МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«Харківський Політехнічний Інститут»  
  
Кафедра Стратегічного Управління

ЗВІТ

з лабораторної роботи № 6

«Використання патернів проектування»

Перевірила: старший викладач  
Вільхівська О. В.  
Виконав: ст. гр. КН-27

Харків, 2018

Лабораторна робота № 6  
Тема: Використання патернів проектуванняЗавдання

**Завдання**

### **Індивідуальне завдання**

Модифікувати раніше створену програму індивідуального завдання лабораторної роботи № 3, побудувавши код з використанням проектних патернів. Обов'язково реалізувати патерни "Facade", "Singleton" "Lazy Initialization" і "Factory Method". Додатково можна застосувати патерни "Abstract Factory" і "Observer", а також будь-які інші патерни на свій розсуд.

Хід роботи

### **Завдання 1.**

*Код програми*

*Вміст AbstractSubject.java*

|  |
| --- |
| **package** task\_1.Model;  **import** java.util.\*;  **public** **abstract** **class** AbstractSubject {  **public** **abstract** **void** setTopics(AbstractTopic[] topics);  **public** **abstract** **void** setName(String str);  **public** **abstract** **void** sortByTopicLength();  **public** **abstract** **void** sortByStudents();  **public** **abstract** **void** ClearTopics();    **public** **abstract** String getName();  **public** **abstract** AbstractTopic[] getTopics();  **public** **abstract** **boolean** getExams();  **public** **abstract** **void** setExams(**boolean** b);  // Factory Method  **public** **abstract** AbstractSubject getInstance();  **public** **double** getAverage() {  **double** average=0;  **for** (AbstractTopic topic: getTopics()) {  average+=topic.getAmount();  }  **return** average/=getTopics().length;  }    **public** **static** AbstractTopic[] addToArray(AbstractTopic[] arr, AbstractTopic item) {  AbstractTopic[] newArr;  **if** (arr!=**null**) {  newArr = **new** AbstractTopic[arr.length+1];  System.*arraycopy*(arr, 0, newArr, 0, arr.length);  }  **else** {  newArr = **new** AbstractTopic[1];  }  newArr[newArr.length-1]=item;  **return** newArr;  }  **public** AbstractTopic getMin() {  AbstractTopic lecture = getTopics()[0];  **for** (**int** i=1; i<getTopics().length; i++) {  **if**(getTopics()[i].getAmount()>lecture.getAmount()) {  lecture = getTopics()[i];  }  }  **return** lecture;  }  **public** AbstractTopic[] findWord(String word) {  System.***out***.println("\nInfo about lecture containing word " + word);  AbstractTopic[] at = {};  **for** (AbstractTopic topic: getTopics()) {  **if** (topic.containsWord(word)) {  System.***out***.println(topic);  *addToArray*(at,topic);  }  }  **return** at;  }  @Override  **public** String toString() { //overloading toString() method  String result = "Name: " + getName() + " Exams: " + getExams() + " Amount of practices: " + getTopics().length;  **for** (**int** i = 0; i < getTopics().length; i++) {  result += "\n" + getTopics()[i];  }  **return** result;  }  @Override  **public** **boolean** equals(Object obj) {  **if** (**this**==obj) **return** **true**;  **if** ((obj==**null**)||!(obj **instanceof** AbstractSubject)) **return** **false**;  AbstractSubject as = (AbstractSubject) obj;  **if** (!getName().equals(as.getName())) **return** **false**;  **return** Arrays.*equals*(getTopics(), as.getTopics());    }  **public** **boolean** addTopic(AbstractTopic topic) {  **if** (getTopics()!=**null**) {  **for** (AbstractTopic p : getTopics()) {  **if** (p.equals(topic)) {  **return** **false**;  }  }  }  setTopics(*addToArray*(getTopics(), topic));  **return** **true**;  }  **protected** AbstractSubject createSubject() {  AbstractSubject result=getInstance();  setName("OOP");  System.***out***.println("\nAdding new practices to " + getName());  System.***out***.println(addTopic(**new** TopicWithData("21,9,2018", "Generics", 8)));  System.***out***.println(addTopic(**new** TopicWithData("2,10,2018", "Exceptions", 3)));  System.***out***.println(addTopic(**new** TopicWithData("16,10,2018", "GUI", 8)));  System.***out***.println(addTopic(**new** TopicWithData("30,10,2018", "Reflection", 7)));  System.***out***.println(addTopic(**new** TopicWithData("30,10,2018", "Reflection", 7)));  **return** **this**;  }  **public** **void** testSubject() {  sortByStudents();  System.***out***.println("\nSorting by students amount: \n" + **this**);  sortByTopicLength();  System.***out***.println("\nSorting by the length of a topic name: \n" + **this**);  findWord("Reflection");  }  **public** **int** topicsCount() {  // **TODO** Auto-generated method stub  **return** 0;  }  **public** AbstractTopic getTopic(**int** i) {  // **TODO** Auto-generated method stub  **return** **null**;  }  **public** **void** setTopic(**int** i, AbstractTopic topic) {  // **TODO** Auto-generated method stub    }  **public** **boolean** addTopic(String name, **int** amount, String date) {  // **TODO** Auto-generated method stub  **return** **false**;  }  } |

