МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИЕРСИТЕТ»
МИРЭА

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Часть 2

Методические указания по выполнению практических работ для студентов, обучающихся по направлениям подготовки 09.03.02 «Информационные системы и технологии» и 09.03.04 «Программная инженерия»

МОСКВА 2017

Составитель Н. В. Зорина, Л. Б. Зорин, О. В. Соболев

Методические указания содержат необходимый для выполнения заданий теоретический материал, задания на практические работы, а также примеры выполнения заданий в виде программ на языке программирования Java. Методические указания предназначены для студентов, обучающихся по направлению 09.03.02 «Информационные системы и технологии» и 09.03.04 «Программная инженерия», продолжающих изучать курс «Объектно-ориентированное программирование».

В авторской редакции

Рецензенты: А.В. Голышко

Московский технологический университет
117454, Москва, проспект Вернадского, 78

Оглавление

ПРАКТИЧЕСКАЯ РАБОТА №1

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В JAVA

ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ:

Цель данной практической работы – освоить на практике работу с классами на Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

1. **Понятие класса**

В Java, класс является определением объектов одного и того же вида. Другими словами, класс это тип данных создаваемый программистом для решения задач. Он представляет из себя шаблон , или прототип, который определяет и описывает статические свойства и динамическое поведение, общие для всех объектов одного и того же вида.

Экземпляр класса - реализация конкретного элемента класса. Другими словами ,экземпляр экземпляра класса . Все экземпляры класса имеют аналогичные свойства , как описано в определении класса . Например, вы можете определить класс с именем "Студент " и создать три экземпляра класса "Студент ": " Петр", " Павел" и " Полина ". Термин " Объект " обычно относится к экземпляру класса . Но он часто используется свободно, которые могут относиться к классу или экземпляру.

Графически представляем Класс как трехкомпонентный контейнер-бокс с отсеками инкапсуляции данных и операций (методами)

Графически можно изобразить Класс как три отсека в боксе, в общем как на рисунке 1.

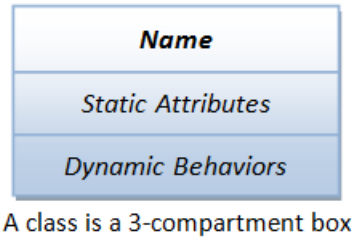


Рис.1 Представление класса

Имя (или сущность) : определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области).

Методы (или поведение, функции, работа с данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“контейнере” или “боксе”).

На рисунке 1.2 показано несколько примеров классов:

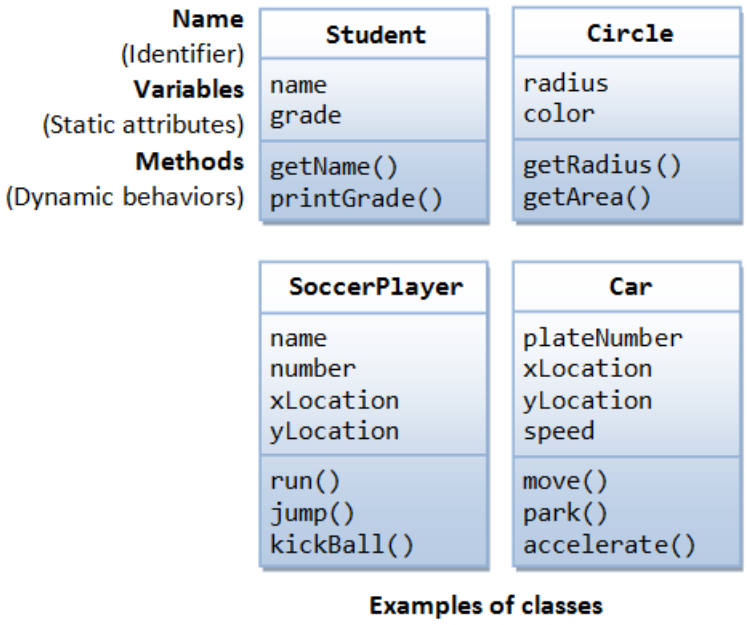


Рис. 1.2 Примеры классов

На рисунке 1.3 показаны два экземпляра класса типа Student "paul" и "peter" .

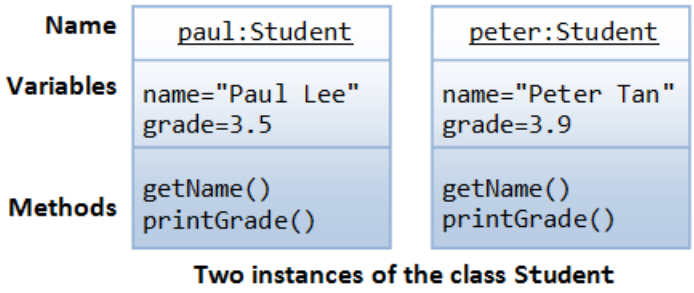


Рис. 1.3 Экземпляры класса Student

Unified Modeling Language (UML) класс и экземпляр диаграмм:

Приведенные выше диаграммы классов описаны в соответствии с UML нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три отсека, один содержит название , два вторых переменные (поля данных класса) и методы , соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр (объект класса) также представляется в виде прямоугольника, разделенного на три отсека, в первом отсеке, надпись с именем экземпляра, показанной в instanceName:Classname и выделенная подчеркиванием (название_экземпляра : имя_класса).

Кратко подведем итоги:

1) Класс определенный программистом, абстрактный тип данных, повторно-используемый программный объект, который имитирует реальные сущности предметной области. Класс контейнер-бокс, который состоит из 3 – отсеков и содержит имя , переменные и методы .

2) Класс инкапсулирует структуры данных (в переменных) и алгоритмы (методы).

3) Значения переменных составляют его состояние. Методы создает свои модели поведения.

Экземпляр это представление (или реализация) конкретного представителя класса.

2. Определение класса

В Java , мы используем класс как ключевое слово, например чтобы определить класс.

Для примера:

```
 1 public class Circle {         // class name
 2    double radius;             // variables
 3    String color;
 4
 5    double getRadius() {...}   // methods
 6    double getArea() {...}
 7 }
 8 public class SoccerPlayer {   // class name
 9    int number;                // variables
10    String name;
11    int x, y;
12
13    void run() {...}           // methods
14    void kickBall() {...}
15 }
```

Синтаксис определения класса в Java:

```
1 [AccessControlModifier] class ClassName {
2    // class body contains definition of variables and methods
3    ...
4 }
```

Давайте разъясним, что такое контроль доступа или спецификатор доступа, например, public и private, позже.

Конвенция кода для класса (Class Naming Convention):

Имя класса должно быть всегда существительным или словосочетание из нескольких слов. Все слова должны с Прописной буквы (верблюжья нотация). Используйте существительное в единственном числе для имени класса. Выберите значимое и самодостаточное имя для класса. Для примера, SoccerPlayer, HttpProxyServer, FileInputStream, PrintStream and SocketFactory.

3. Создание экземпляров класса

Чтобы создать экземпляр класса, вы должны:

Объявить идентификатор экземпляра (имя экземпляра) конкретного класса.

Построить экземпляр (то есть, при выделении памяти для экземпляра и инициализации экземпляра) с помощью оператор "new".

Для примера , предположим, что у нас есть класс с именем Circle, мы можем создавать экземпляры Circle, следующим образом:

```
1 // Declare 3 instances of the class Circle, c1, c2, and c3
2 Circle c1, c2, c3;
3 // Allocate and construct the instances via new operator
4 c1 = new Circle();
5 c2 = new Circle(2.0);
6 c3 = new Circle(3.0, "red");
7 // You can declare and construct in the same statement
8 Circle c4 = new Circle();
```

4. Операция точка “.”

Переменные и методы, принадлежащие к классу формально называется переменные-поля данных класса и методы класса. Для ссылки на переменную-поле данных класса или метод, вы должны:

- Сначала создать экземпляр класса, который вам нужен;
- Затем, использовать оператор точка “.” чтобы сослаться на элемент класса (переменную-поле данных или метод класса)

Например, предположим, что у нас есть класс с именем Circle, с двумя переменными (радиус и цвет) и двумя методами (getRadius () и GetArea ()). Мы создали три экземпляра класса Circle, а именно, C1, C2 и C3 . Чтобы вызвать метод GetArea (), вы должны сначала определить к какой именно сущности вы обращаетесь, об этом собственно говорит говорит c2 , а затем использовать оператор точка , в виде c2.getArea (), для вызова метода GetArea () экземпляра c2.

Например,

```
1 // Declare and construct instances c1 and c2 of the class Circle
2 Circle c1 = new Circle ();
3 Circle c2 = new Circle ();
4 // Invoke member methods for the instance c1 via dot operator
5 System.out.println(c1.getArea());
6 System.out.println(c1.getRadius());
7 // Reference member variables for instance c2 via dot operator
8 c2.radius = 5.0;
9 c2.color = "blue"
```

Вызов метода GetArea () без указания экземпляра не имеет смысла , так как радиус неизвестно какого объекта (может быть много окружностей, у каждой из которых п свой собственный радиус) .

