МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

**ОТЧЕТ**

по лабораторной работе №1

по дисциплине

Шаблоны проектирования программного обеспечения

ВЫПОЛНИЛИ:

Студенты группы \_\_\_22-ИВТ-2\_\_\_\_\_\_\_\_

(шифр группы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Киселев А. Р.

(подпись) (фамилия, и.,о.)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Назаров А. И.

(подпись) (фамилия, и.,о.)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПРОВЕРИЛ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жевнерчук Д. В.

(подпись) (фамилия, и.,о.)

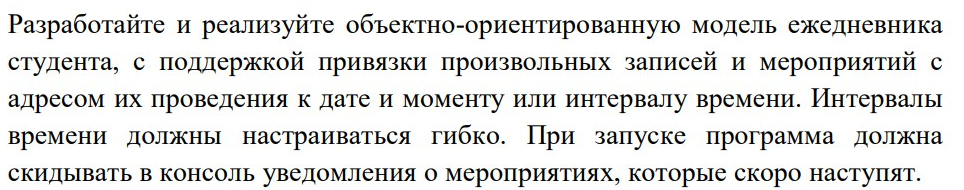
Работа защищена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Нижний Новгород 2024

**Задание**

**Вариант 17**



**Проектное решение**

**Обоснование выбора паттернов**

Основным объектом программы является объект “Задача”. С помощью него можно создать собственные типы, т.к. он является расширяемым.

Управление программой осуществляется с помощью команд.

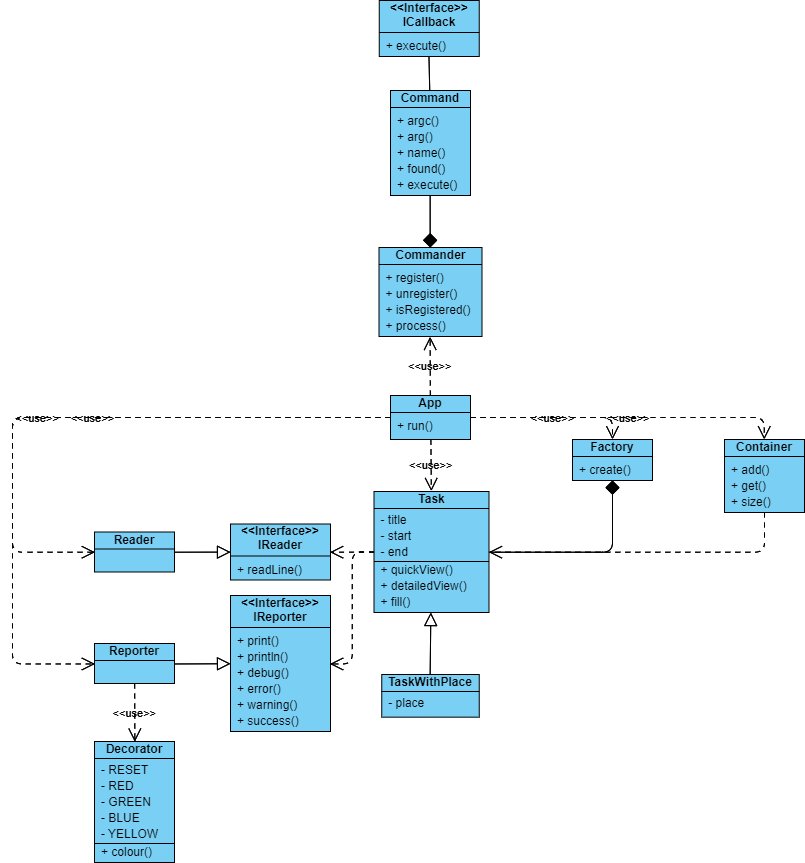
В качестве хранилища используется контейнер (лист, который автоматически сортирует объекты по дате)

Управление вводом/вывода осуществляется с помощью утилиты IOCollection, которая совмещает адаптеры и провайдеры Читатель и Писатель. (В данном приложении они реализованы как провайдер индикации по уровню (ошибка/предупреждение/информация и тд) и адаптер ввода/вывода для системного потока).

Для окраски сообщений на каждом из уровней используется декоратор.

Дополнительно, чтобы упростить создание объектов Задач, используется Фабрика.

