**Лабораторная работа 4**

**Проектирование классов**

**Цель работы:**

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

**Задание:**

Комплектующие PC.

1. Определить иерархию комплектующих PC.
2. Создать объекты комплектующих с различными характеристиками.
3. Собрать из комплектующих готовые PC.
4. Найти PC c наибольшей частотой процессора.

**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.

**Порядок выполнения работы:**

1. Ознакомтесь с теоретическим материалом (на примере Java);
2. Напишите программы по заданию:

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 (для Java- *java code convention)*.  
• Классы должны быть грамотно разложены по пакетам.  
• Консольное меню должно быть минимальным.  
• Для хранения параметров инициализации можно использовать файлы.

**Теоретический материал:**

***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.

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.lab6;*  
*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.

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 01 сентября 2021 года.

*public Employee(String n){  
name=n;  
salary=1;  
hiredate=(new GregorianCalendar(2021,09,1)).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 100 000 у.е. (ссылка на объект сохранена в объектной переменной e1), сотрудник James NeBond был принят на работу 01 сентября 2021 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.lab6;  
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 объектноориентированного подхода не принято использовать прямой доступ к свойствам какоголибо 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 посвоему реагировать на некий общий набор операций.

***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.lab6;  
//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.

**Пример:** текст программы:

**#class Person**package by**.**bsac**.**lab6**.**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**.**lab6**.**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**.**lab6**.**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**.**lab6**.**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**.**lab6**.**runner**;  
import** by**.**bsac**.**lab6**.**model**.**Employee**;  
import** by**.**bsac**.**lab6**.**model**.**Manager**;**public class ManagerTest **{**public static void main**(**String**[]** args**) {**// construct a Manager object  
Manager boss **= new** Manager**(**"Василий Петров"**,** "муж"**,** 20000**,** 2020**,** 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**,** 2019**,** 10**,** 1**);**staff**[**2**] = new** Employee**(**"Иван Васильев"**,** "муж"**,** 10000**,** 2018**,** 3**,** 15**);**// print out information about all Employee objects  
**for (**Employee e **:** staff**) {**e**.**getDescription**();  
}  
}  
}**