*Вміст SubjectWithArray.java*

|  |
| --- |
| **package** task\_1.Model;  **import** java.util.\*;  **public** **class** SubjectWithArray **extends** AbstractSubject {  **private** AbstractTopic[] Topics;  **private** String name;  **private** **boolean** exams;  **public** SubjectWithArray(AbstractTopic[] topics, String name, **boolean** exams) {  **super**();  Topics = topics;  **this**.name = name;  **this**.exams = exams;  }  **public** SubjectWithArray() {  // **TODO** Auto-generated constructor stub  }  @Override  **public** AbstractTopic[] getTopics() {  **return** Topics;  }  @Override  **public** String getName() {  **return** name;  }  @Override  **public** **boolean** getExams() {  **return** exams;  }  @Override  **public** **void** setTopics(AbstractTopic[] topics) {  **this**.Topics = topics;  }  @Override  **public** **void** setName(String str) {  **this**.name = str;  }  @Override  **public** **void** sortByStudents() {  Arrays.*sort*(Topics);  }  @Override  **public** **void** setExams(**boolean** b){ **this**.exams = b; }  @Override  **public** **void** sortByTopicLength(){  Arrays.*sort*(Topics, (t1, t2)->Integer.*compare*(t1.getName().length(),t2.getName().length()));  }  @Override  **public** **void** ClearTopics() {  Topics = **null**;  }  // Overridden Factory Method  @Override  **public** AbstractSubject getInstance() {  **return** **new** SubjectWithArray();  }  **public** **static** **void** main(String[] args) {  // Creating new main abstract class AbstractSubject  AbstractSubject as = **new** SubjectWithArray();  // creating a child class SubjectWithArray in the parent class and testing it  as.createSubject().testSubject();  }  } |

*Вміст SubjectWithList.java*

|  |
| --- |
| **package** task\_1.Model;  **import** java.util.\*;  **public** **class** SubjectWithList **extends** AbstractSubject {  **private** List<AbstractTopic> Topics = **new** ArrayList<>();  **private** String name;  **private** **boolean** exams;  **public** SubjectWithList(List<AbstractTopic> topics, String name, **boolean** exams) {  //super();  Topics = topics;  **this**.name = name;  **this**.exams = exams;  }  **public** SubjectWithList() {  // **TODO** Auto-generated constructor stub  }  @Override  **public** AbstractTopic[] getTopics() {  **return** Topics.toArray(**new** AbstractTopic[Topics.size()]); // пїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅ  }  @Override  **public** String getName() {  **return** name;  }  @Override  **public** **boolean** getExams() {  **return** exams;  }  @Override  **public** **void** setTopics(AbstractTopic[] topics) {  Topics = **new** ArrayList<>(Arrays.*asList*(topics)); // пїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅ  }  @Override  **public** **void** setName(String str) {  **this**.name = str;  }  @Override  **public** **void** sortByStudents() {  Collections.*sort*(Topics);  }  @Override  **public** **void** setExams(**boolean** b){ **this**.exams = b; }  @Override  **public** **void** sortByTopicLength(){  Collections.*sort*(Topics, (t1, t2)->Integer.*compare*(t1.getName().length(),t2.getName().length()));  }  **public** **static** **void** main (String[] args) {  // Creating new main abstract class AbstractSubject  AbstractSubject as = **new** SubjectWithList();  // creating a child class SubjectWithList in the parent class and testing it  as.createSubject().testSubject();  }  @Override  **public** **void** ClearTopics() {  Topics = **null**;  }  // Overridden Factory Method  @Override  **public** AbstractSubject getInstance() {  **return** **new** SubjectWithList();  }    } |

