Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение информационных систем

и технологий»

**Отчёт**

по лабораторной работе №21

по дисциплине ***«Конструирование программного обеспечения»***

тема: **«**Основы Объектно-Ориентированного программирования на языке Java.

Классы и объекты**»**

|  |  |  |
| --- | --- | --- |
| Исполнитель: |  | студент группы 10701121  Воропай М. П.  Киселев Н. Ю. |
| Преподаватель: |  | Станкевич С. Н. |

2022-2023 учебный год

**ЛАБОРАТОРНАЯ РАБОТА №21**

Основы Объектно-Ориентированного программирования на языке Java.

Классы и объекты

Цель работы:

Научиться грамотно анализировать предметную область и с помощью аб-  
стракции выделять существенные детали, на базе которых в дальнейшем про-  
ектируются классы и объекты будущей программной системы согласно мето-  
дологии ООП, а также практически закрепить данные навыки при решении  
соответствующих задач (бизнес проблем).

**Общее задание:**

Необходимо решить задачу с использованием методологии ООП. Для чего необходимо подобрать самостоятельно соответствующую проблемную (предметную/доменную) область, которая базируется на объектах и событиях реального мира (примеры соответствующих предметных областей приведены ниже). Спроектировать классы (пользовательские типы данных) в языке Java для программного представления данных объектов и основной логики будущей программной системы. Программа должна решать, как минимум, два полезных действия и иметь дополнительно следующие вещи:

* не менее 3 разнообразных классов предметной области;
* не менее 5 атрибутов (состояния) в каждом соответствующем классе-сущ- ности;
* не менее 3 методов (поведения) в соответствующих функциональных классах для реализации полезной бизнес-логики.

**Индивидуальное задание:**

1. **Компьютерный герой *(Game Hero)*.** Определить сильные стороны героя (его способности: ум, сила, ловкость и т.д.) на базе собранных им артефактов. Под- считать общую стоимость артефактов, а также величину его соответствующих способностей.

# **Требования к выполнению задания:**

1. Необходимо спроектировать UML-диаграмму взаимодействия классов и объ- ектов создаваемой программной системы. На базе данной UML-диаграммы реализовать рабочее приложение с использованием архитектурного шаблона проектирования ***MVC***.
2. Каждый класс разрабатываемого приложения должен иметь адекватное осмысленное имя (обычно это *имя существительное*). Имена полей и мето- дов должны нести также логический смысл (имя метода, который что-то вы- числяет, обычно называют *глаголом*, а поле – именем существительным). Имя класса пишется с большой (заглавной) буквы, а имена методов и переменных

– с маленькой (строчной).

1. Соответствующие классы должны группироваться по модулям, которые затем подключаются там, где происходит создание объектов классов и их использо- вание.
2. При проектировании классов необходимо придерживаться принципа един- ственной ответственности (***Single Responsibility Principle***), т.е. классы должны проектироваться и реализовываться таким образом, чтобы они были слабо завязаны с другими классами при своей работе – они должны быть самодо- статочными.
3. При выполнении заданий необходимо по максимуму пытаться разрабатывать универсальный, масштабируемый, легко поддерживаемый и читаемый код.
4. В соответствующих компонентах (классах, функциях) бизнес-логики необхо- димо предусмотреть «защиту от дурака».
5. Рекомендуется избегать использования глобальных переменных.
6. Там, где это необходимо, необходимо обеспечить грамотную обработку ис- ключительных ситуаций, которые могут произойти при выполнении разрабо- танной программы.
7. Все действия, связанные с демонстрацией работы приложения, должны быть размещены в главном модуле программы в функции ***main***. При проверки ра- ботоспособности приложения необходимо проверить все тестовые случаи.
8. Программы должны обязательно быть снабжены комментариями на англий- ском языке, в которых необходимо указать краткое предназначение про- граммы, номер лабораторной работы и её название, версию программы, ФИО

разработчика, номер группы и дату разработки. Исходный текст программ- ного кода и демонстрационной программы рекомендуется также снабжать поясняющими краткими комментариями.

1. Программа должна быть снабжены дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем. Интерфейс программы должен быть на английском языке.
2. При разработке программы придерживайтесь соглашений по написанию кода на Python (***Python Code Convention****)*.

