

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СЕРВИСА»
(ФГБОУ ВО «ПВГУС»)

Кафедра «Прикладная информатика в экономике»

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине «Язык программирования Java и средства NetBeans»

Одобрено
Учебно-методическим
Советом университета

Составитель **Малышева Е. Ю.**

Тольятти 2016

УДК 002.52/54(075.8)
ББК -018*32.973я7
Л 12

Рецензент
к.т.н., доц. **Бобровский С. М.**

Л 12 Лабораторный практикум по дисциплине «Язык програм-
мирования Java и средства NetBeans» / сост. Е. Ю. Малышева. –
Тольятти : Изд-во ПВГУС, 2016. – 48 с.

Лабораторный практикум по дисциплине «Язык программирования Java и
средства NetBeans» разработан в соответствии с требованиями ФГОС ВО

УДК 002.52/54(075.8)
ББК -018*32.973я7

© Малышева Е. Ю., составление, 2016
© Поволжский государственный
университет сервиса, 2016

Лабораторная работа № 3. Наследование классов

Цель работы: Научиться создавать приложение с наследованием классов в Java

Задание: разработать приложение в соответствии с указанной моделью классов:

Указания к работе:

Инкапсуляция подразумевает скрытие полей класса и организацию доступа к ним через его методы. Наследование опирается на инкапсуляцию. Оно позволяет строить на основе первоначального класса новые, добавляя в классы новые поля данных и методы. Первоначальный класс называется прародителем (ancestor), новые классы - его потомками (descendants). От потомков, в свою очередь, можно наследовать, получая очередных потомков. И так далее.

Набор классов, связанных отношением наследования, называется иерархией классов. А класс, стоящий во главе иерархии, от которого унаследованы все остальные (прямо или опосредованно), называется базовым классом иерархии.

В Java все классы являются потомками класса Object. То есть он является базовым для всех классов. Тем не менее, если рассматривается поведение, характерное для объектов какого-то класса и всех потомков этого класса, говорят об иерархии, начинающейся с этого класса. В этом случае именно он является базовым классом иерархии.

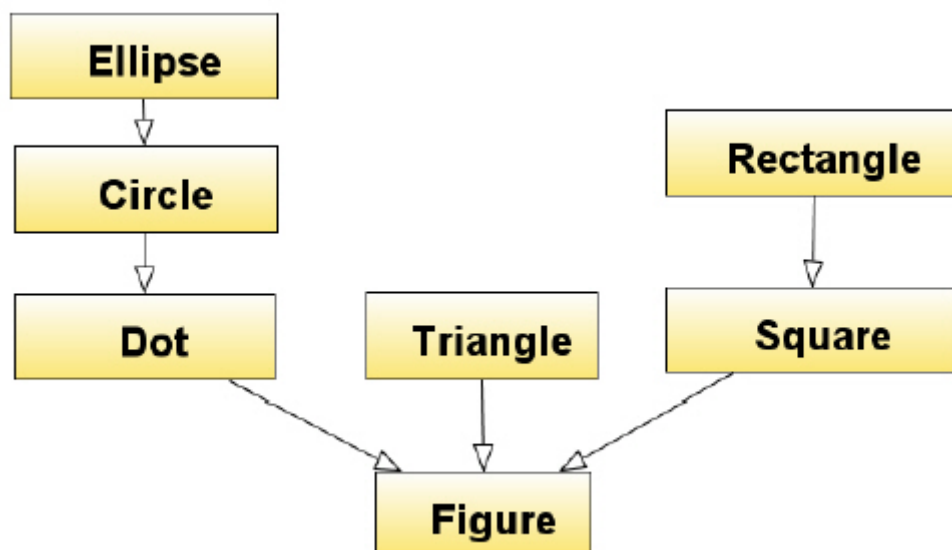
Полиморфизм опирается как на инкапсуляцию, так и на наследование. Как показывает опыт преподавания, это наиболее сложный для понимания принцип. Слово "полиморфизм" в переводе с греческого означает "имеющий много форм". В объектном программировании под полиморфизмом подразумевается наличие кода, написанного для объектов, имеющих тип базового класса иерархии. При этом такой код должен правильно работать для любого объекта, являющегося экземпляром класса из данной иерархии. Независимо от того, где этот класс расположен в иерархии. Такой код и называется полиморфным. При написании полиморфного кода заранее неизвестно, для объектов какого типа он будет работать - один и тот же метод будет исполняться по-разному в зависимости от типа объекта. Пусть, например, у нас имеется класс Figure - "фигура", и в нем заданы методы show() - показать фигуру на экране, и hide() - скрыть ее. Тогда для переменной figure типа Figure вызовы figure.show() и figure.hide() будут показывать или скрывать объект, на который ссылается эта переменная. Причем сам объект "знает", как себя показывать или скрывать, а код пишется на уровне абстракций этих действий.

