ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ В РАЗРАБОТКУ ПРОГРАММ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ JAVA

Язык Java — это объектно-ориентированный язык программирования. Программы написанные на Java могут выполняться на различных операционных системах при наличии необходимого ПО - Java Runtime Environment. Для того чтобы создать программу на языке Java необходимо следующее ПО:

- Java Development Kit (JDK).
- Java Runtime Environment (JRE).
- Среда разработки. Например, NetBeans или IDE IntelliJ IDEA.

Установка ПО

Для того, чтобы скачать ПО, можно воспользоваться следующими ссылками:

1. Программа “IntelliJ IDEA”:https://www.jetbrains.com/idea/download/#section=windows

2. Программа “NetBeans IDE”: https://netbeans.org/downloads/

3. JDK: http://www.oracle.com/technetwork/java/javase/downloads/index.html

По умолчанию, скаченный JDK установится в папку с таким адресом: C:\Program Files\Java

Начало работы с программой

После установки одной из сред разработки (“IntelliJ IDEA” или “NetBeans IDE”) можно начать создавать проекты. Далее будет показано, как начать новый проект на примере программы «IntelliJ IDEA».

1. В открытом окне программы выбираем “Create New Project”.

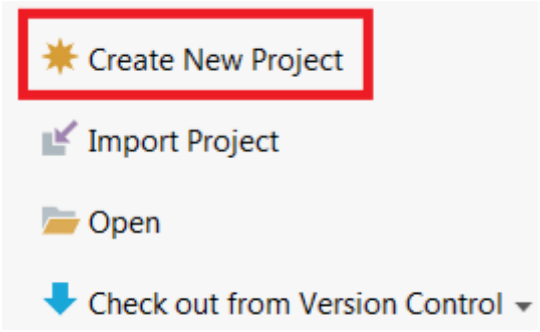

Рисунок 1

2. Щёлкаем «New», чтобы загрузить JDK.

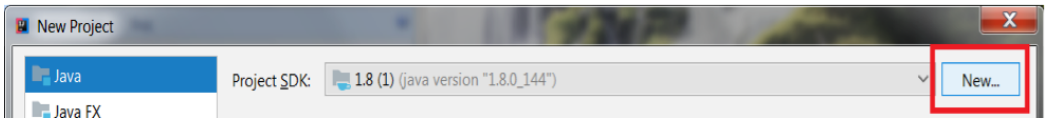

Рисунок 2

3. Из выпадающего списка папок выбираем «Program Files».

Рисунок 3

4. В «Program Files» выбираем папку «Java».

Рисунок 4

5. Далее выбираем папку «jdk...».

Рисунок 5

6. Затем дважды нажимаем «Next» в нижнем правом углу.

Рисунок 6

7. Выбираем название для будущего проекта и затем нажимаем кнопку «Finish».

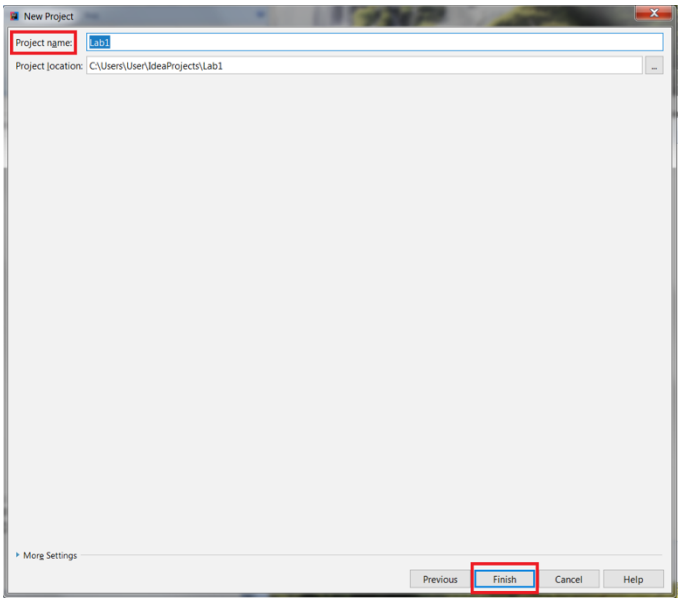

Рисунок 7

8. Щёлкаем правой кнопкой мыши по папке «src» и создаем новый пакет.

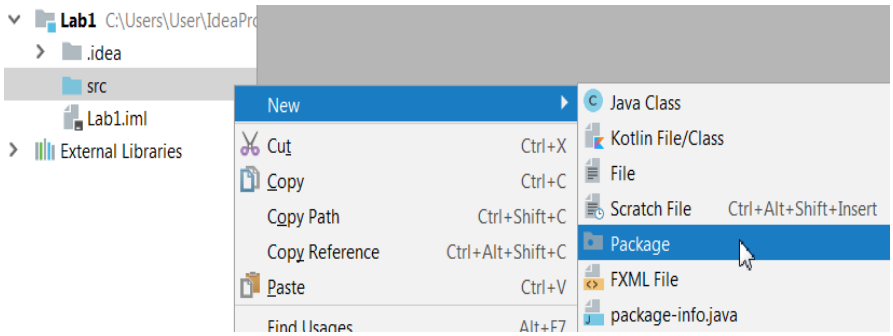

Рисунок 8

9. Вводим название пакета. «Package» – это оператор, который сообщает транслятору, в каком пакете должны определяться содержащиеся в данном файле классы.

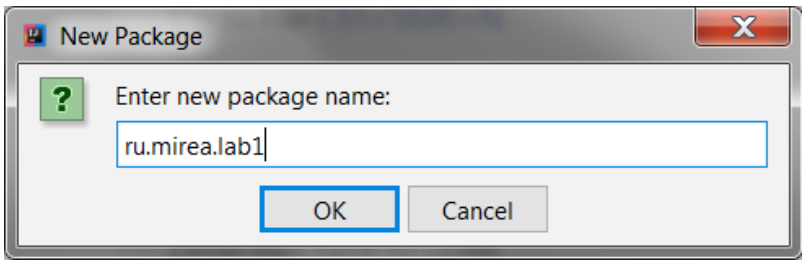

Рисунок 9

10. Щелкаем по созданному пакету правой кнопкой мыши и создаем новый класс.

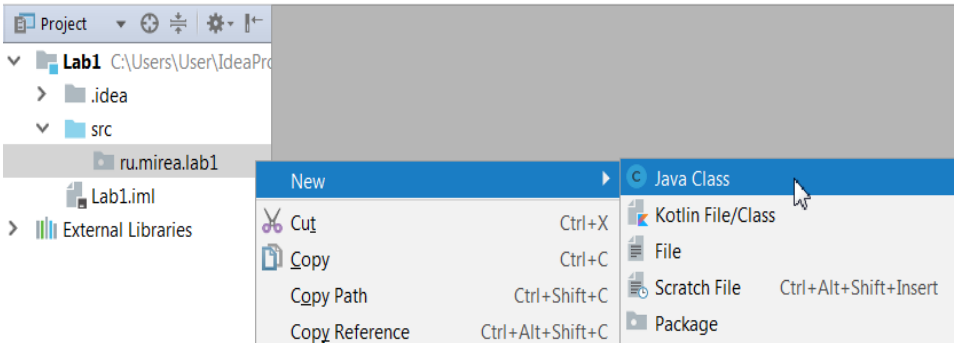

Рисунок 10

11. Новый проект создан. Теперь можно приступать к написанию кода.

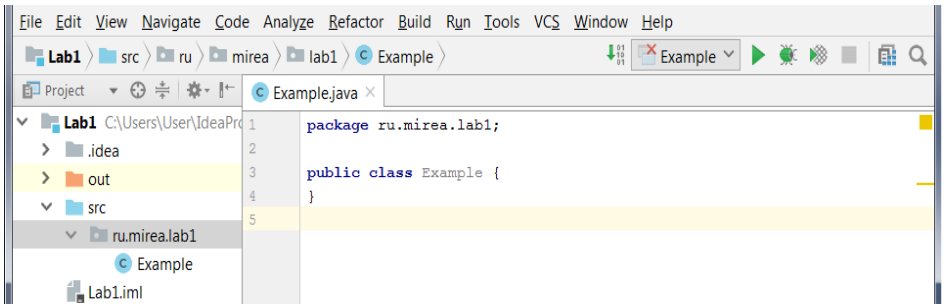

Рисунок 11

Горячие клавиши Intellij IDEA (hot keys)

Таблица 1 – Клавиши редактирования

Ctrl + Space	Список компонентов (класса, метода, переменной)
Ctrl + Shift + Space	Smart code – фильтрует список из методов и переменных ожидаемого типа
Ctrl + Alt + Space	Название любого класса проекта независимо от импортируемых
Ctrl + Shift + Enter	Завершение оператора
Ctrl + P	Сведения о параметрах (в пределах аргументов вызываемого метода)
Ctrl + Q	Быстрый поиск документации
Shift + F1	Внешняя документация
Ctrl + наведение мышью на фрагмент кода	Краткая информация
Ctrl + F1	Показать описания ошибки или предупреждения в каретку
Alt + Insert	Генерация кода (Getters, Setters, Constructors, hashCode/equals, toString)
Ctrl + O	Переопределение метода

Продолжение таблицы 1

Ctrl + I	Реализация методов
Ctrl + Alt + T	Поместить фрагмент кода в (if..else, try..catch, for, synchronized, etc.)
Ctrl + /	Однострочное комментирование / раскомментирование
Ctrl + Shift + /	Многострочное комментирование / раскомментирование
Ctrl + W	Выбирает последовательность возрастающих блоков кода
Alt + Q	Контекстная информация
Alt + Enter	Показать предлагаемое исправление
Ctrl + Alt + L	Форматирование кода
Ctrl + Alt + O	Удалить неиспользуемые импорты
Ctrl + Alt + I	Авто-отступ линии
Tab / Shift + Tab	Отступ / удаление отступа выбранному фрагменту кода
Ctrl + X or Shift + Delete	Вырезать фрагмент кода
Ctrl + C or Ctrl + Insert	Копировать фрагмент кода
Ctrl + V or Shift + Insert	Вставить фрагмент кода из буфера обмена
Ctrl + Shift + V	Вставить последний фрагмент кода из буфера обмена
Ctrl + D	Дублирование строки
Ctrl + Y	Удаление строки
Ctrl + Shift + J	Объединение строк
Ctrl + Enter	Разделение строки
Shift + Enter	Начать с новой строки

Продолжение таблицы 1

Ctrl + Shift + U	Переключить стоящее слово рядом с кареткой в нижний / верхний регистр
Ctrl + Shift +] / [	Выделить код до конца / начала блока
Ctrl + Delete	Удалить слово после каретки
Ctrl + Backspace	Удалить слово перед кареткой
Ctrl + NumPad+/-	Развернуть / свернуть блок кода
Ctrl + Shift + NumPad+	Развернуть все
Ctrl + Shift + NumPad-	Свернуть все
Ctrl + F4	Закрыть активное окно редактора

Таблица 2 – Клавиши поиска и замен

Ctrl + F	Поиск
F3	Искать дальше
Shift + F3	Искать назад
Ctrl + R	Замена
Ctrl + Shift + F	Искать по проекту
Ctrl + Shift + R	Заменить по проекту
Ctrl + Shift + S	Поиск по шаблону
Ctrl + Shift + M	Замена по шаблону

Таблица 3 – Клавиши поиска использования кода

Alt + F7 / Ctrl + F7	Найти использования / Найти использования в файле
Ctrl + Shift + F7	Выделить используемое в файле
Ctrl + Alt + F7	Показать использования

Таблица 4 – Клавиши компиляции и выполнений

Ctrl + F9	Структурирование проекта и сборка измененных файлов
Ctrl + Shift + F9	Компиляция выбранного файла пакета или модуля
Alt + Shift + F10	Выбрать конфигурацию и запустить
Alt + Shift + F9	Выбрать конфигурацию и запустить в debug режиме
Shift + F10	Запуск проекта
Shift + F9	Запуск проекта в debug режиме
Ctrl + Shift + F10	Выполнить в контексте конфигурации из редактора

