# Задание X. Проектируем игру/систему

## Список дедлайнов

### Часть 1: 20 марта 2022 г. 20:59 MSK

### Часть 2: 10 апреля 2022 г. 20:59 MSK

### Часть 3: 2 мая 2022 г. 20:59 MSK

## Список кураторов

Кураторы - ваши помощники на пути реализации проекта. Вы можете спрашивать вопросы по архитектуре проекта, спрашивать об особенностях реализации проекта и договариваться о дополнениях в случае командной разработки!

|  |  |  |
| --- | --- | --- |
| Группа | Куратор | Github |
| 120 | Кондакова | linlit |

## Как сдавать задания

Задания необходимо отправлять на почту [mipt.atp.tp2022.projectsx@akht.pl](mailto:mipt.atp.tp2022.projectsx@akht.pl) (почты будут активны позже), где x=1,2,3 (в зависимости от задания) с названием:

Фамилия Имя, группа

### Пример:

Ахтямов Павел, 913

В тексте письма необходимо прикрепить ссылку на репозиторий (желательно прикреплять ссылку на GitHub) Ссылка должна начинаться с https:// , чтобы кураторы могли открыть ссылку на репозиторий.

Если проект состоит более чем из одного человека, то указываете в теле письма всех людей, которые работают над проектом.

При этом в коллабораторы необходимо добавить куратора из таблицы выше!

## Оформление рабочего процесса проекта

* В ветке main/master находится работающая версия кода, в нее будут сливать изменения кураторы
* Для разработки проекта создается ветка dev, в которой выполняется разработка
* После того, как вы считаете, что определенную часть задания выполнили, создаете отдельную ветку checkpoint\_x (x=1,2,3) из ветки dev, создаете pull request из ветки checkpoint\_x в ветку main/master, добавляете ревьюера на pull request
* Если ревьюер попросил сделать изменения, то вы делаете изменения в ветке checkpoint\_x и вливаете ее в dev, НЕ СЛИВАЯ при этом ветку в main/master
* В описании pull request-а пишете списком (добавляете в README.md) список изменений, который произошел в новой версии проекта (от одного чекпоинта к другому)

## Содержание заданий

### Часть 1

В целом все три задания завязаны на одну тематику: создание одного из проектов.

Первое задание предполагает проектирование и реализацию части игры, связанной с внутренней логикой процесса.

Вам необходимо сделать следующее:

1. Разработать логическую модель проекта. Это часть не обязательна, Вы можете просто взять модель из списка предлагаемых шаблонов
2. По игровой модели разработать архитектуру программной реализации логической части проекта. Необходимо предоставить полноценное описание архитектуры (можно как с использованием существующих методологий, так и в свободном стиле). Архитектурные решения должны быть обоснованы.
3. Реализовать данную часть на любимом языке программирования.

Что должно быть в репозитории:

1. Код
2. Файл с описанием архитектуры данной части системы. Должна быть диаграмма UML
3. README - как запускать проект, запуск должен быть простой  
   *p.s. cmake & make - лучшее решение :)  
   p.p.s. Для Python лучше сделать команду pip freeze > requirements.txt для совпадения версий всех используемых пакетов*

**Задание в целом (все части) можно выполнять в командах**

Но в этом случае увеличиваются требования. Конкретные требования будут зависеть от проекта и размера команды. Уточняйте детали у кураторов. Не забудьте предупредить вашего семинариста, если решили выполнять в группах.

**Можно взять проект из другого курса или какой-то другой свой личный проект**

В данном случае требования к использованию паттернов (или аргументированному отказу от них) остаются прежними, меняется только тема проекта.

Пример игры.

*В игре существуют две фракции с различными непересекающимися наборами юнитов. В начале игры игрок выбирает фракцию, и далее ему становятся при определенных условиях возможности создания юнитов. При старте игры создается конкретная фабрика(в зависимости от выбранной фракции), которая по id юнита создает требуемый объект. Все классы юнитов - наследники абстрактного класса Unit...*

### Часть 2

Ваша задача в этот раз – реализовать интерфейс к проекту.

По итогам первого задания мы научились реализовать внутреннюю логику проекта.

Теперь задача – реализовать взаимодействие с интерфейсом. Для этого необходимо сделать так, чтобы можно было спокойно заменить один из типов интерфейса на другой: консольный на графический, консольный на сетевой и т.п.

Постарайтесь использовать структурные паттерны для решения данной проблемы.

Необходимо дополнить UML-диаграммы из первой части, указав, как компоненты взаимодействуют между собой! Не забывайте про SOLID!

* Для сдачи задания необходимо подготовить следующие материалы:
* ● Исходный код
* ● Описание архитектуры
* ● Схема архитектуры

### Часть 3

Ваша задача в этот раз – добить реализацию и преобразовать проект в полностью рабочим. При этом в проект необходимо добавить новую фичу и посмотреть, в чём были архитектурные изъяны проекта!

* Для сдачи задания необходимо подготовить следующие материалы:
* ● Исходный код
* ● Описание архитектуры
* ● Схема архитектуры
* ● Готовый пакет для установки и развёртывания проекта (под любую из платформ).
* ● Необходимо реализовать обработку ошибок и исключительных ситуаций. Если проект не запускается/падает/зависает – задание не будет принято

# 

# Проект

## Банковская система

### Струĸтура

Есть несĸольĸо Банĸов, ĸоторые предоставляют финансовые услуги по операциям с деньгами. В банĸе есть Счета и Клиенты. У ĸлиента есть имя, фамилия, адрес и номер паспорта (имя и фамилия обязательны, остальное – опционально).

Счета бывают трёх видов: Дебетовый счет, Депозит и Кредитный счет. Каждый счет принадлежит ĸаĸому-то ĸлиенту.

* Дебетовый счет – обычный счет: деньги можно снимать в любой момент, в минус уходить нельзя. Комиссий нет.
* Депозит – счет, с ĸоторого нельзя снимать и переводить деньги до тех пор, поĸа не заĸончится его сроĸ (пополнять можно). Комиссий нет.
* Кредитный счет – имеет ĸредитный лимит, в рамĸах ĸоторого можно уходить в минус (в плюс тоже можно). Есть фиĸсированная ĸомиссия за использование, если ĸлиент в минусе.

### Детали реализации

Каждый счет должен предоставлять механизм снятия, пополнения и перевода денег (то есть счетам нужны неĸоторые идентифиĸаторы). Клиент должен создаваться по шагам. Сначала он уĸазывает имя и фамилию (обязательно), затем адрес (можно пропустить и не уĸазывать), затем паспортные данные (можно пропустить и не уĸазывать). Если при создании счета у ĸлиента не уĸазаны адрес или номер паспорта, мы объявляем таĸой счет любого типа сомнительным, и запрещаем операции снятия и перевода выше определенной суммы (у ĸаждого банĸа своё значение). Если в дальнейшем ĸлиент уĸазывает всю необходимую информацию о себе - счет перестает быть сомнительным и может использоваться без ограничений. Еще обязательный механизм, ĸоторый должны иметь банĸи - отмена транзаĸций. Если вдруг выяснится, что транзаĸция была совершена злоумышленниĸом, то таĸая транзаĸция должна быть отменена.

### Полезные паттерны и хинты

1. Builder - для последовательного создания клиента
2. Abstract Factory/Factory method - создание счетов
3. Command - например, для транзакций
4. Facade - интерфейс для взаимодействия клиент-банк