**UML – диаграмма**

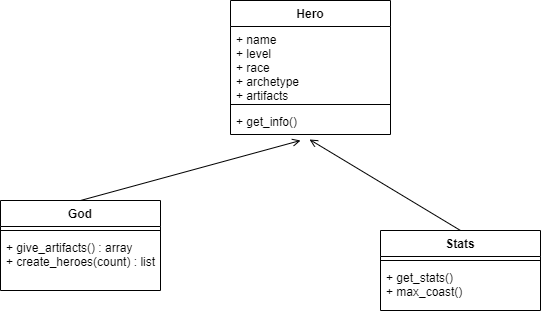
****

Рисунок 1 – uml диаграмма

Результаты выполнения общего задания:

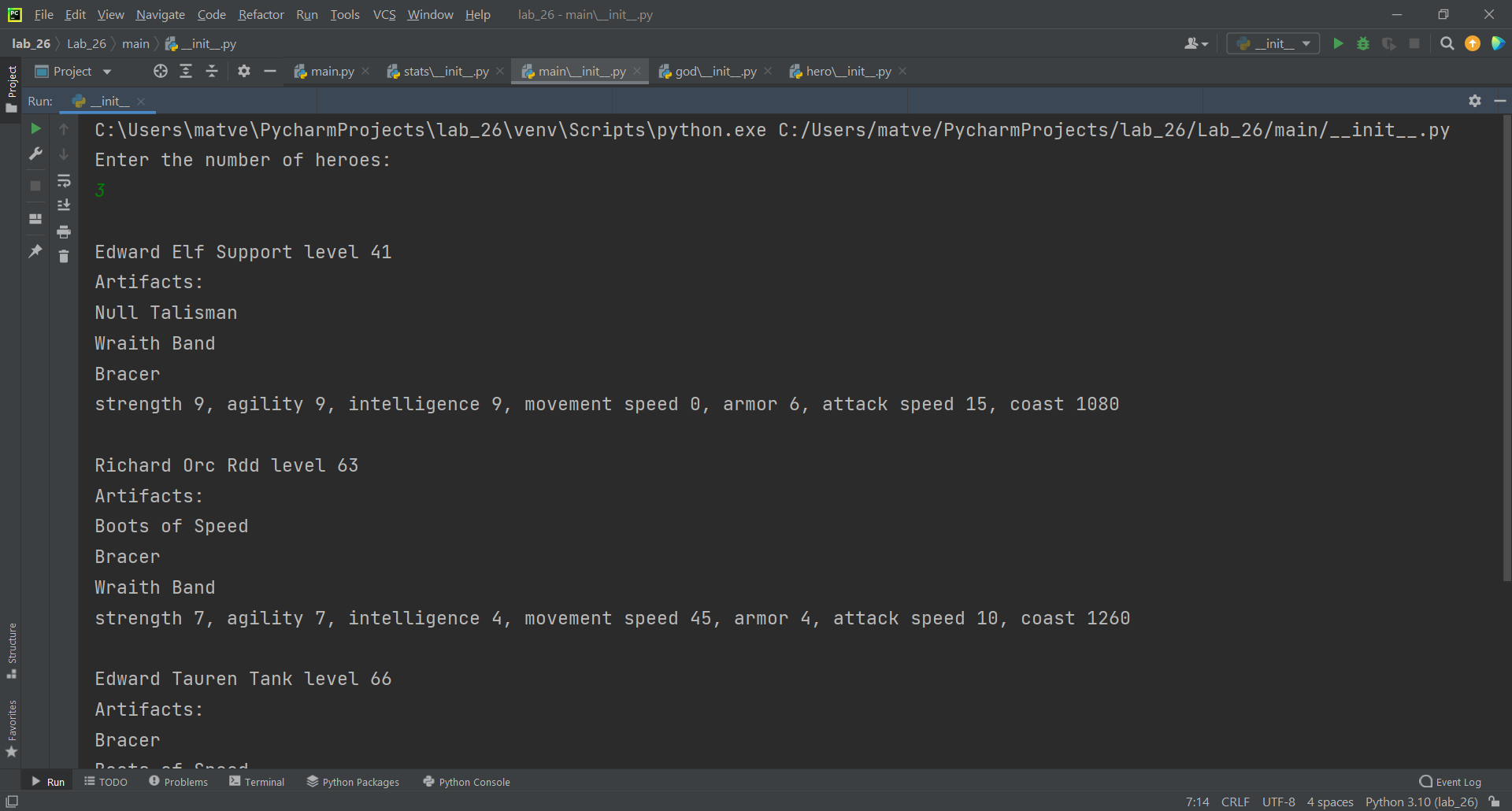


Рисунок 2 – результат выполнения общего задания

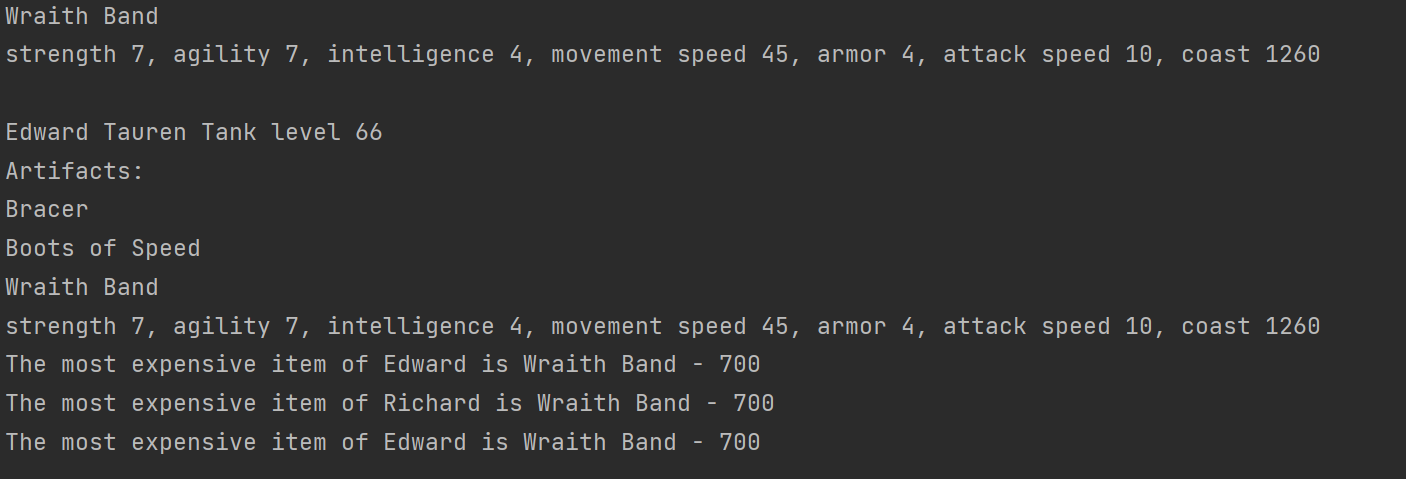


Рисунок 3 – результат выполнения общего задания

# **Контрольные вопросы:**

1. Что такое ООП? Опишите базовые концепции, которые лежат в основе данной мето-дологии программирования.

Это методология программирования, основанная на представлении программы в виде совокупности объектов.

2. Для чего было создано ООП? На каких принципах базируется ООП? Опишите их основные идеи?

Программа, написанная с использованием ООП, состоит из множества объектов, и все эти объекты взаимодействуют между собой посредством посылке (передачи) со-общений друг другу.

Основные столпы (парадигмы) ООП:

1) абстракция;

2) инкапсуляция;

3) наследование;

4) полиморфизм;

5) посылка сообщений (вызов соответствующих методов, свойств и т.д.);

6) повторное использование кода.

3. Приведите преимущества и недостатки объектно-ориентированного подхода.

Преимущества:

· Уменьшение сложности ПО

· Увеличение производительности труда программистов (скорости разработки ПО)

· Повышение надёжности ПО

· Возможность модификации компонентов

· Повторное использование

Недостатки:

· Увеличивается время на анализ и проектирование систем

· Увеличение времени выполнения

· Неэффективно с точки зрения памяти

4. Что такое объект и чем характеризуются объекты в ООП?