Застосування патерну Façade, Singlton та LazyInitialization

*Вміст SubjectsController.java*

|  |
| --- |
| package task\_1.Controller;  import javafx.application.Platform;  import javafx.collections.FXCollections;  import javafx.collections.ObservableList;  import javafx.event.ActionEvent;  import javafx.fxml.FXML;  import javafx.fxml.Initializable;  import javafx.scene.control.\*;  import javafx.scene.control.Alert.AlertType;  import javafx.scene.control.TableColumn.CellEditEvent;  import javafx.scene.control.cell.PropertyValueFactory;  import javafx.scene.control.cell.TextFieldTableCell;  import javafx.stage.FileChooser;  import javafx.util.converter.IntegerStringConverter;  import task\_1.Model.AbstractTopic;  import task\_1.Model.TopicsFacade;  import task\_1.Model.XML.XMLSubject;  import javax.xml.bind.JAXBException;  import java.io.File;  import java.io.IOException;  import java.net.URL;  import java.util.ArrayList;  import java.util.List;  import java.util.ResourceBundle;  public class SubjectsController implements Initializable {  private TopicsFacade facade = TopicsFacade.getInstance();  @FXML public TextField textFieldSubject;  @FXML public TextField textFieldText;  @FXML public TextArea textAreaResults;  @FXML public TableView<AbstractTopic> tableViewTopics;  @FXML public TableColumn<AbstractTopic, String> tableColumnName;  @FXML public TableColumn<AbstractTopic, Integer> tableColumnAmount;  @FXML public TableColumn<AbstractTopic, String> tableColumnDate;  @FXML public void doNew(ActionEvent event) {  facade.doNew();  }  @FXML public void nameChanged(ActionEvent event) {  facade.nameChanged();  }  @FXML public void doSearchByWord(ActionEvent event) {  facade.doSearchByWord();  }  @FXML public void doSearchBySubstring(ActionEvent event) {  facade.doSearchBySubstring();  }  @FXML public void doAbout(ActionEvent event) {  facade.doAbout();  }  @FXML public void doOpen(ActionEvent event) {  facade.doOpen();  }  @FXML  public void doSave(ActionEvent event) {  facade.doSave();  }  @FXML public void doSortByStudentsAmount(ActionEvent event) {  facade.doSortByStudentsAmount();  }  @FXML public void doSortByName(ActionEvent event) {  facade.doSortByName();  }  @FXML public void doRemove(ActionEvent event) {  facade.doRemove();  }  @FXML  public void doExit(ActionEvent event) {  facade.doExit();  }  @FXML public void doAdd(ActionEvent event) {  facade.doAdd();  }  @Override  public void initialize(URL location, ResourceBundle resources) {  facade.setSubjectsController(this);  }  } |

