**Лабораторна робота №3**

**Базові принципи ООП**

**Мета**: Використовуючи теоретичне підґрунтя про базові принципи ООП та підходи використання в Java виконати дії що будуть вказано в завданні до лабораторної роботи.

**Хід роботи**

**Посилання на GitHub:** [OOP-KB-222-Ann-Dryzhak/lab\_03 at main · aanutadrzhk/OOP-KB-222-Ann-Dryzhak · GitHub](https://github.com/aanutadrzhk/OOP-KB-222-Ann-Dryzhak/tree/main/lab_03)

1. **Ticket:** Кожен *ticket* має свій ідентифікатор, ім'я та приблизний час для його виконання. Ці значення надаються через конструктора класу *Ticket*. *Ticket* може вважатися завершеним або незавершеним. Щойно створений *ticket* вважається незавершеним.

package lab\_03;

public class Ticket {

private int id;

private String name;

private int estimate;

private boolean completed;

public Ticket(int id, String name, int estimate) {

this.id = id;

this.name = name;

this.estimate = estimate;

this.completed = false;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

public int getEstimate() {

return estimate;

}

public boolean isCompleted() {

return completed;

}

public void complete() {

completed = true;

}

}

Пояснення:

* Оголошуємо клас Ticket.
* Оголошуємо приватне ціле поле id, яке зберігає ідентифікатор кожного завдання.
* Оголошує приватне рядкове поле name, яке зберігає назву кожного завдання.
* Оголошує приватне ціле поле estimate, яке зберігає оцінку часу, необхідного для виконання кожного завдання.
* Оголошує приватне логічне поле completed, яке вказує, чи завершено завдання.
* public Ticket(int id, String name, int estimate) - оголошуємо конструктор класу Ticket, який приймає параметри id, name та estimate для ініціалізації об'єкта Ticket.
* Присвоюємо полю id значення, передане в конструктор.
* Присвоює полю name значення, передане в конструктор.
* Присвоює полю estimate значення, передане в конструктор.
* Встановлює початкове значення completed як false, що значить, що завдання на момент створення ще не завершене.
* public int getId() { return id; } - цим методом повертаємо значення поля id.
* public String getName() { return name; } - цим методом повертаємо значення поля name.
* public int getEstimate() { return estimate; } - цим методом повертаємо значення поля estimate.
* public boolean isCompleted() { return completed; } - цим методом повертаємо значення поля completed, що вказує, чи завершено завдання.
* public void complete() { completed = true; } - цим методом встановлюємо значення поля completed як true, що означає, що завдання завершено.

1. **UserStory:** *UserStory* (історія користувача) представляє ticket, який може містити залежності. Залежності — це інші екземпляри *UserStory*, які мають бути завершені в першу чергу, щоб потім вдалося завершити і залежне від них *UserStory*. Залежності передаються через конструктора класу *UserStory*.

package lab\_03;

import java.util.ArrayList;

import java.util.List;

class UserStory extends Ticket {

private List<UserStory> dependencies;

public UserStory(int id, String name, int estimate, List<UserStory> dependencies) {

super(id, name, estimate);

this.dependencies = dependencies;

}

public List<UserStory> getDependencies() {

return new ArrayList<>(dependencies);

}

*@Override*

public void complete() {

boolean allDependenciesCompleted = dependencies.stream().allMatch(UserStory::isCompleted);

if (allDependenciesCompleted) {

super.complete();

}

}

*@Override*

public String toString() {

return "[US " + getId() + "] " + getName();

}

}

Пояснення:

* Оголошуємо клас UserStory, який успадковує всі властивості та методи класу Ticket.
* private List<UserStory> dependencies - оголошуємо приватне поле dependencies, яке представляє залежності даної історії користувача від інших історій користувачів.
* public UserStory(int id, String name, int estimate, List<UserStory> dependencies) - оголошуємо конструктор класу UserStory, який приймає параметри id, name, estimate та dependencies для ініціалізації об'єкта UserStory.
* super(id, name, estimate) - викликаємо конструктор батьківського класу Ticket з параметрами id, name та estimate, щоб ініціалізувати поля класу Ticket.
* this.dependencies = dependencies - ініціалізуємо поле dependencies переданим списком залежностей.
* public List<UserStory> getDependencies() - оголошуємо метод getDependencies(), який повертає копію списку залежностей.
* public void complete() - перевизначаємо метод complete(), який перевіряє. (@Override - анотація, що показує перевизначення методу з батьківського класу).
* boolean allDependenciesCompleted = dependencies.stream().allMatch(UserStory::isCompleted) - перевіряємо чи всі залежності є завершеними, використовуючи метод isCompleted() для кожної залежності.
* if (allDependenciesCompleted) { super.complete(); } - якщо всі залежності є завершеними, то викликаємо метод complete() батьківського класу Ticket, щоб завершити історію користувача.
* Перевизначаємо метод toString(), який повертає рядок у вигляді "[US {id}] {name}".

