ЛАБОРАТОРНАЯ РАБОТА №2.
Проектирование классов

1. Цель работы

Получить основные понятия по следующим разделам языка Java:
- объектно-ориентированное программирование;
- создание объектов и классов из стандартной библиотеки Java;
- создание собственных классов.

2. Методические указания

Лабораторная работа направлена на приобретение навыков конструирования и реализации объектов на языке Java.

Требования к результатам выполнения лабораторного практикума:
- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- по завершении выполнения задания составить отчет о проделанной работе.

3. Теоретический материал

Объектно-ориентированное программирование (ООП) в настоящее время стало доминирующей парадигмой программирования, вытеснив «структурные», процедурно-ориентированные подходы, разработанные в 1970 годах. *Java* представляет собой полностью объектно-ориентированный язык.

Объектно-ориентированная программа состоит из объектов. Каждый объект имеет определенную функциональность, которую предоставляет в распоряжение пользователей, а также скрытую реализацию. Многие объекты программ могут быть взяты программистами в готовом виде из стандартных пакетов Java, а некоторые написаны самостоятельно.

Классы и объекты

Класс – это шаблон, или проект, по которому будет сделан объект. При разработке собственных классов необходимо пользоваться абстрагированием от совокупности свойств реальных предметов, и выбирать только те характеристики и свойства предмета, которые удовлетворяют семантике решаемой задачи. Например, если разрабатывается информационная система, которая отвечает за распечатывание квитков при получении сотрудниками заработной платы, то для описания сотрудника в такой системе может быть достаточно указать ФИО сотрудника, размер и дату выдачи заработной платы. Если же речь идет о информационной системе по регистрации и обслуживанию больных в поликлинике, то значения размера и даты выдачи заработной платы больных не имеют никакого значения, а вот информация о месте их проживания, дате рождения и истории болезни играют существенную роль.

Объявление класса на языке *Java* может быть сделано следующим образом:

модификатор class class_name [extends parent_class]{*//тело класса*
//объявление полей класса
тип имя_поля; *//свойства (поля) класса class_name*
//объявление конструктора класса
модификатор class_name(аргументы){тело конструктора};
//объявление методов класса
модификатор тип method_name (аргументы){тело метода};
}

Например, класс сотрудник *Employee* можно описать следующим образом:

package by.bsac.lab2;

```
import java.util.Date;
import java.util.GregorianCalendar;

public class Employee {
   // перечисление полей класса
   private String name; // имя
   private double salary; // размер заработной платы
   private Date hiredate; // дата приема на работу

   // конструктор класса, задача которого – присвоение значений полям класса
   public Employee(String n, double s, int year, int month, int day) {
      name = n;
      salary = s;
      hiredate = (new GregorianCalendar(year, month - 1, day)).getTime();
   }

   // методы класса
   public String getName() {
      // возвращает имя сотрудника
      return name;
   }

   public double getSalary() {
      // возвращает размер заработной платы сотрудника
      return salary;
   }

   public Date getDate() {
      // возвращает дату приема на работу
      return hiredate;
   }
}
```

Объект - это мыслимая или реальная сущность, обладающая характерным поведением, отличительными характеристиками и являющаяся важной в предметной области. Объектом является экземпляр класса созданный путем вызова конструктора класса. Каждый объект обладает состоянием, поведением и уникальностью.

Состояние (state) - совокупный результат поведения объекта, одно из стабильных условий, в которых объект может существовать. В любой конкретный момент времени состояние объекта включает в себя перечень свойств объекта и текущие значения этих свойств.

Поведение (behavior) - действия и реакции объекта, выраженные в терминах передачи сообщений и изменения состояния; видимая извне и воспроизводимая активность объекта.

Уникальность (*identity*) - природа объекта; то, что отличает его от других объектов.

Объекты и объектные переменные

Чтобы работать с объектами, их нужно сначала создать и задать исходное состояние. Затем к этим объектам применяются методы. В языке *Java* для создания новых экземпляров используются конструкторы.