Вміст TopicsFacade

|  |
| --- |
| package task\_1.Model;  import java.io.File;  import java.io.IOException;  import java.util.ArrayList;  import java.util.List;  import javax.xml.bind.JAXBException;  import javafx.application.Platform;  import javafx.collections.FXCollections;  import javafx.collections.ObservableList;  import javafx.event.ActionEvent;  import javafx.fxml.FXML;  import javafx.scene.control.Alert;  import javafx.scene.control.Label;  import javafx.scene.control.TableColumn;  import javafx.scene.control.TableView;  import javafx.scene.control.TextArea;  import javafx.scene.control.TextField;  import javafx.scene.control.Alert.AlertType;  import javafx.scene.control.TableColumn.CellEditEvent;  import javafx.scene.control.cell.PropertyValueFactory;  import javafx.scene.control.cell.TextFieldTableCell;  import javafx.stage.FileChooser;  import javafx.util.converter.IntegerStringConverter;  import task\_1.Controller.SubjectsController;  import task\_1.Model.XML.XMLSubject;  public class TopicsFacade {  private static TopicsFacade instance = null;  private XMLSubject subject = null;  private ObservableList<AbstractTopic> observableList;  SubjectsController subjectsController;  public void setSubjectsController(SubjectsController subjectsController) {  this.subjectsController=subjectsController;  }  // Using Lazy Initialization Pattern  public XMLSubject getSubject() {  if (subject == null) {  subject = new XMLSubject();  }  return subject;  }  // посилання на інші класи моделі, якщо необхідно  // модифкатор private для конструктора унеможливлює створення  // об'єктів не через метод getInstance():  private TopicsFacade() {  }  // реалізація патерну Singleton:  public static TopicsFacade getInstance() {  if (instance == null) {  instance = new TopicsFacade();  }  return instance;  }  // Методи, які відповідають функціям GUI-застосунку  // та вимагають взаємодії з моделлю:    public void nameChanged() {  // Коли користувач змінив дані в textFieldCountry,  // автоматично оновлюємо назву:  getSubject().setName(subjectsController.textFieldSubject.getText());  }    public void doSearchByWord() {  // Оновлюємо дані:  updateSourceData();  subjectsController.textAreaResults.setText("");  for (int i = 0; i < getSubject().topicsCount(); i++) {  AbstractTopic topic = getSubject().getTopic(i);  if (topic.containsWord(subjectsController.textFieldText.getText())) {  showResults(topic);  }  }  }  public void doSearchBySubstring() {  // Оновлюємо дані:  updateSourceData();  subjectsController.textAreaResults.setText("");  for (int i = 0; i < getSubject().topicsCount(); i++) {  AbstractTopic topic = getSubject().getTopic(i);  if (topic.getName().toLowerCase().contains(subjectsController.textFieldText.getText().toLowerCase())) {  showResults(topic);  }  }  }  public void doAbout() {  Alert alert = new Alert(AlertType.INFORMATION);  alert.setTitle("О программе...");  alert.setHeaderText("Данные об учебном курсе");  alert.setContentText("Версія 1.0");  alert.showAndWait();  }  private void showResults(AbstractTopic topic) {  subjectsController.textAreaResults.appendText("Тема: " + topic.getName() + "\n");  subjectsController.textAreaResults.appendText("Студентов: " + topic.getAmount() + "\n");  subjectsController.textAreaResults.appendText("Лекция за: " + topic.getDate() + "\n");  }    public static FileChooser getFileChooser(String title) {  FileChooser fileChooser = new FileChooser();  // Починаємо шукати з поточної теки:  fileChooser.setInitialDirectory(new File("src/task\_1/Model/XML/Samples"));  // Встановлюємо фільтри для пошуку файлів:  fileChooser.getExtensionFilters().add(  new FileChooser.ExtensionFilter("XML-файли (\*.xml)", "\*.xml"));  fileChooser.getExtensionFilters().add(  new FileChooser.ExtensionFilter("Лалалэй (\*.\*)", "\*.\*"));  // Вказуемо заголовк вікна:  fileChooser.setTitle(title);  return fileChooser;  }  public void doOpen() {  FileChooser fileChooser = getFileChooser("Открыть XML-файл");  File file;  if ((file = fileChooser.showOpenDialog(null)) != null) {  try {  getSubject().readFromFile(file.getCanonicalPath());  // Заповнюємо текстові поля прочитаними даними:  subjectsController.textFieldSubject.setText(getSubject().getName());  subjectsController.textAreaResults.setText("");  // Очищаємо та оновлюємо таблицю:  subjectsController.tableViewTopics.setItems(null);  updateTable();  }  catch (IOException e) {  showError("Файл не знайдено");  }  catch (JAXBException e) {  e.printStackTrace();  showError("Неправильний формат файлу");  }  }  }  