В общем, полагают, есть класс, называемый AClass с переменной-полем данных под названием aVariable и способом доступа методом aMethod ( ). Экземпляр называется anInstance и строится для AClass.

Вы можете использовать:

anInstance.aVariable и anInstance.aMethod ().

5. Переменные - поля данных класса

Переменная-поле данных имеет имя (или идентификатор) и тип ; и имеет значение определенного типа (с таблицей на предыдущей главе).

Переменная-поле данных может также быть экземпляром определенного класса (которые будут

обсуждаться позже).

Конвенция об именах переменных: имя переменной должно быть существительным или словосочетанием из нескольких слов. Первое слово в нижнем регистре , а остальные слова пишутся с Прописной букы (двугорбая нотация), например , размер шрифта, roomNumber, Xmax , Ymin и xTopLeft .

Обратите внимание, что имя переменной начинается с буквы в нижнем регистре, в то время как имя класса начинается с заглавной буквы.

Формальный синтаксис для определения переменной в Java:

*[AccessControlModifier] type variableName [= initialValue]**;***

*[AccessControlModifier] type variableName-1 [= initialValue-1] [**,** type variableName-2 [= initialValue-2]] ...* ***;***

Например:

```
1  private double radius;
2  public int length = 1, width =
   1;
```

6. Методы класса

Метод (способ как описано в предыдущем разделе):

- принимает параметры из вызова (как в функции);
- выполняет операции, описанные в теле метода, и;
- возвращает часть результата (или void) в точку вызова.

Формальный синтаксис объявления метода в Java:

```
1 [AccessControlModifier] returnType methodName ([argumentList]) {
2    // method body or implementation
3    ......
4 }
```

Например:

```
1 public double getArea() {
2    return radius*radius*Math.PI;
3 }
```

Конвенция записи имен методов :

Имя метода должно быть глаголом или начинаться глаголом в виде фразы из нескольких слов. Первое слово в нижнем регистре, а остальные слова начинаются с Прописной буквы (двугорбая запись). Например, getRadius ( ) , getParameterValues ( ) .

Обратите внимание, что имя переменной существительное (обозначающий статический атрибут) , в то время как имя метода - глагол (обозначает действие) . Они имеют те же наименования. Тем не менее, вы можете легко отличить их от контекста. Методы могут принимать аргументы в скобках (возможно, нулевой аргумент, в пустых скобках), ноне поля данных. При записи, методы обозначаются парой круглых скобок, например , Println ( ) , GetArea ( ).

7. Теперь соберем все вместе: Пример ООП

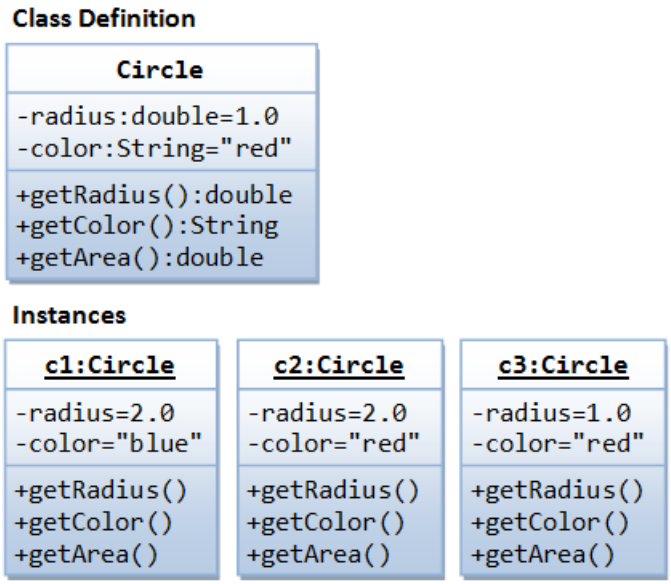


Диаграмма класса и экземпляров класса на рисунке выше

Класс называется Circle и должен быть определен , как показано на диаграмме классов .

Он содержит две переменные: radius (типа double) и color (типа String); и три метода: getRadius ( ) , GetColor ( ) , и GetArea ( ) .

Три экземпляра Circle называются C1 , C2, C3 , и должны быть построены с учетом их соответствующих элементов данных и методов, как показано на схемах UML для экземпляров класса.

Исходные коды для Circle.java:

Circle.java

```
 1 // Define the Circle class
 2 public class Circle {    // Save as "Circle.java"
 3    // Private variables
 4    private double radius;
 5    private String color;
 6
 7    // Constructors (overloaded)
 8    public Circle() {                   // 1st Constructor
 9       radius = 1.0;
10       color = "red";
11    }
12    public Circle(double r) {           // 2nd Constructor
13       radius = r;
14       color = "red";
15    }
16    public Circle(double r, String c) { // 3rd Constructor
17       radius = r;
18       color = c;
19    }
20
```

```
21    // Public methods
22    public double getRadius() {
23       return radius;
24    }
25    public String getColor() {
26       return color;
27    }
28    public double getArea() {
29       return radius*radius*Math.PI;
30    }
31 }
```

Компиляция "Circle.java" в "Circle.class".

Обратите внимание, что в классе Circle нет метода Main () . Следовательно, это не будет программой на Java, и вы не можете запустить класс Circle сам по себе. Класс Circle нужен, чтобы быть строительным блоком и использоваться в других программах.

Дополним нашу программу еще одним классом, который будет демонстрировать работу с нашим классом. Мы напишем TestCircle, в котором будем использовать Circle класс. Класс TestCircle в котором есть метод main() и мы можем теперь запустить программу.

TestCircle.java

```
public class TestCircle {     // Save as "TestCircle.java"
   public static void main(String[] args) {   // Execution entry
1
2  point
3        // Construct an instance of the Circle class called c1
4        Circle c1 = new Circle(2.0, "blue");     // Use 3rd constructor
5        System.out.println("Radius is " + c1.getRadius()  // use dot
6  operator to invoke member methods
7           + " Color is " + c1.getColor()
8           + " Area is " + c1.getArea());
9
10       // Construct another instance of the Circle class called c2
11       Circle c2 = new Circle(2.0);          // Use 2nd constructor
12       System.out.println("Radius is " + c2.getRadius()
13          + " Color is " + c2.getColor()
14          + " Area is " + c2.getArea());
15
16       // Construct yet another instance of the Circle class called
17 c3
18       Circle c3 = new Circle();             // Use 1st constructor
19       System.out.println("Radius is " + c3.getRadius()
20          + " Color is " + c3.getColor()
21          + " Area is " + c3.getArea());
   }
}
```

Запустим TestCircle и увидим результат:

Radius is 2.0 Color is blue Area is 12.566370614359172

Radius is 2.0 Color is red Area is 12.566370614359172

Radius is 1.0 Color is red Area is 3.141592653589793

8. Конструкторы

Конструктор - специальный метод класса, который имеет то же имя, что используется в качестве имени класса. В приведенном выше класса Circle, мы определим три перегруженных версии конструктора " Circle(...) " . Конструктор используется для создания и инициализации всех переменных-полей данных класса. Чтобы создать новый экземпляр класса, вы должны использовать специальный оператор "new" с последующим вызовом одного из конструкторов.

Например,

```
1 Circle c1 = new Circle();
2 Circle c2 = new Circle(2.0);
3 Circle c3 = new Circle(3.0, "red");
```

Конструктор отличается от обычного метода в следующих аспектах:

- название метода-конструктора совпадает с именем класса, а имя класса по конвенции, начинается с заглавной буквы .
- Конструктор не имеет возвращаемого значения типа (или неявно не возвращает). Таким образом, нет объявления типа возвращаемого значения при объявлении.
- Конструктор может быть вызван только через оператор «new». Он может быть использован только один раз, чтобы инициализировать построенный экземпляр. Вы не можете впоследствии вызвать конструктор в теле программы подобно обычным методам (функциям).
- Конструкторы не наследуется (будет объяснено позже).

Конструктор без параметров называется конструктором по умолчанию, который инициализирует переменные-поля данных через их значения по умолчанию. Например, Circle( ) в приведенном выше примере.

9. Перегрузка методов

Перегрузка методов означает, что несколько методов могут иметь то же самое имя метод, но сами методы могут иметь различные реализации (версии) . Тем не менее, различные реализации должны быть различимы по их списком аргументов (либо количество аргументов , или типа аргументов , или их порядок).

Пример: метод average( ) имеет 3 версии с различными списками аргументов. При вызове может использоваться соответствующий выбору вариант, в соответствии с аргументами.

```
1 public class TestMethodOverloading {
2    public static int average(int n1, int n2) {          // A
3       return (n1+n2)/2;
4    }
5
6    public static double average(double n1, double n2) { // B
7       return (n1+n2)/2;
8    }
9
```

```
10    public static int average(int n1, int n2, int n3) {  // C
11       return (n1+n2+n3)/3;
12    }
13
14    public static void main(String[] args) {
15       System.out.println(average(1, 2));      // Use A
16       System.out.println(average(1.0, 2.0)); // Use B
17       System.out.println(average(1, 2, 3));  // Use C
18       System.out.println(average(1.0, 2));   // Use B - int 2
19 implicitly casted to double 2.0
20       // average(1, 2, 3, 4); // Compilation Error - No matching
21 method
22    }
   }
```

Перегрузка конструктора класса Circle

Приведенный выше класс Circle имеет три версии конструктора, которые отличаются списком их параметров, следовательно:

```
1 Circle()
2 Circle(double r)
3 Circle(double r, String c)
```

В зависимости от фактического списка аргументов , используемых при вызове метода, будет вызван соответствующий конструктор. Если ваш список аргументов не соответствует ни одному из определенных методов, вы получите ошибку компиляции.

