Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Севастопольский государственный университет»

ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ JAVA

Методические указания для студентов, обучающихся по направлениям
09.03.02 – «Информационные системы и технологии»,
09.03.03 – «Прикладная информатика»
по учебным планам подготовки бакалавров
дневной и заочной форм обучения

Севастополь
2015

УДК 004.42 (075.8)

Основы объектно-ориентированного программирования на языке Java: методические указания к лабораторной работе №2 по дисциплине “Платформа Java” для студентов / Сост. **С.А. Кузнецов, А.Л. Овчинников —** Севастополь: Изд-во СевГУ, 2015. — 13 с.

Цель указаний: оказание помощи студентам при выполнении лабораторной работы №2 по дисциплине “Платформа Java”.

Методические указания составлены в соответствии с требованиями программы дисциплины «Платформа Java» для студентов дневной и заочной формы обучения и утверждены на заседании кафедры «Информационные системы» протоколом № 1 от 31 августа 2015 года.

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: Кожаев Е.А., канд. техн. наук, доцент кафедры кибернетики и вычислительной техники.

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

В ходе выполнения данной лабораторной работы необходимо ознакомиться с особенностями объектно-ориентированного программирования (ООП) на языке Java, приобрести практические навыки программирования на языке Java с использованием основных принципов ООП.

2. ПОСТАНОВКА ЗАДАЧИ

2.1. Описать абстрактный класс CBuffer, содержащий следующие поля:

- идентификатор буфера (**int** bufID) – уникальный идентификатор буфера;
- размер буфера (**int** bufSize) – максимальный размер буфера;
- количество созданных буферов **(int** BufCount).

Доступ к полям класса CBuffer должны иметь только методы этого класса и методы его потомков. Для организации доступа к этим полям из других классов необходимо реализовать общедоступные методы:

- **int** GetBufCount();
- **int** GetBufID().

Реализовать конструктор класса CBuffer(**int** count), выполняющий инициализацию идентификатора буфера(в качестве идентификатора использовать номер по порядку создаваемого буфера), размера буфера (значением count, передаваемым конструктору), увеличение количества созданных буферов.

В классе CBuffer описать абстрактный метод Generate().

2.2. В соответствии с вариантом задания реализовать дочерний класс для создания буфера, хранящего значения заданного типа T(см. таблицу 4.1). Для хранения значений реализовать поле – массив значений типа T. В конструкторе класса использовать вызов конструктора родительского класса CBuffer, и кроме того создать массив значений типа T(с использованием оператора **new)** и проинициализировать его с импользованием метода Generate().

Реализовать метод Generate(), заполняющий массив случайными числами.

Для генерации случайных чисел необходимо, используя оператор import, подключить пакет java.util.Random. Для использования генератора случайный чисел сначала необходимо создать экземпляр класса Random:

Random random = new Random();

Генерация случайных чисел выполняется методами экземпляра класса Random. Например, для генерации случайного целого числа:

random.nextInt();

2.3. Описать интерфейсы:

1) IBufferPrintable – описывающий методы вывода на экран:

- **public void** PrintInfo() – выводит на экран идентификатор, тип и размер буфера.
- **public void** Print() – выводит на экран содержимое буфера.
- **public void** PrintFirstN(**int** n) – выводит на экран первые n элементов буфера.

- **public void** PrintLastN(**int** n) – выводит на экран последние n элементов буфера.

2) IBufferSortable – описывает метод для сортировки массива:

- **public void** Sort();

3) IBufferComputable – описывает методы для вычисления статистики значений буфера.

- **public void** Max() – вычисляет максимальный элемент буфера;
- **public void** Min() – вычисляет минимальный элемент буфера;
- **public void** Sum() – вычисляет сумму элементов буфера;

4) IBufferStorable – описывает методы для выгрузки буфера в текстовый файл.

- **public void** SaveOneLine(String filename) – сохраняет буфер в файл в одну строку;
- **public void** SaveSeparateLines (String filename) – сохраняет буфер в файл по одному элементу в строке;

2.4. Создать произвольный класс, унаследованный от класса, разработанного при выполнении п. 2.2, и реализующий методы интерфейсов из п. 2.3, необходимых для выполнения задания в соответствии с вариантом (см. таблицу 4.1).

2.5. Реализовать класс Lab2Java, в методе main которого в соответствии с вариантом задания (см. таблицу 4.1) реализовать работу с объектами класса из п. 2.4 с использованием их методов:

- Создать N буферов заданного типа T и размера L;
- Вывести на экран информацию о буферах;
- Вывести на экран первые 10 элементов буферов;
- Вычислить функцию F для каждого буфера и вывести результат на экран;
- Выполнить сортировку буферов методом S;
- Вывести на экран первые 10 элементов буферов;
- Сохранить буферы в файл с использованием метода O.

