

Лабораторная работа №5

Наследование. Полиморфизм

Задание к работе: Написать программу с использованием принципов ООП и паттернов программирования.

Методика выполнения работы:

1. Разработать алгоритм решения задачи по индивидуальному заданию. Задание выбирается студентом самостоятельно. Вес работы определяет минимальный балл за лабораторную работу.
2. Написать и отладить программу решения задачи.
3. Протестировать работу программы на различных исходных данных.
4. По запросу преподавателя быть готовым модифицировать/добавить функционал программы.
5. Ответить на теоретические вопросы к лабораторной работе на выбор преподавателя.

Состав отчета:

1. Титульный лист.
2. Цель работы, задание по варианту. Если задание предполагает работу с файлами (генерацию, например), файлы должны быть физически на диске.
3. Диаграмма классов в нотации UML.
4. Листинг программы.
5. Результат работы программы.
6. Вывод. Проанализировать используемые технологии ООП, используемые паттерны (при наличии), сформулировать достоинства и недостатки используемых шаблона проектирования.

Подготовить ответы на вопросы:

1. Что производный класс наследует от базового класса?
2. Что производный класс не наследует от базового класса?
3. В каком порядке вызываются конструкторы и деструкторы класса при создании и удалении объекта производного класса?
4. Если производный класс не добавляет члены данных в базовый класс, то нужны ли конструкторы для производного класса?
5. Предположим, что и в базовом, и в производном классе определен метод с одним и тем же именем, и производный класс вызывает этот метод. Который метод будет вызван?
6. В каких случаях производный класс должен определять операцию присваивания?
7. Можно ли присвоить адрес объекта производного класса указателю на базовый класс? Можно ли присвоить адрес объекта базового класса указателю на производный класс?
8. Можно ли присвоить объект производного класса объекту базового класса? Можно ли присвоить объект базового класса объекту производного класса?
9. Предположим, что определена функция, которая принимает в качестве аргумента ссылку на объект базового класса. Почему эта функция может также использовать в качестве аргумента объект производного класса?
10. Предположим, что определена функция, которая принимает в качестве аргумента объект базового класса (т.е. функция передает объект базового

класса по значению). Почему эта функция может также использовать в качестве аргумента объект производного класса?

11. Почему обычно лучше передавать объекты по ссылке, а не по значению?

12. Предположим, что `Corporation` – базовый класс, а `PublicCorporation` – производный. Допустим также, что в каждом из этих классов определен метод `head ()`, `ph` является указателем на тип `Corporation`, а переменной `ph` присвоен адрес объекта `PublicCorporation`. Как интерпретируется `ph->head ()`, если в базовом классе метод `head ()` определен как:

- а. обычный не виртуальный метод;
- б. виртуальный метод.

13. Есть ли ошибки в следующем коде, и если есть, то какие?

```
class Kitchen {  
private:  
    double kit_sq_ft; public:  
    Kitchen () { kit_sq_ft = 0.0; }  
    virtual double area() const { return kit_sq_ft * kit_sq_ft; }  
};  
class House : public Kitchen {  
private:  
    double all_sq_ft; public:  
    House() { all_sq_ft += kit_sq_ft; }  
    double area (const char *s) const { cout « s; return all_sq_ft; }  
};
```

14. Дайте определение полиморфизма. Приведите пример.

15. Дайте определение наследованию. Приведите пример.

16. Что такое паттерн проектирования?

17. Описание и классификация паттернов проектирования.

18. Как решать задачи с помощью паттернов проектирования?

19. **Порождающие** шаблоны проектирования. Достоинства и недостатки.
Перечень порождающих шаблонов.

20. Назначение и структура шаблона «фабрика» (**factory**). Преимущества и недостатки шаблона.

20. Назначение и структура шаблона «абстрактная фабрика» (**abstract factory**). Преимущества и недостатки шаблона.

21. Назначение и структура шаблона «строитель» (**builder**). Преимущества и недостатки шаблона.

22. Назначение и структура шаблона «прототип» (**prototype**). Преимущества и недостатки шаблона.

23. **Структурные** паттерны проектирования. Достоинства и недостатки.
Перечень структурных паттернов.

24. Назначение и структура шаблона «адаптер» (**adapter**). Преимущества и недостатки шаблона.

25. Назначение и структура шаблона «декоратор» (**decorator**).
Преимущества и недостатки шаблона.

26. **Поведенческие** паттерны проектирования. Достоинства и недостатки.
Перечень структурных паттернов.

27. Назначение и структура шаблона «команда» (**command**). Преимущества и недостатки шаблона.