10. Public или Private – модификаторы контроля доступа

Контроль за доступом осуществляется с помощью модификатора, он может быть использован для управления видимостью класса или переменных –полей или методов внутри класса. Мы начнем со следующих двух модификаторов управления доступом:

public : класс / переменная / метод доступным и для всех других объектов в системе.

private : класс / переменная / метод доступным и в пределах только этого класса.

Например, в приведенном выше определении Circle, radius переменная-поле данных класса объявлена private . В результате ,radius доступен внутри класса Circle , но не внутри класса TestCircle . Другими словами, вы не можете использовать "*c1.radius*" по отношению к радиусу C1 в классе TestCircle . Попробуйте вставить текст "*System.out.println (c1.radius) ;*" в TestCircle и понаблюдать за сообщением об ошибке.

Попробуйте изменить radius на public, и повторно запустить пример.

С другой стороны, метод getRadius ( ) определяется как public в классе Circle . Таким образом, он может быть вызван в классе TestCircle.

В нотации UML на диаграмме классов обозначается: общественные (public) элементы обозначены со знаком " + " , в то время как частные (private) элементы со знаком " - ".

11. Информация по сокрытию реализации и инкапсуляции

Класс инкапсулирует имя, статические атрибуты и динамического поведение в "3-части бокса - коробки". После того, как класс определен, вы можете запечатать "крышку" и поставить "коробку" на полку для использования другими и использовать самим. Любой желающий может снять "крышку" и использовать это в своем приложении. Это не может быть сделано в традиционном процедурном-ориентированного языка как C, так как статические атрибуты (или переменные) разбросаны на протяжении всей программы и файлов заголовков. Вы не можете "вырезать" куски из части программы на C, подключить в другую программу и ожидает, что запуск программы произойдет без серьезных изменений.

Переменные-поля данных класса, как правило, скрыты от внешнего слоя (то есть, другие классы), с контролем через private доступ модификатора. Доступ к переменным полям класса предоставляются через методы public, например, getRadius () и GetColor ().

Это соответствует принципу сокрытия информации. То есть, объекты могут общаться друг с другом, используя хорошо определенные интерфейсы (публичные методы). Объектам не позволено знать детали реализации других объектов. Детали реализации скрыты или инкапсулированы внутри класса. Сокрытие информации облегчает повторное использование класса.

Основное правило: Не делайте никаких переменных public, если у вас на то нет веских оснований.

ЗАДАНИЯ

Необходимо реализовать простейший класс на языке программирования Java. Добавить метод ToString. Создать класс-тестер для вывода информации об объекте.

Упражнение 1.

Реализуйте простейший класс «Мяч»

Упражнение 2.

Реализуйте простейший класс «Книга»

ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ

Задание: Разработка и реализация класса Dog, который содержит данные экземпляра, которые представляют имя собаки и возраст . Определить конструктор собаки, чтобы принять и инициализировать данные экземпляр. Включите в класс методы получения и записи данных в поля имени и возраста (геттеры и сеттеры). Включите в класс метод для вычисления возраста собаки, метод возвращает возраст собаки в "человеческих" летах (если возраст собаки умножить на семь, то получим возраст человека). Включите метод ToString(), который возвращает строчку в одну описание объекта-собаки. Назначение класса Driver или по другому Tester (можно также использовать для этого класса название - ПитомникСобак, но на англ. языке) демонстрировать работу программы, следовательно в этот класс необходимо поместить основной метод main(), внутри которого создать несколько объектов класса Dog и продемонстрировать работу разработанных в классе методов.

Код программы:

Dog.java

```
import java.lang.*;
public class Dog {
    private String name;
```

```
    private int age;

    public Dog(String n, int a){
        name = n;
        age = a;
    }
    public Dog(String n){
        name = n;
        age = 0;
    }
    public Dog(){
        name = "Pup";
        age = 0;
    }

    public void setAge(int age) {
        this.age = age;
    }

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

    public String getName(String name){
        return name;
    }

    public int getAge() {
        return age;
    }

    public String toString(){
        return this.name+", age "+this.age;
    }

    public void intoHumanAge(){
        System.out.println(name+"'s age in human years is "+age*7+"
years");
    }
}
```

Nursery.java

```
import java.lang.*;
public class Nursery {
    public static void main(String[] args) {
        Dog d1 = new Dog("Mike", 2);
        Dog d2 = new Dog("Helen", 7);
        Dog d3 = new Dog("Bob");
        d3.setAge(1);
        System.out.println(d1);
        d1.intoHumanAge();
        d2.intoHumanAge();
        d3.intoHumanAge();
    }
}
```

ПРАКТИЧЕСКАЯ РАБОТА №2
Проектирование классов: UML-диаграммы

Цель работы: работа с UML-диаграммами классов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Графически представляем класс как трехкомпонентный “контейнер-бокс” с отсеками для инкапсуляции данных и операций (методами составляющими интерфейсную часть класса)

Графически можно изобразить Класс как три отсека в боксе, в общем как на рисунке 1.

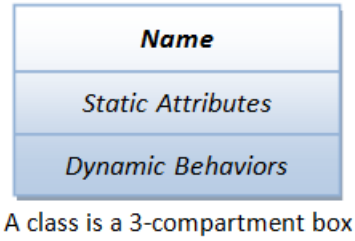


Рис.1 Представление класса

Имя (или сущность) : определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области).

Методы (или поведение, функции, работа с данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“контейнере” или “боксе”).

На рисунке 1.2 показано несколько примеров классов:

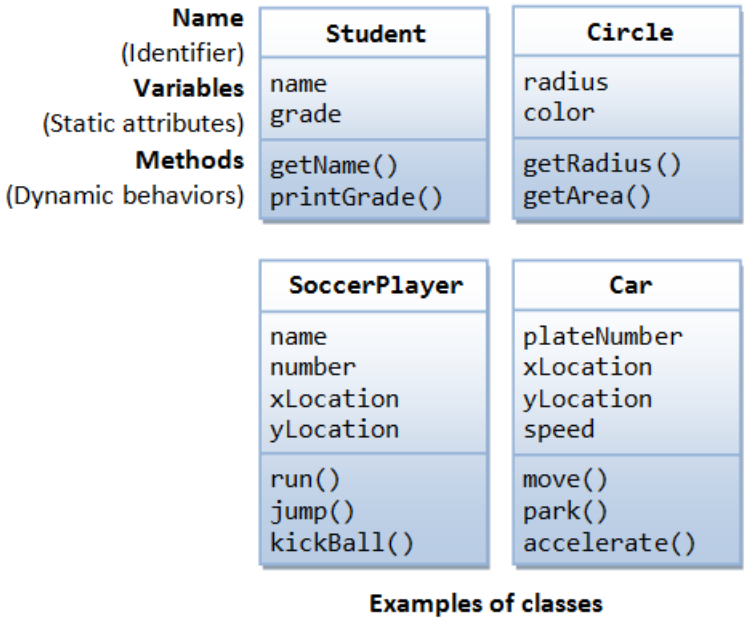


Рис. 1.2 Примеры классов

На рисунке 1.3 показаны два экземпляра класса типа Student "paul" и "peter" .

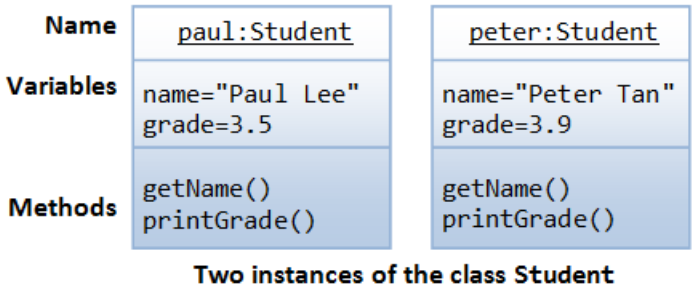


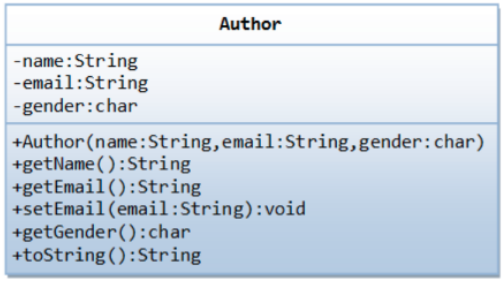
Рис. 1.3 Экземпляры класса Student

Unified Modeling Language (UML) класс и экземпляр диаграмм:

Приведенные выше диаграммы классов описаны в соответствии с UML нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три отсека, один содержит название , два вторых переменные (поля данных класса) и методы , соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр (объект класса) также представляется в виде прямоугольника, разделенного на три отсека, в первом отсеке, надпись с именем экземпляра, показанной в instanceName:Classname и выделенная подчеркиванием (название_экземпляра : имя_класса).