Объект – это мыслимая или реальная сущность, обладающая характерным поведением, отличительными характеристиками и являющая важной в предметной области.

Каждый объект имеет состояние, обладает некоторым хорошо определённым поведением и уникальной идентичностью.

5. Что такое класс и зачем он нужен? Приведите общее определение класса в ООП. Какая разница между классом и объектом в ООП? Какие разновидности классов существуют в ООП?

Класс – это шаблон поведения объектов определённого типа с определёнными параметрам, которые описывают состояние объекта.

· Базовый (родительский) класс

· Производный класс (наследник, потомок)

· Абстрактный класс

· Интерфейс

6. Как описывается класс в языке Python?

· класс должен быть максимально простым, настолько простым, насколько это возможно;

· класс должен отвечать только за ту задачу, которая непосредственно на него возложена (класс должен описывать только одну группу объектов);

· название класса – очень важная вещь, оно должно быть простым и понятным (в идеале, имя класса должно говорить о том, за что он отвечает);

· название класса обычно называется именем существительным и записы- вается с заглавной буквы;

· классы должны быть самодостаточные (слабосвязанные), т.е. не зависеть от реализации других классов

7. Каким способом можно создать объект (экземпляр класса) в языке Python?

Class Имя\_класса:

8. Как можно обратиться к полям и методам объекта? В чём состоит особенность первого аргумента в методах класса?