public static void showError(String message) {  Alert alert = new Alert(AlertType.ERROR);  alert.setTitle(":(");  alert.setHeaderText(message);  alert.showAndWait();  }  private void updateSourceData() {  // Переписуємо дані в модель з observableList  getSubject().ClearTopics();  getSubject().setName(subjectsController.textFieldSubject.getText());  for (AbstractTopic topic : observableList) {  getSubject().addTopic(topic);  }  }  public void doNew() {  subjectsController.tableViewTopics.setPlaceholder(new Label(""));  getSubject();  observableList = null;  subjectsController.textFieldSubject.setText("");  subjectsController.textFieldText.setText("");  subjectsController.textAreaResults.setText("");  subjectsController.tableViewTopics.setItems(null);  subjectsController.tableViewTopics.setPlaceholder(new Label(""));  }  public void doSave() {  FileChooser fileChooser = getFileChooser("Сохранить XML-файл");  File file;  if ((file = fileChooser.showSaveDialog(null)) != null) {  try {  updateSourceData(); // оновлюємо дані в моделі  getSubject().writeToFile(file.getCanonicalPath());  showMessage("Сохранено");  }  catch (Exception e) {  showError("Помилка запису в файл");  }  }  }  public static void showMessage(String message) {  Alert alert = new Alert(AlertType.INFORMATION);  alert.setTitle("");  alert.setHeaderText(message);  alert.showAndWait();  }  public void doSortByStudentsAmount() {  updateSourceData();  getSubject().sortByStudents();  updateTable();  }  public void doSortByName() {  updateSourceData();  getSubject().sortByTopicLength();  updateTable();  }  public void doRemove() {  // Не можемо видалити рядок, якщо немає даних:  if (observableList == null) {  return;  }  // Якщо є рядки, видаляємо останній:  if (observableList.size() > 0) {  observableList.remove(observableList.size() - 1);  }  // Якщо немає рядків, вказуємо, що дані відсутні:  if (observableList.size() <= 0) {  observableList = null;  }  }  public void doExit() {  Platform.exit(); // коректне завершення застосунку JavaFX  }  public void doAdd() {  getSubject().addTopic("",0,"");  updateTable();  }    private void updateTable() {  // Заповнюємо observableList:  List<AbstractTopic> list = new ArrayList<>();  for (int i = 0; i < getSubject().topicsCount(); i++) {  list.add(getSubject().getTopic(i));  }  observableList = FXCollections.observableList(list);  subjectsController.tableViewTopics.setItems(observableList);    // Вказуємо для колонок зв'язану з ними властивість і механізм редагування  // залежно від типу комірок:    subjectsController.tableColumnDate.setCellValueFactory(new PropertyValueFactory<>("Date"));  subjectsController.tableColumnDate.setCellFactory(TextFieldTableCell.forTableColumn());  subjectsController.tableColumnDate.setOnEditCommit(t -> updateDate(t));    subjectsController.tableColumnName.setCellValueFactory(new PropertyValueFactory<>("Name"));  subjectsController.tableColumnName.setCellFactory(TextFieldTableCell.forTableColumn());  subjectsController.tableColumnName.setOnEditCommit(t -> updateName(t));    subjectsController.tableColumnAmount.setCellValueFactory(new PropertyValueFactory<>("Amount"));  subjectsController.tableColumnAmount.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()));  subjectsController.tableColumnAmount.setOnEditCommit(t -> updateAmount(t));  }    private void updateDate(CellEditEvent<AbstractTopic, String> t) {  // Оновлюємо коментарі:  AbstractTopic at = t.getTableView().getItems().get(t.getTablePosition().getRow());  at.setDate(t.getNewValue());  }  private void updateName(CellEditEvent<AbstractTopic, String> t) {  // Оновлюємо коментарі:  AbstractTopic at = t.getTableView().getItems().get(t.getTablePosition().getRow());  at.setName(t.getNewValue());  }  private void updateAmount(CellEditEvent<AbstractTopic, Integer> t) {  // Оновлюємо коментарі:  AbstractTopic at = t.getTableView().getItems().get(t.getTablePosition().getRow());  at.setAmount(t.getNewValue());  }  } |

Висновки

Під час виконання цієї лабораторної роботи я ознайомився з твірними патернами Factory Method, Lazy Initialization, Singleton та структурним патреном Façade. Петерн – це опис взаємодії між об'єктами, що використовується для вирішення певіних задач. У нашому випадку Façade забеспечив надання інтерфейсу до контроллеру, Factory Method дозволив створювати об'єкти спадкоємці з батьківського класу, Lazy Initialization зумовило невеликий приріст у швидкості виконання та Singleton лишив можливості створювати новий об’єкт фасаду, зумовлюючи тим самим певний рівень безпеки.