

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 № 4. Абстрактные классы

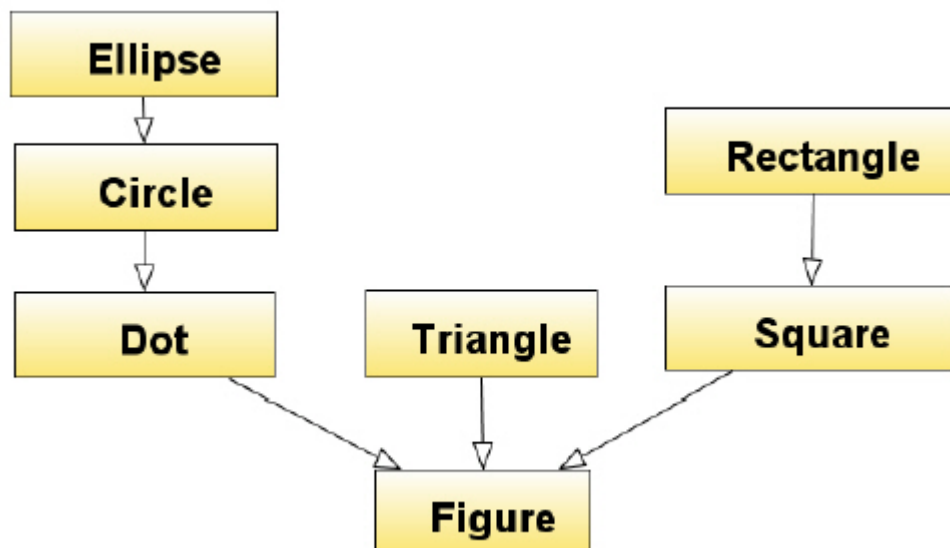
Цель работы: Научиться создавать абстрактные классы в Java

Задание: Разработать приложение с использованием абстрактных классов и интерфейсов.

Указания к работе:

Работа с абстрактными классами

Рассмотрим в качестве примера наследование для классов описанной ранее иерархии фигур. Для простоты выберем вариант, в котором Figure - это класс-прародитель иерархии, Dot его потомок, а Circle - потомок Dot (то есть является "жирной точкой"). Напомним, что имена классов принято начинать с заглавной буквы.



Класс Figure опишем как абстрактный – объектов такого типа создавать не предполагается, так как фигура без указания конкретного вида – это, действительно, чистая абстракция. По той же причине методы show ("показать") и hide ("скрыть") объявлены как абстрактные. Напомним также, что если в классе хоть один метод является абстрактным, это класс обязан быть объявлен как абстрактный.

```
public abstract class Figure { //это абстрактный класс

    int x=0;
    int y=0;
    java.awt.Color color;// awt - пакет для работы с графикой
    java.awt.Graphics graphics;
    java.awt.Color bgColor;
    public abstract void show(); //это абстрактный метод
```

```

public abstract void hide(); //это абстрактный метод

public void moveTo(int x, int y){
    hide();
    this.x= x;
    this.y= y;
    show();
};
}

```

Поля `x` и `y` задают координаты фигуры, а `color` – ее цвет. Соответствующий тип задан в пакете `java.awt`. Поле `graphics` задает ссылку на графическую поверхность, по которой будет идти отрисовка фигуры. Соответствующий тип также задан в пакете `java.awt`. В отличие от полей `x`, `y` и `color` для этого поля при написании класса невозможно задать начальное значение, и оно будет присвоено при создании объекта. То же относится к полю `bgColor` (от "background color") – в нем мы будем хранить ссылку на цвет фона графической поверхности. Цветом фона мы будем выводить фигуру в методе `hide` для того, чтобы она перестала показываться на экране. Это не самый лучший, но зато самый простой способ скрыть фигуру. В дальнейшем при желании реализацию метода можно изменить – это никак не коснется остальных частей программы. В параграфе, посвященном конструкторам, в классе `FilledCircle` мы применим более совершенный способ отрисовки и "скрывания" фигур, основанный на использовании режима рисования XOR ("исключающее или"). Установка этого режима производится методом `setXORMode`. Такой режим можно использовать для всех наших фигур.

Метод `moveTo` имеет реализацию несмотря на то, что класс абстрактный, и в этой реализации используются имена абстрактных методов `show` и `hide`. Этот вопрос будет подробно обсуждаться в следующем параграфе, посвященном полиморфизму.

Рассмотрим теперь, как задается потомок класса `Figure` – класс `Dot` ("Точка"). Для `Dot` классы `Object` и `Figure` будут являться прародителями (суперклассами), причем `Figure` будет непосредственным прародителем. Соответственно, для них класс `Dot` будет являться потомком (подклассом), причем для класса `Figure` – непосредственным потомком. Класс `Dot` расширяет (`extends`) функциональность класса `Figure`: хотя в нем и не появляются новых полей, зато пишется реализация для методов `show` и `hide`, которые в прародительском классе были абстрактными. В классе `Figure` мы использовали классы пакета `java.awt` без импорта этого пакета. В классе `Dot` используется импорт – обычно это удобнее, так как не надо много раз писать длинные имена.

```

package java_gui_example;

import java.awt.*;

public class Dot extends Figure{

    /** Создает новый экземпляр типа Dot */
    public Dot(Graphics graphics,Color bgColor) {
        this.graphics=graphics;
        this.bgColor=bgColor;
    }

    public void show(){
        Color oldC=graphics.getColor();
        graphics.setColor(Color.BLACK);
        graphics.drawLine(x,y,x,y);
        graphics.setColor(oldC);
    }

    public void hide(){
        Color oldC=graphics.getColor();
        graphics.setColor(bgColor);
        graphics.drawLine(x,y,x,y);
        graphics.setColor(oldC);
        ;
    }
}

```

Отметим, что в классе Dot не задаются поля x, y, graphics и метод moveTo – они наследуются из класса Figure. А методы show и hide переопределяются (override) – для них пишется реализация, соответствующая тому, каким именно образом точка появляется и скрывается на экране.