Диаграмма классов:



**Приложение 1**

**Программный код**

src/App.java:  
import commander.Command;  
import commander.Commander;  
  
import organizer.publ;  
import organizer.Task;  
import organizer.util.IOCollection;  
import util.Reader;  
import util.Reporter;  
  
/\*\*  
 \* Контроллер приложения  
 \*/  
public class App {  
 protected IOCollection io = new IOCollection(new Reader(System.in), new Reporter(System.out));  
 protected Factory factory = new Factory(io);  
  
 protected Container container = new Container();  
 protected Commander commander = new Commander();  
  
 public App() {  
 registerCommands();  
 }  
  
 /\*\*  
 \* Функция делает эмуляцию терминала  
 \*/  
 public void run() {  
 while (true) {  
 io.out().print("organizer> ");  
  
 String line = io.in().readLine();  
 if (line.equals("exit")) { // Команда exit всегда имеет выше приоритет, чем остальные  
 break;  
 }  
  
 // Пытаемся обработать введеную строку  
 Command command = commander.process(line);  
  
 // Если не удалось обработать или команда не найдена  
 if (command == null || !command.found()) {  
 io.out().error("Invalid command.");  
 continue;  
 }  
  
 // Выполним команду  
 command.execute();  
  
 // Для того чтобы строки терминала не слились после выполнения, добавим еще одну строку  
 io.out().println("");  
 }  
  
 io.out().println("Bye.");  
 }  
  
 /\*\*  
 \* Регистрация команд  
 \*/  
 protected void registerCommands() {  
 // help  
 commander.register("help", \_ -> {  
 io.out().println("""  
 Available commands:  
 help - View help  
 exit - Exit from application  
  
 create [type] - Create task  
 remove <pos> - Remove task  
 view [id] - View task(s)""");  
 });  
  
 // create [type] (Создание)  
 commander.register("create", command -> {  
 String type = command.arg(1);  
  
 Task task = factory.create(type);  
 if (task == null) {  
 io.out().error("Unable to create task with type: " + type);  
 return;  
 }  
  
 // Запишем в контейнер  
 container.add(task);  
  
 io.out().println("Task was successfully created.");  
 });  
  
 // remove <id> (Удаление)  
 commander.register("remove", command -> {  
 // Отсечем ввод, где только 1 аргумент (т.к. команда требует как минимум 2 аргумента)  
 if (command.argc() < 2) {  
 io.out().error("Usage: remove <id>");  
 return;  
 }  
  
 // Пытаемся извлечь integer из второго аргумента  
 int id;  
  
 try {  
 id = Integer.parseInt(command.arg(1));  
 } catch (Exception e) {  
 io.out().error("Argument <id> must be an integer value.");  
 return;  
 }  
  
 // Проверяем, существует ли задание с таким id  
 Task task = container.get(id);  
 if (task == null) {  
 io.out().error("Task with id=" + id + " not found.");  
 return;  
 }  
  
 // Удаляем  
 container.remove(task);  
  
 io.out().success("Task was removed.");  
 });  
  
 // view [id] (Просмотр)  
 commander.register("view", command -> {  
 // Завершим преждевременно, если заданий не существует.  
 if (container.isEmpty()) {  
 io.out().warning("Please create tasks first.");  
 return;  
 }  
  
 // Если команда имеет 2 и более аргумента  
 if (command.argc() > 1) {  
 // Пытаемся извлечь id  
 int id;  
  
 try {  
 id = Integer.parseInt(command.arg(1));  
 } catch (Exception e) {  
 io.out().error("Argument [id] must be an integer value.");  
 return;  
 }  
  
 // Проверим, существует ли такое задание  
 Task task = container.get(id);  
 if (task == null) {  
 io.out().error("Task with id=" + id + " not found.");  
 return;  
 }  
  
 // Просмотрим детально  
 io.out().print(container.get(id).detailedView());  
 return;  
 }  
  
 // Просмотрим кратко все  
 for (int i = 0; i < container.size(); ++i) {  
 io.out().println(i + ": " + container.get(i).quickView());  
 }  
 });  
 }  
}  
  
  
src/Factory.java:  
import custom.tasks.TaskWithPlace;  
import organizer.Task;  
  
import organizer.util.IOCollection;  
  
/\*\*  
 \* Фабрика представляет собой таблицу, где вместо имени класса используется строковый идентификатор.  
 \*/  
public class Factory {  
 protected IOCollection io;  
  
 public Factory(IOCollection io) {  
 this.io = io;  
 }  
  
 public Task create(String type) {  
 Task task;  
  
 // Если тип пустой, то установим его на базовый класс task  
 if (type.isEmpty()) {  
 type = "default";  
 }  
  
 // Выберем по типу  
 switch (type) {  
 case "default":  
 task = new Task();  
 break;  
 case "place":  
 task = new TaskWithPlace();  
 break;  
 default:  
 return null;  
 }  
  
