Задания по курсу объектно-ориентированного программирования (ООП)

Для выполнения заданий необходимо создать аккаунт на публичном интернет-ресурсе github.com и репозиторий «ООР».

Репозиторий должен быть доступен для чтения лектору(alexander-a-vlasov) и преподавателю, ведущему семинарские занятия.

Задания выполняются на языке Java в программном средстве IntelliJ IDEA версии Community.

Все решения должны сопровождаться набором тестов, проверяющих корректность выполненного задания.

Из каждой группы заданий выполняем только одно.

Для выполнения первого задания необходимо создать проект Task_2_1_1. Следующие задания именовать соответствующим образом.

Зеленым цветом обозначены примеры к заданиям, которые можно использовать в качестве первого теста.

Серым цветом обозначены дополнительные требования к задаче, которые не обязательны к реализации, но приносят дополнительные баллы

Желтым цветом отмечены комментарии к задаче. Дополнительные условия и ограничения.

1. Многопоточные вычисления

1.1. Простые числа

У вас имеется массив целых чисел, необходимо определить есть ли в этом массиве хотя бы одно не простое (делится без остатка только на себя и единицу). Необходимо предоставить три решения задачи: последовательное, параллельное решением с применением класса Thread с возможностью задания количества используемых потоков и параллельным решением с применением ParallelStream.

После выполнения программной реализации, необходимо подготовить тестовый пример с набор простых чисел, который продемонстрирует ускорение вычислений за счет использования многоядерной архитектуры центрального процессора. Полученные времена выполнения программ на созданном тестовом примере необходимо нанести на график (1 точка — последовательное исполнение, n точек — параллельное

исполнение Thread для разного количества используемых ядер, 1 точка — параллельное исполнение ParallelStream).

Преподавателю семинарских занятий объяснить выбор тестового набора данных и полученный график, оценить долю времени последовательного исполнения программы.

вход:
{6,8,7,13,9,4}
выход:
True

вход:
6997901 6997927 6997937 6997967 6998009 6998029 6998039 6998051 6998053 выход:
False

1) Для тестирования необходимо использовать вычислительное устройство с 4 и более ядрами.
2) Полученный график вместе с тестовым набором данных должен быть добавлен в систему контроля версий

2. Автоматизация производства

2.1. Пиццерия

У вас в штате есть «N» пекарей, занимающихся производством пиццы, с разным опытом работы и «M» курьеров, занимающихся доставкой пиццы заказчику, один склад
готовой продукции размера «T». Производственный процесс выглядит

следующим образом:

- поступает заказ на пиццу в общую очередь;
- пекарь нажимает на кнопку и берет заказ в исполнение;
- когда пицца готова, пекарь нажимает на кнопку с готовностью пиццы и пытается зарезервировать место на складе. Если склад полностью заполнен, пекарь ожидает свободное место;
- пекарь передает пиццу на склад и может продолжить работу;
- курьер после очередного заказа обращается на склад и берет одну или несколько пицц в доставку, но не больше объема своего багажника. Если склад пуст ожидает появления готовых пицц;
- после доставки очередной пиццы курьер отмечает что заказ выполнен;
- при выполнении очередного действия, система выводит на стандартный вывод сообщение: [номер заказа], [состояние];
- параметры работников пиццерии загружаем из файла формата JSON.

Преподавателю семинарских занятий необходимо продемонстрировать класс диаграмму с созданными классами(интерфейсами) и отношениями между ними. Объяснить какие из принципов S.O.L.I.D. выполнены в предложенном решении.

Для составления диаграммы классов можно использовать любой редактор диаграмм UML(например https://www.visual-paradigm.com/). Диаграмма должна быть сохранена в системе контроля версий в формате png.

