
                        {
                                //check if id[1] > id[2], then swap
                                if(id1 > id2)
                                {
                                        swap(s, position, position + 1);
                                }
                        }
                        //sort names in alphabetical order
                        else
                        {
                                int length = name2.length();
```

```
                                        //firstly check the length of the names for the loop boundary
                                        if(name1.length() < name2.length())
                                        {
                                                length = name1.length();
                                        }

                                        //check if they are equal up to an element and sort them by length
                                        if(checkIfEqualUpToAnElement(name1, name2, length))
                                        {
                                                swap(s, position, position + 1);
                                                continue;
                                        }

                                        for(int i = 0; i < length; i++)
                                        {

                                                //if char[1] < char[2], then the names are in order and break the loop
                                                if(name1.charAt(i) < name2.charAt(i))
                                                {
                                                        break;
                                                }
                                                //if char[1] == char[2], then we don`t know if the names are in order, so we continue the loop
                                                else if(name1.charAt(i) == name2.charAt(i))
                                                {
                                                        continue;
                                                }
                            //if char[1] < char[2], then the names must be swapped and break the loop
                                                else
                                                {
                                                        swap(s, position, position + 1);
                                                        break;
                                                }
                                        }
                                }//END else
                        }//END for-in
                }//END for-out
}//END sortStudents

//a method that swaps two consecutive elements
```

```
public static void swap(Student[] s, int first, int second)
{
        Student temporary = s[second];
        s[second] = s[first];
        s[first] = temporary;
}//END swap

//a method that checks string equality up to an element
public static boolean checkIfEqualUpToAnElement(String n1, String n2, int l)
{
        for(int i = 0; i < l; i++)
        {
                if(n1.charAt(i) != n2.charAt(i))
                {
                        return false;
                }
        }

        return true;
}//END checkIfEqualUpToAnElement
```

Run your program:

[12]:

```
Student[] students = {
    new Student("John", 3),
    new Student("Alice", 2),
    new Student("Bob", 1),
    new Student("Bob", 5),
    new Student("Cam", 4),
    new Student("Ali", 5)
};

sortStudents(students);

for (Student s : students)
    System.out.println("(" + s.name + "," + Integer.toString(s.id) + ")" );
```

```
(Ali,5)
(Alice,2)
(Bob,1)
(Bob,5)
(Cam,4)
(John,3)
```

1.4 Question 4 [1 mark]

Notice that for the Student class of the previous two questions, you can create two objects s1 and s2 which have identical id and name. In this exercise we will modify the Student class to make creating such two objects impossible. It is for this reason that the constructor of the modified class Student4 below is set to private. This makes it impossible to create objects of this class from outisde. Objects instead will be created by the static method register.

Modify the static method register to check whether a student with this name and id was registered before. If yes, return a reference to the previously created instance of a Student4; if not, create a new instance of Student4 using passed values and return its reference. You are allowed to modify the Student4 class to achieve this.

Define the Main4 class to test your code. You should check whether the register function is returning the correct reference and whether it prints out the names of all registered students.

You can assume that the maximum number of registered students does not exceed 30.

HINT: Keep track of the instances that have been created before by using a static array of type Student4.

Write your answer below:

[1]:

```
public class Student4
{
        String name;
    int id;

    private Student4(String name, int id)
    {
        this.name = name;
        this.id = id;
    }//END Student4


    //objects & methods open to outside modification
    static int numberOfStudents = 0;                                  //␣
↪base case number of students
    final static int MAX_STUDENTS = 30;                               //max␣
↪students set by the format
    static Student4[] registeredStudents = new Student4[MAX_STUDENTS]; //array␣
↪of students by the format


    //use method register to modify the public objects
        public static Student4 register(String name, int id)
    {
                //check if the new student`s name & id are the same with an␣
↪existing one in Student4[] registeredStudents
                for (int i = 0; i < numberOfStudents; i++)
```