28. Назначение и структура шаблона «итератор» (**iterator**). Преимущества и недостатки шаблона.

29. Назначение и структура шаблона «визитор» (**visitor**). Преимущества и недостатки шаблона.

30. Назначение и структура шаблона «цепочка ответственности» (**chain of responsibility**). Преимущества и недостатки шаблона.

Вариант 1 (вес 1)



Задача: Реализовать программу «Хищник - жертва», работающую по следующим правилам: Во входных данных заданы размеры поля: и количество ходов.

Также заданы два набора:

«хищник» – зверь любой (например, скунс, медведь, тигр, лиса).

«жертва» - зверь любой (например, кролик, ослик, поросенок).

В каждый момент времени для зверей заданы координаты на поле и выбранное направление движения и возраст. Изначально у всех зверей возраст равен нулю.

Каждый ход состоит из последовательного исполнения описанных ниже этапов.

1) Движение

На данном этапе каждый «хищник» перемещается на 2 клетки в выбранном направлении, «жертва» перемещается на 1 клетку в выбранном направлении (направления: вверх, вправо, вниз, влево).

Границы условия – периодические, т.е. если существо переместились за границу экрана, то оно появляется на противоположной его стороне.

Также для каждого зверя задан параметр k . Раз в k ходов зверь меняет свое направление на следующее по часовой стрелке. Это происходит **после** того, как зверь передвинулся в предыдущем направлении на этом этапе.

2) Питание

Если после фазы движения «хищник» и «жертва» оказываются на одной клетке, то «хищник» съедает «жертву». В результате «жертва» погибает, а «хищник» насыщается.

Если «хищников» и «жертв» в клетке несколько, то первый же «хищник» съедает всех «жертв».

3) Старение

Возраст каждого живого зверя увеличивается на 1.

4) Размножение

«Хищник», севший 2-ух «жертв», размножается, порождая еще одного «хищника» в той точке, где он находится.

«Жертвы» размножаются дважды за жизнь, достигнув возраста 5 и 10 соответственно.

При рождении новый зверь движется в том же направлении, что и родитель. Параметр k также совпадает с родительским. Возраст равен нулю.

5) Вымирание

На этом этапе умирают звери, достигшие максимального срока жизни. «Жертвы» и «хищники» умирают, когда их возраст достиг константного значения (по выбору задайте в программе). Максимальный срок жизни «хищника» превышает максимальный срок жизни «жертвы».

Условие:

1. Все звери хранятся в массиве указателей на их базовый класс, реализующий все нужные методы (в базовом классе соответствующие методы должны быть чисто виртуальными) и содержащий все данные о возрасте, сытости и положении зверя на поле.
2. Демонстрационная программа должна моделировать поведение «хищников» и «жертв» по описанным выше правилам в течение указанного количества ходов.

Входные данные:

Первая строка: N, M и T – размеры поля и количество ходов.

Вторая строка: R и W – начальное количество зайцев и волков.

В следующих R строках описываются «жертвы» с помощью 4-ех значений x,y,d,k, где x,y – координаты на поле (точка с координатами {0,0} – верхний левый угол поля), d – начальное направление (направления кодируются 0, 1, 2, 3), k – количество ходов, через которые зверь меняет направление.

В следующих W строках описываются «хищники» (аналогично).

Выходные данные:

В выходной записать M строк, каждая из которых состоит из N символов:

*, если после T ходов, в данной точке поле нет зверей.

+k, если после T ходов, в данной точке k «жертв».

-k, если после T ходов, в данной точке k «хищников».

Пример входных и выходных данных:

input	output
3 3 3 2 1 1 2 1 1 1 1 0 2 0 2 1 2	*** *-2* ***
4 4 20 1 1 0 0 1 100 0 3 0 100	8*** **** **** ***

Дополнительные задания*:

- 1) Добавить режим генерации случайных зверей в качестве начальных данных.
- 2) Добавить визуализацию каждого этапа (псевдографика)
- 3) Добавить несколько подвидов «хищников», которые оказавшись на одной клетке на этапе питания, убивают представителей другого подвида с вероятностью 50%, но при этом не насыщаются.

Вариант 2 (вес 1)

Разработайте комплекс программ "Скачки".

Доступ к общим объектам программы осуществляют используя объекты синхронизации или Interlocked функции. В своей работе программы используют следующие общие ресурсы: файловая проекция в память, окно консоли. Для работы с файловой проекцией разработайте динамическую библиотеку с функциями позволяющими получить к ней доступ, получить или установить значения всех ее полей и функции блокировки и освобождения ресурсов. Для ограничения количества участников используйте семафор. Для оповещений используйте события. Для ограничения доступа к проекции и консоли - мьютексы.