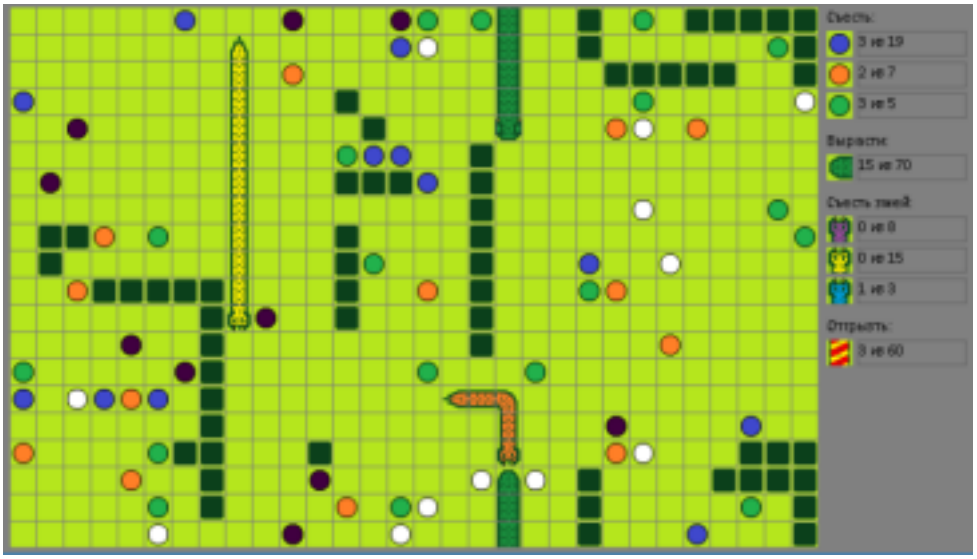
3. Графический пользовательский интерфейс

Необходимо реализовать в игре с использованием Java FX. Наличие тестов для классов, отвечающих за элементы графического пользовательского интерфейса, не обязательно.

3.1. Игра «Змейка»

- змейка (упорядоченный набор связанных звеньев с явно выделенными концами – головой и хвостом) передвигается по полю «N» x «M»;
- в начале игры змейка состоит из одного звена;
- перемещение змейки состоит в добавлении одного звена к ее голове в требуемом направлении (в направлении ее движения) и удалении одного звена хвоста; ● если при перемещении змейки ее голова натыкается на препятствие, то игра проиграна;
- в каждый момент времени на игровом поле находится «T» элементов еды, занимающей одну клетку поля;
- если при перемещении голова змейки натыкается на еду, то змейка ее «съедает» и вырастает на одно звено, а для выполнения предыдущего правила на поле в произвольном свободном месте автоматически появляется новая порция еды; ● выигрыш состоит в достижении змейкой длины в «L» звена.

Пример возможного графического интерфейса программы:



● на поле есть несколько таких же змеек управляемых программой, обладающих разной стратегией поведения;

● в случае, если змейка пользователя пересекает другую змейку, то хвост другой змейки «отгрызается». Если это голова, то змейка съедается полностью.

По согласованию с преподавателем правила игры могут изменяться. Какие изменения необходимо внести в программу, если:

мы хотим в будущем добавить уровни игры (на каждом следующем уровне змейка ускоряется);

изменить поведение «съедание еды», в зависимости от типа еды увеличивать размер змейки на различное количество элементов;

в будущем возможны другие условия победы?

4. Domain-Specific Languages (DSL) — Groovy

Предложить на основе Groovy язык для конфигурирования и автоматического запуска, сборки, проверки и генерации отчета учебных проектов студентов по курсу ООП.

Результатом должно стать консольное приложение, конфигурируемое с помощью разработанного Groovy DSL, и позволяющее выполнять следующие

команды: 1. Для каждой лабораторной работы:

a. запустить процесс компиляции и сгенерировать документацию;
b. проверка соответствия программного кода Google Java Style;
c. запустить тесты, определить количество успешно пройденных проваленных и не выполненных тестов;
d. вычислить суммарное количество полученных балов за решение задачи.

2. Согласно «критериям» рассчитать итоговое количество баллов и оценок на каждую контрольную точку и итоговую аттестацию.
3. Для группы сгенерировать отчет на стандартный вывод в формате HTML.

Конфигурация модели автоматической проверки заданий ООП состоит из четырех частей:

1. Список задач.
2. Группа, студенты в ней
3. план выполнения задач (какие задачи назначены каким студентам, со сроками их сдачи)
4. Общие настройки системы.

Объекты модели

1. Описание задачи. У этого объекта должны быть: идентификатор (уникальное имя), название и количество баллов.
2. Группа. У группы должно быть название.
3. Студент: ник (уникальное имя), ФИО, URL для доступа к репозиторию (имя ветки репозитория по умолчанию «master»).
4. Выданная задача: идентификатор, крайняя дата сдачи задачи.
5. Контрольная отметка: название, дата.
6. Занятия: дата.