Таблица 5 – Клавиши отладки

F8	Шаг обхода
F7	Шаг
Shift + F7	Умный шаг
Shift + F8	Выйти
Alt + F9	Запуск до курсора
Alt + F8	Вычисление выражения
F9	Резюме программы
Ctrl + F8	Переключить точку остановки
Ctrl + Shift + F8	Показать точки остановки

Таблица 6 – Клавиши навигации

Ctrl + N	Перейти к классу
Ctrl + Shift + N	Перейти к файлу
Ctrl + Alt + Shift + N	Перейти к символу

Продолжение таблицы 6

Alt + Right/Left	Переход к следующей / предыдущей вкладки редактора
F12	Вернуться к предыдущему окну инструмента
Esc	Переход к редактору (от окна инструментов)
Shift + Esc	Скрыть активное или последнее активное окно
Ctrl + Shift + F4	Закрыть активное run/messages/find/... окно
Ctrl + G	Переход к номеру строки
Ctrl + E	Последние файлы
Ctrl + Alt + Left/Right	Переход между вкладками назад / вперед
Ctrl + Shift + Backspace	Переход в последнее местоположение Редактора
Alt + F1	Выбор текущего файла или символа в любом режиме
Ctrl + B or Ctrl + Click	Переход к объявлению
Ctrl + Alt + B	Переход к реализации
Ctrl + Shift + I	Открыть быстрый поиск по определению
Ctrl + Shift + B	Переход к объявлению типа
Ctrl + U	Переход к супер-методу или классу
Alt + Up/Down	Переход к предыдущему / следующему методу
Ctrl +] / [	Переход в конец / начало блока

Ctrl + F12	Файловая структура

Продолжение таблицы 6

Ctrl + H	Иерархия Типа
Ctrl + Shift + H	Иерархия метода
Ctrl + Alt + H	Иерархии вызовов
F2 / Shift + F2	Следующая / предыдущая выделенная ошибка
F4 / Ctrl + Enter	Редактирование исходника / Просмотр
Alt + Home	Показ панели навигации
F11	Переключение закладки
Ctrl + #[0-9]	Переход к закладке по номером…
Shift + F11	Показать закладки

Таблица 7 – Клавиши рефакторинга

F5	Копирование
F6	Перемещение
Alt + Delete	Безопасное удаление
Shift + F6	Переименование
Ctrl + F6	Изменить сигнатуру
Ctrl + Alt + N	Встроить
Ctrl + Alt + M	Поместить в метод
Ctrl + Alt + V	Поместить в переменную
Ctrl + Alt + F	Поместить в поле
Ctrl + Alt + C	Поместить в константу
Ctrl + Alt + P	Поместить в параметр

Таблица 8 – Клавиши взаимодействия с VCS

Ctrl + K	Коммит проекта в VCS
Ctrl + T	Обновить проект из VCS
Alt + Shift + C	Посмотреть последние изменения
Alt + BackQuote (`)	Быстрый VCS

Таблица 9 – Интерактивные шаблоны

Ctrl + Alt + J	Окружение с живым шаблоном
Ctrl + J	Вставка живого шаблона
iter	Итерация в Java SDK 1.5
inst	Проверяет тип объекта с InstanceOf
itco	Итерация элементов java.util.Collection
itit	Итерация элементов java.util.Iterator
itli	Итерация элементов java.util.List
psf	public static final
psvm	public static void main
thr	throw new
sout	System.out.println()

ПРАКТИЧЕСКАЯ РАБОТА №1. КЛАССЫ, КАК НОВЫЕ ТИПЫ ДАННЫХ. ПОЛЯ ДАННЫХ И МЕТОДЫ

Цель работы: освоить на практике работу с классами языка программирования 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 объекта типа класс. Другими словами, экземпляр экземпляра класса. Все экземпляры класса имеют аналогичные свойства, как задано 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 можно представить класс в виде UML[1] диаграммы как прямоугольник в виде как трех секций, в котором присутствует секция наименования класса, секция инкапсуляции данных и методов (функций или операций) класса. Пример общего представления диаграммы класса представлен на рисунке 1.1.

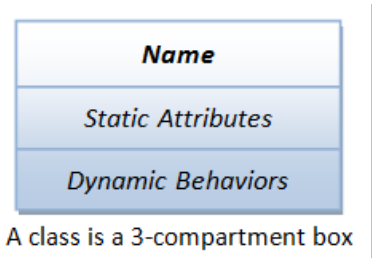

Рисунок 1.1 - Диаграмма класса. Общее представление.

Рассмотрим подробнее диаграмму 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”), представленном на рисунке в виде прямоугольника.

На рисунке 1.2 показано несколько примеров классов. У каждого из них есть имя, переменные класса и методы.

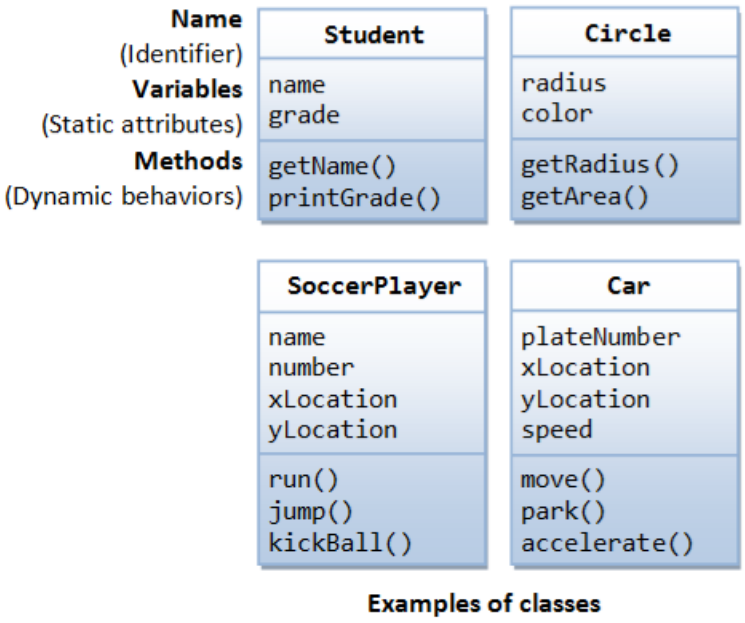

Рисунок 1.2 - Примеры классов

На рисунке 1.3 показаны два экземпляра класса типа Student "paul" и "peter" в виде UML диаграммы экземпляра класса.

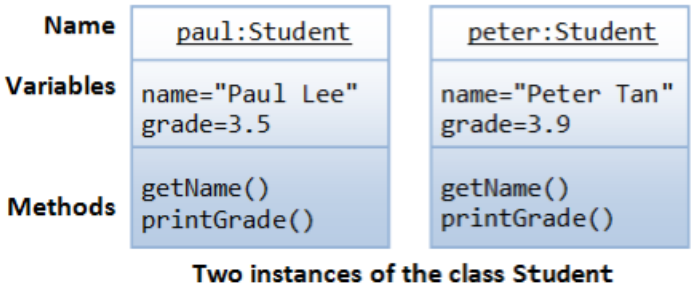

Рисунок 1.3 - Экземпляры класса Student.

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 по определению класса:

1) Класс, тип данных, определяемый программистом, абстрактный тип данных, повторно-используемый программный объект, который имитирует реальные сущности предметной области. Класс можно представить графически в виде контейнера на UML диаграмме, который состоит из трех условных частей и содержит имя, переменные и методы.

2) Класс инкапсулирует статическое состояние объекта, его атрибуты или свойства данных в виде переменных класса и поведение объекта в виде методов, которые могут реализовывать определенные алгоритмы.

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

Экземпляр класса — это представление (или реализация) конкретного представителя класса.

2. Определение класса.

В Java, мы используем слово class как ключевое или служебное слово, например, чтобы задать определение класса. Для примера:

```
public class Circle {    // class name
     double radius;      // variables
     String color;

     double getRadius() {...} // methods
     double getArea() {...}
}
public class SoccerPlayer {   // class name
     int number;                  // variables
     String name;
     int x, y;
```

```
    void run() {...}    // methods
    void kickBall() {...}
}
```

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

Конвенция кода для класса (Class Naming Convention).

Конвенцией кода называют соглашение между программистами о правилах написания кода. Соглашение содержит правила именования переменных и не только. Например, в соответствии с конвенцией кода на Java имя класса должно быть всегда существительным или словосочетанием из нескольких слов. Все слова должны с прописной буквы (так, называемая верблюжья нотация или camel notation). Совет: для имени класса всегда используйте существительное в единственном числе. Выберите значимое и самодостаточное имя для названия класса. Для примера, SoccerPlayer, HttpProxyServer, FileInputStream, PrintStream and SocketFactory будут подходящими именами в определенной предметной области, для которой вы пишете программу.

3. Создание экземпляров класса

Чтобы создать экземпляр класса, вы должны выполнить следующие действия:

• объявить идентификатор экземпляра (имя экземпляра) конкретного класса.

• Сконструировать экземпляр класса (то есть выделить память для

экземпляра и инициализировать его) с помощью оператора "new".

Например, предположим, что у нас есть класс с именем Circle , тогда мы можем создавать экземпляры класса Circle, следующим образом:

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

4. Операция получения доступа к компонентам класса.

Доступ к компонентам класса осуществляется с помощью операции получения доступа, а именно операции точка “.”

Переменные и методы, входящие в состав класса, формально называется переменные-поля данных класса и методы класса и являются компонентами 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).

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,

```
// Declare and construct instances c1 and c2 of the class Circle
Circle c1 = new Circle ();
Circle c2 = new Circle ();
// Invoke member methods for the instance c1 via dot operator
System.out.println(c1.getArea());
System.out.println(c1.getRadius());
// Reference member variables for instance c2 via dot operator
```

```
c2.radius = 5.0;
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 свой собственный радиус) .

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

Вы можете использовать для доступа к открытым полям и методам операцию точка ".", например - anInstance.aVariable и anInstance.aMethod().

5. Переменные - поля данных класса

Переменная-поле данных имеет имя (идентификатор) и тип, а также может иметь значение определенного типа, например базового или типа определенного программистом ранее. Переменная-поле данных может также быть экземпляром определенного класса (которые будут обсуждаться позже).

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

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

Формальный синтаксис для определения переменной в Java:

```
[модификатор_доступа] тип имя_перем [= иниц_знач];
[модификатор_доступа] тип имя_пер-1 [=иниц_знач-1], тип имя_пер-
2 [=иниц_знач-2] ...;
```

Например:

```
private double radius;
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:

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

Конвенция кода о правилах записи имен методов гласит следующее: имя метода должно быть глаголом или начинаться глаголом в виде фразы из нескольких слов, первое слово записывается 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

На рисунке ниже представлена диаграмма класса и его трех экземпляров.

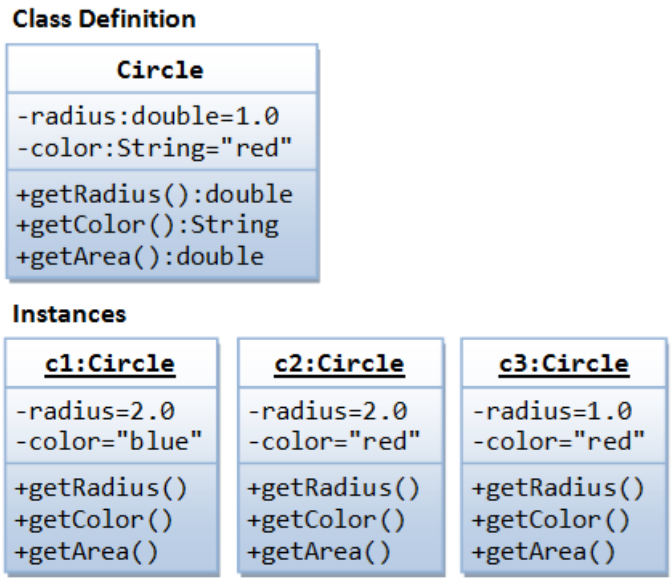

Рисунок 1.4 – Диаграмма класса и его экземпляров.