Основное преимущество объектного программирования по сравнению с процедурным как раз и заключается в возможности написания полиморфного кода. Именно для этого пи-

шется иерархия классов. Полиморфизм позволяет резко увеличить коэффициент повторного использования программного кода и его модифицируемость по сравнению с процедурным программированием.

В качестве примера того, как строится иерархия, рассмотрим иерархию фигур, отрисовываемых на экране - она показана на рисунке. В ней базовым классом является Figure, от которого наследуются Dot - "точка", Triangle - "треугольник" и Square - "квадрат". От Dot наследуется класс Circle - "окружность", а от Circle унаследуем Ellipse - "эллипс". И, наконец, от Square унаследуем Rectangle - "прямоугольник".

Отметим, что в иерархии принято рисовать стрелки в направлении от наследника к прародителю. Такое направление называется Generalization - "обобщение", "генерализация". Стрелки символизируют направление в сторону упрощения.



Часто класс-прародитель называют суперклассом (superclass), а класс-наследник - субклассом (subclass). Но такая терминология подталкивает начинающих программистов к неверной логике: суперкласс пытаются сделать "суперсложным". Так, чтобы его подклассы (это неверно воспринимается синонимом выражению "упрощенные разновидности") обладали упрощенным по сравнению с ним поведением. На деле же **потомки должны обладать более сложным устройством и поведением по сравнению прародителем**. Поэтому в данном учебном пособии предпочтение отдается терминам "прародитель" и "наследник".

Чем ближе к основанию иерархии лежит класс, тем более общим и универсальным (general) он является. И одновременно - более простым. Класс, который лежит в основе иерархии, называется базовым классом этой иерархии. Базовый класс всегда называют именем, которое характеризует все объекты - экземпляры классов-наследников, и которое выражает

наиболее общую абстракцию, применимую к таким объектам. В нашем случае это класс Figure. Любая фигура будет иметь поля данных x и y - координаты фигуры на экране.

Класс Dot ("точка") является наследником Figure, поэтому он будет иметь поля данных x и y , наследуемые от Figure. То есть в самом классе Dot задавать эти поля не надо. От Dot мы наследуем класс Circle ("окружность"), поэтому в нем также имеется поля x и y , наследуемые от Figure. Но появляется дополнительное поле данных. У Circle это поле, соответствующее радиусу. Мы назовем его r . Кроме того, для окружности возможна операция изменения радиуса, поэтому в ней может появиться новый метод, обеспечивающий это действие - назовем его `setSize` ("установить размер"). Класс Ellipse имеет те же поля данных и обеспечивает то же поведение, что и Circle, но в этом классе появляется дополнительное поле данных $r2$ - длина второй полуоси эллипса, и возможность регулировать значение этого поля. Возможен и другой подход, в некотором роде более логичный: считать эллипс сплюснутой или растянутой окружностью. В этом случае необходимо ввести коэффициент растяжения (*aspect ratio*). Назовем его k . Тогда эллипс будет характеризоваться радиусом r и коэффициентом растяжения k . Метод, обеспечивающий изменение k , назовем `stretch` ("растянуть"). Обратим внимание, что исходя из выбранной логики действий метод `scale` должен приводить к изменению поля r и не затрагивать поле k - поэтому эллипс будет масштабироваться без изменения формы.

Каждый из классов этой ветви иерархии фигур можно считать описанием "усложненной точки". При этом важно, что любой объект такого типа можно считать "точкой, которую усложнили". Грубо говоря, считать, что круг или эллипс - это такая "жирная точка". Аналогичным образом Ellipse является "усложненной окружностью"

Аналогично, класс Square наследует поля x и y , но в нем добавляется поле, соответствующее стороне квадрата. Мы назовем его a . У Triangle в качестве новых, не унаследованных полей данных могут выступать координаты вершин треугольника; либо координаты одной из вершин, длины прилежающих к ней сторон и угол между ними, и так далее.

Как располагать классы иерархии, базовый класс внизу, а наследники сверху, образуя ветви дерева наследования, или наоборот, базовый класс сверху а наследники внизу, образуя "корни" дерева наследования - принципиального значения не имеет. По-видимому, на начальном этапе развития объектного программирования применялся первый вариант, почему базовый класс, лежащий в основе иерархии, и получил такое название. Такой вариант выбран в данном учебном пособии, поскольку именно он используется в NetBeans Enterprise Pack. В настоящее время чаще используют второй вариант, когда базовый класс располагают сверху.

