Зразок оформлення титульного аркуша лабораторної роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**ЛАБОРАТОРНА РОБОТА No3**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему

Реалізація породжуючих шаблонів проектування

Варіант No16

Виконав студент

ІІ курсу групи КП-ХХ

Палій Дмитро Володимирович

залікова книжка КП-8321

Перевірив

доцент, к.т.н. Заболотня Т.М.

Оцінка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, підпис)

КИЇВ 2020

**Мета роботи**

Ознаойомлення з основними характеристиками всіх

породжуючих шаблонів («Прототипу», «Фабричного методу»,

«Абстрактної фабрики», «Одинака», «Будівельника»), запам’ятовування

поширених ситуацій, коли використання цих шаблонів є доцільним,

набуття вмінь та навичок реалізації шаблонів під час створення

програмного коду.

**Постановка завдання**

Варіант 16

1) За допомогою шаблона проектування побудувати генератор

комплектів пам’яток для студентів, які можуть роздаватися у приймальній

комісії. Наприклад, для студентів-першокурсників до комплекту

включають карту університету, список необхідних телефонів служб

університету та факультету, інформацію щодо програми навчання до

ступеня «бакалавра», основні правила навчання в вузі (за що відраховують,

за що видають червоний диплом тощо). Для студентів-магістрів до

комплекту включають програму навчання до ступеня «магістр», загальні

поради щодо написання наукових статей, магістерської дисертації тощо.

Сформувати по N комплектів обох видів.

2) Забезпечити створення нових користувачів програми за

допомогою клонування базового прототипу користувача. Полями, які

повинна містити особова картка користувача, можуть бути: логін, пароль,

ПІБ, вік, місто, колір шрифту для спілкування у чаті тощо.

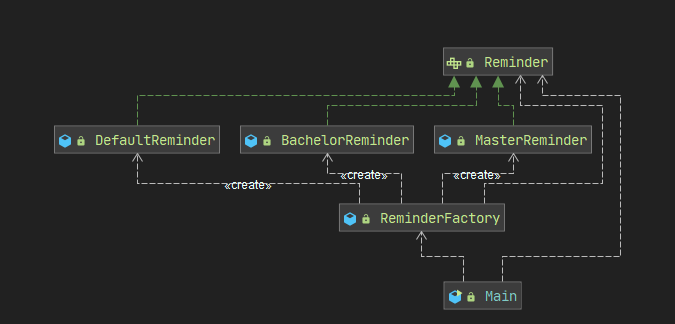
**Обгрунтування вибору шаблонів**

**Завдання 1:** обрано шаблон «Фабричний метод», оскільки клас спроектовано таким чином, щоб специфікація породжуваного об’єкту визначається тільки в нащадках. **Завдання 2:** обрано шаблон «Прототип», оскільки це єдиний породжуючий шаблон, котрий для створення нових об’єктів використовує не явне інстанціювання (ключове слово new), а клонування.

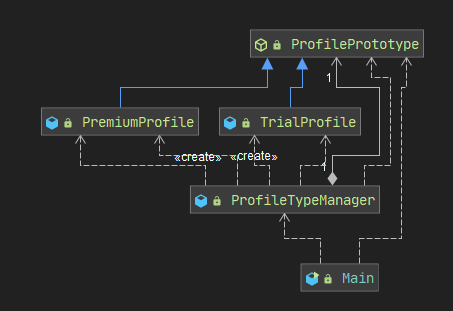
**Тексти коду програм**

Діаграма класів:

**Завдання 1:**

****

**Завдання 2:**

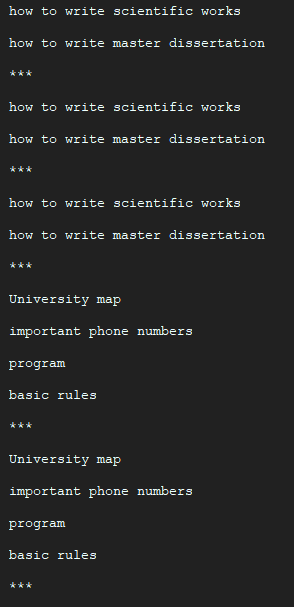


|  |
| --- |
| **Завдання 1** |
| *public class* Main {  *private static final int N* = 10;  *public static void* main(String[] args) {  *var* factory = ReminderFactory.*getInstance*();  *var* masterReminders = *new* ArrayList<*Reminder*>();  *var* bachelorReminders = *new* ArrayList<*Reminder*>();  *for* (*int* i = 0; i < *N*; i++)  {  masterReminders.add(factory.createReminder("master"));  bachelorReminders.add(factory.createReminder("bachelor"));  }  *for* (*Reminder* reminder: masterReminders)  {  System.*out*.println(reminder.printReminder());  System.*out*.println("\*\*\*");  }  *for* (*Reminder* reminder: bachelorReminders)  {  System.*out*.println(reminder.printReminder());  System.*out*.println("\*\*\*");  }  *var* reminder = factory.createReminder("student");  System.*out*.println(reminder.printReminder());  } }  *public class* BachelorReminder *implements Reminder* {  @Override  *public* String printReminder()  {  *return* "University map\n" +  "important phone numbers\n" +  "program\n" +  "basic rules";  } }  *public class* DefaultReminder *implements Reminder* {   @Override  *public* String printReminder() {  *return* "learn hard!";  } }  *public class* MasterReminder *implements Reminder* {  @Override  *public* String printReminder() {  *return* "how to write scientific works\n" +  "how to write master dissertation";  } }  *public interface Reminder* {   *public* String printReminder(); }  *public class* ReminderFactory {  *private static* ReminderFactory *instance* = *null*;  *private* ReminderFactory() {}  *public static* ReminderFactory getInstance()  {  *if* (*instance* == *null*) *instance* = *new* ReminderFactory();  *return instance*;  }  *public Reminder* createReminder(String type)  {  *Reminder* reminder = *null*;  *if* (type.equals("master")) reminder = *new* MasterReminder();  *else if* (type.equals("bachelor")) reminder = *new* BachelorReminder();  *else* reminder = *new* DefaultReminder();  *return* reminder;  } } |

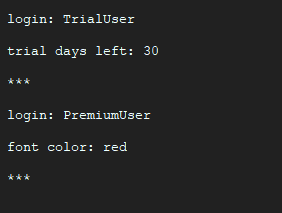
|  |
| --- |
| **Завдання 2** |
| *public class* Main {   *public static void* main(String[] args)  {  ProfileTypeManager.*loadProfiles*();  *var* clonedTrialProfile = ProfileTypeManager.*getProfile*("trial");  clonedTrialProfile.showProfile();  System.*out*.println("\*\*\*");  *var* clonedPremiumPrototype = ProfileTypeManager.*getProfile*("premium");  clonedPremiumPrototype.showProfile();  System.*out*.println("\*\*\*");  } }  *public class* PremiumProfile *extends* ProfilePrototype {  *private* String fontColor;  *public* String getFontColor()  {  *return* fontColor;  }  *public void* setFontColor(String value)  {  *this*.fontColor = value;  }  @Override  *public void* showProfile() {  *super*.showProfile();  System.*out*.println("font color: " + fontColor);  } }  *public abstract class* ProfilePrototype *implements Cloneable* {  *private* String password;  *private* String login;    *public void* setPassword(String value)  {  *this*.password = value;  }  *public void* setLogin(String value)  {  *this*.login = value;  }  *public* Object clone()  {  Object clone = *null*;  *try* {  clone = *super*.clone();  }  *catch* (CloneNotSupportedException e)  {  e.printStackTrace();  }  *return* clone;  }  *public void* showProfile()  {  System.*out*.println("login: " + login);  } }  *public class* ProfileTypeManager {  *private static* Hashtable *profileMap* = *new* Hashtable<String, ProfilePrototype>();   *public static* ProfilePrototype getProfile(String profileType) {  ProfilePrototype profile = (ProfilePrototype) *profileMap*.get(profileType);  *return* (ProfilePrototype) profile.clone();  }  *public static void* loadProfiles() {  *var* premium = *new* PremiumProfile();  premium.setFontColor("red");  premium.setLogin("PremiumUser");  premium.setPassword("default");  *profileMap*.put("premium", premium);  *var* trial = *new* TrialProfile();  trial.setTrialDaysLeft(30);  trial.setLogin("TrialUser");  trial.setPassword("default");  *profileMap*.put("trial", trial);  } }  *public class* TrialProfile *extends* ProfilePrototype {  *private int* trialDaysLeft;  *public int* getTrialDaysLeft()  {  *return* trialDaysLeft;  }  *public void* setTrialDaysLeft(*int* value)  {  *this*.trialDaysLeft = value;  }  @Override  *public void* showProfile()  {  *super*.showProfile();  System.*out*.println("trial days left: " + trialDaysLeft);  } } |

**Результати виконання**

**Завдання 1**

****

**Завдання 2**

****

**Висновки**

У цій лабораторній роботі я ознайомився з основними характеристиками всіх

породжуючих шаблонів («Прототипу», «Фабричного методу»,

«Абстрактної фабрики», «Одинака», «Будівельника»), запам’ятав

поширені ситуації, коли використання цих шаблонів є доцільним,

набув вмінь та навичок реалізації шаблонів під час створення

програмного коду.