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

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 изображенного на UML диаграмме класса Circle.java:

Circle.java

```
// Define the Circle class
public class Circle { // Save as "Circle.java"

     // Private variables
     private double radius;
     private String color;

     // Constructors (overloaded)
     public Circle() { // 1st Constructor
     radius = 1.0;
     color = "red";
     }
     public Circle(double r) { // 2nd Constructor
          radius = r;
          color = "red";
     }

     public Circle(double r, String c) { // 3rd Constructor
          radius = r;
          color = c;
     }
     // Public methods
     public double getRadius() { return radius;
     }
     public String getColor() {
     return color;
     }
     public double getArea() {
     return radius*radius*Math.PI;
     }
```

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 отдельным 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 содержит метод main(), теперь мы можем откомпилировать и запустить программу.

TestCircle.java

```
public class TestCircle { // Save as "TestCircle.java"
     public static void main(String[] args) { // Execution entry
point
          // Construct an instance of the Circle class called c1
          Circle c1 = new Circle(2.0, "blue"); // Use 3rd
constructor
          System.out.println("Radius is " + c1.getRadius() +
"Color is " + c1.getColor() + "Area is " + c1.getArea()); // use
dot operator to invoke member methods
     }
     // Construct another instance of the Circle class called c2
     Circle c2 = new Circle(2.0); // Use 2nd constructor
     System.out.println("Radius is " + c2.getRadius() + "Color is
" + c2.getColor() + "Area is " + c2.getArea());
     // Construct yet another instance of the Circle class called
c3
     Circle c3 = new Circle(); // Use 3rd constructor
     System.out.println("Radius is " + c3.getRadius() + "Color is
" + c3.getColor() + "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 – это 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,

```
Circle c1 = new Circle();
Circle c2 = new Circle(2.0);
```

```
Circle c3 = new Circle(3.0, "red");
```

Конструктор отличается от обычного метода следующим:

- 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, конструктор Circle() в рассмотренном 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 списку их аргументов (либо количество аргументов, или типа аргументов, или их порядок).

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

```
  public class TestMethodOverloading {
     public static int average(int n1, int n2) { // A
     return (n1+n2)/2;
     }

     public static double average(double n1, double n2) { // B
          return (n1+n2)/2;
     }

     public static int average(int n1, int n2, int n3) { // C
          return (n1+n2+n3)/3;
     }

     public static void main(String[] args) {
          System.out.println(average(1, 2)); // Use A
          System.out.println(average(1.0, 2.0)); // Use B
          System.out.println(average(1, 2, 3)); // Use C
          System.out.println(average(1.0,2)); //Use B - int 2
implicitly casted to double 2.0
          // average(1, 2, 3, 4); // Compilation Error - No
matching method
     }
  }
```

Рассмотрим перегрузку конструктора класса Circle.

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

```
Circle()
Circle(double r)
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. Модификаторы контроля доступа- public или private.

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: общедоступные (public) элементы обозначены со знаком "+", в то время как закрытые (private) элементы со знаком " - ".

11. Информация по сокрытию реализации и инкапсуляции.

Класс инкапсулирует имя, статические атрибуты и динамическое поведение в "виде трех частей целого". Можно себе представить класс в виде некоторой коробки, на которой написано имя, внутри нее есть некоторое содержимое. После того, как класс определен, то есть вы туда сто-то положили, написали название на коробке, то можно закрыть "крышку" у этой коробки и поставить ее на полку. В дальнейшем как вы сами можете

использовать эту коробку, так и можете предоставить возможность использовать ее всем желающим. Другими словами, любой желающий может открыть "крышку" этой коробки и использовать ее содержимое, то есть использовать ваш класс в своем приложении. Этого нельзя сделать, программируя в традиционном процедурном-ориентированном стиле, например на языке Си, так как статические атрибуты (или переменные) разбросаны по всей программе и в файлах, заголовки которых вы включаете в код. Вы не сможете "вырезать" куски из части программы на Си, включить их в другую программу и ожидать что такая программа запуститься без внесения в код серьезных изменений.

Переменные-поля данных класса, как правило, скрыты от внешнего слоя (то есть, недоступны другим классам), для этого осуществляется разграничение доступа или контроль доступа с использованием модификатора доступа - private. Доступ к закрытым переменным - полям класса предоставляются через методы, объявленные с модификатором public, например, getRadius () и getColor().

Использование такого подход соответствует принципу сокрытия информации – принципу инкапсуляции. То есть, объекты могут общаться друг с другом, только через хорошо определенные интерфейсы (публичные методы). Объектам не позволено знать детали реализации других объектов. Детали реализации всегда скрыты извне или инкапсулированы внутри класса. Сокрытие информации облегчает повторное использование класса.

Основное правило при создании определения классов: не объявляйте переменные общедоступными – с модификатором доступа public, если у вас на то нет веских оснований, не нарушайте принцип инкапсуляции.

ЗАДАНИЕ

Необходимо реализовать простейший класс на языке программирования Java. Не забудьте добавить метод toString() к вашему классу. Так-же в программе необходимо предусмотреть класс-тестер для тестирования класса и вывода информации об объекте.

Упражнение 1.

Реализуйте простейший класс «Собака».

Упражнение 2.

Реализуйте простейший класс «Мяч».

Упражнение 3.

Реализуйте простейший класс «Книга».

Пример выполнения задания 1.

Задание: Реализуйте класс «Собака, который содержит данные экземпляра, которые представляют имя собаки и ее возраст. Определить конструктор собаки, чтобы принять и инициализировать данные экземпляр. Включите методы получения и установки для имени и возраста. Включите метод вычисления и возвращает возраст собаки в "человеческих" лет (возраст семь раз собаки). Включите метод toString(), который возвращает описание на одну строку собаки. Создание класса драйвера под названием питомника, основной метод конкретизирует и обновляет несколько объектов собаки.

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");
     }
}
```

TestDog.java

```
import java.lang.*;
public class TestDog {
     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. ЦИКЛЫ, УСЛОВИЯ, ПЕРЕМЕННЫЕ И МАССИВЫ В JAVA.

Цель работы: Целями данной работы являются получение практических навыков разработки программ, изучение синтаксиса языка Java, освоение основных конструкций языка Java (циклы, условия, создание переменных и массивов, создание методов, вызов методов), а также научиться осуществлять стандартный ввод/вывод данных.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Язык Java - это объектно-ориентированный язык программирования. Программы написанные на Java могут выполняться на различных операционных системах при наличии необходимого ПО - Java Runtime Environment.

Для того чтобы создать программу на языке Java необходимо следующее ПО:

- Java Development Kit (JDK)
- Java Runtime Environment (JRE)
- Среда разработки. Например NetBeans или IDE IntelliJ IDEA.

Создание программы на Java.

Чтобы начать написание программы необходимо запустить среду разработки. При первом запуске среды обычно нужно указать путь к JDK, чтобы можно было компилировать код и запускать программу. В среде разработки необходимо создать Java проект, после чего необходимо создать пакет и в нем создать какой-либо класс. Также в свойствах проекта нужно указать класс, с которого будет начинаться запуск программы.

В классе, с которого будет начинаться запуск программы обязательно должен быть статический метод main(String[]), который принимает в качестве аргументов массив строк и не возвращает никакого значения.

Пример:

```
package example;
public class Example {
     public static void main(String[] args) {
     }
}
```

В этом примере создан класс Example, располагающийся в пакете example. В нем содержится статический(ключевое слово static) метод main. Массив строк, который передается методу main() - это аргументы командной строки. При запуске Java программы, выполнение начнется с метода main().

Переменные.

Чтобы объявить переменную, необходимо указать тип переменной и ее имя. Типы переменной могут быть разные: целочисленный(long, int, short, byte), число с плавающей запятой(double, float), логический(boolean), перечисление, объектный(Object).

Переменным можно присваивать различные значения с помощью оператора присваивания "=".

Целочисленным переменным можно присваивать только целые числа, а числам с плавающей запятой - дробные. Целые числа обозначаются цифрами от 0 до 9, а дробные можно записывать отделяю целую часть от дробной с помощью точки. Переменным типа float необходимо приписывать справа букву "f", обозначающую, что данное число типа float. Без этой буквы число будет иметь тип double.

Класс String - особый класс в Java, так как ему можно присваивать значение, не создавая экземпляра класса(Java это сделает автоматически). Этот класс предназначен для представления строк. Строковое значение записывается буквами внутри двойных кавычек.

Пример:

```
float length = 2.5f;
double radius = 10024.5;
int meanOfLife = 42;
Object object = new String("Hello, world!"); String b = "Once
compiled, runs everywhere?";
```

С целочисленными переменными можно совершать различные операции: сложение, вычитание, умножение, целое от деления, остаток от деления. Эти операции обозначаются соответственно "+", "-", "*", "/", "%". Для чисел с плавающей запятой применимы операции сложения, вычитания, умножения, деления. Для строк применима операция "+", обозначающая конкатенацию, слияние строк.

Массивы.

Массив — это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу.

Для того чтобы создать массив переменных, необходимо указать квадратные скобки при объявлении переменной массива. После чего необходимо создать массив с помощью оператора new. Необходимо указать в квадратных скобках справа размер массива. Например, чтобы создать массив из десяти целочисленных переменных типа int, можно написать так:

```
int[]b = new int[10];
```

Для того чтобы узнать длину массива, необходимо обратиться к ег свойству length через точку, например b.length.

Для того чтобы получить какой либо элемент массива, нужно указать после имени массива в квадратных скобках индекс, номер элемента. Массивы нумеруются с нуля. Например, чтобы получить 5 элемент массива, можно написать так: b[4].

Условия.

Условие - это конструкция, позволяющая выполнять то или другое действие, в зависимости от логического значения, указанного в условии. Синтаксис создания условия следующий:

```
if(a==b) {
     //Если a равно b, то будут выполняться операторы в этой
области
} else {
     //Если a не равно b, то будут выполняться операторы в этой
области
}
```

Если логическое условие, указанное в скобках после ключевого слова if, истинно, то будет выполняться блок кода, следующий за if, иначе будет выполняться код, следующий за ключевым словом else. Блок else не обязателен и может отсутствовать.

Скобками "{", "}" обозначается блок кода, который будет выполняться. Если в этом блоке всего 1 оператор, то скобки можно не писать(для условий и циклов).

Логическое условие составляется с помощью переменных и операторов равенства, неравенства, больше, меньше, больше или равно, меньше или равно, унарная операция не. Эти операторы обозначаются соответственно "==", "!=", ">", "<", ">=", "<=", "!". Результатом сравнения является логическое значение типа boolean, которое может иметь значение true ("истина") или false ("ложь"). Логические значения могут храниться в переменных типа boolean.

Циклы.

Цикл - это конструкция, позволяющая выполнять определенную часть кода несколько раз. В Java есть три типа циклов for, while, do while. Цикл for - это цикл со счетчиком, обычно используется, когда известно, сколько раз должна выполниться определенная часть кода.

Синтаксис цикла for:

```
for(int i=0;i<10;i++) {
     //Действия в цикле
}
```

В данном примере, в цикле объявлена переменная i, равная изначально 0. После точки с запятой ";" написано условие, при котором будет выполняться тело цикла (пока i<10), после второй точки с запятой указывается как будет изменяться переменная i (увеличиваться на 1 каждый раз с помощью операции инкремента "++"). Прописывать условие, объявлять переменную и указывать изменение переменной в цикле for не обязательно, но обязательно должны быть точки с запятой.

