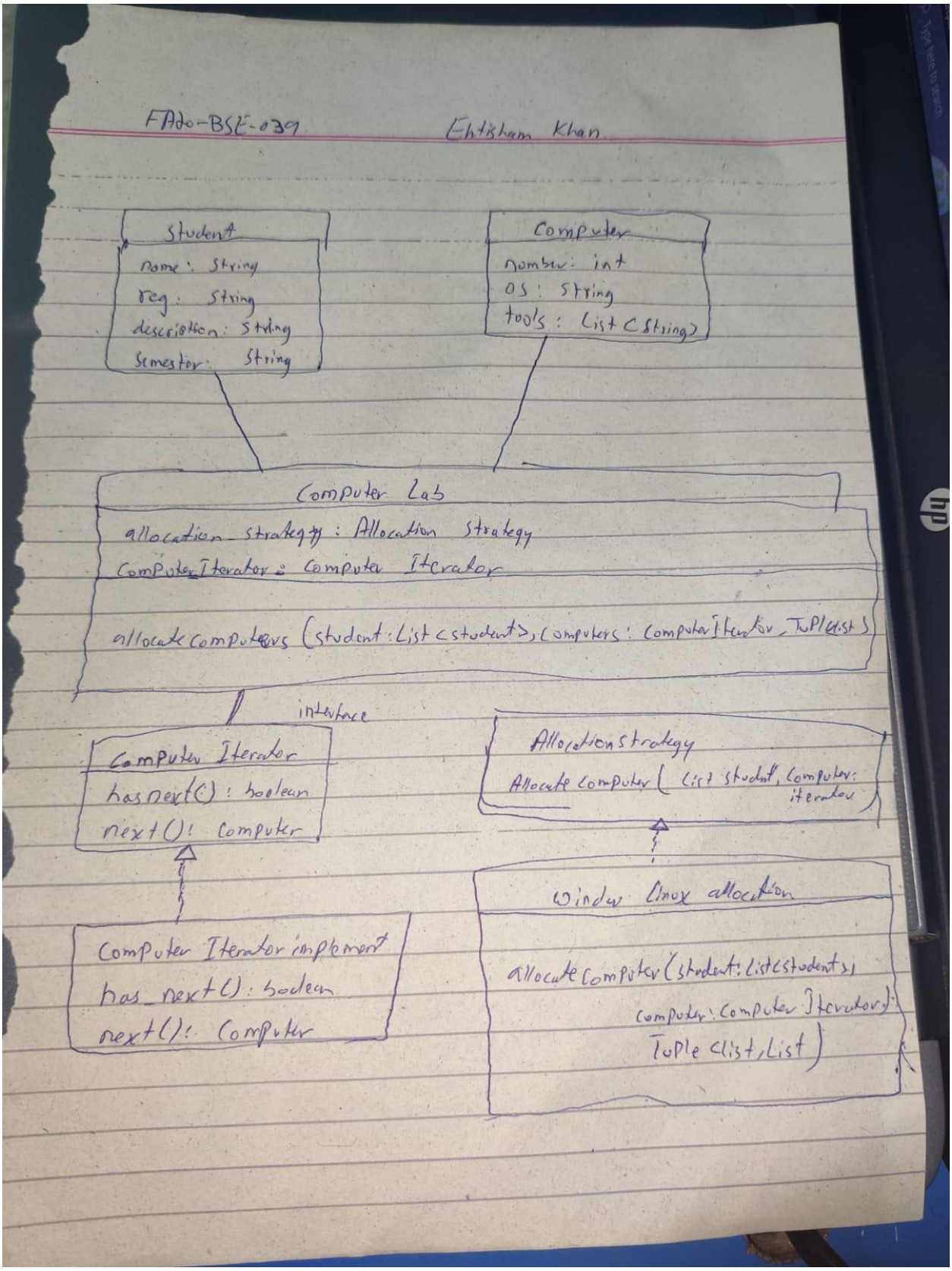
Mid Term Lab:

Name: Ehtisham khan

Reg: FA20-BSE-039

Subject: Design Pattern


FA20-BSE-039
Ehtisham Khan
Student
name: String
reg: String
description: String
Semester: String
Computer
nombr: int
OS: String
tools: List<String>
Computer Lab
allocation_strategy: Allocation Strategy
ComputerIterator: Computer Iterator
allocateComputers(student: List<student>, computers: ComputerIterator, TuPle<List>)
interface
Computer Iterator
hasnext(): boolean
next(): Computer
Allocationstrategy
Allocate Computer(List student, Computer: iterator)
Computer Iterator implement
has_next(): boolean
next(): Computer
Window Linux allocation
allocate Computer(student: List<student>, computer: Computer Iterator): Tuple<List, List>

```
class Student {
    String name;
    String reg;
    String description;
    int semester;

    public Student(String name, String reg, String description, int semester) {
        this.name = name;
        this.reg = reg;
        this.description = description;
        this.semester = semester;
    }

    int getSemester() {
        return semester;
    }
}

class Tuple<X, Y> {
    public final X first;
    public final Y second;

    public Tuple(X first, Y second) {
        this.first = first;
        this.second = second;
```

```
    }
}


class LinuxAllocation implements AllocationStrategy {
    @Override
    public Tuple<List<Tuple<Student, Computer>>, List<Tuple<Student, Computer>>> allocateComputers(List<Student> students, ComputerIterator computers) {
        List<Tuple<Student, Computer>> linuxList = new ArrayList<>();
        List<Tuple<Student, Computer>> windowsList = new ArrayList<>();


        while (students.size() != 0 && computers.hasNext()) {
            Student student = students.remove(0);
            Computer computer = computers.next();


            if (Integer.parseInt(student.reg) % 2 != 0 && computer.os.equalsIgnoreCase("Linux")) {
                linuxList.add(new Tuple<>(student, computer));
            } else if (Integer.parseInt(student.reg) % 2 == 0 && computer.os.equalsIgnoreCase("Windows")) {
                windowsList.add(new Tuple<>(student, computer));
            } else {


            }
        }
```

```
        List<Tuple<Student, Computer>> nonAllocatedStudents = new
ArrayList<>();



        for (Student student : students) {
            nonAllocatedStudents.add(new Tuple<>(student, null));
        }

class ComputerLab {
    private AllocationStrategy allocationStrategy;
    private ComputerIterator computerIterator;


    public ComputerLab(AllocationStrategy allocationStrategy, ComputerIterator
computerIterator) {
        this.allocationStrategy = allocationStrategy;
        this.computerIterator = computerIterator;
    }


    public Tuple<List<Tuple<Student, Computer>>, List<Tuple<Student,
Computer>>> allocateComputers(List<Student> students) {
        return allocationStrategy.allocateComputers(students, computerIterator);
    }
}

class ComputerIteratorImpl implements ComputerIterator {
    private List<Computer> computers;
    private int index;
```

```
    public ComputerIteratorImpl(List<Computer> computers) {
        this.computers = computers;
        this.index = 0;
    }

    public boolean hasNext() {
        return index < computers.size();
    }

    public Computer next() {
        if (this.hasNext()) {
            return computers.get(index++);
        } else {
            throw new NoSuchElementException("No more computers available in the iterator.");
        }
    }
}

interface ComputerIterator {
    boolean hasNext();
    Computer next();
}

class Computer {
    int number;
    String os;
```

```
    List<String> tools;

    public Computer(int number, String os, List<String> tools) {
        this.number = number;
        this.os = os;
        this.tools = tools;
    }

    String getOs() {
        return os;
    }
}

interface AllocationStrategy {
    Tuple<List<Tuple<Student, Computer>>, List<Tuple<Student, Computer>>>
allocateComputers(List<Student> students, ComputerIterator computers);
}
```