Конструктор – специальный метод, предназначенный для создания и инициализации экземпляра класса. Имя конструктора всегда совпадает с именем класса. Следовательно, конструктор класса *Employee* называется *Employee* и объявляется как:

```
public Employee(String n, double s, int year, int month, int day){
name=n;
salary=s;
hiredate=(new GregorianCalendar(year,month-1,day)).getTime();
}
```

В одном классе может быть объявлено несколько конструкторов, если их сигнатуры разные. Например, можно создать конструктор в классе *Employee*, который принимает только имя сотрудника и устанавливает ему заработную плату 1 условная единица. Дата выхода на работу всем таким сотрудникам будет установлена 31 декабря 2015 года.

```
public Employee(String n){
name=n;
salary=1;
hiredate=(new GregorianCalendar(2015,12,31)).getTime();
}
```

Для создания объекта необходимо объявить объектную переменную, затем вызвать конструктор класса. Например, объявим две переменные *e1* и *e2* с типом *Employee*. Создадим двух сотрудников в информационной системе с помощью вызова различных конструкторов.

```
Employee e1 = new Employee("James Bond", 100000, 1950, 1, 1);
Employee e2 = new Employee("James NeBond");
```

В результате вызова конструкторов сотрудник James Bond был принят на работу 1 января 1950 года с заработной платой 100000 у.е. (ссылка на объект сохранена в объектной переменной e1), сотрудник James NeBond был принят на работу 31 декабря 2015 года с заработной платой 1 у.е. (ссылка на объект сохранена в объектной переменной e2), т.к. для его создания использовался второй конструктор, который принимает одно-единственное значение.

Основные понятия ООП – инкапсуляция, наследование и полиморфизм

Наследование (*inheritance*) - это отношение между классами, при котором класс использует структуру или поведение другого (одиночное наследование) или других (множественное наследование) классов.

Наследование вводит иерархию "общее/частное", в которой подкласс наследует от одного или нескольких более общих суперклассов. Подклассы обычно дополняют или переопределяют унаследованную структуру и поведение.

Расширим класс *Employee* следующим образом. Необходимо описать класс, экземпляры которого представляли бы менеджера предприятия.

Менеджер является таким же сотрудником, однако у него есть дополнительное поле – премия. Соответственно, метод, который возвращал в классе сотрудник *Employee* размер заработной платы, больше не подходит для менеджера.

```
package by.bsac.lab2;

class Manager extends Employee { // наследование от класса сотрудник
   private double bonus; // размер премии

// конструктор класса
   public Manager(String n, double s, int year, int month, int day) {
      super(n, s, year, month, day);
// т.к. класс унаследован от другого класса,
// то первой командой в конструкторе класса-
```

```
// потомка необходимо вызвать конструктор
// родителя. Т.к. в скобках после super указано 5
// аргументов, то будет вызван первый
// конструктор Employee
      bonus = 0;
   }
         }
```

Теперь каждый экземпляр класса *Manager* имеет 4 поля – *name, salary, hiredate, bonus.* Определяя подкласс, нужно указать лишь отличия между подклассом (потомком) и суперклассом (родителем). Разрабатывая классы, следует помещать методы общего назначения в суперкласс, а более специальные – в подкласс.

В приведенном выше примере не все методы родительского класса *Employee* подходят для класса *Manager*. В частности, метод *getSalary()* должен возвращать сумму базовой зарплаты и премии. Следовательно, нужно реализовать новый метод, замещающий (*overriding*) метод класса родителя. Сделать это можно добавив метод *getSalary()* в класс *Manager* следующим образом:

```
class Manager extends Employee{
....
public double getSalary() {новое тело метода….}
// перекрытие (замещение)
//метода класса родителя
....
}
```

Новый (замещенный) метод будет выглядеть так:

```
 public double getSalary() {
double basesalary = super.getSalary();
return basesalary + bonus;
    }
```

Инкапсуляция (*encapsulation*) - это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса). При использовании объектно-ориентированного подхода не принято использовать прямой доступ к свойствам какого-либо класса из методов других классов. Для доступа к свойствам класса принято использовать специальные методы этого класса для получения и изменения его свойств.

Открытые члены класса составляют внешний интерфейс объекта. Эта та функциональность, которая доступна другим классам. Закрытыми обычно объявляются все свойства класса, а так же вспомогательные методы, которые являются деталями реализации и от которых не должны зависеть другие части системы. Благодаря сокрытию реализации за внешним интерфейсом класса можно менять внутреннюю логику отдельного класса, не меняя код остальных компонентов системы.

Полиморфизм (*polymorphism*) - положение теории типов, согласно которому имена (например, переменных) могут обозначать объекты разных (но имеющих общего родителя) классов. Следовательно, любой объект, обозначаемый полиморфным именем, может по-своему реагировать на некий общий набор операций.

