**Звіт до Лабораторної роботи №3**

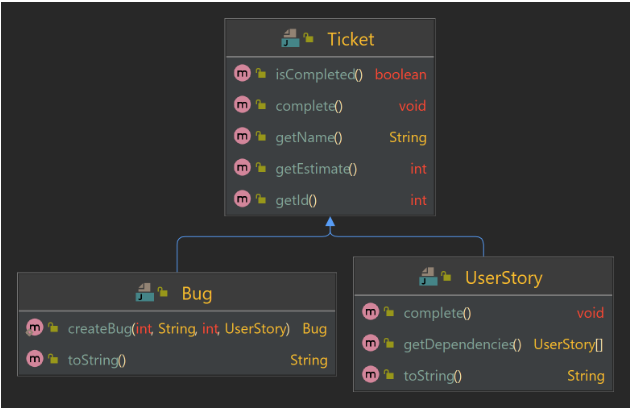
**Тема:** Базові принципи ООП

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

**Завдання для виконання**

У цьому завданні ви формулюватимете бізнес-логіку планування спринту. Спринт - це основна одиниця розробки програмного забезпечення SCRUM. Спринти обмежені за часом, час спринту узгоджується при плануванні. Вам належить організувати планування спринту, що включає завдання, які вам потрібно реалізувати на основі базового класу Ticket. Врахуйте, у цьому випадку спринт приймає лише два підтипи класу Ticket: Bug та UserStory.

Ознайомтеся з діаграмою, яка показує публічний програмний інтерфейс цих класів та їхні відношення:



Ticket

Кожен ticket має свій ідентифікатор, ім'я та приблизний час для його виконання. Ці значення надаються через конструктор класу Ticket.

Ticket може вважатися завершеним або незавершеним. Щойно створений ticket вважається незавершеним.

• getId() - повертає ідентифікатор ticket'у.

• getName() - повертає ім'я ticket'у.

• getEstimate() - повертає оцінку ticket'у.

• isCompleted() - повертає true, якщо ticket завершено, інакше — false.

• complete() - переводить ticket у завершений стан.

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 конструктор класу UserStory.

• complete() - як і метод Ticket#complete() - переводить ticket у завершений стан. Різниця лише в тому, що історія користувача може не бути завершена, якщо його залежність ще не завершена.

• getDependencies() - повертає копію масиву ticket'ів на спринт, що захищена від змін.

• toString() - повертає рядок, що представляє UserStory, використовуючи його ідентифікатор та ім'я. Приклад: з ідентифікатором =1, ім’я UserStory = "Об'єкт реєстрації користувача", отримаємо "[UserStory 1] Об'єкт реєстрації користувача".

Bug

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

• createBug(int id, String name, int estimate, UserStory userStory) - статичний метод створення екземпляра Bug. Повертає null, якщо UserStory має значення null або не завершено. Інакше повертає створений екземпляр Bug.

• toString() - повертає рядкове представлення цього бага, використовуючи ідентифікатор, ім'я багу та ім'я відповідної UserStory. Приклад: з ідентифікатором = 2, ім'ям бага = "Додати повторюваний пароль" і ім'ям відповідної UserStory = "Форма реєстрації" в результаті отримаємо "[Bug 2] Форма реєстрації: Додати повторюваний пароль".

Sprint

Спринти мають обмеження на сумарний час виконання та обмеження кількості ticket'ів, які задаються через конструктор. Спринт не може містити тікет з сумарною оцінкою часу виконання, що перевищує сумарний час виконання 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.

Спринт не повинен приймати:

1. Значення null.

2. ticket'и, які вже завершені.

3. ticket'и, що мають значення оцінки часу виконання, яке, у разі додавання ticket'а, призведе до перевищення сумарного часу виконання спринту.

4. Будь-який ticket, якщо досягнуто межі кількості ticket'ів у спринті.

Опис методів спринта

• addUserStory(UserStory userStory) - приймає userStory, якщо вона не дорівнює null, і не завершена. Повертає true, якщо історія користувача прийнята, у противному випадку false.

