import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

// 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

class Student {

String name;

String reg;

String semester;

Computer computer;

public Student(String name, String 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, List<Computer> computers);

}

// Concrete strategy for Linux allocation

class LinuxAllocationStrategy implements ComputerAllocationStrategy {

@Override

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

List<Computer> linuxComputers = new ArrayList<>();

for (Computer comp : computers) {

if ("Linux".equals(comp.type)) {

linuxComputers.add(comp);

}

}

if (Integer.parseInt(student.reg.substring(1)) % 2 != 0) {

student.computer = linuxComputers.remove(0);

}

}

}

// Concrete strategy for Windows allocation

class WindowsAllocationStrategy implements ComputerAllocationStrategy {

@Override

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

List<Computer> windowsComputers = new ArrayList<>();

for (Computer comp : computers) {

if ("Windows".equals(comp.type)) {

windowsComputers.add(comp);

}

}

if (Integer.parseInt(student.reg.substring(1)) % 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, 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) {

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

}

}

}

// Client code

public class Main {

public static void main(String[] args) {

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

students.add(new Student("Student1", "A1234", "Semester1"));

students.add(new Student("Student2", "B5678", "Semester1"));

List<Computer> computers = new ArrayList<>();

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

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

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

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

// Using LinuxAllocationStrategy

ComputerAllocationStrategy linuxStrategy = new LinuxAllocationStrategy();

ComputerAllocator linuxAllocator = new ComputerAllocator(linuxStrategy);

linuxAllocator.allocateComputers(students, computers);

linuxAllocator.printAllocationList(students);

// Using WindowsAllocationStrategy

ComputerAllocationStrategy windowsStrategy = new WindowsAllocationStrategy();

ComputerAllocator windowsAllocator = new ComputerAllocator(windowsStrategy);

windowsAllocator.allocateComputers(students, computers);

windowsAllocator.printAllocationList(students);

}

}

ForeFront  
import java.util.\*;

// Student class

class Student {

String name;

int regNumber;

int semester;

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

this.name = name;

this.regNumber = regNumber;

this.semester = semester;

}

}

// Computer class

class Computer {

String type;

public Computer(String type) {

this.type = type;

}

}

// Iterator interface

interface ComputerIterator {

boolean hasNext();

Computer next();

}

// Concrete iterator for Linux computers

class LinuxComputerIterator implements ComputerIterator {

private List<Computer> computers;

private int position;

public LinuxComputerIterator(List<Computer> computers) {

this.computers = computers;

this.position = 0;

}

public boolean hasNext() {

return position < computers.size();

}

public Computer next() {

return computers.get(position++);

}

}

// Concrete iterator for Windows computers

class WindowsComputerIterator implements ComputerIterator {

private List<Computer> computers;

private int position;

public WindowsComputerIterator(List<Computer> computers) {

this.computers = computers;

this.position = 0;

}

public boolean hasNext() {

return position < computers.size();

}

public Computer next() {

return computers.get(position++);

}

}

// Client code

public class Main {

public static void main(String[] args) {

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

students.add(new Student("Alice", 101, 3));

students.add(new Student("Bob", 102, 4));

students.add(new Student("Charlie", 103, 2));

students.add(new Student("David", 104, 5));

List<Computer> linuxComputers = new ArrayList<>();

linuxComputers.add(new Computer("Linux1"));

linuxComputers.add(new Computer("Linux2"));

linuxComputers.add(new Computer("Linux3"));

List<Computer> windowsComputers = new ArrayList<>();

windowsComputers.add(new Computer("Windows1"));

windowsComputers.add(new Computer("Windows2"));

windowsComputers.add(new Computer("Windows3"));

ComputerIterator linuxIterator = new LinuxComputerIterator(linuxComputers);

ComputerIterator windowsIterator = new WindowsComputerIterator(windowsComputers);

for (Student student : students) {

if (student.regNumber % 2 != 0 && linuxIterator.hasNext()) {

System.out.println(student.name + " gets " + linuxIterator.next().type);

} else if (student.regNumber % 2 == 0 && windowsIterator.hasNext()) {

System.out.println(student.name + " gets " + windowsIterator.next().type);

} else {

System.out.println("No available computers for " + student.name);

}

}

}

}