

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 № 5. Интерфейсы.

Цель работы: Научиться создавать интерфейсы в Java

Задание: разработать консольное приложение с использованием интерфейсов.

Указания к работе:

Проблемы множественного наследования классов.

Достаточно часто требуется совмещать в объекте поведение, характерное для двух или более независимых иерархий. Но еще чаще требуется писать единый полиморфный код для объектов из таких иерархий в случае, когда эти объекты обладают схожим поведением. Как мы знаем, за поведение объектов отвечают методы. Это значит, что в полиморфном коде требуется для объектов из разных классов вызывать методы, имеющие одинаковую сигнатуру, но разную реализацию. Унарное наследование, которое мы изучали до сих пор, и при котором у класса может быть только один прародитель, не обеспечивает такой возможности. При унарном наследовании нельзя ввести переменную, которая бы могла ссылаться на экземпляры из разных иерархий, так как она должна иметь тип, совместимый с базовыми классами этих иерархий.

В C++ для решения данных проблем используется множественное наследование. Оно означает, что у класса может быть не один непосредственный прародитель, а два или более. В этом случае проблема совместимости с классами из разных иерархий решается путем создания класса, наследующего от необходимого числа классов-прародителей.

Но при множественном наследовании классов возникает ряд трудно разрешимых проблем, поэтому в Java оно не поддерживается. Основные причины, по которым произошел отказ от использования множественного наследования классов: наследование ненужных полей и методов, конфликты совпадающих имен из разных ветвей наследования.

В частности, так называемое ромбовидное наследование, когда у класса A наследники B и C, а от них наследуется класс D. Поэтому класс D получает поля и методы, имеющиеся в классе A, в удвоенном количестве – один комплект по линии родителя B, другой – по линии родителя C.

Конечно, в C++ имеются средства решать указанные проблемы. Но в результате получается заметное усложнение логики работы с классами и объектами, вызывающее логические ошибки, не отслеживаемые компилятором. Поэтому в Java используется множественное наследование с помощью интерфейсов, лишенное практически всех указанных проблем.

Интерфейсы являются важнейшими элементами языков программирования, применяемых как для написания полиморфного кода, так и для межпрограммного обмена.

Интерфейсы являются специальной разновидностью полностью абстрактных классов. То есть таких классов, в которых вообще нет реализованных методов - все методы абстрактные. Полей данных в них также нет, но можно задавать константы (неизменяемые переменные класса). Класс в Java должен быть наследником одного класса-родителя, и может быть наследником произвольного числа интерфейсов. Сами интерфейсы также могут наследоваться от интерфейсов, причем также с разрешением на множественное наследование.

Отсутствие в интерфейсах полей данных и реализованных методов снимает почти все проблемы множественного наследования и обеспечивает изящный инструмент для написания полиморфного кода. Например, то, что все коллекции обладают методами, перечисленными в разделе о коллекциях, обеспечивается тем, что их классы являются наследниками интерфейса `Collection`. Аналогично, все классы итераторов являются наследниками интерфейса `Iterator`, и т.д.

Декларация интерфейса очень похожа на декларацию класса:

```
МодификаторВидимости interface ИмяИнтерфейса
    extends ИмяИнтерфейса1, ИмяИнтерфейса2, ..., ИмяИнтерфейсаN{
        декларация констант;
        декларация заголовков методов;
    }
```

В качестве модификатора видимости может использоваться либо слово `public` – общая видимость, либо модификатор должен отсутствовать, в этом случае обеспечивается видимость по умолчанию - пакетная. В списке прародителей, расширением которых является данный интерфейс, указываются прародительские интерфейсы `ИмяИнтерфейса1`, `ИмяИнтерфейса2` и так далее. Если список прародителей пуст, в отличие от классов, интерфейс не имеет прародителя.

Для имен интерфейсов в Java нет специальных правил, за исключением того, что для них, как и для других объектных типов, имя принято начинать с заглавной буквы. Мы будем использовать для имен интерфейсов префикс `I` (от слова `Interface`), чтобы их имена легко отличать от имен классов.

Объявление константы осуществляется почти так же, как в классе:

```
МодификаторВидимости Тип ИмяКонстанты = значение;
```

В качестве необязательного модификатора видимости может использоваться слово `public`. Либо модификатор должен отсутствовать - но при этом видимость также считается **public**, а не пакетной. Еще одним отличием от декларации в классе является то, что при задании в интерфейсе все поля автоматически считаются окончательными (модификатор `final`

), т.е. без права изменения, и к тому же являющимися переменными класса (модификатор `static`). Сами модификаторы `static` и `final` при этом ставить не надо.