Цикл while - это такой цикл, который будет выполняться, пока логическое выражение, указанное в скобках истинно. Синтаксис цикла while:

```
while(logic) {
     //Тело цикла
}
```

В данном примере тело цикла будет выполняться, пока значение логической переменной logic равно true, то есть истинно.

Цикл do while - это такой цикл, тело которого выполнится хотя бы один раз. Тело выполнится более одного раза, если условие, указанное в скобках истинно.

```
do{
     //Тело цикла
}while(logic);
```

Потоки ввода/вывода и строки в Java, класс String

Для ввода данных используется класс Scanner из библиотеки пакетов

Этот класс надо импортировать в той программе, где он будет использоваться. Это делается до начала открытого класса в коде программы.

В классе есть методы для чтения очередного символа заданного типа со стандартного потока ввода, а также для проверки существования такого символа.

Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный

поток ввода (клавиатура) в Java представлен объектом — System.in. А стандартный поток вывода (дисплей) — уже знакомым вам объектом System.out. Есть ещё стандартный поток для вывода ошибок — System.err

```
importjava.util.Scanner; // импортируем класс
importjava.util.Scanner; // импортируем класс
public class Main {
     public static void main(String[] args) {
          Scanner sc = new Scanner(System.in); // создаём объект
класса  Scanner
          int i = 2;
          System.out.print("Введите целое число: ");
          if(sc.hasNextInt()) { // возвращает истинну, если с
потока ввода  можно  считать целое   число
               i = sc.nextInt(); // считывает целое число с
потока ввода и сохраняем в переменную
               System.out.println(i*2);
          }
          else{
               System.out.println("Вы ввели не целое число");
          }
     }
}
```

Имеется также метод nextLine(), позволяющий считывать целую последовательность символов, т.е. строку, а, значит, полученное через этот метод значение нужно сохранять в объекте класса String. В следующем примере создаётся два таких объекта, потом в них поочерёно записывается ввод пользователя, а далее на экран выводится одна строка, полученная объединением введённых последовательностей символов.

```
Importjava.util.Scanner;
public class Main {
     public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
          String s1, s2;
          s1 = sc.nextLine();
          s2 = sc.nextLine(); System.out.println(s1 + s2);
     }
}
```

Существует и метод hasNext(), проверяющий остались ли в потоке ввода какие-то символы.

- В классе String существует масса полезных методов, которые можно применять к строкам (перед именем метода будем указывать тип того значения, которое он возвращает): intlength() — возвращает длину строки (количество символов в ней);
- boolean isEmpty() — проверяет, пустая листрока;
- String replace(a, b) — возвращает строку, где символ a (литерал или переменная типа char) заменён на символb;
- String toLowerCase() — возвращает строку, где все символы исходной строки преобразованы кстрочным;
- String toUpperCase() — возвращает строку, где все символы исходной строки преобразованы кпрописным;
- boolean equals(s) — возвращает истинну, если строка к которой применён метод, совпадает со строкой s указанной в аргументе метода (с помощью оператора == строки сравнивать нельзя, как и любые другие объекты);
- int indexOf(ch) — возвращает индекс символа ch в строке (индекс это порядковый номер символа, но нумероваться символы начинают с нуля). Если символ совсем не будет найден, то возвратит -1. Если символ встречается в строке несколько раз, то вовзвратит индекс его первоговхождения.
- int lastIndexOf(ch) — аналогичен предыдущему методу, но возвращает индекс последнего вхождения, если смивол встретился в строке несколькораз.
- int indexOf(ch,n) — возвращает индекс символа ch в строке, но начинает проверку с индекса n (индекс это порядковый номер символа, но нумероваться символы начинают с нуля). char charAt(n) — возвращает код символа, находящегося в строке под индексом n (индекс это порядковый номер символа, но нумероваться символы начинают снуля).

```
public class Main {
     public static void main(String[] args){
          String s1 = "firefox";
          System.out.println(s1.toUpperCase()); // выведет
«FIREFOX»
          String s2 = s1.replace('o', 'a');
          System.out.println(s2); // выведет «firefax»
          System.out.println(s2.charAt(1)); // выведет «i»
          int i;
          i = s1.length();
          System.out.println(i); // выведет 7
          i = s1.indexOf('f');
          System.out.println(i); // выведет 0
          i = s1.indexOf('r');
          System.out.println(i); // выведет 2
          i = s1.lastIndexOf('f');
          System.out.println(i); // выведет  4
          i = s1.indexOf('t');
          System.out.println(i); // выведет -1
          i = s1.indexOf('r',3);
          System.out.println(i); // выведет -1
     }
}
```

Пример программы, которая выведет на экран индексы всех пробелов в строке, введенноё пользователем с клавиатуры: importjava.util.Scanner;

```
public class Main {
     public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
          String s = sc.nextLine();
          for(int i=0; i < s.length(); i++) {
               if(s.charAt(i) == ' ') {
                    System.out.println(i);
               }
          }
     }
}
```

Методы в языке Java.

Методы позволяют выполнять блок кода, из любого другого места, где это доступно. Методы определяются внутри классов. Методы могут быть статическими(можно выполнять без создания экземпляра класса), не статическими (не могут выполняться без создания экземпляра класса). Методы могут быть открытыми(public), закрытыми(private). Закрытые методы

могут вызываться только внутри того класса, в котором они определены. Открытые методы можно вызывать для объекта внутри других классов.

При определении метода можно указать модификатор доступа(public, private, protected), а также указать статический ли метод ключевым словом static. Нужно обязательно указать тип возвращаемого значения и имя метода. В скобках можно указать аргументы, которые необходимо передать методу для его вызова. В методе с непустым типом возвращаемого значения нужно обязательно указать оператор return и значение, которое он возвращает. Если метод не возвращает никакого значения, то указывается тип void.

Пример:

```
public static int sum(int a, int b) {
     return a+b;
}
```

В данном примере определен метод, возвращающий сумму двух чисел a и b. Этот метод статический, и его можно вызывать не создавая экземпляра класса, в котором он определен.

Чтобы вызвать этот метод внутри класса, в котором он создан необходимо написать имя метода и передать ему аргументы.

Пример:

```
int s = sum(10,15);
```

ЗАДАНИЯ

1. Вывести на экран сумму чисел массива с помощью циклов for, while, do while.

2. Вывести на экран аргументы командной строки в цикле for.

3. Вывести на экран первые 10 чисел гармонического ряда.

4. Сгенерировать массив целых чисел случайным образом, вывести его на экран, отсортировать его, и снова вывести на экран.

5. Создать метод, вычисляющую факториал числа с помощью цикла, проверить работу метода.

ПРАКТИЧЕСКАЯ РАБОТА №3. ИСПОЛЬЗОВАНИЕ UML ДИАГРАММ В ОБЪЕКТНО- ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

Цель работы: работа с UML-диаграммами классов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Язык моделирования Unified Modeling Language (UML) является стандартом де-факто с 1998 года для проектирования и документирования объектно-ориентированных программ. Средствами UML в виде диаграмм можно графически изобразить класс и экземпляр класса.

Графически представляем класс в виде прямоугольника, разделенного на три области – область именования класса, область инкапсуляции данных и область операций (методы).

Имя (или сущность) : определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области). На рисунке 2.1 приведен общий вид UML диаграммы класса.

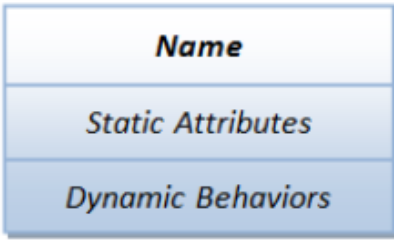

Рисунок 2.1 - Представление класса.

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 (“коробке” или прямоугольнике).

На рисунке 2.2 показано несколько примеров классов:

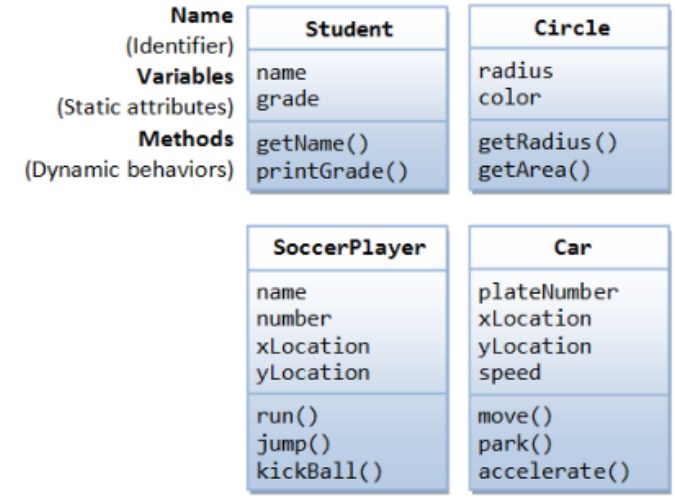

Рисунок 2.2 - Примеры экземпляров классов.

На рисунке 2.3 показаны два экземпляра класса типа Student "paul" и "peter".

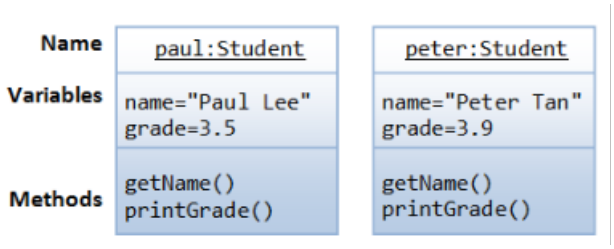

Рисунок 2.3 - Экземпляры класса Student.

Приведенные выше диаграммы классов описаны в соответствии с UML нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три области, одна содержит название, две вторых содержат переменные (поля данных класса) и методы класса, соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр (объект

класса) также представляется в виде прямоугольника, разделенного на три части, в первой части помещается надпись с именем экземпляра, например в instanceName:Classname и выделенная подчеркиванием (название_экземпляра : имя_класса).

ЗАДАНИЯ

Упражнение 1.

По диаграмме класса UML описывающей сущность Автор. Необходимо написать программу, которая состоит из двух классов Author и TestAuthor. Класс Author должен содержать реализацию методов, представленных на диаграмме класса на рисунке 2.4.

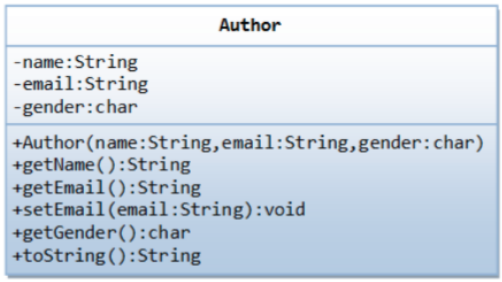

Рисунок 2.4 - Диаграмма класса Author.

Упражнение 2.

По UML диаграмме класса, представленной на рисунке 2.5. написать программу, которая состоит из двух классов. Один из них Ball должен реализовывать сущность мяч, а другой с названием TestBall тестировать работу созданного класса. Класс Ball должен содержать реализацию методов, представленных на UML. Диаграмма на рисунке описывает сущность Мяч написать программу.

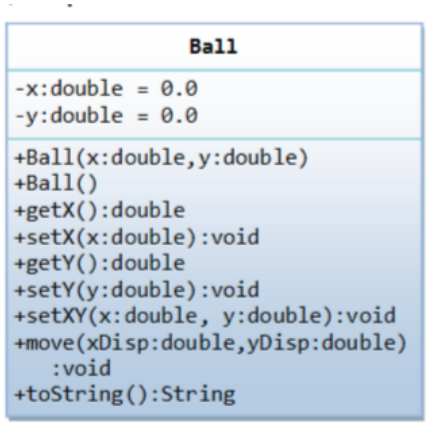

Рисунок 2.5 - Диаграмма класса Ball.

