**Лабораторная работа №3.**

Тема: **Объективно-ориентированное программирование и абстрактный класс. Декомпозиция программы**

Цель работы:

1. **Необходимые теоретические сведения**

Абстрактным называется класс, который содержит один и более абстрактных методов. Абстрактным называется объявленный, но не реализованный метод. Абстрактные классы не могут быть инстанциированы, от них нужно унаследовать, реализовать все их абстрактные методы и только тогда можно создать экземпляр такого класса.

В python существует стандартная библиотека abc, добавляющая в язык абстрактные базовые классы (АБК). АБК позволяют определить класс, указав при этом, какие методы или свойства обязательно переопределить в классах-наследниках.

Возьмем для примера, шахматы. У всех шахматных фигур есть общий функционал, например - возможность фигуры ходить и быть отображенной на доске. Исходя из этого, мы можем создать абстрактный класс Фигура, определить в нем абстрактный метод (в нашем случае - ход, поскольку каждая фигура ходит по-своему) и реализовать общий функционал (отрисовка на доске).

Абстрактные классы призваны предоставлять базовый функционал для классов-наследников. А производные классы уже реализуют этот функционал.

При определении абстрактных классов используется ключевое слово abstract:

*public abstract class Human{*

*private String name;*

*public String getName() { return name; }*

*}*

Но главное отличие состоит в том, что мы не можем использовать конструктор абстрактного класса для создания его объекта. Например, следующим образом:

Кроме обычных методов абстрактный класс может содержать абстрактные методы. Такие методы определяются с помощью ключевого слова abstract и не имеют никакой реализации:

Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе. Также следует учитывать, что если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный.

Зачем нужны абстрактные классы? Допустим, мы делаем программу для обслуживания банковских операций и определяем в ней три класса: Person, который описывает человека, Employee, который описывает банковского служащего, и класс Client, который представляет клиента банка. Очевидно, что классы Employee и Client будут производными от класса Person, так как оба класса имеют некоторые общие поля и методы. И так как все объекты будут представлять либо сотрудника, либо клиента банка, то напрямую мы от класса Person создавать объекты не будем. Поэтому имеет смысл сделать его абстрактным.

*public class Program{*

*public static void main(String[] args) {*

*Employee sam = new Employee("Sam", "Leman Brothers");*

*sam.display();*

*Client bob = new Client("Bob", "Leman Brothers");*

*bob.display();*

*}*

*}*

*abstract class Person {*

*private String name;*

*public String getName() { return name; }*

*public Person(String name){*

*this.name=name;*

*}*

*public abstract void display();*

*}*

*class Employee extends Person{*

*private String bank;*

*public Employee(String name, String company) {*

*super(name);*

*this.bank = company;*

*}*

*public void display(){*

*System.out.printf("Employee Name: %s \t Bank: %s \n", super.getName(), bank);*

*}*

*}*

*class Client extends Person*

*{*

*private String bank;*

*public Client(String name, String company) {*

*super(name);*

*this.bank = company;*

*}*

*public void display(){*

*System.out.printf("Client Name: %s \t Bank: %s \n", super.getName(), bank);*

*}*

*}*

Другим хрестоматийным примером является система геометрических фигур. В реальности не существует геометрической фигуры как таковой. Есть круг, прямоугольник, квадрат, но просто фигуры нет. Однако же и круг, и прямоугольник имеют что-то общее и являются фигурами:

*// абстрактный класс фигуры*

*abstract class Figure{*

*float x; // x-координата точки*

*float y; // y-координата точки*

*Figure(float x, float y){*

*this.x=x;*

*this.y=y;*

*}*

*// абстрактный метод для получения периметра*

*public abstract float getPerimeter();*

*// абстрактный метод для получения площади*

*public abstract float getArea();*

*}*

*// производный класс прямоугольника*

*class Rectangle extends Figure*

*{*

*private float width;*

*private float height;*

*// конструктор с обращением к конструктору класса Figure*

*Rectangle(float x, float y, float width, float height){*

*super(x,y);*

*this.width = width;*

*this.height = height;*

*}*

*public float getPerimeter(){*

*return width \* 2 + height \* 2;*

*}*

*public float getArea(){*

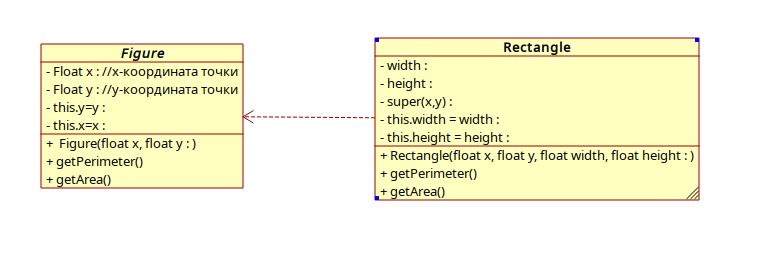
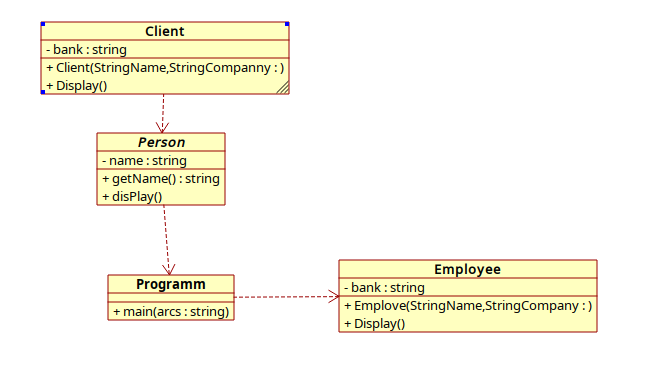
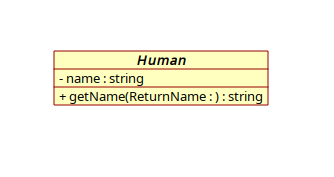
*return width \* height;*

*}*

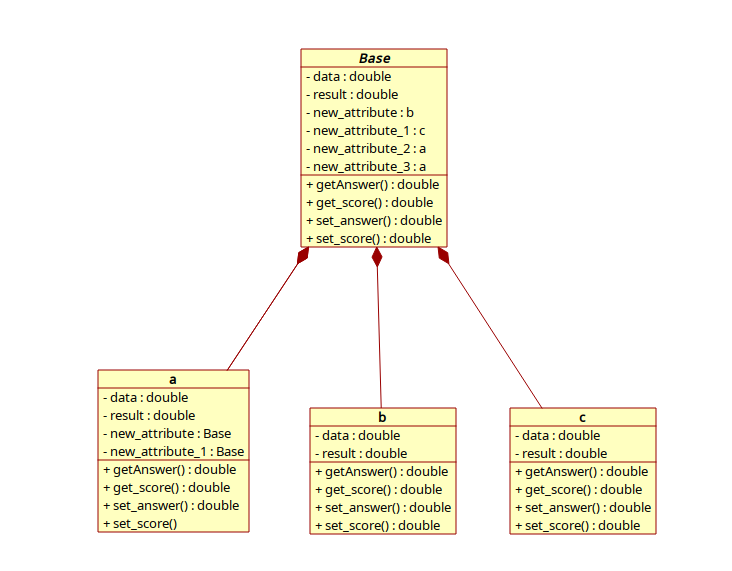
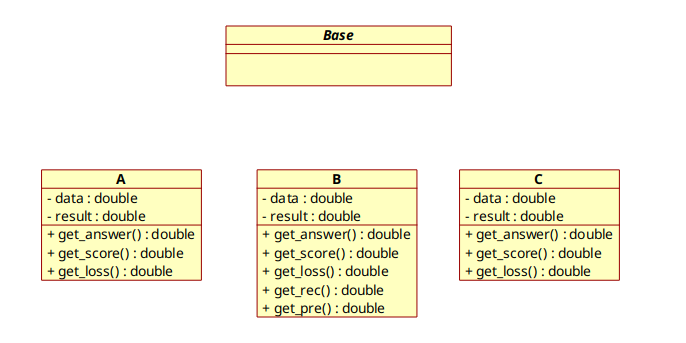
*}*

1. **Индивидуальные задания**

1. Разработать диаграммы UML для примеров из теоретических материалов



2. На диаграмме вам даны 3 класса A, B, C, имеющие сходный (но не одинаковый) интерфейс. Вам необходимо создать абстрактный базовый класс Base и построить корректную схему наследования. При выполнении следует избегать дублирования кода, и стараться следовать SOLID принципам ООП.

Вывод: Мы научились создавать декомпозицию программы