Декларация метода в интерфейсе осуществляется очень похоже на декларацию абстрактного метода в классе – указывается только заголовок метода:

```
МодификаторВидимости Тип ИмяМетода (списокПараметров)
    throws списокИсключений;
```

В качестве модификатора видимости, как и в предыдущем случае, может использоваться либо слово `public`, либо модификатор должен отсутствовать. При этом видимость также считается **public**, а не пакетной. В списке исключений через запятую перечисляются типы проверяемых исключений (потомки `Exception`), которые может возбуждать метод. Часть `throws списокИсключений` является необязательной. При задании в интерфейсе все методы автоматически считаются общедоступными (`public`) абстрактными (`abstract`) методами объектов. Пример задания интерфейса:

```
package figures_pkg;

public interface IScalable {
    public int getSize();
    public void setSize(int newSize);
}
```

Класс можно наследовать от одного родительского класса и от произвольного количества интерфейсов. Но вместо слова `extends` используется зарезервированное слово `implements` – реализует. Говорят, что класс-наследник интерфейса реализует соответствующий интерфейс, так как он обязан реализовать все его методы. Это гарантирует, что объекты для любых классов, наследующих некий интерфейс, могут вызывать методы этого интерфейса. Что позволяет писать полиморфный код для объектов из разных иерархий классов. При реализации возможно добавление новых полей и методов, как и при обычном наследовании. Поэтому можно считать, что это просто один из вариантов наследования, обладающий некоторыми особенностями.

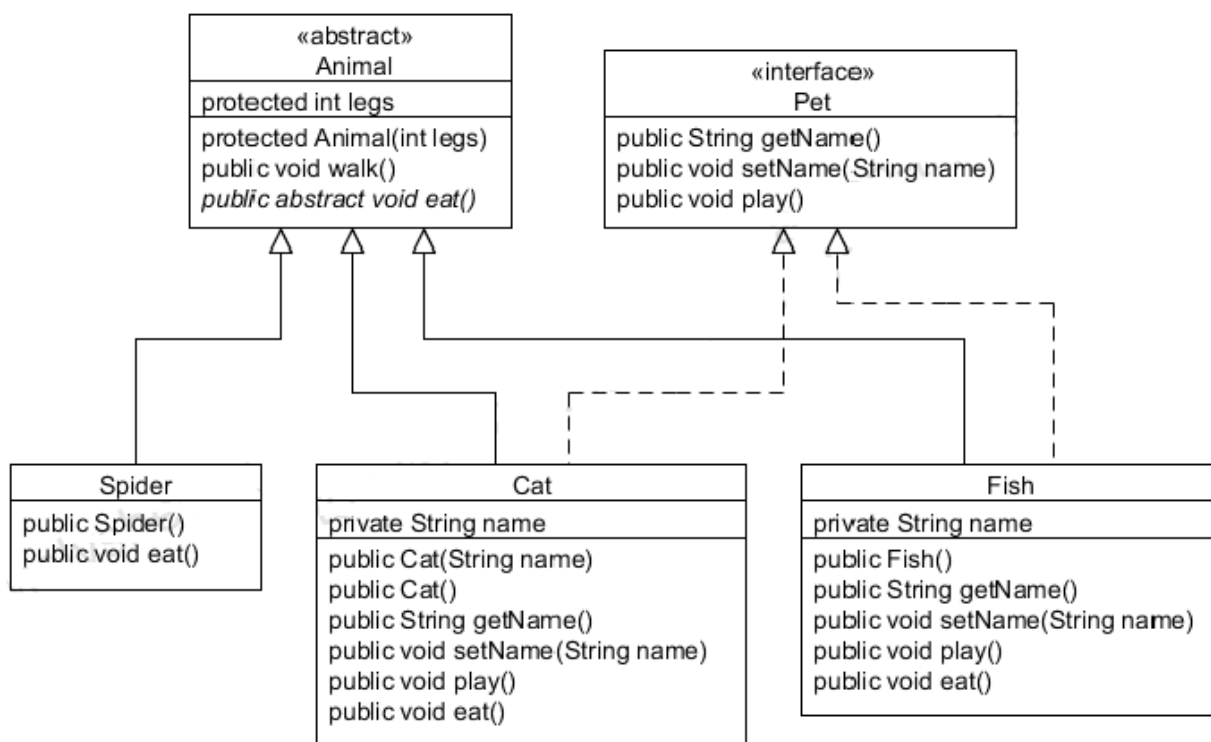
В наследнике класса, реализующего интерфейс, можно переопределить методы этого интерфейса. Повторное указание в списке родителей интерфейса, который уже был унаследован кем-то из прародителей, запрещено. Реализации у сущности типа интерфейс не бывает, как и для абстрактных классов. То есть экземпляров интерфейсов не бывает. Но, как и для абстрактных классов, можно вводить переменные типа интерфейс. Эти переменные могут ссылаться на объекты, принадлежащие классам, реализующим соответствующий интерфейс. То есть классам-наследникам этого интерфейса.