Класс Ball моделирует движущийся мяч. В состав класса входят:

- Две переменные с модификатором private (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) " .

Пример выполнения задания

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

Он содержит:

- три private переменных-поля 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 переменных name, email и gender с заданными значениями. (здесь не будет использоваться конструктор по умолчанию, так как нет значений по умолчанию: ни для имени, ни для электронной почты или пола).
- стандартные методы класса, геттеры/сеттеры: getName(), должны быть объявлены с модификатором public;
- методы getEmail(), setEmail(), and getGender(), нужно упомянуть, что класс не содержит методов сеттеров для полей данных - имени и пола, так как эти атрибуты не могут изменяться;
- метод toString(), который должен возвращать следующий текст "автор - имя (пол) на адрес электронной почты, например, "Ivan Popov(m) at ivPopov@somewhere.com", или "Anna Ivanova(ms) at anIvanova@somewhere.com ", то есть в строке должно быть записано имя[пробел](пол)[пробел]at[пробел]емайл

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 №4. ООП В JAVA. ПОНЯТИЕ КЛАССА.

Цель работы: Цель данной работы - изучить основные концепции объектно-ориентированного программирования, изучить понятие класса и научиться создавать классы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Язык Java - объектно-ориентированный язык программирования. В центре ООП находится понятие объекта. Объект — это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм подтипов — возможность единообразно обрабатывать объекты с различной реализацией при условии наличия общего интерфейса.

Класс в ООП — это в чистом виде абстрактный тип данных, создаваемый программистом. С этой точки зрения объекты являются значениями данного абстрактного типа, а определение класса задаёт внутреннюю структуру значений и набор операций, которые над этими значениями могут быть выполнены. Желательность иерархии классов (а значит, наследования) вытекает из требований к повторному использованию кода — если несколько классов имеют сходное поведение, нет смысла дублировать их описание, лучше выделить общую часть в общий родительский класс, а в описании самих этих классов оставить только различающиеся элементы.

Необходимость совместного использования объектов разных классов, способных обрабатывать однотипные сообщения, требует поддержки полиморфизма — возможности записывать разные объекты в переменные одного и того же типа. В таких условиях объект, отправляя сообщение, может не знать в точности, к какому классу относится адресат, и одни и те же сообщения, отправленные переменным одного типа, содержащим объекты разных классов, вызовут различную реакцию.

Создание классов в Java.

Для того чтобы создать класс в языке Java необходимо создать файл с расширением java. Имя файла должно быть таким же, как и имя создаваемого класса. В созданном файле должен описываться класс. Синтаксис написания класса:

```
<модификатор досупа> class <имя класса> {
    <тело класса>
}
```

В качестве модификатора доступа можно указать ключевое слово public или private. Если указано слово public, то класс будет доступен из других пакетов. Если указано слово private, то класс будет доступен только внутри того пакета, в котором он находится.

В теле класса можно описать методы, переменные, константы, конструкторы класса.

Конструктор - это специальный метод, который вызывается при создании нового объекта. Не всегда удобно инициализировать все переменные класса при создании его экземпляра. Иногда проще, чтобы какие-то значения были бы созданы по умолчанию при создании объекта. По сути конструктор нужен для автоматической инициализации переменных.

Конструктор инициализирует объект непосредственно во время создания. Имя конструктора совпадает с именем класса, включая регистр, а по синтаксису конструктор похож на метод без возвращаемого значения.

В отличие от метода, конструктор никогда ничего не возвращает.

Пример класса, описывающего прямоугольник с высотой height и шириной width.

```
public class Rectangle {
     //Свойства, поля класса
     private float width;
     private float height;
     //Конструктор класса
     public Rectangle(float w, float h) {
          width=w;
          height=h;
     }
     //Метод, возвращающий ширину прямоугольника
     public float getWidth() {
          return width;
     }
     //Метод, возвращающий высоту прямоугольника
     public float getHeight() {
          return height;
     }
     //Метод, устанавливающий ширину прямоугольника
     public void setWidth(float w) {
          width=w;
     }
     //Метод, устанавливающий высоту прямоугольника
     public void setHeight(float h) {
          height=h;
     }
}
```

В данном примере был создан класс с одним конструктором, и методами, меняющими поля класса setWidth(), setHeight() ("сеттеры"), и возвращающие их значение getWidth(), getHeight() ("геттеры").

Если конструкторы в классе отсутствуют, то Java автоматически создает конструктор по умолчанию, который не имеет аргументов.

Создание экземпляра класса.

Для того чтобы создать экземпляр класса необходимо объявить переменную, тип которой соответствует имени класса или суперкласса. После чего нужно присвоить этой переменной значение, вызвав конструктор создаваемого класса с помощью оператора new. Например, можно создать экземпляр класса Rectangle следующим образом:

```
Rectangle rect = new Rectangle(20, 10);
```

После этого можно вызывать методы этого класса для объекта rect, указав имя метода через точку:

```
rect.setWidth(20);
System.out.println("Новая ширина: "+rect.getWidth());
```

ЗАДАНИЯ

1. Создать класс, описывающий модель окружности (Circle). В классе должны быть описаны нужные свойства окружности и методы для получения, изменения этих свойств. Протестировать работу класса в классе CircleTest, содержащим метод статический main(String[] args).

2. Создать класс, описывающий тело человека(Human). Для описания каждой части тела создать отдельные классы(Head, Leg, Hand). Описать необходимые свойства и методы для каждого класса. Протестировать работу класса Human.

3. Создать класс, описывающий книгу (Book). В классе должны быть описаны нужные свойства книги(автор, название, год написания и т. д.)и методы для получения, изменения этих свойств. Протестировать работу класса в классе BookTest, содержащим метод статический main(String[] args).

ПРАКТИЧЕСКАЯ РАБОТА №5. НАСЛЕДОВАНИЕ. АБСТРАКТНЫЕ СУПЕРКЛАССЫ И ИХ ПОДКЛАССЫ В JAVA.

Цель работы: освоить на практике работу с абстрактными классами и наследованием на Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

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

Абстрактный метод внутри абстрактного класса не имеет тела, только прототип. 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.

```
public abstract class Swim {
     // абстрактный метод плавать()
     abstract void neigh();

     // абстрактный класс может содержать и обычный метод
     void run() {
          System.out.println("Куда идешь?");
     }
}
class Swimmer extends Swim { ....
}
```

ЗАДАНИЯ

Упражнение 1.

Создайте абстрактный родительский суперкласс Shape и его дочерние классы (подклассы).

Упражнение 2.

Перепишите суперкласс Shape и его подклассы, так как это представлено на диаграмме Circle, Rectangle and Square рисунка 3.1.

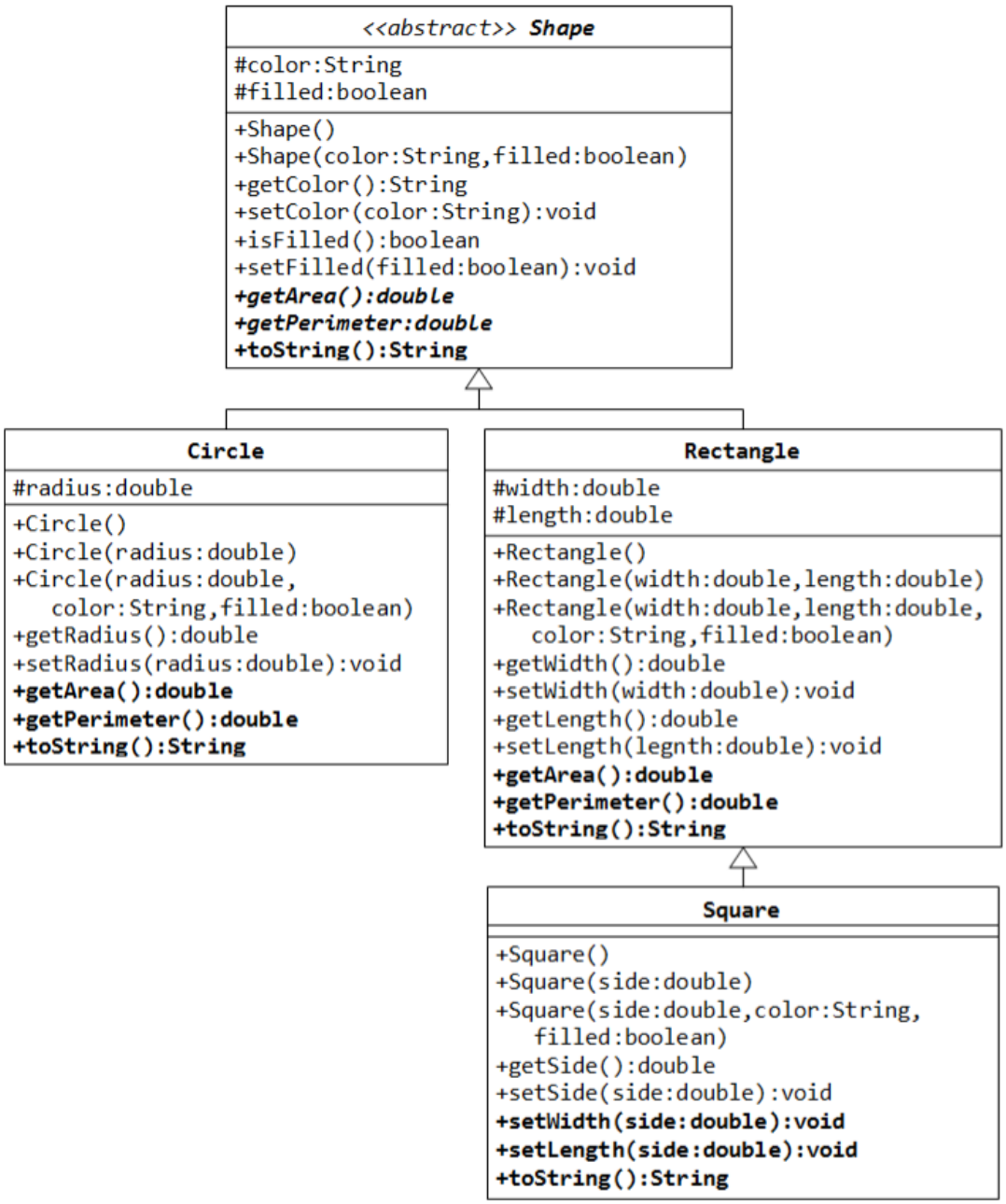

Рисунок 3.1 – Диаграмма суперкласса Shape.

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

• Два поля или переменные класса, объявлены с модификатором ***protected*** color (тип String) и filled (тип boolean). Такие защищенные переменные могут быть доступны в подклассах и классах в одном пакете. Они обозначаются со знаком “#” на 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().

Упражнение 3.