1. **Bug:** Багом вважається ticket, що відноситься до завершеної UserStory. Баги не існують самі по собі, без відповідного екземпляра UserStory.

package lab\_03;

class Bug extends Ticket {

private UserStory userStory;

private Bug(int id, String name, int estimate, UserStory userStory) {

super(id, name, estimate);

this.userStory = userStory;

}

public static Bug createBug(int id, String name, int estimate, UserStory userStory) {

if (userStory != null && !userStory.isCompleted()) {

return new Bug(id, name, estimate, userStory);

}

return null;

}

*@Override*

public String toString() {

return "[Bug " + getId() + "] " + userStory.getName() + ": " + getName();

}

}

Пояснення:

* Оголошує клас Bug, який успадковує всі властивості та методи класу Ticket.
* private UserStory userStory - оголошуємо приватне поле userStory, яке представляє історію користувача, до якої відноситься цей баг.
* Оголошуємо приватний конструктор класу Bug, який є подібним до конструктора класу UserStory.
* public static Bug createBug(int id, String name, int estimate, UserStory userStory) - оголошуємо статичний метод createBug(), який створює новий об'єкт класу Bug.
* if (userStory != null && !userStory.isCompleted()) { return new Bug(id, name, estimate, userStory); } - перевіряємо, чи передана історія користувача не є нульовою та не є завершеною. Якщо умова виконується, створюється новий об'єкт Bug і повертаємо його.
* Якщо передана історія користувача є нульовою або завершеною, метод повертає null.
* Перевизначаємо метод toString(), який повертає рядок у вигляді "[Bug {id}] {userStory.getName()}: {name}".

1. **Sprint:** Спринти мають тимчасову ємність та обмеження кількості ticket'ів, які задаються через конструктор. Спринт не може містити тікет з сумарною оцінкою часу виконання, що перевищує тимчасову ємність спринту. Не допускається, щоб спринт містив більше тикетів, ніж визначено обмеженням кількості тикетів для цього спринту. Спринт повинен приймати ticket'и за допомогою методів *add*\*. Ці методи повертають *true*, коли вхідний ticket був прийнятий у спринт, і *false* в іншому випадку.

package lab\_03;

import java.util.ArrayList;

import java.util.List;

public class Sprint {

private int capacity;

private int ticketLimit;

private List<Ticket> tickets;

public Sprint(int capacity, int ticketLimit) {

this.capacity = capacity;

this.ticketLimit = ticketLimit;

this.tickets = new ArrayList<>();

}

public boolean addUserStory(UserStory userStory) {

if (userStory != null && !userStory.isCompleted() && canAddTicket(userStory)) {

tickets.add(userStory);

return true;

}

return false;

}

public boolean addBug(Bug bug) {

if (bug != null && !bug.isCompleted() && canAddTicket(bug)) {

tickets.add(bug);

return true;

}

return false;

}

public List<Ticket> getTickets() {

return new ArrayList<>(tickets);

}

public int getTotalEstimate() {

int totalEstimate = 0;

for (Ticket ticket : tickets) {

totalEstimate += ticket.getEstimate();

}

return totalEstimate;

}

private boolean canAddTicket(Ticket ticket) {

return !ticket.isCompleted() && getTotalEstimate() + ticket.getEstimate() <= capacity && tickets.size() < ticketLimit;

}

}

Пояснення:

* Оголошуємо клас Sprint.
* Оголошуємо приватне ціле поле capacity, яке представляє максимальну можливу кількість годин роботи в спринті.
* Оголошуємо приватне ціле поле ticketLimit, яке представляє обмеження кількості тикетів у спринті.
* Оголошуємо приватне поле tickets, яке є списком тикетів у спринті.
* public Sprint(int capacity, int ticketLimit) - оголошуємо конструктор класу Sprint, який приймає параметри capacity та ticketLimit і ініціалізує об'єкт спринту з відповідними обмеженнями.
* Присвоюємо полю capacity та ticketLimit значення, передане в конструктор.
* this.tickets = new ArrayList<>() - ініціалізуємо список tickets як порожній ArrayList.
* public boolean addUserStory(UserStory userStory) - оголошуємо метод addUserStory, який додає історію користувача до спринту.
* if (userStory != null && !userStory.isCompleted() && canAddTicket(userStory)) - перевіряємо, чи історія користувача не є нульовою, не завершена і чи можна додати цей тикет до спринту.
* В цьому випаду додаємо історію користувача до списку тикетів спринту та повертаємо true, щоб показати, що тикет було успішно додано до спринту.
* Повертаємо false, якщо додавання тикета було невдалим.
* Аналогічно робимо для методу addBug(Bug bug).
* Далі оголошуємо метод getTickets, який повертає список тикетів у спринті.
* return new ArrayList<>(tickets) - повертаємо копію списку тикетів, щоб забезпечити безпеку даних.
* Оголошуємо метод getTotalEstimate, який обчислює загальну оцінку часу виконання всіх тикетів у спринті.
* Спочатку ініціалізує змінну totalEstimate як нуль.
* for (Ticket ticket : tickets) { totalEstimate += ticket.getEstimate(); - обчислюємо загальну оцінку, додавши оцінку кожного тикета.
* Повертаємо загальну оцінку часу виконання всіх тикетів.
* Оголошуємо приватний метод canAddTicket.
* return !ticket.isCompleted() && getTotalEstimate() + ticket.getEstimate() <= capacity && tickets.size() < ticketLimit - перевіряємо, чи тикет не завершений, чи загальна оцінка часу виконання тикетів плюс оцінка нового тикета не перевищує ємність спринту, і чи не перевищено обмеження на кількість тикетів у спринті.

1. Клас **Main**.

package lab\_03;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Main {

public static void main(String[] args) {

Sprint sprint = new Sprint(100, 5);

UserStory userStory1 = new UserStory(1, "User Story 1", 20, new ArrayList<>());

UserStory userStory2 = new UserStory(2, "User Story 2", 30, Arrays.*asList*(userStory1));

Bug bug = Bug.*createBug*(1, "Bug 1", 10, userStory1);

sprint.addUserStory(userStory1);

sprint.addUserStory(userStory2);

sprint.addBug(bug);

int totalEstimate = sprint.getTotalEstimate();

List<Ticket> tickets = sprint.getTickets();

System.***out***.println("Total Estimate: " + totalEstimate);

System.***out***.println("Tickets:");

for (Ticket ticket : tickets) {

System.***out***.println(ticket);

}

}

}

Пояснення:

* Sprint sprint = new Sprint(100, 5) - створюємо новий об'єкт класу Sprint з максимальною ємністю 100 і обмеженням на кількість тікетів 5.
* UserStory userStory1 = new UserStory(1, "User Story 1", 20, new ArrayList<>()) - створюємо новий об'єкт класу UserStory з ідентифікатором 1, назвою "User Story 1", оцінкою часу виконання 20 і порожнім списком залежностей.
* UserStory userStory2 = new UserStory(2, "User Story 2", 30, Arrays.asList(userStory1)) - створюємо ще один об'єкт класу UserStory з ідентифікатором 2, назвою "User Story 2", оцінкою часу виконання 30 і списком залежностей, що містить першу історію користувача.
* Bug bug = Bug.createBug(1, "Bug 1", 10, userStory1) - створюємо новий об'єкт класу Bug за допомогою статичного методу createBug(). Цей баг має ідентифікатор 1, назву "Bug 1", оцінку часу виконання 10 і залежність від першої історії користувача.
* sprint.addUserStory(userStory1) - додаємо першу історію користувача до спринту. Аналогічно додаємо другу та баг.
* int totalEstimate = sprint.getTotalEstimate() - обчислюємо загальну оцінку часу виконання всіх тікетів у спринті.
* List<Ticket> tickets = sprint.getTickets() -отримуємо список всіх тікетів у спринті.
* System.out.println("Total Estimate: " + totalEstimate) - виводимо загальну оцінку часу виконання у консоль.
* for (Ticket ticket : tickets) { System.out.println(ticket); } - проходимось по кожному тікету у списку та виводимо його у консоль.

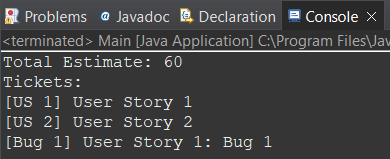


Рис. 1 - Результат виконання програми

**Висновок**. Під час виконання даної лабораторної роботи я використовуючи теоретичне підґрунтя про базові принципи ООП та підходи використання в Java виконала дії, що були вказані в завданні до лабораторної роботи.