

Министерство образования Республики Беларусь
Учреждение образования
«Брестский Государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
по дисциплине “Современные платформы программирования”

Выполнил:

Студент 3 курса

Группы ПО-8

Бондаренко К.А.

Проверил:

Крощенко А.А.

Цель работы: приобрести практические навыки в области объектно-ориентированного проектирования.

Ход работы

Вариант 1

Задание 1. Реализовать указанный класс, включив в него вспомогательный внутренний класс или классы. Реализовать 2-3 метода (на выбор). Продемонстрировать использование реализованных классов.

Создать класс Notepad (записная книжка) с внутренним классом или классами, с помощью объектов которого могут храниться несколько записей на одну дату.

Код программы:

Класс записной книжки:

```
import java.util.*;
import java.lang.StringBuilder;

public class Notepad {
    Map<Date, Note> entries;

    public Notepad() {
        entries = new HashMap<>();
    }

    public void addEntry(Date date, String text) {
        if (!entries.containsKey(date)) {
            entries.put(date, new Note());
        }

        Note entry = entries.get(date);
        entry.addNote(text);
    }

    public ArrayList<String> getEntryByDate(Date date) {
        if (!entries.containsKey(date)) {
            return new ArrayList<String>();
        }
        return entries.get(date).getNote();
    }

    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        entries.forEach((date, entry) -> {
            builder.append(date);
            builder.append("\n");
            ArrayList<String> notes = entry.getNote();
```

```

        for (String note : notes) {
            builder.append(note);
            builder.append("\n");
        }
    });

    return builder.toString();
}

private class Note {
    private ArrayList<String> entry;

    public Note() {
        entry = new ArrayList<String>();
    }

    public void addNote(String text) {
        entry.add(text);
    }

    public ArrayList<String> getNote() {
        return entry;
    }
}
}

```

Класс с методом main:

```

import java.util.Date;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Notepad notepad = new Notepad();
        notepad.addEntry(new Date(), "Запись 1 Дата 1");
        notepad.addEntry(new Date(), "Запись 2 Дата 1");

        Thread.sleep(2000);

        notepad.addEntry(new Date(), "Запись 1 Дата 2");

        System.out.println(notepad);
    }
}

```

Результаты работы программы:

```

Sun Feb 25 15:03:50 MSK 2024
Запись 1 Дата 2
Sun Feb 25 15:03:48 MSK 2024
Запись 1 Дата 1
Sun Feb 25 15:03:48 MSK 2024
Запись 2 Дата 1

```

Задание 2. Реализовать агрегирование. При создании класса агрегируемый класс объявляется как атрибут (локальная переменная, параметр метода). Включить в каждый класс 2-3 метода на выбор. Продемонстрировать использование разработанных классов.

Создать класс Строка, используя классы Слово, Символ.

Код программы:

Класс символа:

```
public class Symbol {
    private char symbol;

    public Symbol(char symbol) {
        this.symbol = symbol;
    }

    public char getSymbol() {
        return symbol;
    }

    public void setSymbol(char symbol) {
        this.symbol = symbol;
    }

    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false;
        if (!super.equals(object)) return false;
        Symbol symbol1 = (Symbol) object;
        return symbol == symbol1.symbol;
    }

    public int hashCode() {
        return java.util.Objects.hash(super.hashCode(), symbol);
    }

    public char toChar() {
        return symbol;
    }
}
```

Класс слова:

```
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Objects;

public class Word {
    private ArrayList<Symbol> symbols;
```

```

public Word(Symbol[] symbols) {
    this.symbols = new ArrayList<>();
    for (Symbol symbol : symbols) {
        if (symbol.getSymbol() != ' ') {
            this.symbols.add(symbol);
        }
    }
}

public Symbol[] getSymbols() {
    return (Symbol[]) symbols.toArray(Symbol[]::new);
}

public void setSymbols(Symbol[] symbols) {
    this.symbols.clear();
    for (Symbol symbol : symbols) {
        if (symbol.getSymbol() == ' ') {
            continue;
        }
        if (this.symbols != null) {
            this.symbols.add(symbol);
        }
    }
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Word word = (Word) o;
    return Objects.equals(symbols, word.symbols);
}

@Override
public int hashCode() {
    return Objects.hash(symbols);
}

@Override
public String toString() {
    StringBuilder builder = new StringBuilder();
    for (Symbol symbol : symbols) {
        builder.append(symbol.toChar());
    }
    return builder.toString();
}

public Symbol symbolAt(int index) throws Exception {
    if (index < 0 || index > symbols.size()) {
        throw new Exception("Неверный индекс!");
    }
    return symbols.get(index);
}
}

