**Техническое задание**

Продолжим работать над трекером задач. Вам предстоит добавить в него новую функциональность и провести рефакторинг написанного кода с учётом изученных принципов ООП.

В этом спринте вы изучили методы тестирования и библиотеку JUnit, и теперь существующий код нужно покрыть тестами, а для нового кода, который вам ещё предстоит добавить в проект, мы рекомендуем писать тесты сразу. Некоторые особенно важные для тестирования кейсы мы опишем ниже.

**Обобщаем класс «Менеджер»**

При проектировании классов и их взаимодействия бывает полезно разделить описание функций класса на интерфейс и реализацию. Из темы об абстракции и полиморфизме вы узнали, что такое интерфейсы и как их использовать для выделения значимой функциональности классов (абстрагирования). При таком подходе набор методов, который должен быть у объекта, лучше вынести в интерфейс, а реализацию этих методов — в класс, который его реализует. Теперь нужно применить этот принцип к менеджеру задач.

Класс TaskManager станет интерфейсом. В нём нужно собрать список методов, которые должны быть у любого объекта-менеджера. Вспомогательные методы, если вы их создавали, переносить в интерфейс не нужно.

Созданный ранее класс менеджера нужно переименовать в InMemoryTaskManager. Именно то, что менеджер хранит всю информацию в оперативной памяти, и есть его главное свойство, позволяющее эффективно управлять задачами. Внутри класса должна остаться реализация методов. При этом важно не забыть имплементировать TaskManager, ведь в Java класс должен явно заявить, что он подходит под требования интерфейса.

**История просмотров задач**

Добавьте в программу новую функциональность — нужно, чтобы трекер отображал последние просмотренные пользователем задачи. Для этого добавьте метод getHistory в TaskManager и реализуйте его — он должен возвращать последние 10 просмотренных задач. Просмотром будем считать вызов тех методов, которые получают задачу по идентификатору, — getTask(int id), getSubtask(int id) и getEpic(int id). От повторных просмотров избавляться не нужно.

Пример формирования истории просмотров задач после вызовов методов менеджера:

У метода getHistory не будет параметров. Это значит, что он формирует свой ответ, анализируя исключительно внутреннее состояние полей объекта менеджера. Подумайте, каким образом и какие данные вы запишете в поля менеджера для возможности извлекать из них историю посещений. Так как в истории отображается, к каким задачам было обращение в методах getTask, getSubtask и getEpic, эти данные в полях менеджера будут обновляться при вызове этих трёх методов.

Обратите внимание, что просмотрен может быть любой тип задачи. То есть возвращаемый список задач может содержать объект одного из трёх типов на любой своей позиции. Чтобы описать ячейку такого списка, нужно вспомнить о полиморфизме и выбрать тип, являющийся общим родителем обоих классов.

**Утилитарный класс**

Со временем в приложении трекера появится несколько реализаций интерфейса TaskManager. Чтобы не зависеть от реализации, создайте утилитарный класс Managers. На нём будет лежать вся ответственность за создание менеджера задач. То есть Managers должен сам подбирать нужную реализацию TaskManager и возвращать объект правильного типа.

У Managers будет метод getDefault. При этом вызывающему неизвестен конкретный класс — только то, что объект, который возвращает getDefault, реализует интерфейс TaskManager.

**Сценарий для проверки**

Начиная с этого урока обязательными для проверки кода становятся юнит-тесты. Сценарии, представленные в этом разделе ТЗ, опциональны. Вы можете реализовать их для более простой демонстрации возможностей программы.

Например, в главном классе можно реализовать такой несложный сценарий:

Создать несколько задач разного типа.

Вызвать разные методы интерфейса TaskManager и напечатать историю просмотров после каждого вызова. Если код рабочий, то история просмотров задач будет отображаться корректно.

Вы можете использовать этот код (с небольшими модификациями конкретно под ваш класс TaskManager).

private static void printAllTasks(TaskManager manager) {

System.out.println("Задачи:");

for (Task task : manager.getTasks()) {

System.out.println(task);

}

System.out.println("Эпики:");

for (Task epic : manager.getEpics()) {

System.out.println(epic);

for (Task task : manager.getEpicSubtasks(epic.getId())) {

System.out.println("--> " + task);

}

}

System.out.println("Подзадачи:");

for (Task subtask : manager.getSubtasks()) {

System.out.println(subtask);

}

System.out.println("История:");

for (Task task : manager.getHistory()) {

System.out.println(task);

}

}

**Сделайте историю задач интерфейсом**

В этом спринте возможности трекера ограничены — в истории просмотров допускается дублирование, и она может содержать только десять задач. В следующем спринте вам нужно будет убрать дубли и расширить её размер. Чтобы подготовиться к этому, проведите рефакторинг кода.

Создайте отдельный интерфейс для управления историей просмотров — HistoryManager. У него будет два метода: add(Task task) должен помечать задачи как просмотренные, а getHistory — возвращать их список.

Объявите класс InMemoryHistoryManager и перенесите в него часть кода для работы с историей из класса InMemoryTaskManager. Новый класс InMemoryHistoryManager должен реализовывать интерфейс HistoryManager.

Добавьте в служебный класс Managers статический метод HistoryManager getDefaultHistory. Он должен возвращать объект InMemoryHistoryManager — историю просмотров.

Проверьте, что теперь InMemoryTaskManager обращается к менеджеру истории через интерфейс HistoryManager и использует реализацию, которую возвращает метод getDefaultHistory.

Ещё раз всё протестируйте!

**Покрываем проект тестами**

Вы прошли тему о юнит-тестировании и теперь можете перейти к полноценной автоматизированной проверке вашего кода. Вам нужно протестировать все созданные классы проекта.

Для этого вам потребуется:

Набор библиотек тестирования. Здесь вам поможет инструкция ниже и второй урок темы «Unit-тесты», а точнее — пошаговая инструкция по добавлению в проект библиотек JUnit.

Анализ технического задания. На основе этого и предыдущего задания (финального проекта четвёртого спринта) сформулируйте ключевые моменты работы классов Manager и Task и основные требования к их функциональности.

Организация кода. Для каждого класса создайте соответствующий тест-класс в пакете внутри каталога test. Если его не подсвечивает IDEA, кликните правой кнопкой мыши и выберите пункт Mark Directory as → Test Sources.

**Не нужно покрывать тестами весь код. Даже для такого небольшого проекта тестирование может занять значительное время. Но всё же обратите внимание на некоторые нюансы, которые необходимо проверить:**

проверьте, что экземпляры класса Task равны друг другу, если равен их id;

проверьте, что наследники класса Task равны друг другу, если равен их id;

проверьте, что объект Epic нельзя добавить в самого себя в виде подзадачи;

проверьте, что объект Subtask нельзя сделать своим же эпиком;

убедитесь, что утилитарный класс всегда возвращает проинициализированные и готовые к работе экземпляры менеджеров;

проверьте, что InMemoryTaskManager действительно добавляет задачи разного типа и может найти их по id;

проверьте, что задачи с заданным id и сгенерированным id не конфликтуют внутри менеджера;

создайте тест, в котором проверяется неизменность задачи (по всем полям) при добавлении задачи в менеджер

убедитесь, что задачи, добавляемые в HistoryManager, сохраняют предыдущую версию задачи и её данных.

А вообще — смело тестируйте самостоятельно всё, что посчитаете нужным. Но не забывайте о том, что нужно покрыть тестами основные методы менеджера и самих задач. Тестов много не бывает!