Абстрактный класс

Абстрактные классы используются, чтобы создать класс с реализацией метода по умолчанию для подклассов. Абстрактный класс может иметь абстрактные методы как с реализацией, так и без реализации.

Чтобы создать абстрактный класс, нужно дописать ему ключевое слово *abstract* при объявлении класса. Нельзя создать экземпляр абстрактного класса, поэтому такие классы

являются базой для подклассов, которые реализуют абстрактные методы и переопределяют или используют реализованные методы абстрактного класса.

```
package by.bsac.lab2;

//абстрактный класс
public abstract class Person {
    private String name;
    private String gender;

    public Person(String nm, String gen){
        this.name=nm;
        this.gender=gen;
    }

    //абстрактный метод
    public abstract void work();

    @Override
    public String toString(){
        return "Имя: " + this.name + "Пол: " + this.gender;
    }
}
```

Обратите внимание, что *work()* - это абстрактный метод без реализации.

Ход работы. Полный текст программы:

#class Person

```
package by.bsac.lab2.model;

public abstract class Person {
    private String name;
    private String gender;

    public Person(String name, String gender) {
        this.name = name;
        this.gender = gender;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
```

```
    }

    public abstract void work();

    public abstract void getDescription();

    @Override
    public String toString() {
        return "Имя: " + this.name + "Пол: " + this.gender;
    }
}
```

#class Employee

```
package by.bsac.lab2.model;

import java.util.Date;
import java.util.GregorianCalendar;

public class Employee extends Person {
    private double salary;
    private Date hireDate;

    public Employee(String name, String gender, double salary, int year, int month, int day) {
        super(name, gender);
        this.salary = salary;
        GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
        hireDate = calendar.getTime();
    }

    public double getSalary() {
        return salary;
    }

    public Date getHireDate() {
        return hireDate;
    }

    public void raiseSalary(double byPercent) {
        double raise = salary * byPercent / 100;
        salary += raise;
    }

    @Override
    public void work() {
        System.out.println("Working...");
    }

    @Override
    public void getDescription() {
        System.out.println("Employee with name=" + getName() + " with salary=" + getSalary());
    }
}
```

#class Manager

```
package by.bsac.lab2.model;

public class Manager extends Employee {
    private double bonus;

    public Manager(String name, String gender, double salary, int year, int month, int day) {
        super(name, gender, salary, year, month, day);
        bonus = 0;
    }

    public double getSalary() {
        double basesalary = super.getSalary();
        return basesalary + bonus;
    }

    public void setBonus(double bonus) {
        this.bonus = bonus;
    }

    @Override
    public void getDescription() {
        System.out.println("Manager with name=" + getName() + " with salary=" + getSalary());
    }
}
```

#class Student

```
package by.bsac.lab2.model;

public class Student extends Person {
    private int course;

    public Student(String name, String gender, int course) {
        super(name, gender);
        this.course = course;
    }

    public int getCourse() {
        return course;
    }

    @Override
    public void work() {
        System.out.println("Studying...");
    }

    @Override
    public void getDescription() {
        System.out.println("Student with name=" + getName() + " on course=" + getCourse());
```

```
    }
}
```

#class ManagerTest

```
package by.bsac.lab2.runner;

import by.bsac.lab2.model.Employee;
import by.bsac.lab2.model.Manager;

public class ManagerTest {

    public static void main(String[] args) {
        // construct a Manager object
        Manager boss = new Manager("Василий Петров", "муж", 20000, 2010, 9, 1);
        boss.setBonus(5000);

        Employee[] staff = new Employee[3];
        // fill the staff array with Manager and Employee objects
        staff[0] = boss;
        staff[1] = new Employee("Петр Соколов", "муж", 15000, 2012, 10, 1);
        staff[2] = new Employee("Иван Васильев", "муж", 10000, 2015, 3, 15);
        // print out information about all Employee objects
        for (Employee e : staff) {
            e.getDescription();
        }
    }
}
```

Рекомендации по проектированию классов

- Всегда храните данные в переменных, объявленных как *private*;
- Всегда инициализируйте данные;
- Не используйте в классе слишком много простых типов;
- Не для всех полей надо создавать методы доступа и модификации;
- Используйте стандартную форму определения класса;
- Разбивайте на части слишком большие классы;
- Выбирайте для классов и методов осмысленные имена.