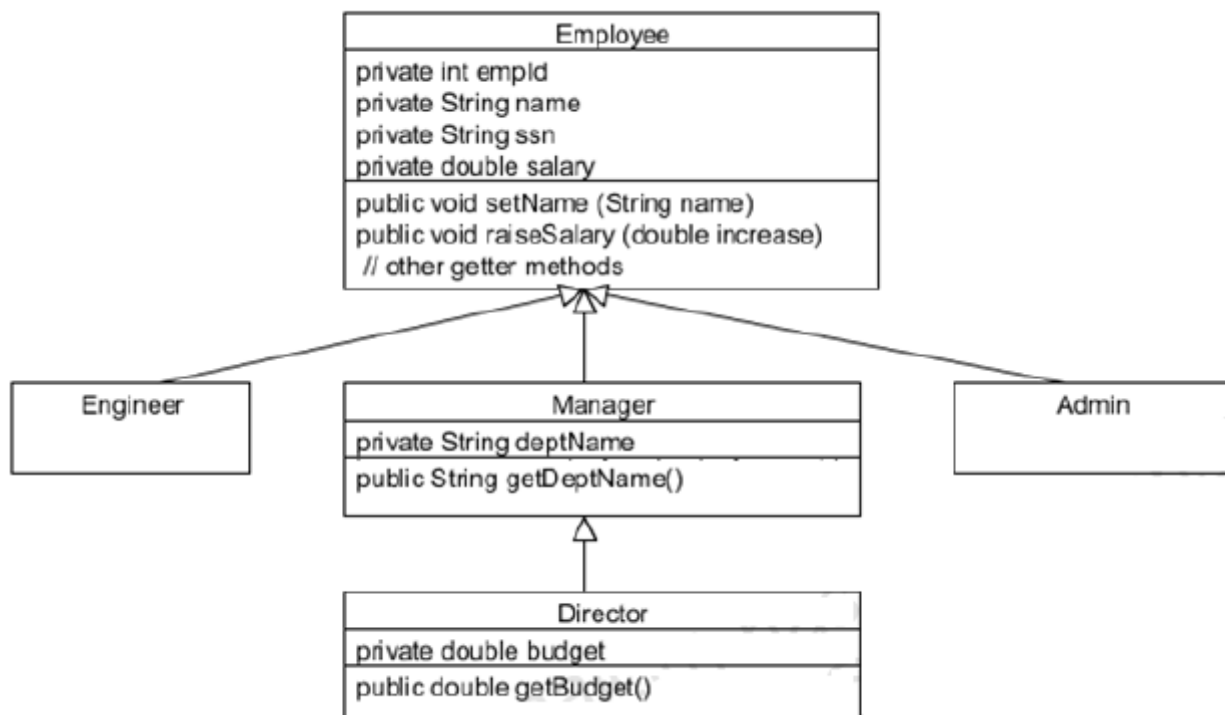
Каждый объект класса-потомка при любых значениях полей должен рассматриваться как экземпляр класса-прародителя, и с тем же поведением на уровне абстракции действий. Но только с некоторыми изменениями на уровне реализации этих действий. В концепции наследования основное внимание уделяется поведению объектов. Объекты с разным поведением имеют другой тип. А значения полей данных характеризуют состояние объекта, но не его тип.

По своему поведению любой объект-эллипс вполне может рассматриваться как экземпляр типа "Окружность" и даже вести себя в точности как окружность. Но не наоборот - объекты типа Окружность не обладает поведением Эллипса. Мы намеренно используем заглавные буквы для того, чтобы не путать классы с объектами.

Продумывание того, как устроены классы, то есть какие в них должны быть поля и методы (без уточнения об конкретной реализации этих методов), и описание того, какая должна быть иерархия наследования, называется проектированием. Это сложный процесс, и он обычно гораздо важнее написания конкретных операторов в реализации (кодирования).

Ход работы:

Разработаем приложения в соответствии со следующей моделью классов:



1. Скопируйте папку с проектом из предыдущей работы и переименуйте проект.
2. Примените *инкапсуляцию* к классу `Employee`. Для этого:
 - замените модификаторы доступа полей с `public` на `private`;

- замените конструктор без параметров на конструктор с параметрами

```
public Employee(int empId, String name, String ssn, double salary) {
    this.empId = empId;
    this.name = name;
    this.ssn = ssn;
    this.salary = salary;
}
```

- уберите все методы записи данных в поля («сеттеры»), кроме метода setName
- добавьте метод увеличения зарплаты raiseSalary:

```
public void raiseSalary(double increase){
    if (increase>0){
        salary += increase;
    }
}
```

3. Создайте в том же пакете подкласс класса Employee и назовите его Manager:

```
public class Manager extends Employee {

}
```