Структура репозитория студентов соответствует учебным заданиям по курсу ООП.

Правила модели

1) Структура заданий соответствуют требованиям курса ООП.
2) Лабораторная работа готовая к сдаче если: успешно компилируется, генерируется документация, тесты выполнены или пропущены.
3) Часть конфигураций может не меняться в течение нескольких проведений курса (например, список задач), часть – быть актуальной только в течение семестра (состав группы), часть – меняться постоянно (информация о сданных работах, дополнительных баллах за работу, посещаемость). Должна быть возможность указывать разные по времени жизни конфигурации в разных файлах (импортировать более долгоживущие настройки в менее долгоживущие)
4) Все проекты хранятся в GIT-репозитории, в будущем возможно подключение других систем контроля версий. У пользователя могут отсутствовать права на доступ к заданному репозиторию.

5) Структура репозитория
    a. Для большинства репозиториев ветка master.
    b. Каждая папка в корне репозитория соответствует номеру задачи, в названии которой закодирован: номер семестра, номер группы, номер задачи. В большинстве репозиториев студентов правила кодирования названия директорий совпадает
6) Существуют длительные тесты, которые нужно принудительно отключить для некоторых лабораторных работ
7) Некоторые задачи могут не выдаваться студентам
8) Для каждой задачи существует срок, до которого нужно сдать задачу
9) Для группы студентов нужно составить отчет в формате HTML.
10) Существует N-контрольных точек, в которых выставляется текущая успеваемость в автоматическом режиме в зависимости от количества набранных балов, всегда должна существовать итоговая аттестация.
11) Посещаемость студентов определяется по наличию операций «commit» в течение заданной недели.
12) Итоговая оценка зависит от количества набранных баллов и посещаемости занятий.
13) В случае если задача сдана с опозданием, запускается алгоритм корректировки итоговых баллов в зависимости от количества дополнительных дней.
14) Управляющие действия для каждой группы:
    a. студент сдал задачу (идентификатор студента, дата, количество баллов, текстовое сообщение);
    b. студент получил дополнительные баллы за работу (дата, количество баллов может быть положительным и отрицательным, текстовое сообщение);
    c. принудительная отметка присутствия/отсутствия студента на занятии.

Замечания по реализации

1) Объектная модель предметной области и выполнение команд приложения должны быть написаны на Java; методы конфигурации этих объектов (DSL) должны быть написаны на Groovy (рекомендуется ознакомиться с разделами 1-5 документации по DSL на Groovy: http://docs.groovy-lang.org/docs/latest/html/documentation/core-domain-specific-languages.html)
2) Приложение должно работать по той же логике, что и Gradle: при поступлении команды ищет в рабочей директории скрипт на Groovy с предопределённым именем, считывает из него конфигурацию и выполняет соответствующую команду. Например:
    a. Для указанной в конфигурации группы скачивает репозитории всех студентов в группе в поддиректорию рабочей директории
    b. ищет решение указанной по id задачи в репозитории указанного по id студента, компилирует его, выполняет тесты и выдаёт информацию

либо об успешном прохождении тестов, либо количество и названия проваленных тестов

c. генерирует отчёт по указанной в конфигурации посещаемости и т.д.

3) Работа с репозиториями должна осуществляться через консольный клиент git; использовать API GitHub не нужно

a. Считается, что работа с git идёт от имени пользователя, указанного в git config --global; у этого пользователя может не быть доступа к указанным репозиториям студентов

b. Допускается работать с git, настроенным на отсутствие запросов аутентификации пользователя; в этом случае перед запуском необходимо проверять, что это действительно так

Пример вывода отчёта по группе:

Гр XXXXX

Успеваемость

	LAB_1_1_1 К-1 К-2				Total
	Build	Style	Doc	Tests Credit Add Total	
Студ. №1	+			6/1/ 2/0 -1 1 7/3 10/3	15/4
Студ. №2	-	+	+	5/0/0 3/0 3 6 10/4 15/5	19/5

Посещаемость

	7.04	14.04	21.04 К-1 К-2	Total
Студ. №1	Н	Н		
Студ. №2	Н	+		