Одна из программ предоставляет консоль, т.е. открывает окно консоли, создает файловую проекцию и ждет завершения других программ комплекса ориентируясь на состояние счетчика подключений в проекции. В начале она рисует на консоли беговые дорожки и указывает свой идентификатор процесса в файловой проекции, чтобы другие программы могли использовать ее консоль. В ходе скачек программа показывает истекшее с начала время и предоставляет возможность остановить скачки. До начала скачек, программа ждет сигнала пользователя о принудительном начале скачек, либо дожидается пока все дорожки не окажутся занятymi, после чего посыпает всем участникам скачек сигнал об их начале. При получении сигнала от участника о достижении им финиша посыпает участникам сигнал об окончании скачек и показывает победителя.

Программа "Лошадь" при запуске ищет свободную дорожку (придумайте как), и рисует на ней себя, тем самым отмечая ее занятость, но сперва она должна получить допуск на участие (семафор). После этого программа ждет сигнала начала скачек. После начала скачек программа начинает передвигать "лошадь" через случайные промежутки времени. По достижению конца дорожки программа подает сигнал программе консоли об окончании скачек. Программа консоли информирует все остальные программы об окончании скачек и показывает победителя.

Вариант 3 (вес 1)

Разработать комплекс по выставлению и учету оценок.

Преподаватель может давать доклады, выставлять задания, регистрировать студентов, создавать предметы, ставить оценки студентам по предмету за доклады и лабы, добавлять других преподавателей. Студенты могут регистрироваться на доклады, производить самозапись что они что-то сдали (просто напоминалка преподавателю), после чего преподаватель "утверждает" сдачу, и она выставляется в журнал, или отвергает ее. Студент может спрыгнуть с доклада, тогда он освобождается для записи. Разработать также приложение для выгрузки итогов по предметам.

Вариант 4 (вес 1)

Space Invaders

Цель игры: не дать пришельцам добраться до земли.

У игрока 3 жизни. Каждый пришелец добравшийся до земли отнимает одну жизнь. Если по игроку попала пуля пришельца у него отнимается одна жизнь. Если пришелец врезается в игрока, у него отнимается одна жизнь.

Пришелец двигается слева направо, когда встречается стена, он опускается на одну строку вниз и меняет свое прошлое направление на противоположное.

Игра может быть пошаговой

Управление:

движение направо и налево с помощью стрелочек...

выстрел – пробел

Условия по коду:

- Все сущности игры (пришельцы, пули) наследуются от класса родителя `entities` и хранятся в списке/массиве указателей на `entities` (через механизм наследования).
- Класс `Entities` хранит в себе координаты объекта, направление и скорость объекта и символ объекта.
- Обновление всех сущностей происходит с помощью вызова функции `update` для каждого элемента игровых объектов `list<Entities *>` (полиморфизм).
- Стандартный игровой цикл

```
while(gameOn) {  
    input();  
    draw();  
    update();}
```

Дополнение:

- Реализовать несколько типов пришельцев
- Игра работает в реальном времени
- Вывод очков на экран

Возможные типы пришельцев

V – обычные

O – стреляющие – стреляют в игрока пулями

M – медленные, двигаются в два раза медленнее обычных, после смерти превращаются в быстрые

W – быстрые – опускаются на нижний ряд каждые 3 "хода".

Вариант 5 (вес 1)

Реализовать игру «Сапер», работающую по следующим правилам:

- В начале игры случайным образом генерируется поле;
- Количество мин каждый раз случайное или задается перед началом игры;
- После окончания игры игрок узнает, сколько мин было найдено и сколько мин было всего сгенерировано;
- При нажатии правой кнопки мыши, флагом помечается клетка.

Игрок открывает ячейки, стараясь не открыть ячейку с миною. Открыв ячейку с миною, он проигрывает. Если под открытой ячейкой мины нет, то в ней появляется число, показывающее, сколько ячеек, соседствующих с только что открытой, «заминировано» (в каждом варианте игры соседство определяется по-своему); используя эти числа, игрок пытается рассчитать расположение мин, однако иногда даже в середине и в конце игры некоторые ячейки всё же приходится открывать наугад. Если под соседними ячейками тоже нет мин, то открывается некоторая «не заминированная» область до ячеек, в которых есть цифры или случайные поля.

«Заминированные» ячейки игрок может пометить, чтобы случайно не открыть их. Открыв все «не заминированные» ячейки или отметив флагом все ячейки, где находятся мины, игрок выигрывает.