• addBug(Bug bugReport) - приймає bug, якщо він не дорівнює null і не завершений. Повертає true, якщо баг прийнято, інакше false.

• getTickets() - повертає захищену копію масиву ticket'ів на спринт. Переконайтеся, що ticket'и розташовані в тому ж порядку, як вони були прийняті в спринт.

• getTotalEstimate() - повертає суму оцінок часу виконання всіх ticket'ів, прийнятих на спринт.

**Хід роботи**

Для виконання цієї роботи мною було створено 4 класи Ticket, UserStory, Bug, Sprint, а також клас Main із головною процедурою main.

Клас Ticket (завдання) – батьківський клас для UserStory та Bug. Він містить спільні з дочірніми класами поля (id, name, estimate, completed). Його методи дозволяють отримати інформацію про тікет і змінити його статус. Ticket самостійно не може бути створений (abstract), бо існує лише у вигляді підкласів UserStory та Bug.

Поля класу Ticket:

id – унікальний ідентифікатор тікета. Поле має тип final, отже його не можна змінити після ініціалізації.

name – назва тікета (UserStory або Bug). Також типу final.

estimate – оцінка часу виконання (в годинах), тип final, тобто задається один раз при створенні.

completed – статус виконання (true – завершено, false – незавершено). Початково false.

При створенні нового об’єкта Ticket (завдання) конструктор ініціалізує його id, name та estimate, які більше не змінюються. Змінна completed завжди початково false, тобто щойно створений тікет вважається незавершеним.

getId() – повертає унікальний ідентифікатор тікета.

getName() – повертає назву тікета.

getEstimate() – повертає оцінку часу виконання.

isCompleted() – повертає true, якщо тікет завершений, і false, якщо ні.

complete() – Метод позначає тікет як завершений (completed = true). Для UserStory цей метод перевизначається (@Override), оскільки історія користувача може мати залежності, які потрібно виконати перед завершенням.

Код класу Ticket:

package task\_1;

abstract class Ticket {  
 private final int id;  
 private final String name;  
 private final 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() {  
 this.completed = true;  
 }  
}

Клас UserStory розширює клас Ticket та представляє історію користувача в рамках спринту. Історією користувача називають вимоги/побажання користувачів до інтерфейсу програмного продукту. Основна особливість UserStory – вона може мати залежності у вигляді інших UserStory, які потрібно завершити першими, щоб ця історія користувача могла бути виконана.

Клас UserStory наслідує клас Ticket, (про це вказує ключове слово extends при оголошенні класу) тому він має всі його поля (id, name, estimate, completed) і методи (getId(), getName(), getEstimate(), isCompleted(), complete()).

dependencies (залежності) — це список історій користувачів (UserStory), які повинні бути завершені перед виконанням цієї конкретної історії. Поле dependencies оголошено як final, що означає, що посилання на список не зміниться після створення об'єкта.

Для роботи із списком залежностей перед оголошенням класу UserStory імпортуються наступні бібліотеки:

java.util.ArrayList - масив-список List зі змінним розміром масиву. Реалізує всі можливі операції зі списком і дозволяє всі елементи, включаючи null.

java.util.Collections - цей клас складається виключно зі статичних методів, які працюють із колекціями або повертають їх.

Java.util.List - упорядкована колекція, де користувач має точний контроль над тим, куди в списку вставляти кожен елемент. Користувач може отримувати доступ до елементів за їх цілим індексом (позицією в списку), а також шукати елементи в списку.

На початку викликається конструктор батьківського класу Ticket через super(id, name, estimate), передаючи унікальний ідентифікатор, назву та оцінку часу виконання.

Якщо список dependencies передано в конструктор не порожній, створюється його копія (new ArrayList<>(dependencies)). Якщо список dependencies == null (порожній), то створюється порожній список (new ArrayList<>()). Це робиться для того, щоб уникнути можливості зміни залежностей ззовні.