```

Класс строки:

```
import java.util.Arrays;

public class MyString {
    Word[] words;

    public MyString(Word[] words) {
        this.words = words;
    }

    public Word[] getWords() {
        return words;
    }

    public void setWords(Word[] words) {
        this.words = words;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        MyString myString = (MyString) o;
        return Arrays.equals(words, myString.words);
    }

    @Override
    public int hashCode() {
        return Arrays.hashCode(words);
    }

    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        for (Word word : words) {
            builder.append(word.toString()).append(' ');
        }
        return builder.toString();
    }

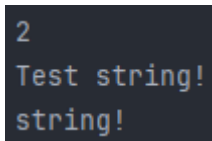
    public int length() {
        return words.length;
    }

    public Word wordAt(int index) throws Exception {
        if (index < 0 || index > words.length) {
            throw new Exception("Неверный индекс!");
        }
        return words[index];
    }
}
```

Класс с методом main:

```
public class Main {  
    public static void main(String[] args) throws Exception {  
        Word wordTest = new Word(new Symbol[] {  
            new Symbol('T'),  
            new Symbol('e'),  
            new Symbol('s'),  
            new Symbol('t')  
        });  
        Word wordString = new Word(new Symbol[] {  
            new Symbol('s'),  
            new Symbol('t'),  
            new Symbol('r'),  
            new Symbol('i'),  
            new Symbol('n'),  
            new Symbol(' '),  
            new Symbol('g'),  
            new Symbol('!')  
        });  
  
        MyString str = new MyString(new Word[] {  
            wordTest,  
            wordString  
        });  
  
        System.out.println(str.length());  
        System.out.println(str);  
        System.out.println(str.wordAt(1));  
    }  
}
```

Результаты работы программы:



```
2  
Test string!  
string!
```

Задание 3. Построить модель программной системы с применением отношений (обобщения, агрегации, ассоциации, реализации) между классами. Задать атрибуты и методы классов. Реализовать (если необходимо) дополнительные классы. Продемонстрировать работу разработанной системы.

Система Факультет. Преподаватель объявляет запись на Курс. Студент записывается на Курс, обучается и по окончании Преподаватель выставляет Оценку, которая сохраняется в Архиве. Студентов, Преподавателей и Курсов при обучении может быть несколько.

Код программы:

Класс факультета:

```
import java.util.ArrayList;

public class Faculty {
    private String name;
    private ArrayList<Course> courses;

    public Faculty(String name) {
        this.name = name;
        courses = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public void addCourse(Course course) {
        courses.add(course);
    }

    public void removeCourse(Course course) {
        courses.remove(course);
    }

    public void setCourses(ArrayList<Course> courses) {
        this.courses = courses;
    }

    public ArrayList<Course> getCourses() {
        return courses;
    }
}
```

Класс преподавателя:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

public class Teacher {
    private String name;
    private ArrayList<Course> courses;

    public Teacher(String name) {
        this.name = name;
        courses = new ArrayList<>();
    }

    public String getName() {
        return name;
    }
}
```



```

    }

    public void announceRegistryForCourse(Course course) {
        courses.add(course);
        course.setTeacher(this);
        System.out.println("Прием на курс " + course.getName());
    }

    public void stopTeachingCourse(Course course) {
        System.out.println("Завершение курса " + course.getName());
        Scanner scanner = new Scanner(System.in);