4. Добавьте в него поле deptName

```
private String deptName;
```

5. Добавьте конструктор класса с параметрами, который вызывает конструктор родительского класса и инициализирует значение поля deptName :

```
public Manager(int empId, String name, String ssn, double salary, String deptName) {
    super(empId, name, ssn, salary);
    this.deptName = deptName;
}
```

6. Добавьте в него метод чтения данных из поля getDeptName:

```
public String getDeptName() {
    return deptName;
}
```

7. Создайте в том же пакете подкласс класса `Employee` и назовите его `Admin`. Запишите в него конструктор с параметрами:

```
public class Admin extends Employee {  
  
    public Admin(int empId, String name, String ssn, double  
salary) {  
        super(empId, name, ssn, salary);  
    }  
  
}
```

8. Создайте в том же пакете подкласс класса `Employee` и назовите его `Engineer`. Запишите в него конструктор с параметрами:

```
public class Engineer extends Employee {  
    public Engineer(int empId, String name, String ssn, double  
salary) {  
        super(empId, name, ssn, salary);  
    }  
}
```

9. Создайте в том же пакете подкласс класса `Manager` и назовите его `Director`:

```
public class Director extends Manager {  
  
}
```

10. Добавьте в него поле `budget`

```
private double budget;
```

11. Добавьте конструктор класса с параметрами, который вызывает конструктор родительского класса и инициализирует значение поля `budget`:

```
public Director(int empId, String name, String ssn, double  
salary, String deptName, double budget) {  
    super(empId, name, ssn, salary, deptName);  
    this.budget = budget;  
}
```

12. Добавьте в него метод чтения данных из поля `budget`:

```
public double getBudget() {  
    return budget;  
}
```

13. Сохраните все классы

14. Добавьте в процедуру main класса EmployeeTest команды импорта созданных классов

```
import com.example.domain.Admin;
import com.example.domain.Director;
import com.example.domain.Engineer;
import com.example.domain.Manager;
```

15. Запишите в процедуру main класса EmployeeTest команды создания объектов созданных классов и заполнение их полей

```
Engineer eng = new Engineer(101, "Jane Smith", "012-34-5678",
120_345.27);
Manager mgr = new Manager(207, "Barbara Johnson", "054-12-
2367", 109_501.36, "US Marketing");
Admin adm = new Admin(304, "Bill Munroe", "108-23-2367",
75_002.34);
Director dir = new Director(12, "Susan Wheeler", "099-45-
2340", 120_567.36, "Global Marketing", 1_000_000.00);
```

16. Добавьте в класс EmployeeTest статический метод отображения данных объекта, представленного по ссылке класса Employee, родительского для всех вновь созданных классов

```
private static void printEmployee(Employee emp) {
    System.out.println("Employee ID: " + emp.getEmpId());
    System.out.println("Employee Name: " + emp.getName());
    System.out.println("Employee Soc Sec # " +
emp.getSsn());
    System.out.println("Employee salary: " +
emp.getSalary());
}
```

17. Добавьте в класс EmployeeTest команды отображения данных созданных объектов

```
printEmployee(eng);
printEmployee(mgr);
printEmployee(adm);
printEmployee(dir);
```

18. Сохраните все классы и запустите приложение.

Вопросы для самоконтроля

1. Что такое наследование?
2. Что такое иерархия классов?
3. Как наследуются поля и методы классов?

Варианты заданий

<i>Вариант</i>	<i>Класс</i>
1	Сотрудник, 3 поля – 3 класса наследника
2	Студент, 2 поля – 2 класса наследника
3	Товар, 3 поля– 4 класса наследника
4	Собака, 2 поля– 3 класса наследника
5	Геометрическая фигура, 3 поля– 3 класса наследника
6	Программное обеспечение, 4 поля– 2 класса наследника
7	Аппаратное обеспечение, 3 поля– 3 класса наследника
8	Город, 2 поля– 2 класса наследника
9	Страна, 2 поля– 2 класса наследника
10	Книга, 3 поля – 4 класса наследника

***Литература:** 1, 2, 3, 4, 5.*

Учебное издание

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине «**Язык программирования Java и средства NetBeans**»

Составитель

Малышева Елена Юрьевна

Издается в авторской редакции.

Подписано в печать с электронного оригинал-макета 30.06.2016.

Бумага офсетная. Печать трафаретная. Усл. печ. л. 3,0.

Тираж 500 экз. Заказ 78/01.

Издательско-полиграфический центр

Поволжского государственного университета сервиса.

445677, г. Тольятти, ул. Гагарина, 4.

rio@tolgas.ru, тел. (8482) 222-650.

Электронную версию этого издания

вы можете найти на сайте ЭБС ПВГУС <http://elib.tolgas.ru/>.