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());
```

Продолжение листинга

```
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());
```

Упражнение 4.

Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable.

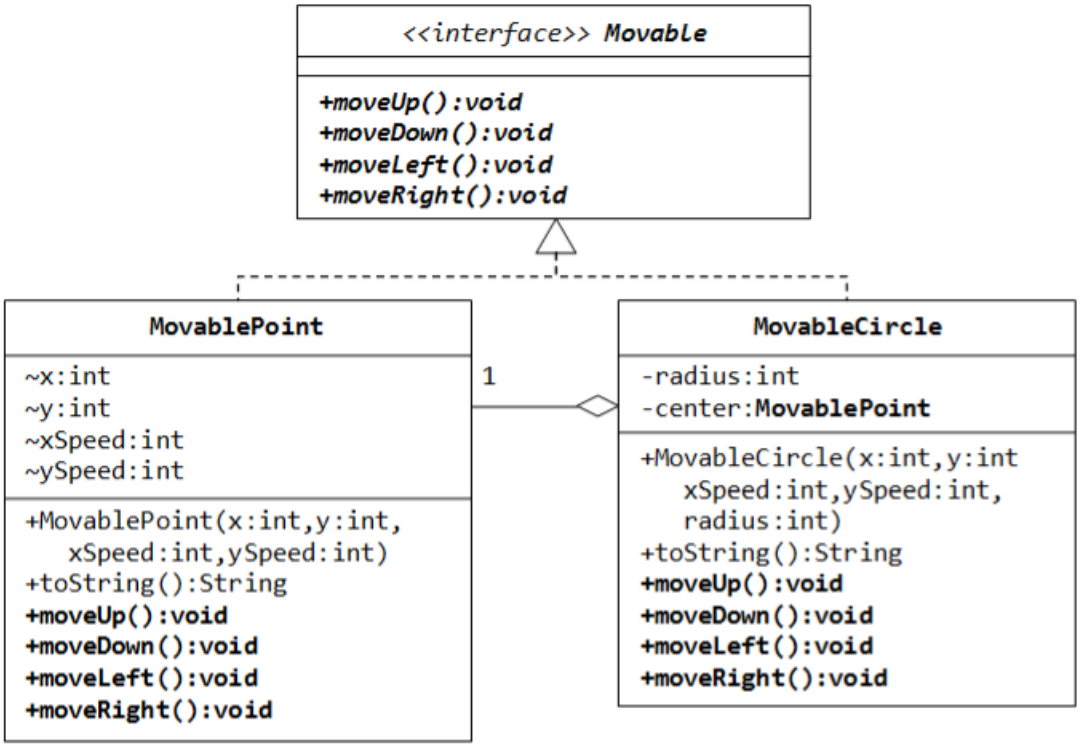

Рисунок 3.2. – Диаграмма реализации итерфейса Movable.

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

Упражнение 5.

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).

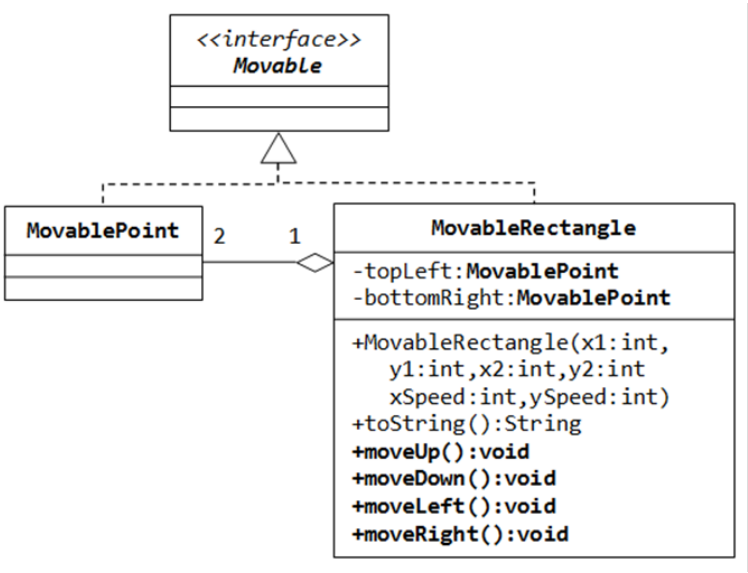

Рисунок 3.3 – Диаграмма класса MovableRectangle.

ПРИМЕР РЕШЕНИЯ ЗАДАНИЯ 1.

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;
           }
     }
```

ПРАКТИЧЕСКАЯ РАБОТА№6. НАСЛЕДОВАНИЕ В JAVA

Цель работы: цель данной работы - изучить понятие наследования, и научиться реализовывать наследование в Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Одним из ключевых аспектов объектно-ориентированного программирования является наследование. С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого. Например, имеется следующий класс Person, описывающий отдельного человека:

```
public class Person {
     private String name;
     private String surname;

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

     public Person(String name, String surname){
          this.name=name;
          this.surname=surname;
     }
     public void displayInfo(){
          System.out.println("Имя: " + name + " Фамилия: " +
surname);
     }
}
```

И, возможно, впоследствии мы решили расширить имеющуюся систему и классов, добавив в нее класс, описывающий сотрудника предприятия - класс Employee. Так как этот класс реализует тот же функционал, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником) от класса Person, который, в свою очередь, называется базовым классом или родителем:

```
Class Employee extends Person{
}
```

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово extends, после которого идет имя базового класса. Для класса Employee базовым является Person, и поэтому

класс Employee наследует все те же поля и методы, которые есть в классе Person.

В классе Employee могут быть определены свои методы и поля, а также конструктор. Способность к изменению функциональности, унаследованной от базового класса, назвывается полиморфизмом и является одинм из ключевых аспектов объектно-ориентированного программирования наряду с наследованием и инкапсуляцией.

Например, переопределим метод displayInfo() класса Person в классе Employee:

```
class Employee extends Person{
     private String company;
     public Employee(String name, String surname, String company)
{
          super(name, surname);
          this.company=company;
     }
     public void displayInfo(){
          super.displayInfo();
          System.out.println("Компания: " + company);
     }
}
```

Класс Employee определяет дополнительное поле для хранения компании, в которой работает сотрудник. Кроме того, оно также устанавливается в конструкторе.

Так как поля name и surname в базовом классе Person объявлены с модификатором private, то мы не можем к ним напрямую обратиться из класса Employee. Однако в данном случае нам это не нужно. Чтобы их установить, мы обращаемся к конструктору базового класса с помощью ключевого слова super, в скобках после которого идет перечисление передаваемых аргументов.

С помощью ключевого слова super мы можем обратиться к любому члену базового класса - методу или полю, если они не определены с модификатором private.

Также в классе Employee переопределяется метод displayInfo() базового класса. В нем с помощью ключевого super также идет обращение к методу

displayInfo(), но уже базового класса, и затем выводится дополнительная информация, относящаяся только к Employee.

Используя обращение к методом базового класса, можно было бы переопределить метод displayInfo() следующим образом:

```
public void displayInfo(){
     System.out.println("Имя: " + super.getName() + " Фамилия: "
+ super.getSurname() + " Компания: " + company);
}
```

При этом нам необязательно переопределять все методы базового класса. Например, в данном случае мы не переопределяем методы getName() и getSurname(). Поэтому для этих методов класс- наследник будет использовать реализацию из базового класса. И в основной программе мы можем эти методы использовать:

```
public static void main(String[] args) {
     Employee empl = new Employee("Tom", "Simpson", "Oracle");
     empl.displayInfo();
     String firstName = empl.getName();
     System.out.println(firstName);
}
```

Хотя наследование очень интересный и эффективный механизм, но в некоторых ситуациях его применение может быть нежелательным. И в этом случае можно запретить наследование с помощью ключевого слова final. Например:

```
public final class Person {
}
```

Если бы класс Person был бы определен таким образом, то следующий код был бы ошибочным и не сработал, так как мы тем самым запретили наследование:

```
class Employee extends Person{
}
```

Кроме запрета наследования можно также запретить переопределение отдельных методов. Например, в примере выше переопределен метод displayInfo(), запретим его переопределение:

```
public class Person {
     //.....................
     public final void displayInfo(){
          System.out.println("Имя: " + name + " Фамилия: " +
surname);
     }
}
```

В этом случае в классе Employee надо будет создать метод с другим именем для вывода информации об объекте.

Абстрактные классы

Кроме обычных классов в Java есть абстрактные классы. Абстрактный класс похож на обычный класс. В абстрактном классе также можно определить поля и методы, в то же время нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов- наследников. А производные классы уже реализуют этот функционал.

При определении абстрактных классов используется ключевое слово abstract:

```
public abstract class Human{
     private int height;
     private double weight;

     public int getHeight() { return height; }
     public double getWeight() { return weight; }
}
```

Кроме обычных методов абстрактный класс может содержать абстрактные методы. Такие методы определяются с помощью ключевого слова abstract и не имеют никакого функционала:

```
public abstract void displayInfo();
```

Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе. Также следует учитывать, что если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный.

Зачем нужны абстрактные классы? Допустим, мы делаем программу для обсулживания банковских операций и определяем в ней три класса: Person, который описывает человека, Employee, который описывает банковского

служащего, и класс Client, который представляет клиента банка. Очевидно, что классы Employee и Client будут производными от класса Person, так как оба класса имеют некоторые общие поля и методы. И так как все объекты будут представлять либо сотрудника, либо клиента банка, то напрямую мы от класса Person создавать объекты не будем. Поэтому имеет смысл сделать его абстрактным.

```
public abstract class Person {
     private String name;
     private String surname;
     public String getName() { return name; }
     public String getSurname() { return surname; }
     public Person(String name, String surname){
          this.name=name;
          this.surname=surname;
     }
     public abstract void displayInfo();
}
class Employee extends Person{
     private String bank;
     public Employee(String name, String surname, String company)
{
          super(name, surname);
          this.bank=company;
     }
     public void displayInfo(){
          System.out.println("Имя: " + super.getName() + "
Фамилия: " + super.getSurname() + " Работает в банке: " + bank);
     }
}
class Client extends Person{
     private String bank;
     public Client(String name, String surname, String company) {
          super(name, surname);
          this.bank=company;
     }
     public void displayInfo(){
          System.out.println("Имя: " + super.getName() + "
Фамилия: " + super.getSurname() + " Клиент банка: " + bank);
     }
}
```

ЗАДАНИЯ

1. Создать абстрактный класс, описывающий посуду(Dish). С помощью наследования реализовать различные виды посуды, имеющие свои свойства и методы. Протестировать работу классов.

2. Создать абстрактный класс, описывающий собак(Dog). С помощью наследования реализовать различные породы собак. Протестировать работу классов.

3. Создать абстрактный класс, описывающий мебель. С помощью наследования реализовать различные виды мебели. Также создать класс FurnitureShop, моделирующий магазин мебели. Протестировать работу классов.

ПРАКТИЧЕСКАЯ РАБОТА №7.СОЗДАНИЕ GUI. СОБЫТИЙНОЕ ПРОГРАММИРОВАНИЕ В JAVA.

Цель работы: введение в событийное программирование на языке Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Данная практическая работа посвящена закреплению практических навыков по созданию приложений на Java с использованием следующих элементов GUI:

- Текстовые поля и области ввода текста;
- Менеджеры компоновки компонентов;
- Слушатель мыши;
- Создание меню.

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 создания объекта класса JTextField:

JTextField jta = new JTextField (10);

В параметрах 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();
      }
}
```

Рисунок 4.1

Важное замечание

Ответственность за выполнение проверки на наличие ошибок в коде лежит полностью на программисте, например, чтобы проверить, произойдет ли ошибка, когда 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 TextArea похож на TextField, 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.
```

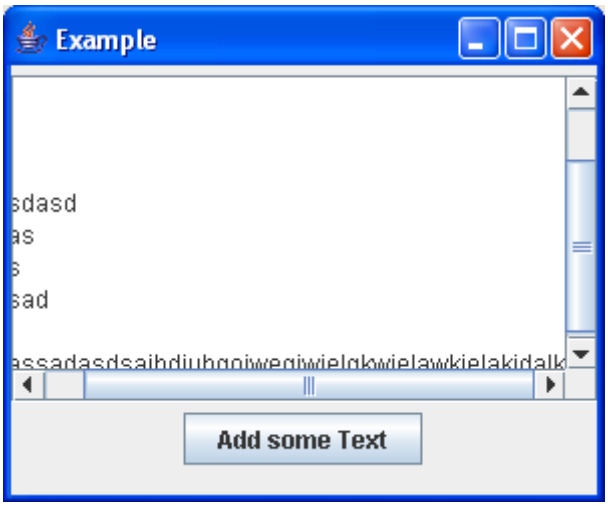

Рисунок 4.2

Менеджеры компоновки компонентов или Layout Менеджеры.