Конструктор Dot(Graphics graphics, Color bgColor) занимается созданием объекта типа Dot и инициализацией его полей.

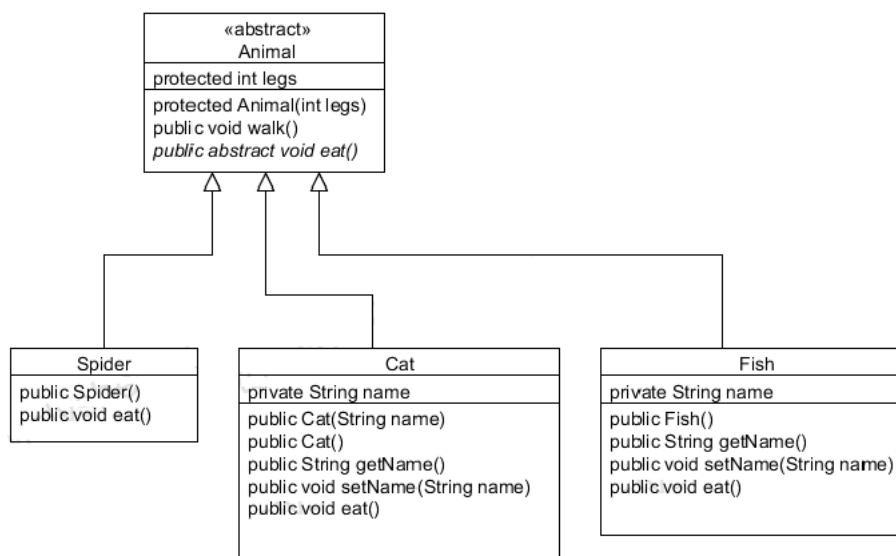
В методах show и hide используются методы объекта graphics. В методе show сначала во временной переменной oldC сохраняется информация о текущем цвете рисования. Затем в качестве текущего цвета устанавливается черный цвет (константа java.awt. Color.BLACK).

Затем вызывается метод, рисующий точку, в качестве него используется рисование линии с совпадающими началом и концом.

После чего восстанавливается первоначальный цвет рисования. Это необходимо для того, чтобы не повлиять на поведение других объектов, пользующихся для каких-либо целей текущим цветом.

Ход работы:

Разработаем приложение в соответствии с указанной моделью классов:



1. Создайте проект Java. Назовите пакет `com.example`, а главный класс `PetMain`
2. Создайте пакет `com.example.domain`, а в нем абстрактный класс `Animal` с полем `legs`, конструктором с аргументами и методом `walk`:

```
public abstract class Animal {
    protected int legs;
    protected Animal(int legs) {
        this.legs = legs;
    }
    public void walk() {
        System.out.println("Это животное гуляет на " + legs +
            "ногах.");
    }
}
```

3. Добавьте в пакет `com.example.domain` класс `Spider` наследующий классу `Animal`, без полей и с конструктором без аргументом

```
public class Spider extends Animal {
    public Spider() {
        super(8);
    }
}
```

4. Добавьте в пакет `com.example.domain` класс `Cat`, наследующий классу `Animal`, с полем `name`, конструктором с одним аргументом, конструктором без аргументов, методом `getName`

```
public class Cat extends Animal {
    private String name;
    public Cat(String name) {
        super(4);
        this.name = name;
    }
    public Cat() {
        this("Мурка");
    }
    public String getName() {
        return name;
    }
}
```

5. Добавьте в пакет `com.example.domain` класс `Fish`, наследующий классу `Animal`, с полем `name`, конструктором без аргументов, методами `setName` и `getName` и напишите в нем метод `walk`:

```
public class Fish extends Animal {
    private String name;
    public Fish() {
        super(0);
    }

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

```

    }
    @Override
    public void walk() {
        System.out.println("Рыбки, конечно, гулять не могут –
они плавают");
    }
}

```

6. Добавьте в класс PetMain команды импорта классов

7. Добавьте в процедуру main класса PetMain команды создания объектов созданных классов и заполнение их полей

```

Animal a;
Spider s = new Spider();
s.walk();
Cat c = new Cat("Том");
c.walk();
a = new Cat();
a.walk();
Fish f = new Fish();
f.setName("Гуппи");
f.walk();
a = new Fish();
a.walk();

```

8. Сохраните все классы и запустите приложение. Обратите внимание, как работает метод walk для рыбки для ссылки типа Fish и типа Animal

9. Добавьте в абстрактный класс Animal абстрактный метод eat:

```

public abstract void eat();

```

10. Реализуйте этот метод в дочерних классах Spider, Cat и Fish:

```

@Override
public void eat() {
    System.out.println("Паук есть мух");
}

@Override
public void eat() {
    System.out.println("Кошки любят есть пауков и рыбок");
}

```

```
}  
@Override  
public void eat() {  
    System.out.println("Рыбки едят червяков");  
}
```

11. Добавьте в процедуру main класса PetMain команды вызова метода eat для созданных объектов
12. Сохраните все классы и запустите приложение. Обратите внимание, как работает метод eat для ссылки типа Animal.

Вопросы для самоконтроля

1. Что такое абстрактный класс?
2. Что такое абстрактный метод?
3. Можно ли создавать объекты абстрактного класса?

Литература: 1, 2, 3, 4, 7.

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/>.