ЗАДАНИЯ

По диаграмме класса UML описывающей сущность Автор написать программу которая состоит из двух классов Author и TestAuthor. Класс Author должен содержать реализацию методов, представленных на диаграмме.



Класс, называемый Author (с англ. Автор) моделирует сущность предметной области – автор книги, как показано на диаграмме классов. Он содержит:

* три private переменных-полей данных класса: name (типа String) , email (типа String) , и gender (типа char, которая может принимать три значения либо 'M', если автор книги мужчина, 'F' – если автор книги женщина, или 'U' если пол автора неизвестен, - вы можете также использовать для реализации логическую переменную под названием male для обозначения пола автора, которая будет принимать значение истина или ложь) .

* Один конструктор для инициализации переменых name, email и gender с заданными значениями. (Тут не будет конструктора по умолчанию так как нет значений по умолчанию ни для для имени , ни для электронной почты или пола).

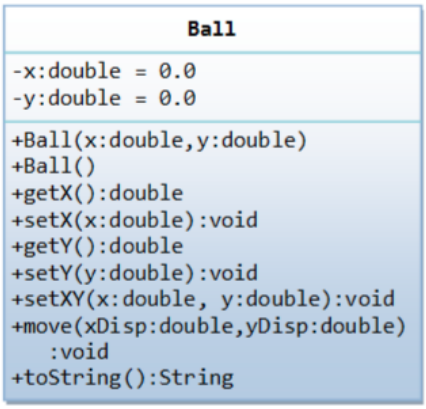
* Public методы Геттеры/сеттеры: getName(),

getEmail(), setEmail(), and getGender(). (Нужно упомянуть, что там не будет сеттеров для имени и пола, так как эти атрибуты не могут изменяться).

* Метод ToString (), которая должен возвращать следующий текст "автор - имя (пол) на адрес электронной почты " , например, " Tan Ah Teck (m) at ahTeck@somewhere.com ", или "Sue Grant (ms) at suGrant@somewhere.com ", то есть в строке должно быть записано имя[пробел](пол)[пробел]at[пробел]емайл

ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ

По диаграмме класса UML описывающей сущность Мяч написать программу которая состоит из двух классов Ball и TestBall. Класс Ball должен содержать реализацию методов, представленных на диаграмме.



Класс Ball моделирует движущийся мяч. В состав класса входят:

* Две private переменные (поля данных класса) x, y, которые описывают положение мяча на поле.

* Конструкторы, public методы получения и записи значений для private переменных.

* Метод setXY (), который задает положение мяча и метод setXYSpeed(), чтобы задать скорость мяча

* Метод move() , позволяет переместить мяч, так что что увеличивает x и y на данном участке на xDisp и yDisp, соответственно.

* Метод ToString(), который возвращает "Ball @ (x , y) " .

Ball.java

```
package balls;
public class Ball {
    private double x = 0.0;
    private double y = 0.0;

    public Ball(){}
    public Ball(double x, double y){
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }
```

```
    public double getY() {
        return y;
    }

    public void setX(double x) {
        this.x = x;
    }

    public void setY(double y) {
        this.y = y;
    }

    public void setXY(double x, double y){
        this.x = x;
        this.y = y;
    }

    public void move( double xDisp, double yDisp){
        x+=xDisp;
        y+=yDisp;
    }

    @Override
    public String toString() {
        return "Ball @ ("+this.x+", "+this.y+").";
    }
}
```

TestBall.java

```
package balls;
public class TestBall {
    public static void main(String[] args) {
        Ball b1 = new Ball(100, 100);
        System.out.println(b1);
        b1.move(30, 15);
        System.out.println(b1);
    }
}
```

ПРАКТИЧЕСКАЯ РАБОТА №3

Абстрактные суперклассы и его подклассы в Java.

ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ:

Цель данной практической работы – освоить на практике работу с абстрактными классами и наследованием на Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы помечаются ключевым словом **abstract**.

Абстрактный метод не завершён. Он состоит только из объявления и не имеет тела:

abstract void yourMethod();

По сути, мы создаём шаблон метода. Например, можно создать абстрактный метод для вычисления площади фигуры в абстрактном классе Фигура. А все другие производные классы от главного класса могут уже реализовать свой код для готового метода. Ведь площадь у прямоугольника и треугольника вычисляется по разным алгоритмам и универсального метода не существует.

Если вы объявляете класс, производный от абстрактного класса, но хотите иметь возможность создания объектов нового типа, вам придётся предоставить определения для всех абстрактных методов базового класса. Если этого не сделать, производный класс тоже останется абстрактным, и компилятор заставит пометить новый класс ключевым словом **abstract**.

Абстрактный класс не может содержать какие-либо объекты, а также абстрактные конструкторы и абстрактные статические методы. Любой подкласс абстрактного класса должен либо реализовать все абстрактные методы суперкласса, либо сам быть объявлен абстрактным

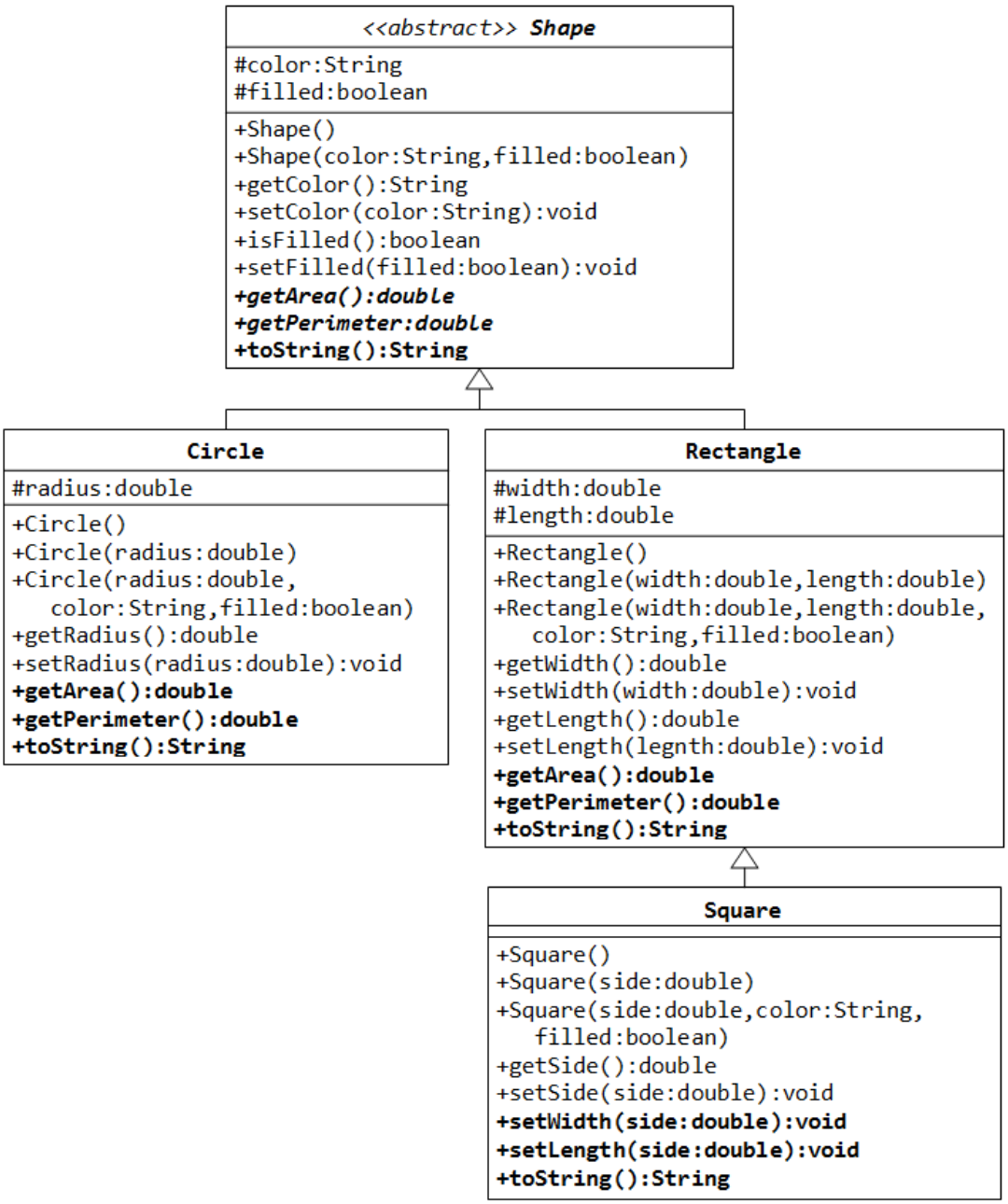
```
1 public abstract class Swim {
2     // абстрактный метод плавать()
3     abstract void neigh();
4 
5     // абстрактный класс может содержать и обычный метод
6     void run() {
7         System.out.println("Куда идешь?");
8     }
9 }
10 
11 class Swimmer extends Swim {
12 
13 }
```

ЗАДАНИЯ

Задание 1.