Метод complete() перевизначає (@Override) метод complete() з класу Ticket. Перед тим як відмітити UserStory як завершену, метод перевіряє всі її залежності. Якщо хоча б одна залежність (dependency) ще не завершена (!dependency.isCompleted()), метод просто виходить (return). Якщо всі залежності завершені, викликається super.complete(), що змінює статус completed на true, тобто робить UserStory завершеною.

Метод getDependencies() повертає список історій користувачів List<UserStory> у вигляді Collections.unmodifiableList(). Метод unmodifiableList() з імпортованого класу Collections робить сисок List<UserStory> захищеним від змін ззовні. Це запобігає можливості випадково або навмисно змінити список залежностей після створення об'єкта.

Метод toString() повертає текстове представлення UserStory, де:

getId() — ідентифікатор історії.

getName() — назва історії.

Текстове представлення UserStory має такий вид:

[UserStory 1] Реєстрація користувачів

Код класу UserStory:

package task\_1;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
class UserStory extends Ticket {  
 private final List<UserStory> dependencies;  
  
 public UserStory(int id, String name, int estimate, List<UserStory> dependencies) {  
 super(id, name, estimate);  
 this.dependencies = dependencies != null ? new ArrayList<>(dependencies) : new ArrayList<>();  
 }  
  
 @Override  
 public void complete() {  
 for (UserStory dependency : dependencies) {  
 if (!dependency.isCompleted()) {  
 return;  
 }  
 }  
 super.complete();  
 }  
  
 public List<UserStory> getDependencies() {  
 return Collections.*unmodifiableList*(dependencies);  
 }  
  
 @Override  
 public String toString() {  
 return "[UserStory " + getId() + "] " + getName();  
 }  
}

Клас Bug є підкласом Ticket і представляє помилку (баг), яка відноситься до конкретної історії користувача (UserStory). Основна особливість Bug полягає в тому, що він може бути створений лише для завершеної UserStory.

Bug теж успадковує Ticket, тому він має всі його поля (id, name, estimate, completed) і методи (getId(), getName(), getEstimate(), isCompleted(), complete()).

relatedUserStory — це об'єкт UserStory, до якого прив'язаний цей баг. Це поле має тип final, що означає, що воно має бути визначене під час створення об'єкта і не може змінюватися після цього.

Конструктор Bug() – приватний (private), що означає, що екземпляри Bug не можуть бути створені безпосередньо через new Bug(). Викликається конструктор батьківського класу Ticket, передаючи йому id, name і estimate. Поле relatedUserStory ініціалізується переданим об'єктом UserStory.

Метод createBug() використовується для створення об'єкта Bug, але з певними перевірками. Якщо userStory == null, тобто не вказана відповідна історія користувача, повертається null. Якщо userStory ще не завершена (!userStory.isCompleted()), повертається null. Якщо всі умови дотримані, створюється новий Bug за допомогою приватного конструктора і повертається в процедуру, яка його викликала. Це гарантує, що баги можуть бути створені лише для завершених історій користувача.

Метод toString() повертає текстове представлення об'єкта Bug у наступному вигляді (окрім ідентифікатора багу виводиться також ім’я пов’язаної з багом історії користувача, і, накінець, назва самого багу):

[Bug 3] Авторизація : Помилка входу

Код класу Bug:

package task\_1;  
  
class Bug extends Ticket {  
 private final UserStory relatedUserStory;  
  
 private Bug(int id, String name, int estimate, UserStory relatedUserStory) {  
 super(id, name, estimate);  
 this.relatedUserStory = relatedUserStory;  
 }  
  
 public static Bug createBug(int id, String name, int estimate, UserStory userStory) {  
 if (userStory == null || !userStory.isCompleted()) {  
 return null;  
 }  
 return new Bug(id, name, estimate, userStory);  
 }  
  
 @Override  
 public String toString() {  
 return "[Bug " + getId() + "] " + relatedUserStory.getName() + ": " + getName();  
 }  
}

Клас Sprint використовується для організації спринтів (коротких завершених проміжних етапів розробки програмного продукту) у методології SCRUM. Він містить список завдань (tickets), які можна додати до спринту за певних умов, і які повинні бути виконані протягом цього спринту.