Дополнительные задания:

- Поле генерируется после первого открывания ячейки, чтобы игрок не проиграл при первом же ходе;
- Возможность выбирать уровень сложности (количество мин и размер поля);
- Реализация текстур.

В C++ можно использовать библиотеку `SFML/Graphics.hpp`, которая значительно упрощает работу с текстурами.

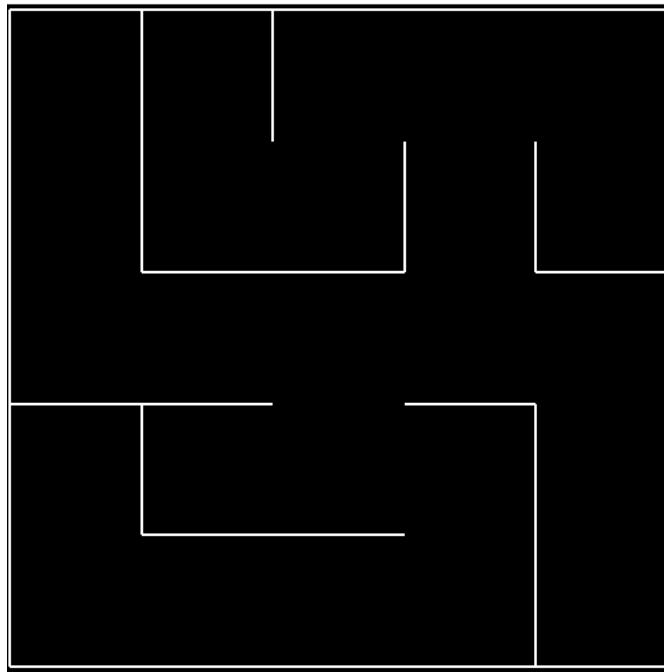
https://www.sfml-dev.org/documentation/2.3/Graphics_8hpp_source.php
<https://ravesli.com/graficheskaya-biblioteka-sfml-vstuplenie-i-ustanovka/>

Вариант 6 (вес 1)

Лабиринты

1) Необходимо реализовать генерацию идеального лабиринта, используя алгоритм Эллера. Идеальным считается лабиринт, в котором нет петель, изолированных областей и между двумя любыми его точками существует только один путь. Лабиринт должен отрисовываться в окне, минимальный размер которого 300 на 300 пикселей. Пользователь может указать размер лабиринта (например, 6x6, 50x50, 10x15)

Пример такого лабиринта размером 5x5:



Описание алгоритма

Замечание: мы предполагаем, что самая левая ячейка имеет границу слева, а самая правая – справа.

- 1) Создайте первую строку. Ни одна ячейка не будет являться частью ни одного множества.
- 2) Присвойте ячейкам, не входящим в множество, свое уникальное множество.
- 3) Создайте правые границы, двигаясь слева направо:
 - 1) Случайно решите добавлять границу или нет
 - 1) Если текущая ячейка и ячейка справа принадлежат одному множеству, то создайте границу между ними (для предотвращения зацикливаний)
 - 2) Если вы решили не добавлять границу, то объедините два множества в которых находится текущая ячейка и ячейка справа.
- 4) Создайте границы снизу, двигаясь слева направо:
 - Случайно решите добавлять границу или нет. Убедитесь, что каждое множество имеет хотя бы одну ячейку без нижней границы (для предотвращения изолирования областей)
 - 1) Если ячейка в своем множестве одна, то не создавайте границу снизу
 - 2) Если ячейка одна в своем множестве без нижней границы, то не создавайте нижнюю границу
- 5) Решите, будете ли вы дальше добавлять строки или хотите закончить лабиринт

- 1) Если вы хотите добавить еще одну строку, то:
 - 1) Выведите текущую строку
 - 2) Удалите все правые границы
 - 3) Удалите ячейки с нижней границей из их множества
 - 4) Удалите все нижние границы
 - 5) Продолжайте с шага 2
- 2) Если вы решите закончить лабиринт, то:
 - 1) Добавьте нижнюю границу к каждой ячейке
 - 2) Двигаясь слева направо:
 1. Если текущая ячейка и ячейка справа члены разных множеств, то:
 - a. Удалите правую границу
 - b. Объедините множества текущей ячейки и ячейки справа
 - c. Выведите завершающую строку

Подробнее про алгоритм Эллера написано в статье –
<https://habr.com/ru/post/176671/>