Абстрактный суперкласс Shape и его подклассы

Задание перепишите суперкласс Shape и его подклассы так как это представлено на диаграмме Circle, Rectangle and Square



В этом задании, класс Shape определяется как абстрактный класс , который содержит:

Два ***protected*** (защищенных) переменных color(String) и filled(boolean). Защищенные переменные могут быть доступны в подклассах и классах в одном пакете . Они обозначаются со знаком '#' в диаграмме классов в нотации языка UML.

Методы геттеры и сеттеры для всех переменных экземпляра класса, и метод toString () .

Два абстрактных метода getArea() и getPerimeter() выделены курсивом в диаграмме класса).

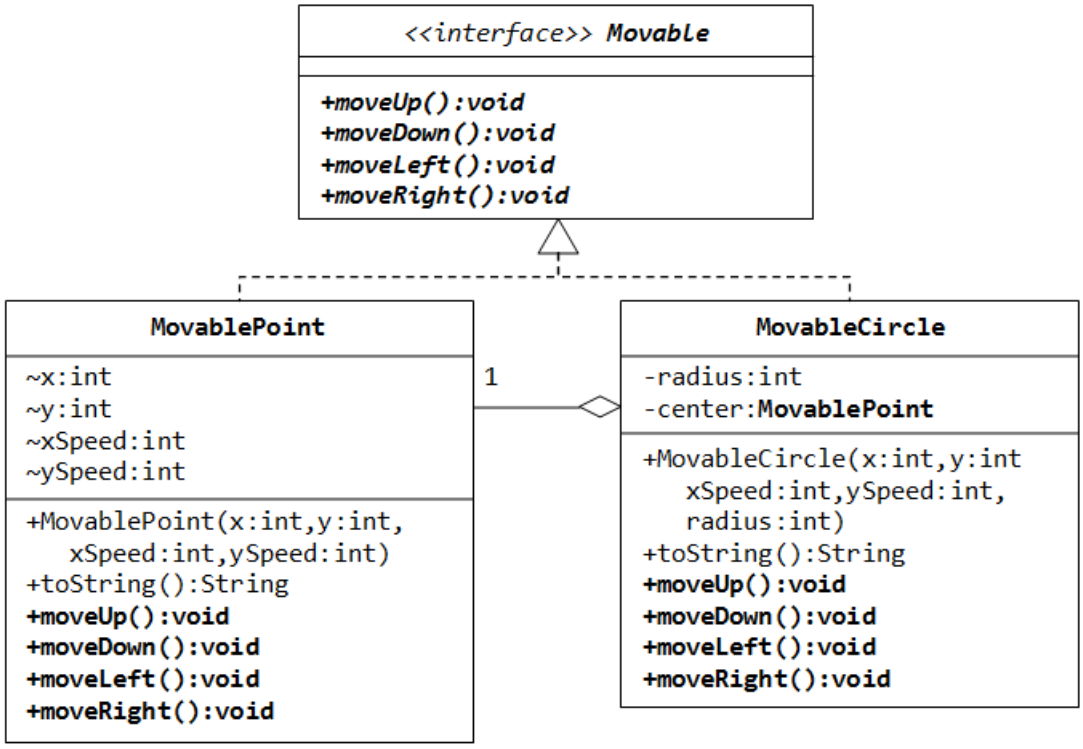
В подклассах Circle(круг) и Rectangle(прямоугольник) должны переопределяться абстрактные методы getArea() и getPerimeter(), чтобы обеспечить их надлежащее выполнение для конкретных экземпляров типа подкласс. Также необходимо для каждого подкласса переопределить toString() .

Вам нужно написать тестовый класс, чтобы самостоятельно это проверить, Необходимо объяснить полученные результаты и связать их с понятием ООП - полиморфизм. Некоторые объявления могут вызвать ошибки компиляции. Объясните полученные ошибки, если таковые имеются.

```
Shape s1 = new Circle(5.5, "RED", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;                    // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "RED", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Задание 2.

Вам нужно написать два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable .




```
public interface Movable { // saved as "Movable.java"
public void moveUp();
......
}
```

ПРИМЕР РЕШЕНИЯ ЗАДАНИЯ

Реализации класса Circle:

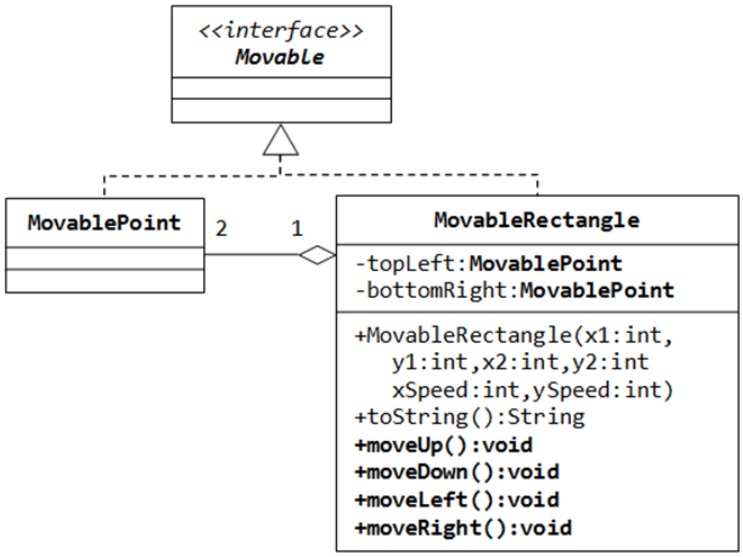
Circle.java

```
package shape;
import java.math.*;
public class Circle extends Shape{
    protected double radius;
    public Circle(){
        this.filled = false;
        this.color = "blue";
        radius = 1;
    }
    public Circle(double radius){
        this.filled = false;
        this.color = "blue";
        this.radius = radius;
    }
    public Circle(double radius, String color, boolean filled){
        this.radius = radius;
        this.color = color;
        this.filled = filled;
    }
    public double getRadius() {
        return radius;
    }
    public void setRadius(double radius) {
        this.radius = radius;
    }
```

```
    @Override
    public double getArea() {
        return Math.PI*radius*radius;
    }
    @Override
    public double getPerimeter() {
        return 2*Math.PI*radius;
    }
    @Override
    public String toString() {
        return "Shape: circle, radius: "+this.radius+", color:
"+this.color;
    }
}
```

2)

Напишите новый класс MovableRectangle (движущийся прямоугольник). Его можно представить как две движущиеся точки MovablePoints (представляющих верхняя левая и нижняя правая точки) и реализующие интерфейс Movable. Убедитесь, что две точки имеет одну и ту же скорость (нужен метод это проверяющий).



ПРАКТИЧЕСКАЯ РАБОТА №4
СОБЫТИЙНОЕ ПРОГРАММИРОВАНИЕ

Цель работы: Введение в событийное программирование

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Text Fields and Text Areas.
2. Layout Менеджеры (BorderLayout и GridLayout)
3. MouseListeners.
4. Menus.
5. Exercises.

Text Fields - текстовое поле или поля для ввода текста (можно ввести только одну строку). Примерами текстовых полей являются поля для ввода логина и пароля, например используемые, при входе в электронную почту.

Пример содания объекта класса JTextField:

```
JTextField jta = new JTextField (10);
```

С параметре конструктора задано число 10, это количество символов, которые могут быть видны в текстовом поле. Текст веденный в поле `JText` может быть возвращен с помощью метода `getText()`. Также в поле можно записать новое значение с помощью метода `setText(String s)`.

Как и у других компонентов, мы можем изменять цвет и шрифт текста в текстовом поле.

Пример 1

```
class LabExample extends JFrame
{
      JTextField jta = new JTextField(10);
      Font fnt = new Font("Times new roman",Font.BOLD,20);
      LabExample()
      {
            super("Example");
            setLayout(new FlowLayout());
            setSize(250,100);
            add(jta);
            jta.setForeground(Color.PINK);
            jta.setFont(fnt);
            setVisible(true);
      }

      public static void main(String[]args)
      {
            new LabExample();
      }
}
```

Важная замечание

Ответственность за выполнение проверки на наличие ошибок в коде лежит полностью на программисте, например, чтобы проверить произойдет ли ошибка, когда в в качестве входных данных в JTextField ожидается ввод числа. Компилятор не будет ловить такого рода ошибку, поэтому ее необходимо обрабатывать пользовательским кодом.

Выполните следующий пример и наблюдайте за результатом, когда число вводится в неправильном формате:

Пример 2

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class LabExample extends JFrame
{
      JTextField jta1 = new JTextField(10);
      JTextField jta2 = new JTextField(10);
      JButton button = new JButton(" Add them up");

      Font fnt = new Font("Times new roman",Font.BOLD,20);

      LabExample()
      {
            super("Example");
            setLayout(new FlowLayout());
            setSize(250,150);
            add(new JLabel("1st Number"));
            add(jta1);
            add(new JLabel("2nd Number"));
            add(jta2);
```

```
            add(button);


            button.addActionListener(new ActionListener()
            {
                  public void actionPerformed(ActionEvent ae)
                  {
                        try
                        {
                              double x1 =
Double.parseDouble(jta1.getText().trim());
                              double x2 =
Double.parseDouble(jta2.getText().trim());


                              JOptionPane.showMessageDialog(null, "Result
= "+(x1+x2),"Alert",JOptionPane.INFORMATION_MESSAGE);


                        }
                        catch(Exception e)
                        {
                               JOptionPane.showMessageDialog(null, "Error
in Numbers !","alert" , JOptionPane.ERROR_MESSAGE);
                        }
                  }
            });


            setVisible(true);
      }


      public static void main(String[]args)
      {
            new LabExample();
      }
}
```