```
            {
                        if(registeredStudents[i].name.equals(name) && registeredStudents[i].id == id)
                        {
                                return registeredStudents[i]; //return the reference of the pre-existing student
                        }
            }

                Student4 new_student = new Student4(name,id);
                addNewRegisteredStudent(new_student);

                return new_student; //return the reference to the new student
    }//END register

        //add the new student by using pass by reference
        public static void addNewRegisteredStudent(Student4 new_student)
        {
                if(numberOfStudents < MAX_STUDENTS - 1)
                {
                        registeredStudents[numberOfStudents] = new_student;
                        numberOfStudents++;
                }
        }//END addNewRegisteredStudent
}//END Student4
```

[3]:

```
public class Main4
{
    public static void main(String[] args)
    {
        Student4 student1 = Student4.register("John", 123);
        Student4 student2 = Student4.register("Jane", 456);
        Student4 student3 = Student4.register("John", 123);
        Student4 student4 = Student4.register("Jane", 456);
        Student4 student5 = Student4.register("Cate", 389);

        //== compares the references to the objects
        System.out.println(student1 == student3); // true
        System.out.println(student1 == student2); //false
        System.out.println(student2 == student4); // true
        System.out.println(student4 == student5); //false

        //print the register
        for (int i = 0; i < Student4.numberOfStudents; i++)
        {
            System.out.println(Student4.registeredStudents[i].name + " " + Student4.registeredStudents[i].id);
        }

    }//END main
}//END Main4
```

Run your program:

[4]:

```
Main4.main(null);
```

```
true
false
true
false
John 123
Jane 456
Cate 389
```

1.5 Question 5 [1 mark]

Consider the class `Employee` below, it has two instance variables `name` of type `String` and `manager` of type `Employee`.

A *district manager* is an employee that does not have any manager, i.e. its `manager` instance variable is set to `null`. Write the method `getDistrictManager()` which returns the district manager of the given employee.

In other words if we have employees `jane`, `joe`, and `john`, such that `jane` is the manager of `joe` and `joe` is the manager of `john`, and moreover `jane` does not have a manager; then calling `john.getDistrictManager()` should return an object reference to `jane`.

Finally test your code with the example below.

Write your answer below:

[3]:

```
class Employee
{
    String name;
    Employee manager;

    Employee(String name, Employee manager)
    {
        this.name = name;
        this.manager = manager;
    }

    public Employee getDistrictManager()
    {
            //create an object temp_employee that stores the current employee
            Employee temp_employee = new Employee(name,manager);

            if(manager != null) //if not null we begin the recursive method
            {
                    temp_employee = manager.getDistrictManager();
                    //when the recursion gets to its closing process
                    //the final manager is returned, so temp_employee becomes␣
 ↪the [final manager]
                    //this means that with every closing iteration of the␣
 ↪recursion
                    //temp_employeee keeps on becoming the [final manager]

                    //System.out.println(1);
                    //System.out.println(name);
                //System.out.println("---");

            }

            //if manager is null, then the current employee is its own manager,
            //so temp_employee returns the employee itself
            return temp_employee;
    }//END getDistrictManager
}//END Employee
```

[2]:

```
class Main5
{
    public static void main(String[] args)
    {
            //1
        Employee jane = new Employee("jane", null);
        Employee joes = new Employee("joes", jane);
        Employee john = new Employee("john", joes);
        //2
        Employee kiri = new Employee("kiri", null);
        Employee kris = new Employee("kris", kiri);
        Employee pres = new Employee("pres", kris);
        Employee kami = new Employee("kami", pres);
        Employee lili = new Employee("lili", kami);

        //example 1
        System.out.println(john.getDistrictManager().name); // should print jane
        System.out.println(jane.getDistrictManager().name); // should print jane
        //example 2
        System.out.println(lili.getDistrictManager().name); // should print kiri
        System.out.println(pres.getDistrictManager().name); // should print kiri

    }//END main
}//END Main5
```

Run your program:

[4]:

```
Main5.main(null);
```

```
jane
jane
kiri
kiri
```