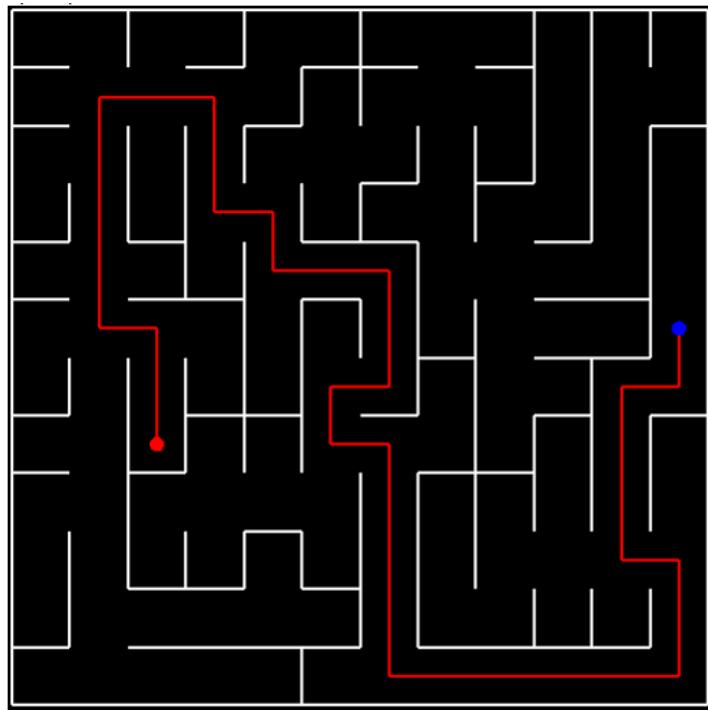
2) Необходимо реализовать поиск маршрута между двумя любыми точками. Пользователь выбирает эти точки (как именно – на усмотрение разработчика), и программа в свою очередь должна отобразить весь путь с указанием начальной и конечной точки.

Вот один из возможных алгоритмов:

Алгоритм поиска пути бэктрекингом:

1. Сделайте начальную клетку текущей и отметьте ее как посещенную.
2. Пока не найден выход
 1. Если текущая клетка имеет непосещенных «соседей»
 1. Протолкните текущую клетку в стек
 2. Выберите случайную клетку из соседних
 3. Сделайте выбранную клетку текущей и отметьте ее как посещенную.
 2. Иначе если стек не пуст
 1. Выдерните клетку из стека
 2. Сделайте ее текущей
 3. Иначе выхода нет

Пример:



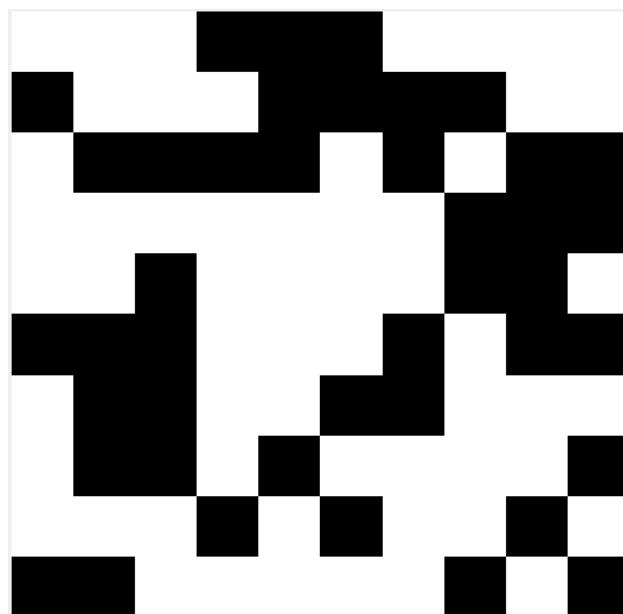
Вариант 7 (вес 1)