JTextArea

Компонент TextAreas похож на TextFields, но в него можно вводить более одной строки. В качестве примера TextArea можно рассмотреть текст, который мы набираем в теле сообщения электронной почты

Пример 3

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class TextAreaExample extends JFrame
{
      JTextArea jta1 = new JTextArea(10,25);
      JButton button = new JButton("Add some Text");
      public TextAreaExample()
      {
            super("Example");
            setSize(300,300);
            setLayout(new FlowLayout());
            add(jta1);
            add(button);
            button.addActionListener(new ActionListener()
            {
                  public void actionPerformed(ActionEvent ae)
                  {
                        String txt =
JOptionPane.showInputDialog(null,"Insert some text");
                        jta1.append(txt);
                  }
            });
      }
      public static void main(String[]args)
      {
            new TextAreaExample().setVisible(true);
      }
```

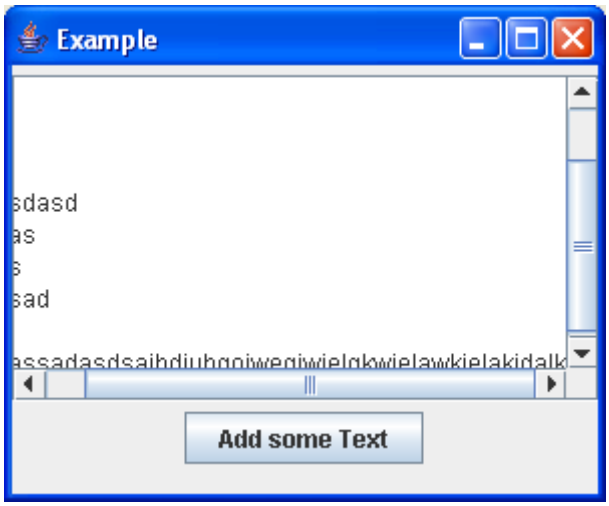
```
}
```

Замечание

Мы можем легко добавить возможность прокрутки к текстовому полю, добавив его в контейнер с именем `JScrollPane` следующим образом:

```
JTextArea txtArea = new JTextArea(20,20)

JScrollPane jScroll = new JScrollPane(txtArea);

// …

add(Scroll); // we add the scrollPane and not the text area.
```

Попробуйте выполнить сами!

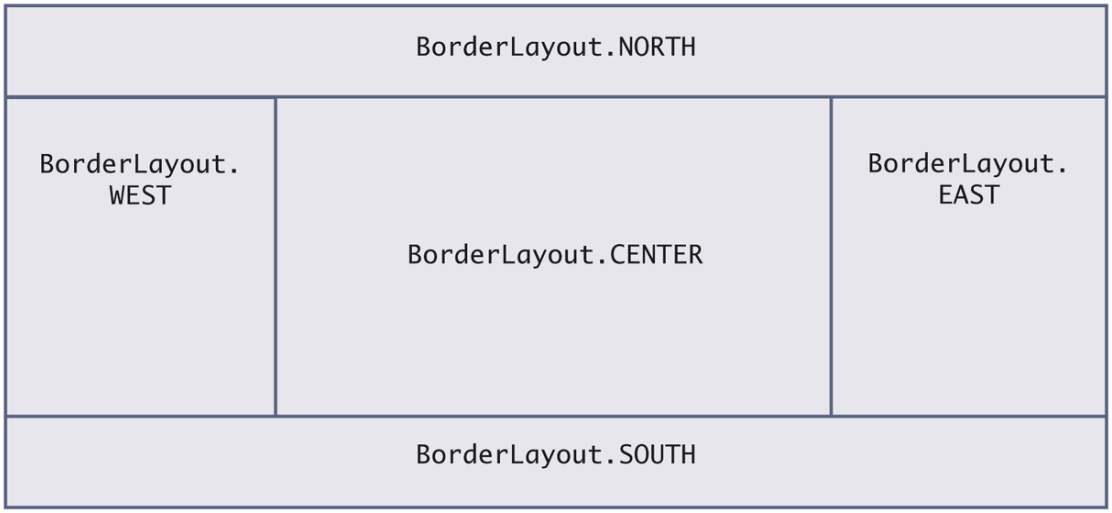


Layout Менеджеры:

BorderLayout:

Разеляет компонент на пять областей (WEST, EAST, NOTH, SOUTH and Center). Другин компоненты могут быть добавлены в любой из этих компонентов пятерками.

Display 17.8 **BorderLayout Regions**



Метод для добавления в контейнер, который есть у менеджера BorderLayout отличается и выглядит следующим образом:

```
add( comp , BorderLayout.EAST);
```

Обратите внимание, что мы можем например добавить панели JPanel в эти области и затем добавлять компоненты этих панелей. Мы можем установить расположение этих JPanel используя другие менеджеры

GridLayout менеджер

С помощью менеджера GridLayout компонент может принимать форму таблицы, где можно задать число строк и столбцов.

1	2	3	4
5	6	7	8

9	10	11	12

Если компоненту GridLayout задать 3 строки и 4 столбца, то компоненты будут принимать форму таблицы, показанной выше, и будут всегда будут добавляться в порядке их появления.

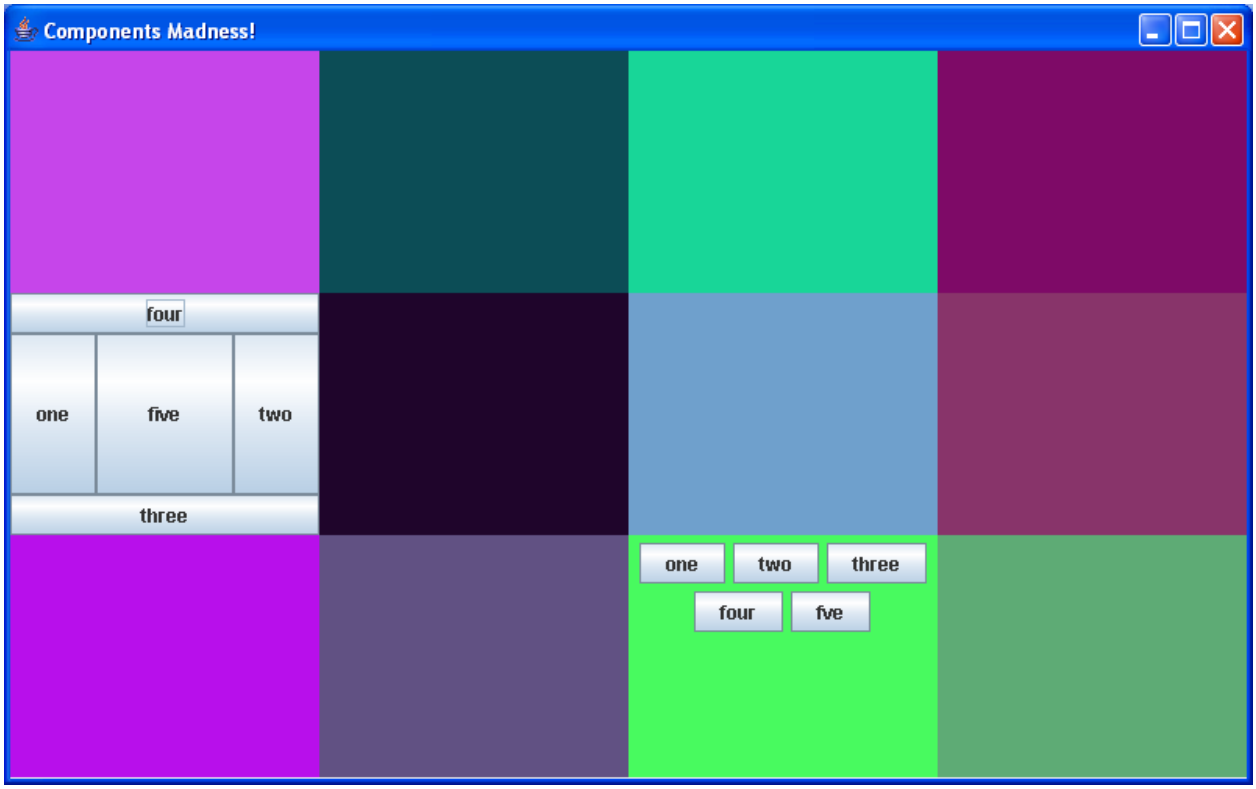
Следующий пример иллюстрирует смесь компоновки различных компонентов

Пример 4

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;



class BorderExample extends JFrame
{
      JPanel[] pnl = new JPanel[12];

      public BorderExample()
      {
            setLayout(new GridLayout(3,4));
            for(int i = 0 ; i < pnl.length ; i++)
            {
                  int r = (int) (Math.random() * 255);
                  int b = (int) (Math.random() * 255);
                  int g = (int) (Math.random() * 255);
                  pnl[i] = new JPanel();
                  pnl[i].setBackground(new Color(r,g,b));
                  add(pnl[i]);
            }

            pnl[4].setLayout(new BorderLayout());
            pnl[4].add(new JButton("one"),BorderLayout.WEST);
            pnl[4].add(new JButton("two"),BorderLayout.EAST);
            pnl[4].add(new JButton("three"),BorderLayout.SOUTH);
            pnl[4].add(new JButton("four"),BorderLayout.NORTH);
            pnl[4].add(new JButton("five"),BorderLayout.CENTER);

            pnl[10].setLayout(new FlowLayout());
            pnl[10].add(new JButton("one"));
            pnl[10].add(new JButton("two"));
```

```
            pnl[10].add(new JButton("three"));
            pnl[10].add(new JButton("four"));
            pnl[10].add(new JButton("fve"));

            setSize(800,500);
      }
      public static void main(String[]args)
      {
            new BorderExample().setVisible(true);
      }
}
```

