**Лабораторна робота №3**

**Тема**: 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**

**Завдання 1**

**Умова:**

|  |  |
| --- | --- |
| 1 | У цьому завданні ви формулюватимете бізнес-логіку планування спринту. [Спринт](https://en.wikipedia.org/wiki/Scrum_(software_development)#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. |
|  |  |

**Виконання:** Клас Ticket є базовим для всіх типів задач у системі. Він зберігає унікальний ідентифікатор, назву задачі, оцінку часу виконання та статус завершення. Метод complete() позначає задачу як завершену. Цей клас слугує основою для більш спеціалізованих типів задач.

Клас UserStory успадковує від Ticket і додає функціонал для роботи з залежностями. Кожна історія користувача містить список інших UserStory, які мають бути завершені перед її виконанням. Метод complete() перевіряє, чи всі залежності вже завершені, і лише тоді позначає задачу як виконану. Список залежностей повертається через getDependencies() у захищеному від змін вигляді. При виведенні об’єкта використовується формат [US {id}] {назва}.

Клас Bug також наслідує Ticket, але представляє баги, пов’язані з завершеними історіями користувачів. Створюється через статичний метод createBug(), який перевіряє, чи відповідна UserStory існує та завершена. Якщо ні — метод повертає null. При виведенні баг відображається у форматі [Bug {id}] {назва UserStory}: {назва бага}.

Клас Sprint керує додаванням задач до спринту з урахуванням обмежень: максимальної часової ємності (timeCapacity) та ліміту кількості задач (ticketLimit). Методи addUserStory() та addBug() перевіряють, чи задача не є null, чи не завершена, чи не перевищує ліміти спринту. Якщо всі умови виконані — задача додається до списку. Список усіх задач повертається через getTickets() у незмінному вигляді, а сумарна оцінка часу — через getTotalEstimate().

Клас Main демонструє роботу системи: створює історії користувачів із залежностями, додає їх до спринту разом із багами, а потім виводить загальний час та список задач.

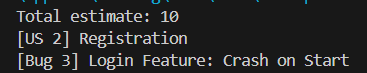
**Взаємодія між класами:**

* UserStory та Bug базуються на Ticket, використовуючи його поля та методи.
* Sprint контролює додавання задач, перевіряючи їхню коректність та обмеження спринту.
* Завершення UserStory можливе лише після виконання всіх її залежностей, що забезпечує логічну послідовність робіт.

**Текст програми**:

|  |
| --- |
| public class Ticket {      private static int counter = 1;      private final int id;      private String name;      private int estimate;      private boolean isCompleted;      public Ticket(String name, int estimate) {          this.id = counter++;          this.name = name;          this.estimate = estimate;          this.isCompleted = false;      }        public int getId() { return id; }      public String getName() { return name; }      public int getEstimate() { return estimate; }      public boolean isCompleted() { return isCompleted; }      public void complete() {          this.isCompleted = true;      }      @Override      public String toString() {          return "Ticket{" +                  "id=" + id +                  ", name='" + name + '\'' +                  ", estimate=" + estimate +                  ", isCompleted=" + isCompleted +                  '}';      }  }  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class UserStory extends Ticket {      private List<UserStory> dependencies;      public UserStory(String name, int estimate, List<UserStory> dependencies) {          super(name, estimate);          this.dependencies = new ArrayList<>(dependencies != null ? dependencies : Collections.emptyList());      }      @Override      public void complete() {          for (UserStory dependency : dependencies) {              if (!dependency.isCompleted()) return;          }          super.complete();      }      public List<UserStory> getDependencies() {          return Collections.unmodifiableList(new ArrayList<>(dependencies));      }      @Override      public String toString() {          return "[US " + getId() + "] " + getName();      }  }  public class Bug extends Ticket {      private UserStory userStory;      private Bug(String name, int estimate, UserStory userStory) {          super(name, estimate);          this.userStory = userStory;      }      public static Bug createBug(int id, String name, int estimate, UserStory userStory) {          if (userStory == null || !userStory.isCompleted()) return null;          return new Bug(name, estimate, userStory);      }      @Override      public String toString() {          return "[Bug " + getId() + "] " + userStory.getName() + ": " + getName();      }  }  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class Sprint {      private final int timeCapacity;      private final int ticketLimit;      private final List<Ticket> tickets = new ArrayList<>();      private int totalEstimate = 0;      public Sprint(int timeCapacity, int ticketLimit) {          this.timeCapacity = timeCapacity;          this.ticketLimit = ticketLimit;      }      public boolean addUserStory(UserStory userStory) {          if (userStory == null || userStory.isCompleted()) return false;          if (tickets.size() >= ticketLimit) return false;          if (totalEstimate + userStory.getEstimate() > timeCapacity) return false;          tickets.add(userStory);          totalEstimate += userStory.getEstimate();          return true;      }      public boolean addBug(Bug bug) {          if (bug == null || bug.isCompleted()) return false;          if (tickets.size() >= ticketLimit) return false;          if (totalEstimate + bug.getEstimate() > timeCapacity) return false;          tickets.add(bug);          totalEstimate += bug.getEstimate();          return true;      }      public List<Ticket> getTickets() {          return Collections.unmodifiableList(new ArrayList<>(tickets));      }      public int getTotalEstimate() {          return totalEstimate;      }  }  import java.util.Arrays;  public class Main {      public static void main(String[] args) {          Sprint sprint = new Sprint(100, 10);            UserStory story1 = new UserStory("Login Feature", 5, null);          UserStory story2 = new UserStory(                  "Registration",                  8,                  Arrays.asList(story1)          );          story1.complete();          Bug bug = Bug.createBug(2, "Crash on Start", 2, story1);          sprint.addUserStory(story2);          sprint.addBug(bug);          System.out.println("Total estimate: " + sprint.getTotalEstimate());          sprint.getTickets().forEach(System.out::println);      }  } |

**Результат виконання:**

****

**Висновок:** під час виконання роботи я використовуючи теоретичне підґрунтя про базові принципи ООП та підходи використання в Java виконав дії що будуть вказано в завданні до лабораторної роботи.