Менеджер BorderLayout:

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

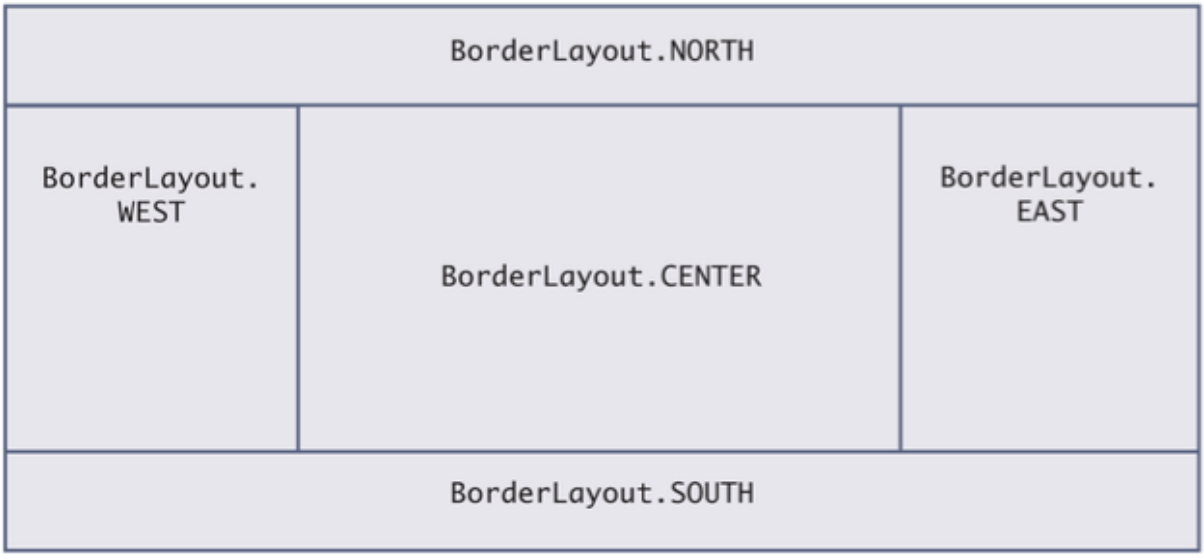

Рисунок 4.3

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.

Менеджер GridLayout.

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

Таблица 4.1.

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);
    }
}
```

Код, представленный выше, будет иметь вид, который указан на рисунке 4.4.

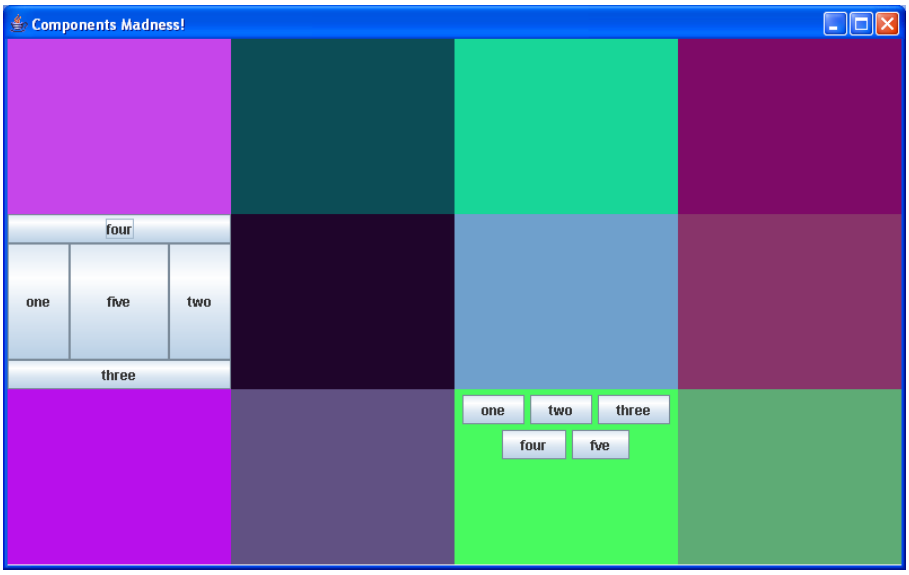

Рисунок 4.4 – Пример выполнения кода

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.

Менеджер 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 результат, отраженный на рисунке 4.5.

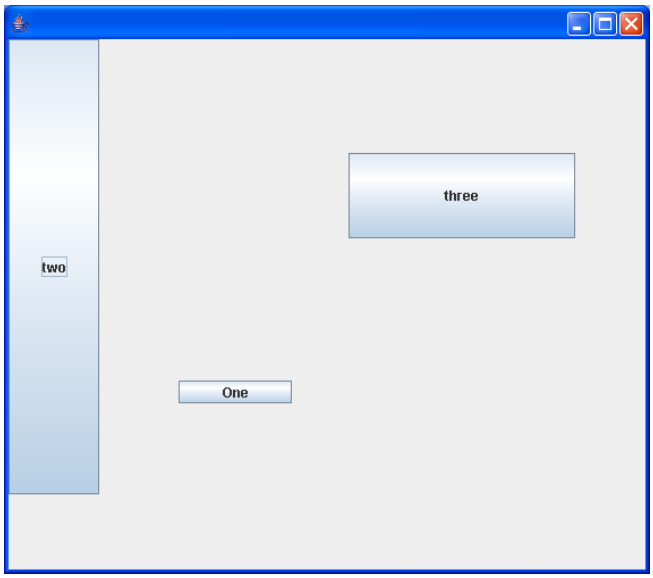

Рисунок 4.5 – Пример работы программы при setLayout равным нулю

Пример 5.

```
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);
    }
}
```

Слушатель событий мыши MouseListener.

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

Таблица 4.2 - Методы класса MouseListener.

Методы класса		
Возвращаемое значение	Прототип метода	Описание
void	mouseClicked(MouseEvent e)	Вызывается, когда кнопка мыши была нажата (нажата и отпущена) на области компонента.
void	mouseEntered(MouseEvent e)	Вызывается, когда мышь входит в область компонент.
void	mouseExites(MouseEvent e)	Вызывается, когда мышь выходит из области компонента.
void	mousePressed(MouseEvent e)	Вызывается при нажатии кнопки мыши на область компонента.
void	mouseReleased(MouseEvent e)	Вызывается, когда над областью компонента отпущена кнопка мыши.

Слушатель MouseListener можно добавить к компоненту следующим образом:

Component.addMouseListener(listener);

Здесь слушатель является экземпляром класса, который реализует интерфейс MouseListener. Обратите внимание, что он должен обеспечивать

выполнение всех методов, перечисленных в таблице 2 в данной практической работе.

Пример 6.

```
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);
    }
}
```

Рисунок 4.6 – Пример выполнения программы 6

Создание меню

Добавление меню в программе Java происходит несложно. Java определяет три компонента для обработки:

- JMenuBar: который представляет собой компонент, который содержит меню.
- JMenu: который представляет меню элементов для выбора.
- JMenuItem: представляет собой элемент, который можно кликнуть из меню.
- 

Рисунок 4.7 – Меню в языке Java

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

Упражнение 1.

Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид. Создайте JFrame приложение у которого 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”;

Всякий раз, когда пользователь нажимает на кнопку AC Milan, результат будет увеличиваться для Милана, сначала 1 X 0, затем 2 X 0 и так далее. Last Scorer означает последнюю забившую команду. В этом случае: AC Milan. Если пользователь нажимает кнопку для команды Мадрид, то счет приписывается ей. Победителем становится команда, которая имеет больше кликов кнопку на соответствующую, чем другая.

ПРАКТИЧЕСКАЯ РАБОТА №8. ИНТЕРФЕЙСЫ В JAVA.

Цель работы: Цель данной лабораторной работы - изучить понятие интерфейса, научиться создавать интерфейсы в Java и применять их в программах.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Механизм наследования очень удобен, но он имеет свои ограничения. В частности мы можем наследовать только от одного класса, в отличие, например, от языка C++, где имеется множественное наследование.

В языке Java подобную проблему позволяют решить интерфейсы. Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.

Чтобы определить интерфейс, используется ключевое слово interface.

Определим следующий интерфейс:

```
public interface Printable{
void print();
}
```

Интерфейс может определять различные методы, которые, так же как и абстрактные методы абстрактных классов не имеют реализации. В данном случае объявлен только один метод.

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

И также при объявлении интерфейса надо учитывать, что только один интерфейс в файле может иметь тип доступа public. А его название должно совпадать с именем файла. Остальные интерфейсы (если такие имеются в файле java) не должны иметь модификаторов доступа.

Интерфейс может определять различные методы, которые, так же как и абстрактные методы абстрактных классов не имеют реализации. В данном случае объявлен только один метод.

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

И также при объявлении интерфейса надо учитывать, что только один интерфейс в файле может иметь тип доступа public. А его название должно совпадать с именем файла. Остальные интерфейсы (если такие имеются в файле java) не должны иметь модификаторов доступа.

Чтобы класс применил интерфейс, надо использовать ключевое слово implements:

```
class Book implements Printable{
	String name;
	String author;
	int year;
	Book(String name, String author, int year){
		this.name = name;
		this.author = author;
		this.year = year;
	}
	public void print() {
		System.out.printf("Книга '%s' (автор %s) была издана в
%d году \n", name, author, year);
	}
}
```

При этом надо учитывать, что если класс применяет интерфейс, то он должен реализовать все методы интерфейса, как в случае выше реализован метод print.

Потом в главном классе мы можем использовать данный класс и его метод print:

```
Book b1 = new Book("Война и мир", "Л. Н. Толстой", 1863);
b1.print();
```

В тоже время мы не можем напрямую создавать объекты интерфейсов, поэтому следующий код не будет работать:

```
Printable pr = new Printable();
pr.print();
```

Одним из преимуществ использоваия интерфейсов является то, что они позволяют добавить в приложение гибкости. Например, в дополнение к классу

Book определим еще один класс, который будет реализовывать интерфейс Printable:

```
public class Journal implements Printable {
     private String name;
     String getName(){
          return name;
     }
     Journal(String name){
          this.name = name;
     }
     public void print() {
          System.out.printf("Журнал '%s'\n", name);
     }
}
```

Класс Book и класс Journal связаны тем, что они реализуют интерфейс Printable. Поэтому мы динамически в программе можем создавать объекты Printable как экземпляры обоих классов:

```
Printable printable = new Book("Война и мир", "Л. Н. Толстой",
1863);
printable.print();
printable = new Journal("Хакер");
printable.print();
```

И также как и в случае с классами, интерфейсы могут использоваться в качестве типа параметров метода или в качестве возвращаемого типа:

```
public static void main(String[] args) {
     Printable printable = createPrintable("Компьютерра",false);
     printable.print();
     read(new Book("Отцы и дети", "И. Тургенев", 1862));
     read(new Journal("Хакер"));
}
static void read(Printable p){
     p.print();
}
static Printable createPrintable(String name, boolean option){
     if(option){
          return new Book(name, "неизвестен", 2015);
     }
     else{
          return new Journal(name);
     }
}
```

Метод read() в качестве параметра принимает объект интерфейса Printable, поэтому в этот метод мы можем передать как объект Book, так и объект Journal.

Метод createPrintable() возвращает объект Printable, поэтому также мы вожем возвратить как объект Book, так и Journal.

ЗАДАНИЯ

1. Создать интерфейс Nameable, с методом getName(), возвращающим имя объекта, реализующего интерфейс. Проверить работу для различных объектов (например, можно создать классы, описывающие разные сущности, которые могут иметь имя: планеты, машины, животные и т. д.).

2. Реализовать интерфейс Priceable, имеющий метод getPrice(), возвращающий некоторую цену для объекта. Проверить работу для различных классов, сущности которых могут иметь цену.

ПРАКТИЧЕСКАЯ РАБОТА №9. ПРОГРАММИРОВАНИЕ РЕКУРСИИ В JAVA

Цель работы: разработка и программирование рекурсивных алгоритмов на языке Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В контексте языка программирования рекурсия — это некий активный метод (или подпрограмма) вызываемый сам по себе непосредственно, или вызываемой другим методом (или подпрограммой) косвенно. 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 не произойдет остановка рекурсии (а точнее возврат к последнему вызову функции). Всё решение сводится к поиску решения для 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 базового случая или условия;
- условие продолжения или шага рекурсии — способ сведения сложной задачи к более простым подзадачам.

  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 интернет при объяснении понятия рекурсии часто даются примеры решения задач нахождения чисел Фибоначчи и Ханойская башня

Представлены задачи с различным уровнем сложности. Попробуйте их решить, самостоятельно используя подход, описанный выше.

При решении попробуйте думать рекурсивно и ответить на вопросы:

- какой базовый случай или условие задается в задаче?
- какой шаг рекурсии или рекурсивное условие?

ЗАДАНИЯ

Упражнения по этой теме выполняются следующим образом: каждый учащийся выполняет от 3 до 5 задач, начиная с номера варианта задания, который соответствует номеру учащегося в журнале группы.

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 не убывания 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.

Функция 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 ЗАДАЧИ.

Задание: найти точную степень двойки. Дано натуральное число N. Выведите слово YES, если число N является точной степенью двойки, или слово NO в противном случае.

Решение:

```
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 №10. ТЕХНИКИ СОРТИРОВКИ В JAVA

Цель работы: освоение на практике методов сортировки с использованием приемов программирования на объектно-ориентированном языке Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Сортировка — это процесс упорядочивания списка элементов (организация в определенном порядке) исходного списка элементов, который возможно организован в виде контейнера или храниться в виде коллекции.