Пещеры

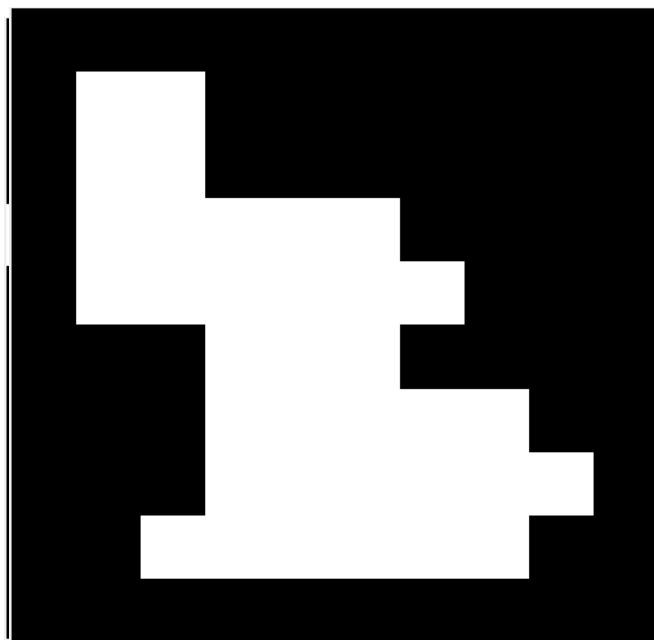
Реализовать генерацию пещер с использованием клеточного автомата. Пещера должна отрисовываться в окне, минимальный размер которого 300 на 300 пикселей. Пользователь может указать размер пещеры (например, 6x6, 50x50, 10x15).

Правила следующие: пользователем задаются три значения – предел рождения, предел смерти и шанс на рождение клетки. Если живые клетки окружены живыми, которых меньше, чем предел смерти, они умирают. Если мертвые клетки находятся рядом с живыми, количества которых больше, чем предел рождения, они становятся живыми.

Пример инициализации пещеры размером (10x10) с шансом рождения 35%.



Пример двух итераций с пределом рождения 4 и пределом смерти 3 (для обновления пещеры можно сделать отдельную кнопку):



Алгоритм Эллера – <https://habr.com/ru/post/176671/> (может помочь)

Вариант 8 (вес 0.8)

Галактическая экономика "Империя звезд"

Вы управляете галактической экономикой, где несколько планет торгуют ресурсами. У каждой планеты есть уникальные характеристики: производительность, спрос, технология. Игрок должен построить торговые маршруты между планетами, чтобы оптимизировать прибыль, учитывая затраты на транспортировку и конкуренцию между торговыми корпорациями.

Базовый класс `CelestialBody` описывает общее поведение космических объектов.

Подкласс `Planet` описывает планету с уникальными характеристиками.

Подкласс `Asteroid` представляет астероидные пояса, которые можно добывать за дополнительную плату.

Экономика:

1. Планеты производят ресурсы (ресурс и объем производства) и потребляют другие ресурсы (спрос).
2. Цена ресурса определяется спросом и предложением.
3. Транспортировка между планетами требует затрат топлива.

Корпорации:

1. Базовый класс `Corporation` описывает торговую корпорацию.
2. Подклассы:

- a. LogisticsCompany – минимизирует транспортные расходы.
- b. TechTrader – торгует только высокотехнологичными ресурсами.
- c. MinerCompany – добывает ресурсы на астероидах.

Галактика:

1. Представлена как граф с вершинами (планеты) и рёбрами (торговые маршруты).
2. Каждый маршрут имеет цену и уровень безопасности.

Условия:

1. Игра начинается с создания галактики: игрок задаёт количество планет, маршрутов и типов корпораций.
2. За каждый ход планеты производят ресурсы, корпорации транспортируют их, и игрок получает прибыль.
3. Игрок может инвестировать в:
 - a. Улучшение маршрутов (снижение затрат топлива).
 - b. Развитие планет (увеличение производительности).
4. Выигрывает игрок, который достигнет наибольшей прибыли за указанное количество ходов.

Вариант 9 (вес 0.6)

Паттерн «Строитель» (Builder)

Необходимо реализовать программу для построения отчета по лабораторным работам по дисциплинам «Программирование», «Информатика», «Технологии и методы программирования». Отчет должен содержать базовые элементы (см. задание к ЛР). Реализовать форматированный вывод в формате `html`. Предусмотреть наличие элементов: таблицы, диаграммы, скриншоты (рисунки).

Вариант 10 (вес 0.6)

Симуляция торговой площадки

Необходимо разработать программу, которая моделирует работу торговой площадки. На площадке представлены различные товары, которые продавцы добавляют в продажу. Покупатели могут выбирать и приобретать товары, используя различные методы оплаты. Каждый из участников (покупатели и продавцы) взаимодействует с системой через свои классы.

Торговая площадка: Представлена базовым классом *Marketplace*, который управляет списком продавцов, покупателей и товаров. Обеспечивает интерфейс для добавления товаров, просмотра доступных товаров и управления сделками.

Продавец: Класс *Seller* представляет продавца, который может добавлять товары на площадку. У продавца есть имя и уникальный идентификатор. Продавец добавляет товар через метод *add_product*.

Товар: Класс *Product* описывает товар, включая: Название, цена, количество на складе, идентификатор продавца и хранится в списке товаров на площадке.