4. Порядок выполнения работы

- изучить предлагаемый теоретический материал;
- напишите программы на языке *Java* в соответствии с вашим вариантом.

Создать консольное приложение, удовлетворяющее следующим требованиям:

• Использовать возможности ООП: классы, наследование, полиморфизм, инкапсуляция.

• Каждый класс должен иметь отражающее смысл название и информативный состав.

• Наследование должно применяться только тогда, когда это имеет смысл.

• При кодировании должны быть использованы соглашения об оформлении кода *java code convention*.

• Классы должны быть грамотно разложены по пакетам.

• Консольное меню должно быть минимальным.

• Для хранения параметров инициализации можно использовать файлы.

+ 1 коллекция, 1 интерфейс

5. Индивидуальные задания

1. Цветочница. Определить иерархию цветов. Создать несколько объектов-цветов. Собрать букет (используя аксессуары) с определением его стоимости. Провести сортировку цветов в букете на основе уровня свежести. Найти цветок в букете, соответствующий заданному диапазону длин стеблей.

2. Новогодний подарок. Определить иерархию конфет и прочих сладостей. Создать несколько объектов-конфет. Собрать детский подарок с определением его веса. Провести сортировку конфет в подарке на основе одного из параметров. Найти конфету в подарке, соответствующую заданному диапазону содержания сахара.

3. Домашние электроприборы. Определить иерархию электроприборов. Включить некоторые в розетку. Подсчитать потребляемую мощность. Провести сортировку приборов в квартире на основе мощности. Найти прибор в квартире, соответствующий заданному диапазону параметров.

4. Шеф-повар. Определить иерархию овощей. Сделать салат. Подсчитать калорийность. Провести сортировку овощей для салата на основе одного из параметров. Найти овощи в салате, соответствующие заданному диапазону калорийности.

5. Звукозапись. Определить иерархию музыкальных композиций. Записать на диск сборку. Подсчитать продолжительность. Провести перестановку композиций диска на основе принадлежности к стилю. Найти композицию, соответствующую заданному диапазону длины треков.

6. Камни. Определить иерархию драгоценных и полудрагоценных камней. Отобрать камни для ожерелья. Подсчитать общий вес (в каратах) и стоимость. Провести сортировку камней ожерелья на основе ценности. Найти камни в ожерелье, соответствующие заданному диапазону параметров прозрачности.

7. Мотоциклист. Определить иерархию амуниции. Экипировать мотоциклиста. Подсчитать стоимость. Провести сортировку амуниции на основе веса. Найти элементы амуниции, соответствующие заданному диапазону параметров цены.

8. Транспорт. Определить иерархию подвижного состава железнодорожного транспорта. Создать пассажирский поезд. Подсчитать общую численность пассажиров и багажа. Провести сортировку вагонов поезда на основе уровня комфортности. Найти в поезде вагоны, соответствующие заданному диапазону параметров числа пассажиров.

9. Авиакомпания. Определить иерархию самолетов. Создать авиакомпанию. Посчитать общую вместимость и грузоподъемность. Провести сортировку самолетов компании по дальности полета. Найти самолет в компании, соответствующий заданному диапазону параметров потребления горючего.

10. Таксопарк. Определить иерархию легковых автомобилей. Создать таксопарк. Подсчитать стоимость автопарка. Провести сортировку автомобилей парка по расходу топлива. Найти автомобиль в компании, соответствующий заданному диапазону параметров скорости.

11. Страхование. Определить иерархию страховых обязательств. Собрать из обязательств деривати. Подсчитать стоимость. Провести сортировку обязательств в деривативе на основе уменьшения степени риска. Найти обязательство в деривативе, соответствующее заданному диапазону параметров.

12. Мобильная связь. Определить иерархию тарифов мобильной компании. Создать список тарифов компании. Подсчитать общую численность клиентов. Провести сортировку тарифов на основе размера абонентской платы. Найти тариф в компании, соответствующий заданному диапазону параметров.

13. Фургон кофе. Загрузить фургон определенного объема грузом на определенную сумму из различных сортов кофе, находящихся, к тому же, в разных физических состояниях (зерно, молотый, растворимый в банках и пакетиках). Учитывать объем кофе вместе с упаковкой. Провести сортировку товаров на основе соотношения цены и веса. Найти в фургоне товар, соответствующий заданному диапазону параметров качества.