 // Заполним  
 task.fill(io);  
  
 return task;  
 }  
}  
  
  
src/Main.java:  
/\*\*  
 \* Точка входа приложения  
 \*/  
public class Main {  
 public static void main(String[] args) {  
 App app = new App();  
 app.run();  
 }  
}  
  
src/commander/Command.java:  
package commander;  
  
import commander.iface.ICallback;  
  
/\*\*  
 \* Оболочка над аргументами для безопасного использования  
 \*/  
public class Command {  
 /\*\*  
 \* Аргумента  
 \*/  
 protected String[] args;  
  
 /\*\*  
 \* Обратный вызов  
 \*/  
 protected ICallback cb;  
  
 public Command(String[] args) {  
 this.args = args;  
 }  
  
 public Command(String[] args, ICallback cb) {  
 this.args = args;  
 this.cb = cb;  
 }  
  
 /\*\*  
 \* Кол-во аргументов  
 \*/  
 public int argc() {  
 return args.length;  
 }  
  
 /\*\*  
 \* Получить аргумент по индексу  
 \*/  
 public String arg(int pos) {  
 return argc() > pos ? args[pos] : "";  
 }  
  
 /\*\*  
 \* Имя команды (нулевой аргумент)  
 \*/  
 public String name() {  
 return arg(0);  
 }  
  
 /\*\*  
 \* Найдена ли команда  
 \*/  
 public boolean found() {  
 return cb != null;  
 }  
  
 /\*\*  
 \* Выполняет обратный вызов  
 \*/  
 public void execute() {  
 if (found()) {  
 cb.execute(this);  
 }  
 }  
}  
  
  
src/commander/Commander.java:  
package commander;  
  
import commander.iface.ICallback;  
import commander.util.ArgBuilder;  
  
import java.util.HashMap;  
  
/\*\*  
 \* Контроллер команд  
 \*/  
public class Commander {  
 protected HashMap<String, ICallback> commands = new HashMap<>();  
  
 /\*\*  
 \* Регистрация команд и тд  
 \*/  
  
 public void register(String name, ICallback cb) {  
 commands.put(name, cb);  
 }  
  
 public void unregister(String name) {  
 commands.remove(name);  
 }  
  
 public boolean isRegistered(String name) {  
 return commands.containsKey(name);  
 }  
  
 /\*\*  
 \* Парсинг команды  
 \* Извлекаем из линии все аргументы  
 \*/  
 public Command process(String line) {  
 if (line.isEmpty()) {  
 return null;  
 }  
  
 boolean in\_scope = false;  
 ArgBuilder builder = new ArgBuilder();  
  
 for (char c : line.toCharArray()) {  
 if (c == '"') {  
 in\_scope = !in\_scope;  
 continue;  
 }  
  
 if (c == ' ' && !in\_scope) {  
 builder.push();  
 continue;  
 }  
  
 builder.add(c);  
 }  
  
 builder.push();  
  
 String[] args = builder.build();  
  
 if (args.length == 0) {  
 return null;  
 }  
  
 if (!commands.containsKey(args[0])) {  
 return new Command(args);  
 }  
  
 return new Command(args, commands.get(args[0]));  
 }  
}  
  
  
src/organizer/Container.java:  
package organizer;  
  
import java.time.LocalDateTime;  
import java.util.AbstractList;  
import java.util.ArrayList;  
  
/\*\*  
 \* Контейнер для задач.  
 \* Представляет собой ArrayList, но при каждом добавлении элемента происходит сортировка по начальной дате задачи.  
 \*/  
public class Container extends AbstractList<Task> {  
 protected ArrayList<Task> container = new ArrayList<>();  
  
 @Override  
 public void add(int index, Task task) {  
 container.add(task);  
 container.sort((o1, o2) -> {  
 LocalDateTime t1 = o1.getStart();  
 LocalDateTime t2 = o2.getStart();  
  
 if (t1.isEqual(t2)) {  
 return 0;  
 }  
  
 return t1.isAfter(t2) ? 1 : -1;  
 });  
 }  
  
 @Override  
 public Task get(int index) {  
 if (0 <= index && index < size()) {  
 return container.get(index);  
 }  
 return null;  
 }  
  
 @Override  
 public int size() {  
 return container.size();  
 }

@Override  
 public Task remove(int index) {  
 return container.remove(index);  
 }  
}  
  
  
src/organizer/Task.java:  
package organizer;  
  
import organizer.iface.ISetter;  
import organizer.util.IOCollection;  
import organizer.util.TimeFormatter;  
  
import java.time.LocalDateTime;  
  