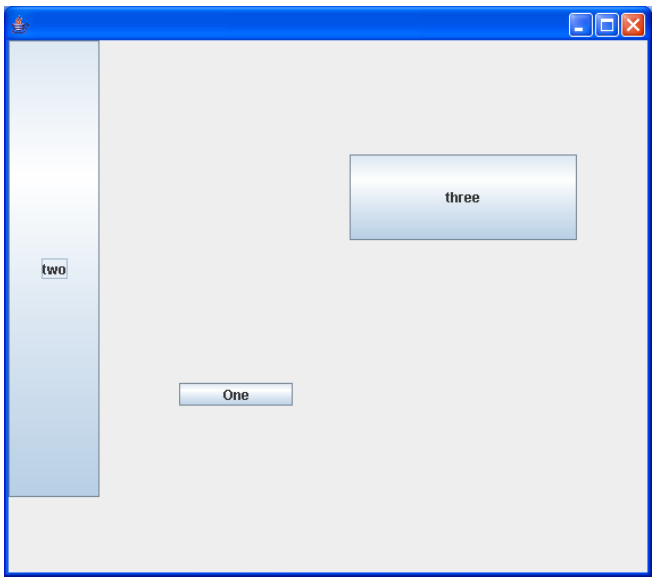
Вот такой будет иметь вид, представленный выше код



Заметьте, что JFrame имеет GridLayout размера 3 на 4 (таблица), в то время как JPanel размером (2, 1) имеет менеджер BorderLayout. A JPanel (3, 3) имеет FLowLayout.

Null Layout Manager

Иногда бывает нужно изменить размер и расположение компонента в контейнере. Таким образом, мы должны указать программе не использовать никакой менеджер компоновки, то есть (setLayout (нуль)). Так что мы получим что-то вроде этого:



Пример5

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class NullLayout extends JFrame
{
      JButton but1 = new JButton("One");;
      JButton but2 = new JButton("two");;
      JButton but3 = new JButton("three");;

      public NullLayout()
      {
            setLayout(null);
            but1.setBounds(150,300,100,20); // added at 150,300 width =
100, height=20
            but2.setSize(80,400); // added at 0,0 width = 80, height=400
            but3.setLocation(300,100);
            but3.setSize(200,75);
            // those two steps can be combined in one setBounds method
call
            add(but1);
```

```
            add(but2);
            add(but3);
            setSize(500,500);
      }
      public static void main(String[]args)
      {
            new NullLayout().setVisible(true);
      }
}
```

Слушатели событий мыши MouseListener

Мы можем реализовыать слушателей мыши и также слушателей клавиатуры на компонентах. Интерфейс MouseListener имеет следующие методы:

Method Summary	
void	**mouseClicked**(MouseEvent e) Invoked when the mouse button has been clicked (pressed and released) on a component.
void	**mouseEntered**(MouseEvent e) Invoked when the mouse enters a component.
void	**mouseExited**(MouseEvent e) Invoked when the mouse exits a component.
void	**mousePressed**(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	**mouseReleased**(MouseEvent e) Invoked when a mouse button has been released on a component.

MouseListener можно добавить к компоненту следующим образом:

Component.addMouseListener(listener);

Где слушатель является экземпляром класса, который реализует интерфейс MouseListener. Обратите внимание, что он должен обеспечивать выполнение всех методов, перечисленных в таблице .

Example6

```
import java.awt.*;
```

```
import java.awt.event.*;
import javax.swing.*;

class MyMouse extends JFrame
{
      JLabel lbl = new JLabel("");
      public MyMouse()
      {
            super("Dude! Where's my mouse ?");
            setSize(400,400);
            setLayout(new BorderLayout());
            add(lbl,BorderLayout.SOUTH);
            addMouseListener(new MouseListener()
            {
                  public void mouseExited(MouseEvent a){}
                  public void mouseClicked(MouseEvent a)
{lbl.setText("X="+a.getX()+" Y="+a.getY());}
                  public void mouseEntered(MouseEvent a) {}
                  public void mouseReleased(MouseEvent a) {}
                  public void mousePressed(MouseEvent a) {}

            });
      }
      public static void main(String[]args)
      {
            new MyMouse().setVisible(true);
      }
}
```


Dude! Where's my mouse ?
X=164 Y=203

Меню

Добавление меню в программе Java проста. Java определяет три компонента для обработки этих

• JMenuBar: который представляет собой компонент, который содержит меню.

• JMenu: который представляет меню элементов для выбора.

• JMenuItem: представляет собой элемент, который можно кликнуть из меню.

Подобно компоненту Button (на самом деле MenuItems являются подклассами класса AbstractButton). Мы можем добавить ActionListener к ним так же, как мы делали с кнопками

ЗАДАНИЯ

Создайте JFrame приложение у которо есть следующие компоненты GUI:

Одна кнопка JButton labeled “AC Milan”

Другая JButton подписана “Real Madrid”

Надпись JLabel содержит текст “Result: 0 X 0”

Надпись JLabel содержит текст “Last Scorer: N/A ”

Надпись Label содержит текст “Winner: DRAW”;

Теперь всякий раз, когда вы нажимаете на кнопку AC Milan, результат будет увеличиваться для Милана, чтобы стать сначала 1 X 0, затем 2 X 0. Last Scorer означает последнюю забившую команду. В этом случае: AC Milan. И победителем становится команда, которая имеет больше кликов кнопку на соответствующую, чем другая.

ПРИМЕРЫ РЕШЕНИЯ ЗАДАНИЙ

В теоретических сведениях.

ПРАКТИЧЕСКАЯ РАБОТА №5
РЕКУРСИЯ

Цель работы: Изучение работы с рекурсией.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В первую очередь надо понимать, что рекурсия — это своего рода перебор. Вообще говоря, всё то, что решается итеративно можно решить рекурсивно, то есть с использованием рекурсивной функции.

Так же как и у перебора (цикла) у рекурсии должно быть условие остановки — Базовый случай (иначе также как и цикл рекурсия будет работать вечно — infinite). Это условие и является тем случаем к которому рекурсия идет (шаг рекурсии). При каждом шаге вызывается рекурсивная функция до тех пор пока при следующем вызове не сработает базовое условие и произойдет остановка рекурсии(а точнее возврат к последнему вызову функции). Всё решение сводится к решению базового случая. В случае, когда рекурсивная функция вызывается для решения сложной задачи (не базового случая) выполняется некоторое количество рекурсивных вызовов или шагов, с целью сведения задачи к более простой. И так до тех пор пока не получим базовое решение.

Итак рекурсивная функция состоит из

Условие остановки или же Базовый случай

Условие продолжения или Шаг рекурсии — способ сведения задачи к более простым.

Рассмотрим это на примере нахождения факториала:

```
public class Solution {
    public static int recursion(int n) {
        // условие выхода
        // Базовый случай
        // когда остановиться повторять рекурсию ?
        if (n == 1) {
            return 1;
        }
        // Шаг рекурсии / рекурсивное условие
        return recursion(n - 1) * n;
    }
    public static void main(String[] args) {
        System.out.println(recursion(5)); // вызов рекурсивной функции
    }
}
```

Тут Базовым условием является условие когда n=1. Так как мы знаем что 1!=1 и для вычисления 1! нам ни чего не нужно. Чтобы вычислить 2! мы можем использовать 1!, т.е. 2!=1!*2. Чтобы вычислить 3! нам нужно 2!*3… Чтобы вычислить n! нам нужно (n-1)!*n. Это и является шагом рекурсии.

Иными словами, чтобы получить значение факториала от числа n, достаточно умножить на n значение факториала от предыдущего числа.

В сети при обьяснении рекурсии также даются задачи нахождения чисел Фибоначчи и Ханойская башня

Представлены задачи с различным уровнем сложности

Попробуйте их решить самостоятельно используя метод описанный выше

При решении попробуйте думать рекурсивно и ответить на вопросы:

Какой базовый случай в задаче?

Какой Шаг рекурсии или рекурсивное условие?

ЗАДАНИЯ

Задания на рекурсию

1. Треугольная последовательность

Дана монотонная последовательность, в которой каждое натуральное число k встречается ровно k раз: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4,…

По данному натуральному n выведите первые n членов этой последовательности. Попробуйте обойтись только одним циклом for.

2. От 1 до n

Дано натуральное число n. Выведите все числа от 1 до n.

3. От A до B

Даны два целых числа A и B (каждое в отдельной строке). Выведите все числа от A до B включительно, в порядке возрастания, если A < B, или в порядке убывания в противном случае.

4. Заданная сумма цифр

Даны натуральные числа k и s. Определите, сколько существует k-значных натуральных чисел, сумма цифр которых равна d. Запись натурального числа не может начинаться с цифры 0.

В этой задаче можно использовать цикл для перебора всех цифр, стоящих на какой-либо позиции.

5. Сумма цифр числа

Дано натуральное число N. Вычислите сумму его цифр.

При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).

6. Проверка числа на простоту

Дано натуральное число n>1. Проверьте, является ли оно простым. Программа должна вывести слово YES, если число простое и NO, если число составное. Алгоритм должен иметь сложность O(logn).

Указание. Понятно, что задача сама по себе нерекурсивна, т.к. проверка числа n на простоту никак не сводится к проверке на простоту меньших чисел. Поэтому нужно сделать еще один параметр рекурсии: делитель числа, и именно по этому параметру и делать рекурсию.

7. Разложение на множители

Дано натуральное число n>1. Выведите все простые множители этого числа в порядке неубывания с учетом кратности. Алгоритм должен иметь сложность O(logn)

8. Палиндром

Дано слово, состоящее только из строчных латинских букв. Проверьте, является ли это слово палиндромом. Выведите YES или NO.

При решении этой задачи нельзя пользоваться циклами, в решениях на питоне нельзя использовать срезы с шагом, отличным от 1.

9. Без двух нулей

Даны числа a и b. Определите, сколько существует последовательностей из a нулей и b единиц, в которых никакие два нуля не стоят рядом.

10. Разворот числа

Дано число n, десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке.

При решении этой задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика.

Фунция должна возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.

11. Количество единиц

Дана последовательность натуральных чисел (одно число в строке), завершающаяся двумя числами 0 подряд. Определите, сколько раз в этой последовательности встречается число 1. Числа, идущие после двух нулей, необходимо игнорировать.

В этой задаче нельзя использовать глобальные переменные и параметры, передаваемые в функцию. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметров.

12. Вывести нечетные числа последовательности

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Выведите все нечетные числа из этой последовательности, сохраняя их порядок.

В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции.

13. Вывести члены последовательности с

нечетными номерами

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Выведите первое, третье, пятое и т.д. из введенных чисел. Завершающий ноль выводить не надо.

В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции.

14. Цифры числа слева направо

Дано натуральное число N. Выведите все его цифры по одной, в обычном порядке, разделяя их пробелами или новыми строками.

При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика

15. Цифры числа справа налево

Дано натуральное число N. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами или новыми строками.

При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика.

16. Количество элементов, равных максимуму

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Определите, какое количество элементов этой последовательности, равны ее наибольшему элементу.

В этой задаче нельзя использовать глобальные переменные. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметра. В программе на языке Python функция возвращает результат в виде кортежа из нескольких чисел и функция вообще не получает никаких параметров. В программе на языке C++ результат записывается в переменные, которые передаются в функцию по ссылке. Других параметров, кроме как используемых для возврата значения, функция не получает.

Гарантируется, что последовательность содержит хотя бы одно число (кроме нуля)

17. Максимум последовательности

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Определите наибольшее значение числа в этой последовательности.

В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция возвращает единственное значение: максимум считанной последовательности. Гарантируется, что последовательность содержит хотя бы одно число (кроме нуля).

ПРИМЕР РЕШЕНИЯ ЗАДАНИЙ

«Точная степень двойки»

Дано натуральное число N. Выведите слово YES, если число N является точной степенью двойки, или слово NO в противном случае.

```
public class Rec1 {

    public static int recursion(double n) {

        if (n == 1) {
            return 1;
        }
        else if (n > 1 && n < 2) {
```

```
            return 0;
        }
        else {
            return recursion(n / 2);
        }
    }
    public static void main(String[] args) {
        double n = 64;

        if (recursion(n) == 1) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }
}
```

ПРАКТИЧЕСКАЯ РАБОТА №6
Техники сортировки

Цель работы: Целью данной практической работы освоить на практике сортировки различными методами.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

(Quick Sort) Быстрая сортировка

* Массив A[1..n] азбивается на два непустых подмассивов по отношению к "опорному элементу”

* Два суб-массивы сортируются рекурсивно посредством Quick Sort.

(Merge Sort) Сортировка слиянием

* Разделить массив A[1..n] на 2 равные части

* Сортировка слиянием двух подмассивов (рекурсивно)

* Объединить (соединить) два отсортированных подмассива

ЗАДАНИЯ

Задание 1.

Написать тестовый класс, который создает массив класса Student и сортирует массив iDNumber.

Задание 2.

Напишите класс SortingStudentsByGPA который реализует интерфейс Comparator аким образом, что она сортирует студентов с их итоговым баллом в порядке убывания.

Задание 3.

Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках.

ПРИМЕР РЕШЕНИЯ ЗАДАНИЙ

Сортировка вставками:

```
public static void insertionSort(int[] arr) {
    for(int i = 1; i < arr.length; i++){
        int currElem = arr[i];
        int prevKey = i - 1;
        while(prevKey >= 0 && arr[prevKey] > currElem){
            arr[prevKey+1] = arr[prevKey];
            prevKey--;
        }
        arr[prevKey+1] = currElem;
    }
}
```

ПРАКТИЧЕСКАЯ РАБОТА №7
Очереди

Цель работы: Изучение работы с очередями.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Конструкторы	
Stack() Создаёт пустой стек.	
Методы	
boolean	***empty()*** Служит для проверки стека на наличие элементов — он возвращает true, если стек пуст.
Object	***peek()*** Возвращает верхний элемент, не удаляя его из стека.
Object	***pop()*** Извлекает верхний элемент удаляя его из стека.
Object	***push(Object item)*** Помещает элемент в вершину стека.
int	***search(Object o)*** Метод ищет заданный элемент в стеке, возвращая количество операций pop, которые требуются для того чтобы перевести искомый элемент в вершину стека. Если заданный элемент в стеке отсутствует, этот метод возвращает -1.

ЗАДАНИЯ

Задача 1 Игра в "пьяницу"

В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт - проигрывает.

Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту ("шестерка берет туза").

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

Напишите программу, которая моделирует игру в пьяницу и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9.

Входные данные

Программа получает на вход две строки: первая строка содержит 5 карт первого игрока, вторая - 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

Выходные данные

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 106 ходов игра не заканчивается, программа должна вывести слово botva.

Пример ввода

1 3 5 7 9

2 4 6 8 0

second 5

ПРИМЕР РЕШЕНИЯ ЗАДАНИЯ

(Используя arraydeque)

Реализуйте аналогичные очереди процедуры, реализующие стек, и на их основе напишите нижеописанную программу

С помощью стека реализуйте следующий диалог. На вход программе подается последовательность чисел. С ней происходит следующее:

* если число положительное, то оно помещается в стек, на выход программы печатается 0,

* если число 0, то, если стек не пуст, из него извлекается стоящее там число и печатается на выход программы, если же стек пуст, печатается -1.

```
import java.util.ArrayDeque;
```

```
public class SimpleStack {

    public static String work(int word[]){
        String res = "";

        ArrayDeque<Integer> wordArray = new ArrayDeque<Integer>();

        for(int i = 0; i<word.length;i++){
            if(word[i]>0){
                wordArray.add(word[i]);
                res+=",0";
            }
            else if(word[i] == 0){
                if(wordArray.isEmpty()){
                    res+= ",-1";
                }
                else{
                    res+="," + (int) wordArray.pop();
                }
            }
        }
        return res;
    }

    public static void main(String[] args) {
        int word[] =new int[]{0,1,8,4,0,3,0,8,8,3,7} ;
        System.out.println(work(word));
    }
}
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Зорина Н.В. Курс лекций по Объектно-ориентированному программированию на Java, МИРЭА, Москва, 2016
2. Программирование на языке Java: работа со строками и массивами. Методические указания. [Электронный ресурс] : Учебно-методические пособия — Электрон. дан. — СПб. : ПГУПС, 2015. — 24 с.
3. Кожомбердиева, Г.И. Программирование на языке Java: создание графического интерфейса пользователя: учеб. пособие. [Электронный ресурс] : Учебные пособия / Г.И. Кожомбердиева, М.И. Гарина. — Электрон. дан. — СПб.: ПГУПС, 2012. — 67 с.
4. Вишневская, Т.И. Технология программирования. Часть 1. [Электронный ресурс] / Т.И. Вишневская, Т.Н. Романова. — Электрон. дан. — М. : МГТУ им. Н.Э. Баумана, 2007. — 59 с.