Процесс сортировки основан на упорядочивании конкретных значений, например:

- сортировка списка результатов экзаменов баллов в порядке возрастания результата;
- сортировка списка людей в алфавитном порядке по фамилии. Есть много алгоритмов для сортировки списка элементов, которые различаются по эффективности.

Алгоритм сортировки вставками.

Работа метода сортировки состоит из следующих шагов:

- выбрать любой элемент из списка элементов и вставить его в надлежащее место в отсортированный подсписок;
- повторять предыдущий шаг, до тех пор, пока все элементы не будут вставлены.

Более детально:

- рассматриваем первый элемент списка как отсортированный подсписок (то есть первый элемент списка);

- вставим второй элемент в отсортированный подсписок, сдвигая первый элемент по мере необходимости, чтобы освободить место для вставки нового элемента;
- вставим третий элемент в отсортированный подсписок (из двух элементов), сдвигая элементы по мере необходимости;
- повторяем до тех пор, пока все значения не будут вставлены на свои соответствующие позиции.

Алгоритм быстрой сортировки (Quick Sort).

Состоит из последовательного выполнения двух шагов:

- массив A[1..n] разбивается на два непустых подмассивов по отношению к "опорному элементу”;
- два подмассива сортируются рекурсивно посредством Quick Sort.

Алгоритм сортировка слиянием (Merge Sort).

Состоит из последовательного выполнения трех шагов:

- разделить массив A[1..n] на 2 равные части;
- провести сортировку слиянием двух подмассивов (рекурсивно);
- объединить (соединить) два отсортированных подмассива.

Использование полиморфизма в сортировке.

Техника программирования сортировок в Java отличается от написания алгоритмов на процедурных языках программирования. При написании кода большим преимуществом является использование основного принципа ООП – полиморфизма. Напомним, что класс, который реализует интерфейс Comparable определяет метод compareTo(), чтобы определить относительный порядок своих объектов.

Таким образом мы можем использовать полиморфизм, чтобы разработать обобщенную сортировку для любого набора Comparable объектов.

При разработке класса, реализующего метод сортировки, нужно помнить, что метод принимает в качестве параметра массив объектов типа Comparable или фактически полиморфных ссылок.

Таким образом, один метод может быть использован для сортировки любых объектов, например: People (людей), Books (книг), или любой каких-либо других объектов.

Методу сортировки все-равно, что именно он будет сортировать, ему только необходимо иметь возможность вызвать метод compareTo().

Это обеспечивается использованием в качестве типа формального параметра интерфейса Comparable или интерфейсной ссылки.

Кроме того, таким образом каждый класс “для себя” решает, что означает для одного объекта, быть меньше, чем другой.

ЗАДАНИЯ

Упражнение 1.

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

Упражнение 2.

Напишите класс SortingStudentsByGPA который реализует интерфейс Comparator таким образом, чтобы сортировать список студентов по их итоговым баллам в порядке убывания с использованием алгоритма быстрой сортировки.

Упражнение 3.

Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках с использованием алгоритма сортировки слиянием.

Пример решения задачи

```
public class Sorting {
public static void selectionSort (Comparable[] list) {
     int min;
     Comparable temp;
     for (int index = 0; index < list.length-1; index++){
          min = index;
          for (int scan = index+1; scan < list.length; scan++){
               if (list[scan].compareTo(list[min]) < 0){
                    min = scan;
               }
         temp = list[min];
         list[min] = list[index];
         list[index] = temp;
    }
}
```

ПРАКТИЧЕСКАЯ РАБОТА №11. ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ КОНТЕЙНЕРНЫХ КЛАССОВ ПРИ ПРОГРАММИРОВАНИИ НА JAVA

Цель работы: изучение на практике приемов работы со стандартными контейнерными классами Java Collection Framework.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Java Collections Framework — это набор связанных классов и интерфейсов, реализующих широко используемые структуры данных — коллекции. На вершине иерархии в Java Collection Framework располагаются 2 интерфейса: Collection и Map. Эти интерфейсы разделяют все коллекции, входящие в фреймворк на две части по типу хранения данных: простые последовательные наборы элементов и наборы пар «ключ — значение» (словари).

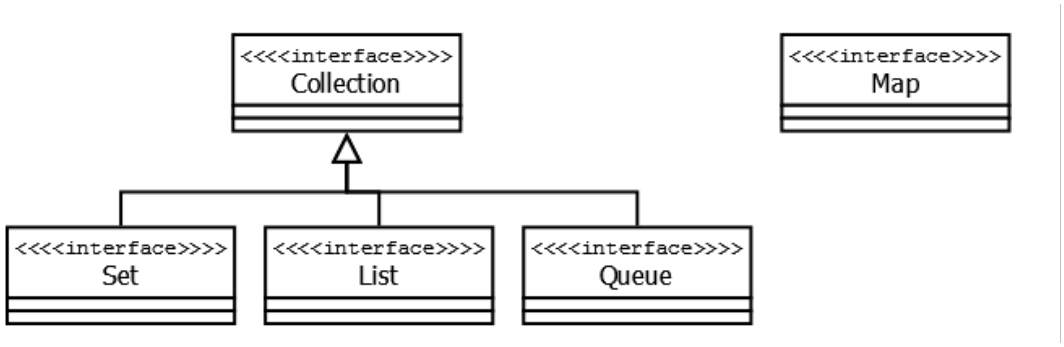

Рисунок 7.1 – Пример Java Collections

Vector — реализация динамического массива объектов. Позволяет хранить любые данные, включая null в качестве элемента. Vector появился в JDK версии Java 1.0, но, как и Hashtable, эту коллекцию не рекомендуется использовать, если не требуется достижения потокобезопасности. Потому как в Vector, в отличии от других реализаций List, все операции с данными являются синхронизированными. В качестве альтернативы часто применяется аналог — ArrayList.

Stack — данная коллекция является расширением коллекции Vector. Была добавлена в Java 1.0 как реализация стека LIFO (last-in-first-out). Является частично синхронизированной коллекцией (кроме метода добавления push()). После добавления в Java 1.6 интерфейса Deque, рекомендуется использовать именно реализации этого интерфейса, например ArrayDeque.

ArrayList — как и Vector является реализацией динамического массива объектов. Позволяет хранить любые данные, включая null в качестве элемента. Как можно догадаться из названия, его реализация основана на обычном массиве. Данную реализацию следует применять, если в процессе работы с коллекцией предполагается частое обращение к элементам по индексу. Из-за особенностей реализации обращение к элементам по индексу, которое выполняется за константное время O(1). Использование данной коллекции рекомендуется избегать, если требуется частое удаление/добавление элементов в середине коллекции.

LinkedList — вид реализации List. Позволяет хранить любые данные, включая null. Данная коллекция реализована на основе двунаправленного связного списка (каждый элемент списка имеет ссылки на предыдущий и следующий). Добавление и удаление элемента из середины, доступ по индексу, значению происходит за линейное время O(n), а из начала и конца за константное время O(1). Ввиду реализации, данную коллекцию можно использовать как абстрактный тип данных — стек или очередь. Для этого в ней реализованы соответствующие методы.

Интерфейс Set — Представляет собой неупорядоченную коллекцию, которая не может содержать одинаковые элементы и является программной моделью математического понятия «множество».

Интерфейс Queue — Этот интерфейс описывает коллекции с предопределённым способом вставки и извлечения элементов, а именно — очереди FIFO (first-in-first-out). Помимо методов, определённых в интерфейсе Collection, определяет дополнительные методы для извлечения и добавления элементов в очередь.

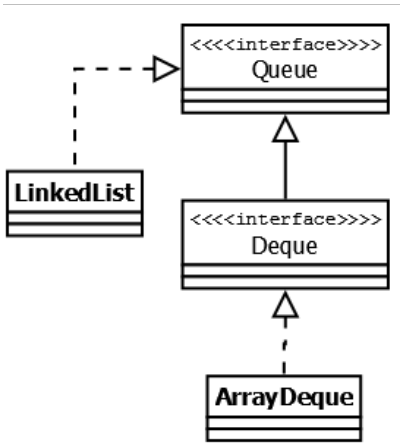

Рисунок 7.2 – Пример LinkedList

PriorityQueue — является единственной прямой реализацией интерфейса Queue (была добавлена, как и интерфейс Queue, в Java 1.5), не считая класса LinkedList, который так же реализует этот интерфейс, но был реализован намного раньше. Особенностью данной очереди является возможность управления порядком элементов. По умолчанию, элементы сортируются с использованием «natural ordering», но это поведение может быть переопределено при помощи объекта Comparator, который задаётся при создании очереди. Данная коллекция не поддерживает null в качестве элементов.

ArrayDeque — реализация интерфейса Deque, который расширяет интерфейс Queue методами, позволяющими реализовать конструкцию вида LIFO (last-in-first-out). Интерфейс Deque и реализация ArrayDeque были добавлены в Java 1.6. Эта коллекция представляет собой реализацию с использованием массивов, подобно ArrayList, но не позволяет обращаться к элементам по индексу и хранение null.

Как заявлено в документации, эта коллекция работает быстрее чем Stack, если используется как LIFO коллекция, а также быстрее чем LinkedList, если используется как FIFO коллекция.

ЗАДАНИЯ

Напишите программу в виде консольного приложения, которая моделирует карточную игру «пьяница» и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую; карта «0» побеждает карту «9».

Карточная игра “ В пьяницу”. В этой игре карточная колода раздается поровну двум игрокам. Далее они открывают по одной верхней карте, и тот, чья карта старше, забирает себе обе открытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт, - проигрывает.

Для простоты будем считать, что все карты различны по номиналу и что самая младшая карта побеждает самую старшую карту (“шестерка берет туз”).

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).

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.

13579 24680

Пример вывода.

second 5

Упражнение 1.

Используйте для организации хранения структуру данных Stack.

Упражнение 2.

Используйте для организации хранения структуру данных Queue.

Упражнение 3.

Используйте для организации хранения структуру данных Dequeue.

Упражнение 3.

Используйте для организации хранения структуру данных DoubleList.

Упражнение 4*.

Реализуйте более усложненный вариант решения задачи из упражнения Реализация должна иметь интерактивный интерфейс для взаимодействия с пользователем.

Пример решения задания

Реализуйте аналогичные очереди процедуры, реализующие стек, и на их основе напишите программу по заданию. 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.

Данная реализация выполнена с использованием ArrayDeque.

```
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));
     }
}
```