Name Muhammad Hammad

Reg Fa20-bse-031

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

// Iterator interface

interface ComputerAllocationIterator {

boolean hasNext();

Object next();

}

// Concrete iterator for student list

class StudentIterator implements ComputerAllocationIterator {

private List<Student> students;

private int index;

public StudentIterator(List<Student> students) {

this.students = students;

this.index = 0;

}

@Override

public boolean hasNext() {

return index < students.size();

}

@Override

public Object next() {

Student student = students.get(index);

index++;

return student;

}

}

// Aggregate interface

interface ComputerAllocationAggregate {

ComputerAllocationIterator createIterator();

}

// Concrete aggregate for student list

class StudentAggregate implements ComputerAllocationAggregate {

private List<Student> students;

public StudentAggregate(List<Student> students) {

this.students = students;

}

@Override

public ComputerAllocationIterator createIterator() {

return new StudentIterator(students);

}

}

// Student class with integer registration number

class Student {

String name;

int reg;

String semester;

Computer computer;

public Student(String name, int reg, String semester) {

this.name = name;

this.reg = reg;

this.semester = semester;

}

}

// Computer class

class Computer {

String type;

String id;

public Computer(String type, String id) {

this.type = type;

this.id = id;

}

}

// Strategy interface

interface ComputerAllocationStrategy {

void allocateComputer(Student student, Map<String, List<Computer>> computers);

}

// Concrete strategy for Linux allocation

class LinuxAllocationStrategy implements ComputerAllocationStrategy {

@Override

public void allocateComputer(Student student, Map<String, List<Computer>> computers) {

List<Computer> linuxComputers = computers.get("Linux");

if (!linuxComputers.isEmpty() && student.reg % 2 != 0) {

student.computer = linuxComputers.remove(0);

}

}

}

// Concrete strategy for Windows allocation

class WindowsAllocationStrategy implements ComputerAllocationStrategy {

@Override

public void allocateComputer(Student student, Map<String, List<Computer>> computers) {

List<Computer> windowsComputers = computers.get("Windows");

if (!windowsComputers.isEmpty() && student.reg % 2 == 0) {

student.computer = windowsComputers.remove(0);

}

}

}

// Context class

class ComputerAllocator {

private ComputerAllocationStrategy strategy;

public ComputerAllocator(ComputerAllocationStrategy strategy) {

this.strategy = strategy;

}

public void allocateComputers(List<Student> students, Map<String, List<Computer>> computers) {

ComputerAllocationIterator iterator = new StudentAggregate(students).createIterator();

while (iterator.hasNext()) {

Student student = (Student) iterator.next();

strategy.allocateComputer(student, computers);

}

}

public void printAllocationList(List<Student> students) {

for (Student student : students) {

if (student.computer != null) {

System.out.println(student.name + " - " + student.computer.type + " computer " + student.computer.id);

} else {

System.out.println(student.name + " - No computer allocated");

}

}

}

}

// Client code

public class Fa23LabMidTErm {

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student("Student1", 10, "Semester1"));

students.add(new Student("Student2", 7, "Semester1"));

students.add(new Student("Student1", 11, "Semester1"));

students.add(new Student("Student2", 18, "Semester1"));

students.add(new Student("Student1", 31, "Semester1"));

students.add(new Student("Student2", 40, "Semester1"));

Map<String, List<Computer>> computers = new HashMap<>();

computers.put("Linux", new ArrayList<>());

computers.put("Windows", new ArrayList<>());

computers.get("Linux").add(new Computer("Linux", "L1"));

computers.get("Linux").add(new Computer("Linux", "L2"));

computers.get("Windows").add(new Computer("Windows", "W1"));

computers.get("Windows").add(new Computer("Windows", "W2"));

computers.get("Linux").add(new Computer("Linux", "L3"));

computers.get("Windows").add(new Computer("Windows", "W3"));

// Using LinuxAllocationStrategy

System.out.println("Using LinuxAllocationStrategy");

ComputerAllocationStrategy linuxStrategy = new LinuxAllocationStrategy();

ComputerAllocator linuxAllocator = new ComputerAllocator(linuxStrategy);

linuxAllocator.allocateComputers(students, computers);

linuxAllocator.printAllocationList(students);

// Using WindowsAllocationStrategy

System.out.println("Using WindowsAllocationStrategy");

ComputerAllocationStrategy windowsStrategy = new WindowsAllocationStrategy();

ComputerAllocator windowsAllocator = new ComputerAllocator(windowsStrategy);

windowsAllocator.allocateComputers(students, computers);

windowsAllocator.printAllocationList(students);

}

}