Перед описом класу Sprint імпортуються бібліотеки Java для роботи зі списками завдань, коротко описані вище в описі класу UserStory.

timeCapacity – максимальна часова тривалість спринту (у годинах).

ticketLimit – максимальна кількість завдань, які можна додати в спринт.

tickets – список завдань (List<Ticket>), що містить усі додані тікети (завдання).

totalEstimate – загальний час виконання всіх завдань у спринті.

Конструктор Sprint() ініціалізує timeCapacity, ticketLimit, створює порожній список tickets.

Метод getTotalEstimate() початково дорівнює 0, оскільки спринт ще не містить завдань.

Метод addUserStory() перевіряє, чи можна додати UserStory у спринт через виклик методу addTicket().

Метод addBug() працює аналогічно addUserStory(), але для Bug.

Метод addTicket() додає нове завдання до списку до списку тікетів (tickets), при виконанні таких умов: поточне завдання ticket не пусте і не завершене; кількість завдань в списку tickets.size() не перевищує встановлений ліміт ticketLimit;

Сумарний час на виконання існуючих в списку завдань getTotalEstimate() не перевищує ліміт часу timeCapacity, при додаванні до сумарного часу getTotalEstimate() часу на вионання поточного тікета (завдання) ticket.getEstimate().

Метод getTickets() повертає копію списку tickets. Щоб уникнути зміни внутрішніх даних об'єкта Sprint, застосовується метод – Collections.unmodifiableList(tickets).

Метод getTotalEstimate() повертає значення загального часу виконання спринта: totalEstimate. В цьому методі спочатку ініціалізуються цілі змінні totalEstimate=0 та лічильник циклу і=0. Потім в циклі перебираються всі завдання спринта (об’єкти ticket класу Ticket із списку tickets). Для кожного і-го завдання ticket знаходиться значення часу на його виконання через звернення до цього завдання tickets.get(i) та виклик методу getEstimate(), який повертає час виконання тікета. При кожному проході циклу лічильник збільшується на 1 (і++), а сумарний час на виконання всього спринта totalEstimate збільшується на час виконання поточного завдання. Після завершення циклу повертається значення totalEstimate.

Код класу Sprint:

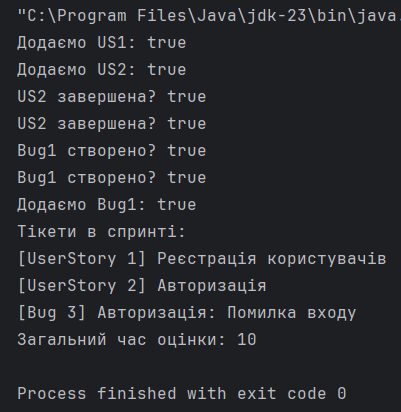
package task\_1;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
class Sprint {  
 private final int timeCapacity;  
 private final int ticketLimit;  
 private final List<Ticket> tickets;  
  
 public Sprint(int timeCapacity, int ticketLimit) {  
 this.timeCapacity = timeCapacity;  
 this.ticketLimit = ticketLimit;  
 this.tickets = new ArrayList<>();  
 }  
  
 public boolean addUserStory(UserStory userStory) {  
 return addTicket(userStory);  
 }  
  
 public boolean addBug(Bug bugReport) {  
 return addTicket(bugReport);  
 }  
  
 private boolean addTicket(Ticket ticket) {  
 if (ticket == null || ticket.isCompleted()) {  
 return false;  
 }  
 if (tickets.size() >= ticketLimit || getTotalEstimate() + ticket.getEstimate() > timeCapacity) {  
 return false;  
 }  
 tickets.add(ticket);  
 return true;  
 }  
  
 public List<Ticket> getTickets() {  
 return Collections.*unmodifiableList*(tickets);  
 }  
  
 public int getTotalEstimate() {  
 int totalEstimate = 0;  
 int i=0;  
 for (Ticket ticket : tickets) {  
 totalEstimate = totalEstimate + tickets.get(i).getEstimate();  
 i++;  
 }  
 return totalEstimate;  
 }