        ArrayList<Student> students = course.getStudents();
        Iterator<Student> iterator = students.iterator();
        while (iterator.hasNext()) {
            Student student = iterator.next();
            System.out.println("Поставьте оценку студенту " + student.getName() + " по предмету " +
course.getName());
            Grade grade = new Grade(scanner.nextInt());
            Archive.addRecord(student, course, grade);
            iterator.remove();
            student.stopStudying(course);
        }
        courses.remove(course);
    }
}

```

Класс курса:

```

import java.util.ArrayList;

public class Course {
    private String name;
    private Teacher teacher;
    private ArrayList<Student> students;

    public Course(String name) {
        this.name = name;
        students = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public void setTeacher(Teacher teacher) {
        this.teacher = teacher;
    }

    public ArrayList<Student> getStudents() {
        return students;
    }

    public void enrollStudent(Student student) {

```

```

        students.add(student);
        student.startStudying(this);
    }

    public void excludeStudent(Student student) {
        students.remove(student);
        student.stopStudying(this);
    }
}

```

Класс студента:

```

public class Student {
    private String name;

    public Student(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void startStudying(Course course) {
        System.out.println("Студент " + name + " начал изучать курс " + course.getName());
    }

    public void stopStudying(Course course) {
        System.out.println("Студент " + name + " закончил изучать курс " + course.getName());
    }
}

```

Класс оценки:

```

public class Grade {
    private int value;

    public Grade(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}

```

Класс архива:

```

import java.util.HashMap;

public class Archive {
    private static HashMap<Student, HashMap<Course, Grade>> records = new HashMap<>();

    public static void addRecord(Student student, Course course, Grade grade) {

```

```

        records.computeIfAbsent(student, k -> new HashMap<>()).put(course, grade);
    }

    public HashMap<Course, Grade> getStudentRecords(Student student) {
        return records.get(student);
    }

    public static void printAllRecords() {
        StringBuilder builder = new StringBuilder();
        for (HashMap.Entry<Student, HashMap<Course, Grade>> entry : records.entrySet()) {
            Student student = entry.getKey();
            HashMap<Course, Grade> studentRecords = entry.getValue();
            builder.append("Студент: ").append(student.getName()).append("\n");
            for (HashMap.Entry<Course, Grade> recordEntry : studentRecords.entrySet()) {
                Course course = recordEntry.getKey();
                Grade grade = recordEntry.getValue();
                builder.append("\tКурс: ").append(course.getName()).append(", Оценка: ");
            }
            builder.append(grade.getValue()).append("\n");
        }
        System.out.println(builder.toString());
    }
}

```

Класс с методом main:

```

public class Main {
    public static void main(String[] args) {
        Faculty faculty = new Faculty("Факультет электронно-информационных систем");

        Course course = new Course("Основы алгоритмизации и программирования");
        faculty.addCourse(course);

        Teacher teacher = new Teacher("Щербаков Марк Егорович");
        teacher.announceRegistryForCourse(course);

        Student student1 = new Student("Степанова Алиса Константиновна");
        Student student2 = new Student("Жаров Артём Ильич");

        course.enrollStudent(student1);
        course.enrollStudent(student2);

        teacher.stopTeachingCourse(course);

        Archive.printAllRecords();
    }
}

```

Результаты работы программы:

```
Прием на курс Основы алгоритмизации и программирования
Студент Степанова Алиса Константиновна начал изучать курс Основы алгоритмизации и программирования
Студент Жаров Артём Ильич начал изучать курс Основы алгоритмизации и программирования
Завершение курса Основы алгоритмизации и программирования
Поставьте оценку студенту Степанова Алиса Константиновна по предмету Основы алгоритмизации и программирования
4
Студент Степанова Алиса Константиновна закончил изучать курс Основы алгоритмизации и программирования
Поставьте оценку студенту Жаров Артём Ильич по предмету Основы алгоритмизации и программирования
6
Студент Жаров Артём Ильич закончил изучать курс Основы алгоритмизации и программирования
Стундент: Жаров Артём Ильич
    Курс: Основы алгоритмизации и программирования, Оценка: 6
Стундент: Степанова Алиса Константиновна
    Курс: Основы алгоритмизации и программирования, Оценка: 4
```

Вывод: приобрели практические навыки в области объектно-ориентированного проектирования.