# Лабораторна робота №3

**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 до лабораторної роботи

Для виконання лабораторної роботи необхідно використовувати репозиторій на github, що був створений при виконанні завдання до першої лабораторної роботи. Результати роботи необхідно буде завантажити на github для перевірки.

При використанні IDE створити проект під назвою lab03 для розміщення результатів виконання завдань. У випадку використання звичайного текстового редактору створити окрему директорію lab03 для виконання завдань.

Виконайте наступні завдання(кожне завдання має бути виконано в окремому файлі):

|  |  |
| --- | --- |
| 1 | У цьому завданні ви формулюватимете бізнес-логіку планування спринту. [Спринт](https://en.wikipedia.org/wiki/Scrum_(software_development)" \l "Sprint) - це основна одиниця розробки програмного забезпечення [SCRUM](https://en.wikipedia.org/wiki/Scrum_(software_development)). 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 "[US 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 та ім'я відповідної UserStory. Приклад: з ідентифікатором = 2, ім'ям бага = "Додати пароль, що повторюється" і ім'ям відповідної UserStory = "Форма реєстрації" в результаті отримаємо "[Bug 2] Форма реєстрації: Додати повторюваний пароль".   **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.  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 переповнення тимчасової ємності 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. |
|  |  |

Сформуйте звіт який містить код програм та результати тестування.

Надайте посилання на github для перевірки.

Код:

package lab3;  
  
import java.util.ArrayList;  
import java.util.List;  
  
// Базовий клас Ticket  
abstract class Ticket {  
 private final int id;  
 private final String name;  
 private final int estimate;  
 private boolean completed = false;  
  
 public Ticket(int id, String name, int estimate) {  
 this.id = id;  
 this.name = name;  
 this.estimate = estimate;  
 }  
  
 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;  
 }  
}  
  
// Клас UserStory  
class UserStory extends Ticket {  
 private final UserStory[] dependencies;  
  
 public UserStory(int id, String name, int estimate, UserStory[] dependencies) {  
 super(id, name, estimate);  
 this.dependencies = dependencies != null ? dependencies.clone() : new UserStory[0];  
 }  
  
 @Override  
 public void complete() {  
 for (UserStory dependency : dependencies) {  
 if (!dependency.isCompleted()) {  
 return; // Не завершувати, якщо залежність не виконана  
 }  
 }  
 super.complete();  
 }  
  
 public UserStory[] getDependencies() {  
 return dependencies.clone();  
 }  
  
 @Override  
 public String toString() {  
 return "[US " + getId() + "] " + getName();  
 }  
}  
  
// Клас Bug  
class Bug extends Ticket {  
 private final UserStory relatedStory;  
  
 private Bug(int id, String name, int estimate, UserStory relatedStory) {  
 super(id, name, estimate);  
 this.relatedStory = relatedStory;  
 }  
  
 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() + "] " + relatedStory.getName() + ": " + getName();  
 }  
}  
  
// Клас Sprint  
class Sprint {  
 private final int timeCapacity;  
 private final int ticketLimit;  
 private final List<Ticket> tickets = new ArrayList<>();  
  
 public Sprint(int timeCapacity, int ticketLimit) {  
 this.timeCapacity = timeCapacity;  
 this.ticketLimit = ticketLimit;  
 }  
  
 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) {  
 return false;  
 }  
 if (getTotalEstimate() + ticket.getEstimate() > timeCapacity) {  
 return false;  
 }  
 tickets.add(ticket);  
 return true;  
 }  
  
 public Ticket[] getTickets() {  
 return tickets.toArray(new Ticket[0]);  
 }  
  
 public int getTotalEstimate() {  
 int total = 0;  
 for (Ticket ticket : tickets) {  
 total += ticket.getEstimate();  
 }  
 return total;  
 }  
}  
  
// Головний клас Main  
public class Main {  
 public static void main(String[] args) {  
 // Створення UserStory без залежностей  
 UserStory us1 = new UserStory(1, "Реєстрація користувача", 5, new UserStory[0]);  
 us1.complete();  
  
 // Перевірка залежностей (метод використано)  
 UserStory[] dependencies = us1.getDependencies();  
 System.out.println("Кількість залежностей: " + dependencies.length);  
  
 // Створення Bug на основі завершеної UserStory  
 Bug bug1 = Bug.createBug(2, "Виправити перевірку пароля", 3, us1);  
  
 // Створення Sprint  
 Sprint sprint = new Sprint(10, 3);  
  
 // Додаємо задачі в Sprint з перевіркою  
 if (sprint.addUserStory(us1)) {  
 System.out.println("UserStory додано до спринту.");  
 } else {  
 System.out.println("UserStory НЕ додано до спринту.");  
 }  
  
 if (bug1 != null && sprint.addBug(bug1)) {  
 System.out.println("Bug додано до спринту.");  
 } else {  
 System.out.println("Bug НЕ додано до спринту.");  
 }  
  
 // Виводимо задачі у спринті  
 System.out.println("Тікети у спринті:");  
 for (Ticket ticket : sprint.getTickets()) {  
 System.out.println(ticket);  
 }  
  
 // Виводимо загальну оцінку  
 System.out.println("Загальна оцінка часу: " + sprint.getTotalEstimate());  
 }  
}

1. **Клас Ticket (абстрактний)**

Це **базовий клас** для всіх задач.

Має поля: id, name, estimate (час виконання) і completed (чи завершено).

Методи:

getId(), getName(), getEstimate() — гетери.

isCompleted() — перевірка статусу.

complete() — позначає задачу як завершену.

2. **Клас UserStory extends Ticket**

У UserStory можуть бути **залежності** від інших історій (dependencies).

В complete() — задача завершується **лише тоді**, коли всі залежні UserStory завершені.

Методи:

getDependencies() — повертає копію залежностей.

toString() — повертає рядок типу [US 1] Назва.

3. **Клас Bug extends Ticket**

Bug має бути прив’язаний до **вже завершеної** UserStory.

Створюється тільки через createBug(...), який повертає null, якщо UserStory не завершена або null.

Метод:

toString() — формат: [Bug 2] НазваUserStory: Назва бага.

4. **Клас Sprint**

Sprint має **обмеження**:

timeCapacity — максимальний час (у годинах або умовних одиницях),

ticketLimit — максимальна кількість задач.

Методи:

addUserStory(...) / addBug(...) — додає задачі в спринт.

Валідація:

не null,

не завершена,

* + не перевищує ліміт часу чи кількості.

getTickets() — список задач.

getTotalEstimate() — сума оцінок усіх задач.

5. **Головний клас Main (метод main)**

Створюється UserStory і одразу завершується (бо немає залежностей).

Створюється Bug, прив’язаний до завершеної UserStory.

Створюється спринт із обмеженнями.

Додаються задачі у спринт (якщо проходять перевірки).

Виводяться всі задачі, додані в спринт.

Обчислюється та виводиться загальний час задач у спринті.