/\*\*  
 \* Базовый класс задания  
 \*/  
public class Task {  
 // Имеет 3 поля: заголовок, дата начала и дата окончания  
 protected String title;  
  
 protected LocalDateTime start = LocalDateTime.MIN;  
 protected LocalDateTime end = LocalDateTime.MIN;  
  
 /\*\*  
 \* Get/Set  
 \*/  
 public String getTitle() {  
 return title;  
 }  
  
 public boolean setTitle(String title) {  
 if (title.isEmpty()) {  
 return false;  
 }  
  
 this.title = title;  
 return true;  
 }  
  
 public LocalDateTime getStart() {  
 return start;  
 }  
  
 public boolean setStart(LocalDateTime start) {  
 this.start = start;  
  
 if (start.isAfter(end)) {  
 end = start;  
 }  
  
 return true;  
 }  
  
 public LocalDateTime getEnd() {  
 return end;  
 }  
  
 public boolean setEnd(LocalDateTime end) {  
 if (!end.isBefore(start)) {  
 this.end = end;  
 return true;  
 }  
 return false;  
 }  
  
 /\*\*  
 \* Быстрый просмотр  
 \*/  
 public String quickView() {  
 return getTitle() + " (" + TimeFormatter.retrieve(getStart()) + " - " + TimeFormatter.retrieve(getEnd()) + ")";  
 }  
  
 /\*\*  
 \* Детальный просмотр  
 \*/  
 public String detailedView() {  
 return String.format("""  
 Title: %s  
 Time: (%s - %s)""", getTitle(), TimeFormatter.retrieve(getStart()), TimeFormatter.retrieve(getEnd()));  
 }  
  
 /\*\*  
 \* Заполняет данные используя адаптер  
 \*/  
 public void fill(IOCollection io) {  
 // Ввод заголовка  
 enter(io, "Enter title", "Unable to set title.",  
 this::setTitle);  
  
 // Ввод даты начала  
 enter(io, "Enter start (" + TimeFormatter.format + ")", "Please enter valid datetime.", line -> {  
 LocalDateTime time;  
  
 try {  
 time = TimeFormatter.retrieve(line);  
 } catch (Exception e) {  
 return false;  
 }  
  
 return setStart(time);  
 });  
  
 // Ввод даты конца  
 enter(io, "Enter end (" + TimeFormatter.format + ")", "Please enter valid datetime.", line -> {  
 LocalDateTime time;  
  
 try {  
 time = TimeFormatter.retrieve(line);  
 } catch (Exception e) {  
 return false;  
 }  
  
 return setEnd(time);  
 });  
 }  
  
 /\*\*  
 \* Данная функция упрощает заполнение, требуя ввод до тех пор, пока он не будет верным.  
 \*/  
 protected void enter(IOCollection io, String welcome, String error, ISetter setter) {  
 while (true) {  
 io.out().print(welcome + ": ");  
  
 if (setter.trySet(io.in().readLine())) {  
 break;  
 }  
  
 io.out().error(error + " Please try again.");  
 }  
 }  
}  
  
  
src/util/Decorator.java:  
package util;  
  
/\*\*  
 \* Декоратор с палитрой из 5 цветов  
 \*/  
public class Decorator {  
 public static final String RESET = colour(0);  
 public static final String RED = colour(31);  
 public static final String GREEN = colour(32);  
 public static final String YELLOW = colour(33);  
 public static final String BLUE = colour(34);  
  
 public static String colour(int num) {  
 return "\u001B[" + num + "m";  
 }  
}  
  
  
src/util/Reader.java:  
package util;  
  
import organizer.iface.IReader;  
  
import java.io.BufferedReader;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
  
/\*\*  
 \* Адаптер ввода  
 \*/  
public class Reader implements IReader {  
 protected BufferedReader reader;  
  
 public Reader(InputStream stream) {  
 reader = new BufferedReader(new InputStreamReader(stream));  
 }  
   
 @Override  
 public String readLine() {  
 String line;  
  
 try {  
 line = reader.readLine();  
 } catch (Exception e) {  
 return "";  
 }  
   
 return line;  
 }  
}  
  
  
src/util/Reporter.java:  
package util;  
  
import organizer.iface.IReporter;  
  
import java.io.PrintStream;  
  
/\*\*  
 \* Адаптер вывода  
 \*/  
public class Reporter implements IReporter {  
 protected PrintStream stream;  
  
 public Reporter(PrintStream stream) {  
 this.stream = stream;  
 }  
  
 @Override  
 public void print(String message) {  
 stream.print(Decorator.RESET + message);  
 }  
  
 @Override  
 public void println(String message) {  
 stream.println(Decorator.RESET + message);  
 }  
  