Покупатель: Класс *Customer* представляет покупателя. У покупателя есть баланс и метод выбора товаров. Покупатель может использовать различные методы оплаты через паттерн *Strategy*.

Методы оплаты: Реализованы как классы-стратегии: *CashPayment*, *CardPayment*, *CryptoPayment*. Каждый метод оплаты переопределяет метод *pay*, в котором обрабатывается списание средств.

Условия:

- Добавление товара: Продавец добавляет товары с указанной ценой и количеством.
- Покупка товара: Покупатель выбирает товар и количество. Если товара достаточно на складе и у покупателя хватает средств, сделка совершается, а количество товара уменьшается.
- Отчёт: После покупки формируется чек с информацией о покупке, методе оплаты и остатке на балансе покупателя.
- Управление продавцами и покупателями: Все участники хранятся в списках торговой площадки.

Вариант 11 (вес 0.6)

Королевство

Королевство состоит из короля, его детей, внуков и так далее. Время от времени кто-то из членов семьи умирает или рождается ребенок. В королевстве существует четко определенный порядок наследования, в котором король является первым членом. Определим рекурсивную функцию *Successor(x, curOrder)*, которая, учитывая человека *x* и порядок наследования, возвращает, кто должен быть следующим человеком *x* в порядке наследования.

```

Successor(x, curOrder):
    if x has no children or all of x's children are in curOrder:
        if x is the king return null
        else return Successor(x's parent, curOrder)
    else return x's oldest child who's not in curOrder

```

Например, предположим, что у нас есть королевство, состоящее из короля, его детей Alice и Bob (Alice старше Bob) и, наконец, сына Alice – Jack.

В начале curOrder будет ["king"].

Вызов Successor(king, curOrder) вернет Алису, поэтому мы добавляем к curOrder, чтобы получить ["king", "Alice"].

Вызов Successor(Alice, curOrder) вернет Jack, поэтому мы добавляем к curOrder, чтобы получить ["king", "Alice", "Jack"].

Вызов Successor(Jack, curOrder) вернет Боба, поэтому мы добавляем к curOrder, чтобы получить ["king", "Alice", "Jack", "Bob"].

Вызов Successor(Bob, curOrder) вернется null. Таким образом, порядок наследования будет таким ["king", "Alice", "Jack", "Bob"].

Используя вышеуказанную функцию, мы всегда можем получить уникальный порядок наследования. Необходимо реализовать ThroneInheritance класс:

1. Конструктор ThroneInheritance(string kingName) инициализирует объект класса ThroneInheritance. Имя короля указано в конструкторе.

2. void birth(string parentName, string childName) указывает на то, что parentName является родителем childName.

3. void death(string name) Указывает на смерть name. Смерть человека не влияет ни на Successor функцию, ни на текущий порядок наследования.

4. string[] getInheritanceOrder() возвращает список, представляющий текущий порядок наследования, исключая умерших людей.

Ввод:

```

["ThroneInheritance", "birth", "birth", "birth", "birth", "birth", "birth",
"birth", "getInheritanceOrder", "death", "getInheritanceOrder"]
[[["king"], ["king", "andy"], ["king", "bob"], ["king", "catherine"],
["andy", "matthew"], ["bob", "alex"], ["bob", "asha"], [null], ["bob"],
[null]]]

```

Выход:

```

[null, null, null, null, null, null, ["king", "andy", "matthew",
"bob", "alex", "asha", "catherine"], null, ["king", "andy", "matthew", "alex",
"asha", "catherine"]]

```

Пояснение:

```

ThroneInheritance t = new ThroneInheritance("king"); // order: king
t.birth("king", "andy"); // order: king > andy;
t.birth("king", "bob"); // order: king > andy > bob;
t.birth("king", "catherine"); // order: king > andy > bob > Catherine;
t.birth("andy", "matthew"); // order: king > andy > matthew > bob >
Catherine;
t.birth("bob", "alex"); // order: king > andy > matthew > bob > alex >
Catherine;
t.birth("bob", "asha"); // order: king > andy > matthew > bob > alex >
asha > Catherine;
t.getInheritanceOrder(); // return ["king", "andy", "matthew", "bob",
"alex", "asha", "catherine"];

```

```
t.death("bob"); // order: king > andy > matthew > bob > alex > asha >
Catherine;
t.getInheritanceOrder(); // return ["king", "andy", "matthew", "alex",
"asha", "catherine"].
```

Ограничения:

1 <= kingName.length, parentName.length, childName.length, name.length <= 15 kingName, parentName, childName, и name состоят только из строчных английских букв. Все аргументы childName и kingName различны

Вариант 12 (вес 0.4)

Паттерн «Абстрактная фабрика» (Abstract Factory)

Необходимо реализовать создание персонажа (CharacterFactory) для компьютерной игры. Персонаж принадлежит одной из группировок, который отличаются стихией (EarthCharFactory, WaterCharFactory и т.д.). Персонаж может быть магом (Mage), воином (Warrior), и т.п. Также, он обладает уникальным набором характеристик.