}

Результат виконання написаної програми буде залежати від конкретного сценарію використання класів. Для перевірки роботи програми я створив головний клас Main який містить головну функцію main, з якої викликаютсья всі інші класи та їх об’єкти й методи.

package task\_1;  
import java.util.List;  
  
public class Main {  
 public static void main(String[] args){  
 // 1. Створюємо UserStory без залежностей  
 UserStory us1 = new UserStory(1, "Реєстрація користувачів", 5, null);  
 UserStory us2 = new UserStory(2, "Авторизація", 3, List.*of*(us1));  
  
 // 2. Створюємо Sprint із обмеженням 10 годин і максимум 3 завдання  
 Sprint sprint = new Sprint(10, 3);  
  
 // 3. Додаємо UserStory в спринт  
 System.*out*.println("Додаємо UserStory1: " + sprint.addUserStory(us1)); // true  
 System.*out*.println("Додаємо UserStory2: " + sprint.addUserStory(us2)); // true  
  
 // 4. Завершуємо UserStory 1  
 us1.complete();  
  
 // 5. Пробуємо завершити UserStory 2 (не вийде, бо залежить від US1)  
 us2.complete();  
 System.*out*.println("UserStory2 завершена? " + us2.isCompleted()); // false  
  
 // 6. Завершуємо залежність (повторно), тепер US2 можна завершити  
 us2.complete();  
 System.*out*.println("UserStory2 завершена? " + us2.isCompleted()); // true  
  
 // 7. Пробуємо створити Bug до US2 (яка ще не завершена)  
 Bug bug1 = Bug.*createBug*(3, "Помилка входу", 2, us2);  
 System.*out*.println("Bug1 створено? " + (bug1 != null)); // false  
  
 // 8. Завершуємо US2 і створюємо баг  
 us2.complete();  
 bug1 = Bug.*createBug*(3, "Помилка входу", 2, us2);  
 System.*out*.println("Bug1 створено? " + (bug1 != null)); // true  
  
 // 9. Додаємо баг до спринту  
 System.*out*.println("Додаємо Bug1: " + sprint.addBug(bug1)); // true  
  
 // 10. Виводимо всі завдання в спринті  
 System.*out*.println("Тікети в спринті:");  
 for (Ticket t : sprint.getTickets()) {  
 System.*out*.println(t);  
 }  
  
 // 11. Загальний час виконання спринта  
 System.*out*.println("Загальний час виконання спринта: " + sprint.getTotalEstimate());  
 }  
}

Результат виконання головного модуля програми Main:



Спочатку створюються UserStory:

US1 ("Реєстрація користувачів") не має залежностей.

US2 ("Авторизація") залежить від US1.

Потім створюється Sprint з обмеженням часу = 10годин на 3 тікети (завдання)

Далі додається UserStory до спринту. Обидві історії US1 та US2 успішно додаються.

Далі програма здійснює спробу завершити UserStory. US1 завершується без проблем. А от US2 не завершується, бо US1 ще не відмічена як завершена. Після повторного виклику complete() для US2 вона завершується.

Далі програма здійснює спробу створити баг. Bug1 не створюється, бо US2 ще не завершена. Після завершення US2 Bug1 успішно створюється.

Далі програма пробує додати Bug1 у спринт.

Далі виводяться тікети у спринті:

[UserStory 1] Реєстрація користувачів

[UserStory 2] Авторизація

[Bug 3] Авторизація: Помилка входу

В результаті отримується загальний час оцінки: 5 + 3 + 2 = 10.

**Висновки.** Під час виконання цієї лабораторної роботи я навчився використовувати такі базові концепції ООП як інкапсуляція, наслідування та поліморфізм. Навчився працювати із залежностями між об'єктами, використовувати статичні методи для створення об'єктів за умовами, реалізовувати перевірки вхідних даних та обмеження у класах ть організовувати модель SCRUM для розробки програмного забезпечення.

Посилання на GitHub репозиторій:

<https://github.com/diachenkom/OOP-KB-231-Diachenko-Maksym>