 @Override  
 public void debug(String message) {  
 stream.println(Decorator.BLUE + message + Decorator.RESET);  
 }  
  
 @Override  
 public void error(String message) {  
 stream.println(Decorator.RED + message + Decorator.RESET);  
 }  
  
 @Override  
 public void warning(String message) {  
 stream.println(Decorator.YELLOW + message + Decorator.RESET);  
 }  
  
 @Override  
 public void success(String message) {  
 stream.println(Decorator.GREEN + message + Decorator.RESET);  
 }  
}  
  
  
src/commander/iface/ICallback.java:  
package commander.iface;  
  
import commander.Command;  
  
/\*\*  
 \* Интерфейс обратного вызова команды  
 \*/  
public interface ICallback {  
 void execute(Command command);  
}  
  
  
src/commander/util/ArgBuilder.java:  
package commander.util;  
  
import java.util.ArrayList;  
  
/\*\*  
 \* Сборщик аргумента  
 \*/  
public class ArgBuilder {  
 protected ArrayList<String> args = new ArrayList<>();  
 protected StringBuilder builder = new StringBuilder();  
  
 /\*\*  
 \* Добавляет символ в аргумент  
 \*/  
 public void add(char c) {  
 builder.append(c);  
 }  
  
 /\*\*  
 \* Записывает аргумент в массив  
 \*/  
 public void push() {  
 if (!builder.isEmpty()) {  
 args.add(builder.toString());  
 builder.setLength(0);  
 }  
 }  
  
 /\*\*  
 \* Возвращает массив аргументов  
 \*/  
 public String[] build() {  
 return args.toArray(String[]::new);  
 }  
}  
  
  
src/custom/tasks/TaskWithPlace.java:  
package custom.tasks;  
  
import organizer.Task;  
import organizer.util.IOCollection;  
  
/\*\*  
 \* Расширяем базовый класс задания, добавляя в него поле "место"  
 \*/  
public class TaskWithPlace extends Task {  
 protected String place;  
  
 public String getPlace() {  
 return place;  
 }  
  
 public boolean setPlace(String place) {  
 this.place = place;  
 return true;  
 }  
  
 @Override  
 public String detailedView() {  
 return String.format("""  
 %s  
 Place: %s""", super.detailedView(), getPlace());  
 }  
  
 @Override  
 public void fill(IOCollection io) {  
 super.fill(io); // Заполняем поля из родительского класса  
  
 // Просим ввести поле "место"  
 enter(io, "Enter place", "Please enter a valid place.", this::setPlace);  
 }  
}  
  
  
src/organizer/iface/IReader.java:  
package organizer.iface;  
  
/\*\*  
 \* Интерфейс адаптера ввода  
 \*/  
public interface IReader {  
 String readLine();  
}  
  
  
src/organizer/iface/IReporter.java:  
package organizer.iface;  
  
/\*\*  
 \* Интерфейс адаптера вывода  
 \*/  
public interface IReporter {  
 void print(String message);  
 void println(String message);  
  
 void debug(String message);  
 void error(String message);  
 void warning(String message);  
 void success(String message);  
}  
  
  
src/organizer/iface/ISetter.java:  
package organizer.iface;  
  
/\*\*  
 \* Интерфейс для функции organizer.Task::enter  
 \*/  
public interface ISetter {  
 boolean trySet(String line);  
}  
  
  
src/organizer/util/IOCollection.java:  
package organizer.util;  
  
import organizer.iface.IReader;  
import organizer.iface.IReporter;  
  
/\*\*  
 \* Пара адапетров ВВОД + ВЫВОД  
 \*/  
public class IOCollection {  
 protected IReader reader;  
 protected IReporter reporter;  
  
 public IOCollection(IReader reader, IReporter reporter) {  
 this.reader = reader;  
 this.reporter = reporter;  
 }  
  
 public IReader in() {  
 return reader;  
 }  
  
 public IReporter out() {  
 return reporter;  
 }  
}  
  
  
src/organizer/util/TimeFormatter.java:  
package organizer.util;  
  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
  
/\*\*  
 \* Позволяет удобно конвертировать LocalDateTime в строку и обратно  
 \*/  
public class TimeFormatter {  
 public static String format = "yyyy-MM-dd HH:mm";  
  
 public static DateTimeFormatter formatter() {  
 return DateTimeFormatter.ofPattern(format);  
 }  
  
 public static String retrieve(LocalDateTime ldt) {  
 return ldt.format(formatter());  
 }  
  
 public static LocalDateTime retrieve(String string) {  
 return LocalDateTime.parse(string, formatter());  
 }  
}

**Приложение 2**

**Результаты тестирования**