При написании программы допускается расширение классов необходимыми полями и методами.

3. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Основные теоретические положения ООП даны в курсе «Объектно-ориентированное программирование», поэтому лишь напомним основные понятия ООП и остановимся на отличительных особенностях их реализации на языке Java.

3.1. Классы и объекты в Java

Класс является абстрактным типом данных, определяемым пользователем, и представляет собой модель реального объекта в виде данных и

функций для работы с ними. Данные класса называются ***полями*** (по аналогии с полями структуры), а функции класса — ***методами***. Поля и методы называются элементами класса.

Ниже приведен пример описания класса на языке Java.

```
class MyClass {
    // "Поле"
    String name = "Example";
     // "Конструктор"
    public MyClass(String name) {
        this.name = name;
    }
     // "Метод"
    public String getName() {
        return name;
    }
}
```

Объект (экземпляр) – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом. Конкретный объект некоторого класса можно создать с помощью оператора **new.**

Конструктор — это специальный метод класса, который вызывается автоматически в момент создания объектов этого класса. Имя конструктора в Java совпадает с именем класса.

```
MyClass my = new MyClass("Example 2");
```

3.2. Наследование в Java

Наследование — это отношение между классами, при котором один класс расширяет функциональность другого. Это значит, что он автоматически перенимает все его поля и методы, а также добавляет некоторые свои.

Расширение (наследование) родительского класса осуществляется при помощи ключевого слова **extends.** При этом для Java возможно наследование только от одного класса-родителя. Класс-наследник может переопределять методы родителя. ***Переопределением*** называют объявление метода, сигнатура которого совпадает с одним из методов родительского класса. Иногда при переопределении бывает полезно воспользоваться результатом работы родительского метода, для этого применяется слово **super.**

Например, ниже в реализация конструктора класса будет вызывать конструктор родительского класса.

```
public class MyClassChild extends MyClass {
        public MyClassChild(String name) {
                super(name);
        }
}
```

3.3. Инкапсуляция и модификаторы в Java

Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе, и скрыть детали реализации от пользователя. При помощи трех модификаторов доступа можно задать различные уровни инкапсуляции класса – **private, public, protected**.

Перед словом **class** можно записать модификаторы класса. Это одно из слов **public**, **abstract**, **final**, **strictfp**. Перед именем вложенного класса можно поставить, кроме того, модификаторы **protected**, **private**, **static**.

Описание поля может начинаться с одного или нескольких необязательных модификаторов **public**, **protected**, **private**, **static**, **final**, **transient**, **volatile**.

Описание метода может начинаться с модификаторов **public**, **protected**, **private**, **abstract**, **static**, **final**, **synchronized**, **native**, **strictfp**.

Модификаторы доступа

Доступ к любому члену класса — полю или методу — может быть ограничен. Для этого перед его объявлением ставится ключевое слово **private**. Оно означает, что к этому члену класса нельзя будет обратиться из методов других классов. Ключевое слово **public** может употребляться в тех же случаях, но имеет противоположный смысл. Оно означает, что данный член класса является доступным. Если это поле, его можно использовать в выражениях или изменять при помощи присваивания, а если метод, его можно вызывать. Ключевое слово **protected** означает, что доступ к полю или методу имеет сам класс и все его потомки. Если при объявлении члена класса не указан ни один из перечисленных модификаторов, используется модификатор по умолчанию (default). Он означает, что доступ к члену класса имеют все классы, объявленные в том же пакете.

Модификатор static

Любой член класса можно объявить статическим, указав перед его объявлением ключевое слово **static**. Статический член класса «разделяется» между всеми его объектами. Для поля это означает, что любое изменение его значения, сделанное одним из объектов класса, сразу же «увидят» все остальные объекты. Метод, объявленный с модификатором **static**, не изменяет никаких полей класса, кроме статических. Для обращения к статическому члену класса можно использовать любой объект этого класса. Более того, это обращение можно осуществлять даже тогда, когда не создано ни одного такого объекта. Вместо имени объекта можно просто указывать имя класса.

Модификатор final

Любое поле класса можно объявить неизменяемым, указав перед его объявлением ключевое слово **final**. Неизменяемому полю можно присвоить значение только один раз (обычно это делается сразу при объявлении).

