ЛАБОРАТОРНАЯ РАБОТА №3
НАСЛЕДОВАНИЕ В JAVA

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Цель данной лабораторной работы - изучить понятие наследования, и научиться реализовывать наследование в Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Одним из ключевых аспектов объектно-ориентированного программирования является наследование. С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого. Например, имеется следующий класс Person, описывающий отдельного человека:

```
public class Person {

    private String name;
    private String surname;

    public String getName() { return name; }
    public String getSurname() { return surname; }

    public Person(String name, String surname){

        this.name=name;
        this.surname=surname;
    }

    public void displayInfo(){

        System.out.println("Имя: " + name + " Фамилия: " + surname);
    }
}
```

И, возможно, впоследствии мы решили расширить имеющуюся систему и классов, добавив в нее класс, описывающий

сотрудника предприятия - класс Employee. Так как этот класс реализует тот же функционал, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником) от класса Person, который, в свою очередь, называется базовым классом или родителем:

```
class Employee extends Person{

}
```

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово extends, после которого идет имя базового класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же поля и методы, которые есть в классе Person.

В классе Employee могут быть определены свои методы и поля, а также конструктор. Способность к изменению функциональности, унаследованной от базового класса, назвывается полиморфизмом и является одинм из ключевых аспектов объектно-ориентированного программирования наряду с наследованием и инкапсуляцией.

Например, переопределим метод displayInfo() класса Person в классе Employee:

```
class Employee extends Person{

    private String company;

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

        super(name, surname);
        this.company=company;
    }

    public void displayInfo(){

        super.displayInfo();
        System.out.println("Компания: " + company);
```

```
    }
}
```

Класс Employee определяет дополнительное поле для хранения компании, в которой работает сотрудник. Кроме того, оно также устанавливается в конструкторе.

Так как поля name и surname в базовом классе Person объявлены с модификатором private, то мы не можем к ним напрямую обратиться из класса Employee. Однако в данном случае нам это не нужно. Чтобы их установить, мы обращаемся к конструктору базового класса с помощью ключевого слова super, в скобках после которого идет перечисление передаваемых аргументов.

С помощью ключевого слова super мы можем обратиться к любому члену базового класса - методу или полю, если они не определены с модификатором private.

Также в классе Employee переопределяется метод displayInfo() базового класса. В нем с помощью ключевого super также идет обращение к методу displayInfo(), но уже базового класса, и затем выводится дополнительная информация, относящаяся только к Employee.

Используя обращение к методом базового класса, можно было бы переопределить метод displayInfo() следующим образом:

```
public void displayInfo(){

    System.out.println("Имя: " + super.getName() + " Фамилия: "
        + super.getSurname() + " Компания: " + company);
}
```

При этом нам необязательно переопределять все методы базового класса. Например, в данном случае мы не переопределяем методы getName() и getSurname(). Поэтому для этих методов класс-наследник будет использовать реализацию из базового класса. И в основной программе мы можем эти методы использовать:

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

    Employee empl = new Employee("Tom", "Simpson", "Oracle");
```

```
    empl.displayInfo();
    String firstName = empl.getName();
    System.out.println(firstName);
}
```

Хотя наследование очень интересный и эффективный механизм, но в некоторых ситуациях его применение может быть нежелательным. И в этом случае можно запретить наследование с помощью ключевого слова final. Например:

```
public final class Person {
}
```

Если бы класс Person был бы определен таким образом, то следующий код был бы ошибочным и не сработал, так как мы тем самым запретили наследование:

```
class Employee extends Person{
}
```

Кроме запрета наследования можно также запретить переопределение отдельных методов. Например, в примере выше переопределен метод displayInfo(), запретим его переопределение:

```
public class Person {

    //.....................

    public final void displayInfo(){

        System.out.println("Имя: " + name + " Фамилия: " + surname);
    }
}
```

В этом случае в классе Employee надо будет создать метод с другим именем для вывода информации об объекте.

Абстрактные классы

Кроме обычных классов в Java есть абстрактные классы. Абстрактный класс похож на обычный класс. В абстрактном классе также можно определить поля и методы, в то же время нельзя создать объект или экземпляр абстрактного класса. 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 int height;
    private double weight;

    public int getHeight() { return height; }
    public double getWeight() { return weight; }
}
```

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

```
public abstract void displayInfo();
```

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 обсулживания 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 abstract class Person {

    private String name;
    private String surname;
```

```
    public String getName() { return name; }
    public String getSurname() { return surname; }

    public Person(String name, String surname){

        this.name=name;
        this.surname=surname;
    }

    public abstract void displayInfo();
}

class Employee extends Person{

    private String bank;

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

        super(name, surname);
        this.bank=company;
    }

    public void displayInfo(){

        System.out.println("Имя: " + super.getName() + " Фамилия: "
                + super.getSurname() + " Работает в банке: " + bank);
    }
}

class Client extends Person
{
    private String bank;

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

        super(name, surname);
```

```
        this.bank=company;
    }

    public void displayInfo(){

        System.out.println("Имя: " + super.getName() + " Фамилия: "
                + super.getSurname() + " Клиент банка: " + bank);
    }
}
```

ВАРИАНТЫ ЗАДАНИЙ

1. Создать абстрактный класс, описывающий посуду(Dish). С помощью наследования реализовать различные виды посуды, имеющие свои свойства и методы. Протестировать работу классов.

2. Создать абстрактный класс, описывающий собак(Dog). С помощью наследования реализовать различные породы собак. Протестировать работу классов.

3. Создать абстрактный класс, описывающий мебель. С помощью наследования реализовать различные виды мебели. Также создать класс FurnitureShop, моделирующий магазин мебели. Протестировать работу классов.