Правила совместимости таковы: переменной типа интерфейс можно присваивать ссылку на объект любого класса, реализующего этот интерфейс. С помощью переменной типа интерфейс разрешается вызывать только методы, декларированные в данном интерфейсе, а не любые методы данного объекта. Если, конечно, не использовать приведение типа.

Основное назначение переменных интерфейсного типа – вызов с их помощью методов, продекларированных в соответствующем интерфейсе. Если такой переменной назначена ссылка на объект, можно гарантировать, что из этого объекта разрешено вызывать эти методы, независимо от того, какому классу принадлежит объект. Ситуация очень похожа с полиморфизмом на основе виртуальных и динамических методов объектов. Но гарантией работоспособности служит не одинаковость сигнатуры методов в одной иерархии, а одинаковость сигнатуры методов в разных иерархиях – благодаря совпадению с декларацией одного и того же интерфейса. Обязательно, чтобы методы были методами объектов – полиморфизм на основе методов класса невозможен, так как для вызовов этих методов используется статическое связывание (на этапе компиляции).

Ход работы:

Продолжаем работать с указанной моделью классов. Добавляем в нее интерфейс:



1. Добавьте в пакет `com.example.domain` интерфейс `Pet` с методами `getName`, `setName`, `play`:

```

interface Pet {
    public String getName();

```

```

        public void setName (String name);
        public void play ();
    }

```

2. Измените программный код класса Cat

```

public class Cat extends Animal implements Pet {
    @Override
    public String getName() {
        return name;
    }
    @Override
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public void play() {
        System.out.println(name + " любит играть с веревоч-
кой");
    }
}

```

3. Измените программный код класса Fish

```

public class Fish extends Animal implements Pet {
    ...
    @Override
    public String getName() {
        return name;
    }
    @Override
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public void play() {
        System.out.println("Рыбка просто плавает");
    }
}

```

4. Добавьте в процедуру main класса PetMain команды вызова метода play для созданных объектов. Добавьте импорт интерфейса Pet и интерфейсные ссылки (ссылки типа Pet)

```
Animal a;  
//test a spider with a spider reference  
Spider s = new Spider();  
s.eat();  
s.walk();  
Cat c = new Cat("Tom");  
c.eat();  
c.walk();  
    c.play();  
a = new Cat();  
a.eat();  
a.walk();  
Pet p;  
p = new Cat();  
p.setName("Mr. Whiskers");  
    p.play();  
Fish f = new Fish();  
f.setName("Guppy");  
f.eat();  
f.walk();  
f.play();  
a = new Fish();  
a.eat();  
a.walk();
```

5. Сохраните все классы и запустите приложение

ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ

1. Добавьте в класс PetMain статический метод playWithAnimal с аргументом типа Animal с проверкой, реализует ли данное животное интерфейс Pet. Если это домашний питомец, то используйте кастинг (явное приведение к интерфейсному типу Pet). Если животное не является домашним питомцем, то надо вывести сообщение, что с ним играть опасно:

```
public static void playWithAnimal(Animal a) {
```

```

        if (a instanceof Pet) {
            Pet p = (Pet) a;
            p.play();
        } else {
            System.out.println("Опасно, дикое животное");
        }
    }
}

```

- Добавьте в процедуру main класса PetMain вызов статического метода playWithAnimal для животных разного типа:

```

playWithAnimal(s);
playWithAnimal(c);
playWithAnimal(f);

```

- Сохраните все классы и запустите приложение

Варианты заданий: Создайте собственного питомца, который является наследником приведенных в приложении абстрактного класса и интерфейса.

Вариант	Название питомца для своего класса
1	белка
2	хомячок
3	собака
4	енот
5	морская свинка
6	попугай
7	бурундук
8	черепаха
9	кролик
10	шиншилла

Вопросы для самоконтроля

- Что такое множественное наследование?
- Что такое интерфейс?
- Для чего применяется интерфейс?

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/>.