14. Игровая комната. Подготовить игровую комнату для детей разных возрастных групп. Игрушек должно быть фиксированное количество в пределах выделенной суммы денег. Должны встречаться игрушки родственных групп: маленькие, средние и большие машины, куклы, мячи, кубики. Провести сортировку игрушек в комнате по одному из параметров. Найти игрушки в комнате, соответствующие заданному диапазону параметров.

15. Налоги. Определить множество и сумму налоговых выплат физического лица за год с учетом доходов с основного и дополнительного мест работы, авторских вознаграждений, продажи имущества, получения в подарок денежных сумм и имущества, переводов из-за границы, льгот на детей и материальной помощи. Провести сортировку налогов по сумме.

16. Счета. Клиент может иметь несколько счетов в банке. Учитывать возможность блокировки/разблокировки счета. Реализовать поиск и сортировку счетов. Вычисление общей суммы по счетам. Вычисление суммы по всем счетам, имеющим положительный и отрицательный балансы отдельно.

17. Туристические путевки. Сформировать набор предложений клиенту по выбору туристической путевки различного типа (отдых, экскурсии, лечение, шопинг, круиз и т. д.) для оптимального выбора. Учитывать возможность выбора транспорта, питания и числа дней. Реализовать выбор и сортировку путевок.

18. Кредиты. Сформировать набор предложений клиенту по целевым кредитам различных банков для оптимального выбора. Учитывать возможность досрочного погашения кредита и/или увеличения кредитной линии. Реализовать выбор и поиск кредита.

19. Комплектующие PC. Определить иерархию комплектующих PC. Создать объекты комплектующих с различными характеристиками. Собрать из комплектующих готовые PC. Найти PC с наибольшей частотой процессора.

20. Академия. Определить иерархию внутри академии. Создать несколько объектов (преподаватель, учащийся и т.д.). Собрать из учащихся несколько групп. Провести сортировку групп по количеству студентов. Найти учащегося с наибольшим средним балом.

21. Академия. Определить иерархию внутри академии. Создать несколько объектов преподавателей, заведующих кафедрой, декан. Собрать составы нескольких кафедр. Провести сортировку кафедр на основе численности преподавателей.

22. Дом. Определить иерархию дома. Создать несколько объектов-квартир. Собрать дом из этих квартир. Провести сортировку квартир в букете на основе количества комнат. Найти квартиру в доме, соответствующую заданному количеству комнат.

23. Квартира. Определить иерархию квартиры. Создать несколько объектов-комнат. Собрать квартиру. Провести сортировку комнат в квартире на основе их размеров (м2). Найти комнату в квартире, соответствующую заданному размеру.

24. Страна. Определить иерархию городов. Создать несколько объектов-городов. Собрать страну. Провести сортировку городов в стране на основе количества жителей. Найти города в стране, соответствующие заданному диапазону количества жителей.

25. Звездная система. Определить иерархию звездной системы. Создать несколько объектов-планет. Собрать звездную систему. Провести сортировку планет на основе удаленности от солнца. Найти планету в заданном диапазоне дистанции от солнца.

26.Фотоальбом. Определить иерархию фотоальбома. Создать несколько объектов-фотографий. Собрать фотоальбом. Провести сортировку страниц в фотоальбоме на основе количества фотографий на них. Найти страницу в фотоальбоме, с наибольшим количеством фотографий.

27. Год. Определить иерархию года. Создать несколько объектов-месяцев. Собрать год. Провести сортировку месяцев в году на основе средней температуры. Найти месяц в году, соответствующий заданному диапазону температур.

28. Улица. Определить иерархию улицы. Создать несколько объектов-домов.

Собрать улицу. Провести сортировку домов на улице на основе количества квартир. Найти дом на улице, соответствующий заданному диапазону количества квартир.

29. Город. Определить иерархию города. Создать несколько объектов-районов. Собрать город. Провести сортировку районов в букете на основе количества домов. Найти район в городе, соответствующий заданному диапазону количества домов.

30. Корабль. Определить иерархию корабля. Создать несколько объектов-кают. Собрать корабль. Провести сортировку кают в букете на основе количества коек. Найти каюту на коробе, соответствующую заданному числу коек.

6. Контрольные вопросы

1. Дайте определение Классу. Приведите пример объявления класса на языке Java.
2. Перечислите и дайте определение основным понятиям объектно-ориентированного программирования.
3. Для чего используется абстрактный класс?