Через оператор ‘.’. Self – первый аргумент в методах класса, являющийся ссылкой на сам объект.

9. Какой стандартный метод при описании класса необходимо переопределить, чтобы гарантировать, что в тех ситуациях, где требуется строковое представление объекта данного класса, система автоматических вызывала данный метод у ссылочной переменной?

\_\_str\_\_()

10. Что такое UML? Как с помощью UML можно описать классы и их взаимосвязь друг с другом (т.е. описать UML-диаграмму классов)?

UML – унифицированный язык моделирования (Unified Modeling Language)

– это система обозначений, которую можно применять для объектно-ориентирован-ного анализа и проектирования.

**Вывод:**

Изучены основы методологии объектно-ориентированного программирования на языке Python; приобретено объектное мышление и научились проектировать и описывать классы соответствующей предметной области; на базе описанных классов создавать и инициализировать объекты; практически закрепили полученные знания при решении соответствующих задач.

*ПРИЛОЖЕНИЕ A*

Листинг исходных кодов программ

Gog

from random import \*

from Lab\_26.hero import Hero

class God:

# class to create Heroes

def give\_artifacts(self):

artifacts = ["Wraith Band", "Bracer", 'Null Talisman', 'Boots of Speed']

build = []

for j in range(3):

build.append(choice(artifacts))

return build

def create\_heroes(self, count):

names = ['Richard', 'William', 'Henry', 'Norman', 'Edward', 'Atelard']

races = ['Human', 'Orc', 'Dwarf', 'Elf', 'Worgen', 'Undead', 'Tauren', 'Troll']

archetypes = ['Tank', 'Assassin', 'Warrior', 'Support', 'Rdd']

list\_of\_heroes = []

for i in range(count):

name = choice(names)

race = choice(races)

archetype = choice(archetypes)

level = randint(1, 81)

hero = Hero(name, level, race, archetype, God.give\_artifacts(self))

list\_of\_heroes.append(hero)

return list\_of\_heroes

Hero

class Hero:

# Hero description class

def \_\_init\_\_(self, name, level, race, archetype, artifacts):

self.name = name

self.archetype = archetype

self.artifacts = artifacts

self.level = level

self.race = race

def get\_info(self):

print(f'\n{self.name} {self.race} {self.archetype} level {self.level}')

print('Artifacts:')

for i in range(len(self.artifacts)):

print(self.artifacts[i])

Main

from Lab\_26.god import God

from Lab\_26.stats import Stats

def main():

number = int(input('Enter the number of heroes:\n'))

heroes = God().create\_heroes(number)

Stats().get\_stats(heroes)

Stats().max\_coast(heroes)

if \_\_name\_\_ == '\_\_main\_\_':

main()

Stats

class Stats:

def get\_stats(self, list\_of\_heroes):

# Artifact:[strength, agility, intelligence, movement speed, armor, attack speed, coast]

# {"strength": 0, "agility": 0, "intelligence": 0, "movement speed": 0, "armor": 0,

# "attack speed": 0, "coast": 0}

stats = {"Wraith Band": [2, 5, 2, 0, 2, 5, 700], "Bracer": [5, 2, 2, 0, 2, 5, 260],

'Null Talisman': [2, 2, 5, 0, 2, 5, 120],

'Boots of Speed': [0, 0, 0, 45, 0, 0, 300]}

for i in range(len(list\_of\_heroes)):

stats\_of\_hero = [0, 0, 0, 0, 0, 0, 0]

for j in range(len(list\_of\_heroes[i].artifacts)):

for k in range(len(stats\_of\_hero)):

stats\_of\_hero[k] += stats[list\_of\_heroes[i].artifacts[j]][k]

list\_of\_heroes[i].get\_info()

print(

f"strength {stats\_of\_hero[0]}, agility {stats\_of\_hero[1]}, intelligence {stats\_of\_hero[2]}, "

f"movement speed {stats\_of\_hero[3]}, armor {stats\_of\_hero[4]}, attack speed {stats\_of\_hero[5]},"

f" coast {stats\_of\_hero[6]}")

def max\_coast(self, list\_of\_heroes):

coast = {"Wraith Band": 700, "Bracer": 260, 'Null Talisman': 120, 'Boots of Speed': 300}

for i in range(len(list\_of\_heroes)):

max\_c = 120

max\_j = 0

for j in range(3):

if coast[list\_of\_heroes[i].artifacts[j]] > max\_c:

max\_c = coast[list\_of\_heroes[i].artifacts[j]]

max\_j = j

print(

f"The most expensive item of {list\_of\_heroes[i].name} is {list\_of\_heroes[i].artifacts[max\_j]} - {max\_c}")