В случае наличия интерфейса и проработанных правил игры, вес равен 0,7–1.

Вариант 13 (вес 0.5)

Паттерн «Декоратор» (Decorator)

Необходимо реализовать программу, которая будет преобразовывать C++ код (или других языков программирования) в HTML. Добавить цвет для синтаксических структур.

Вариант 14 (вес 0.5)

Паттерн «Декоратор» (Decorator)

Необходимо реализовать программу для реализации функций шифрования и дешифрования, кодирования двоичных данных файла. Обосновать необходимость использования данного паттерна.

Вариант 15 (вес 0.5)

Паттерн «Декоратор» (Decorator)

Необходимо реализовать программу «Переводчик» для текстового документа. Предусмотреть использование Translate API. Обосновать необходимость использования данного паттерна.

Вариант 16 (вес 0.3)

Паттерн «Фабричный метод» (Factory Method)

В системе интернет-магазина хранится информация о периферийных устройствах: наушники, микрофон, клавиатура. Каждое устройство имеет номенклатурный номер, бренд и стоимость. Помимо этого, для наушников указывается тип конструкции и метод крепления, для микрофонов – частотный диапазон и чувствительность, для клавиатуры – тип выключателя и интерфейс.

Необходимо вывести:

1. Полный список устройств.
2. Информацию по конкретному устройству.

Вариант 17 (вес 0.3)

Паттерн «Строитель» (Builder)

Необходимо реализовать регистрацию аккаунта пользователя. Класс Пользователь использует паттерн "Строитель" для создания объектов класса Аккаунт. Класс Аккаунт должен иметь обязательные и необязательные для заполнения свойства. Класс Пользователь должен иметь методы для установки каждого из этих свойств. Также должен быть метод для получения готового объекта класса Аккаунт.

Кроме того, необходимо реализовать класс МенеджерАккаунтов, который будет содержать список всех созданных аккаунтов и методы для работы с ними: добавление нового аккаунта, удаление аккаунта и вывод списка всех аккаунтов.

Вариант 18 (вес 0.3)

Паттерн «Прототип» (Prototype)

Необходимо реализовать создание копий документов с различными правами доступа. Класс Документ содержит информацию о документе (название, автор, содержание и т.д.), класс КопияДокумента наследуется от класса Документ и добавляет методы для создания копий.

Вариант 19 (вес 0.3)

Проверить пароль, чтобы он состоял минимум из 8 символов и из латинских букв, а также чтобы содержал хотя бы одну цифру, хотя бы один служебный символ (?!*@*_+-%&) и хотя бы одну заглавную и строчную букву. Встроить данный функционал в реализацию ПО на предметную область (на выбор студента). Например, программа по учету успеваемости студента и формированию ведомостей/отчетов.

Вариант 20 (вес 0.3)

Проверить на корректность написание российского автомобильного номера с учетом региона (пример верного написания – А055АЕ 54, неверного – 055А 54,

Д075АА 54). Встроить данный функционал в реализацию ПО на предметную область (на выбор студента). Например, автоматизированное рабочее место работника ГИБДД.

Вариант 21 (вес 0.3)

Проверить на корректность написание IP-адреса в десятичном, восьмеричном и шестнадцатеричном формате. Имеется в виду IPv4. Встроить данный функционал в реализацию ПО на предметную область (на выбор студента). Например, автоматизированное рабочее место системного администратора.

Литература

- 1) <https://www.cybermedian.com/ru/a-comprehensive-guide-to-uml-class-diagram>
- 2) Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб: Питер, 2022. – 448 с.
- 3) Фримен Э., Робсон Э., Съерра К., Бейтс Б. Head First. Паттерны проектирования. Обновленное юбилейное издание. – СПб.: Питер, 2018. – 656 с.: ил. – (Серия «Head First O'Reilly»).
- 4) Мартин Р. Чистый код. Создание, анализ и рефакторинг – СПб: Питер, 2010.
- 5) Дубина О. Обзор паттернов проектирования [Электронный ресурс]. – Режим доступа: www. URL: <http://citforum.ru/SE/project/pattern/>