-Частное учреждение образования
Колледж бизнеса и права

УТВЕРЖДАЮ
Заведующий
методическим кабинетом
_____Е.В. Паскал
«___»_____2021

Специальность: 2-40 01 01 «Программное обеспечение информационных технологий»	Дисциплина: «Основы кроссплатформенного программирования»

ЛАБОРАТОРНАЯ РАБОТА № 4
Инструкционно-технологическая карта

Тема: «Наследование и интерфейсы в языке Java»

Цель: Научиться создавать интерфейсы, обращаться к ним, научиться работать с наследованием в языке Java

Время выполнения: 2 часа

СОДЕРЖАНИЕ РАБОТЫ

1. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое множественное наследование?
2. Что такое интерфейс?
3. Для чего применяется интерфейс?
4. Каким образом описывается интерфейс?
5. Когда целесообразно применять интерфейсы?
6. Опишите принцип наследования.
7. Синтаксис наследования в Java. Какие способы раскладки блоков существуют?

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ДЛЯ ВЫПОЛНЕНИЯ РАБОТЫ

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.

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

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;
}
```

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;

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

МодификаторВидимости Тип ИмяМетода(списокПараметров)
throws списокИсключений;
abstract)

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.

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).

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.

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.

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".

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 2 - 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 прилегающих к ней сторон и угол между ними, и так далее.

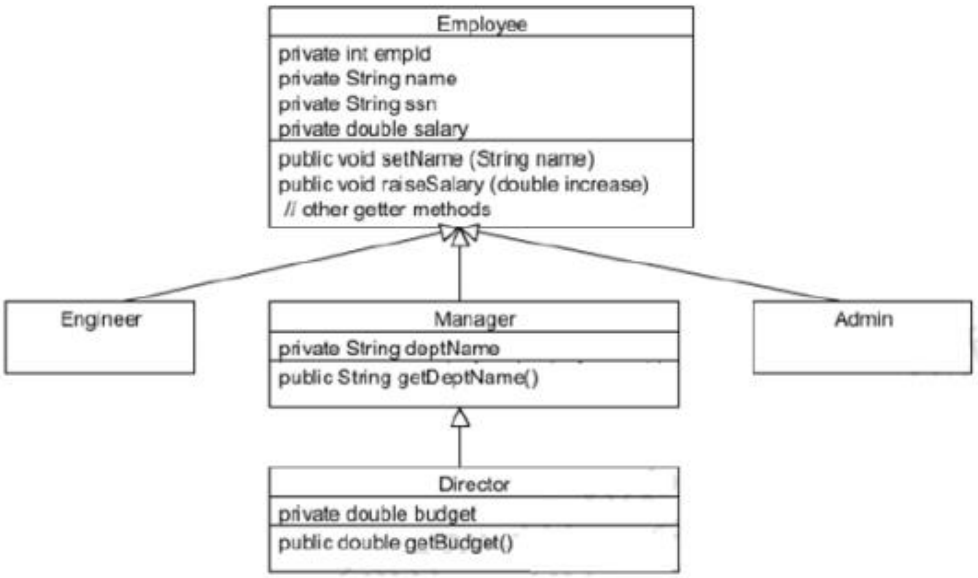
Каждый объект класса-потомка при любых значениях полей должен рассматриваться как экземпляр класса-прародителя, и с тем же поведением на уровне абстракции действий.

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.

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 инициирует 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 инициирует 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.

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 веревочкой");
} }
```

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

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучите теоретические сведения.

2. 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
11. еж
12. змея
13. мышь
14. крыса
15. тушканчик
16. хамелеон
17. геккон
18. хорек
19. улитка
20. сурок
21. лемур
22. ласка
23. ленивец
24. сурикат
25. кролик
26. лама
27. утка

ЗАДАЧА 2. Создать 3 класса(базовый) и 2 предка которые описывают некоторых работников с почасовой оплатой (один из предков) и фиксированой оплатой (второй предок).

Описать в базовом классе абстрактный метод для расчета среднемесячной зарплаты.

Для «почасовщиков» формула для расчета такая: «среднемесячная зарплата = 20.8*8*ставка в час», для работников с фиксированой оплатой «среднемесячная зарплата = фиксированой месячной оплате».

а)Упорядочить всю последовательность рабочих по убыванию среднемесячной зарплаты.

При совпадении зарплаты – упорядочить данные в алфавитном порядке по имени. Вывести идентификатор работника, имя и среднемесячную зарплату для всех елементов списка.

b)Вывести первые 5 имен работников из полученого выше списка (задача А).

c)Вывести последние 3 идентификаторы работников из полученого више списка (задача А).

d)Организовать запись и чтение колекции в/из файл(а)

e)Организовать обработку некоректного формата входного файла

3. Выполненные задания сдайте на проверку в папку, указанную преподавателем.

4. ДОМАШНЕЕ ЗАДАНИЕ

стр. 155-165

5. ЛИТЕРАТУРА

Альфред В., Ахо Компиляторы. Принципы, технологии и инструментарий, Вильямс, 2015.

Преподаватель А.Н.Воронцова

Рассмотрено на заседании цикловой комиссии программного обеспечения информационных технологий
Протокол № ____ от «___» _____ 2020
Председатель ЦК _____В.Ю.Михалевич