Константы в языке Java очевидным образом описываются путем совмещения модификаторов static и final. Например, мы можем объявить константу PI, написав:

```
final static double PI = 3.14;
```

Если ключевое слово **final** указать перед объявлением метода, это будет обозначать, что метод нельзя переопределять при наследовании (т.е. данная версия метода будет окончательной). Перед объявлением класса модификатор **final** ставится в том случае, если необходимо запретить от него наследования.

3.3. Полиморфизм в Java

Полиморфизм — это возможность класса выступать в программе в роли любого из своих предков, несмотря на то, что в нем может быть изменена реализация любого из методов.

Иногда имеет смысл описать только заголовок метода, без его тела, и таким образом объявить, что данный метод будет существовать в этом классе. Реализацию этого метода, то есть его тело, можно описать позже. Классы, в которых объявлены только заголовки методов без конкретной реализации, называются ***абстрактными***. Для объявления абстрактного класса применяется ключевое слово **abstract**.

```
// Базовая арифметическая операция
abstract class Operation {
  public abstract int calculate(int a, int b);
}
// Сложение
class Addition extends Operation {
  public int calculate(int a, int b) {
    return a+b;
   }
}
// Вычитание
class Subtraction extends Operation {
   public int calculate(int a, int b) {
      return a-b;
   }
}
```

3.4. Интерфейсы

Интерфейс – это набор методов класса, доступных для использования другими классами. Интерфейсом класса будет являться набор всех его публичных методов в совокупности с набором публичных атрибутов. По сути, интерфейс специфицирует класс, чётко определяя все возможные действия над ним.

Концепция абстрактных методов позволяет предложить альтернативу множественному наследованию. В Java класс может иметь только одного родителя, поскольку при множественном наследовании могут возникать конфликты, которые запутывают объектную модель. Интерфейсы допускают множественное наследование.

Объявление интерфейсов очень похоже на упрощенное объявление классов. Оно начинается с заголовка. Сначала указываются модификаторы. Интерфейс может быть объявлен как **public** и тогда он будет доступен для общего использования, либо модификатор доступа может не указываться, в этом случае интерфейс доступен только для классов своего пакета. Модификатор **abstract** для интерфейса не требуется, поскольку все интерфейсы являются абстрактными. Его можно указать, но делать этого не рекомендуется, чтобы не загромождать код. Далее записывается ключевое слово **interface** и имя интерфейса. После этого может следовать ключевое слово **extends** и список интерфейсов, от которых будет наследоваться объявляемый интерфейс. Родительских типов может быть много, главное, чтобы не было повторений и чтобы отношение наследования не образовывало циклической зависимости.

Можно заметить, что интерфейс, с точки зрения реализации, — это просто чистый абстрактный класс, то есть класс, в котором не определено ничего, кроме абстрактных методов.

Ниже приводится пример интерфейса, который содержит четыре метода для вычисления и получения площади и периметра фигуры.

```
public interface IShape
{
   public double getSquare();
   public void calculateSquare();
   public double getPerimeter();
   public void calculatePerimeter();
}
```

Реализация интерфейса конкретным классом выполняется при помощи ключевого слова **implements**. Каждый класс может реализовывать любые доступные интерфейсы. При этом в классе должны быть реализованы все абстрактные методы, появившиеся при наследовании от интерфейсов или родительского класса, чтобы новый класс мог быть объявлен неабстрактным. Рекомендуется к переопределяемым методам добавлять аннотацию **@Override**. **(*Java-аннотация*** — в языке Java специальная форма синтетических метаданных, которая может быть добавлена в исходный код. Аннотации используются для анализа кода, компиляции или выполнения).

@Override проверяет, переопределен ли метод. Вызывает предупреждение компиляции, если метод не найден в родительском классе.

Далее приводится описание класса, описывающего круг, и реализующего интерфейс IShape.

```
public class Circle implements IShape
{
   protected double square;
   protected double perimeter;
   private double _radius;
      public Circle(double radius)
   {
      _radius = radius;
   }
```

```
    @Override
  public double getSquare()
  {
    this.calculateSquare();
    return this.square;
  }
  @Override
  public void calculateSquare()
  {
    this.square = Math.PI * Math.sqrt(_radius);
  }
  @Override
  public double getPerimeter()
  {
    calculatePerimeter();
    return this.perimeter;
  }
  @Override
  public void calculatePerimeter()
  {
    this.perimeter = 2 * Math.PI * _radius;
  }
  @Override
  public String toString()
  {
    return "Parametrs this circle: Square = " + this.square + ", Perimeter = " +
this.perimeter;
  }
}
```

3.5. Отличительные особенности ООП в Java

1) *В Java используются только динамические объекты*, статических объектов нет. Это означает, что, добавляя переменную объектного типа, создаётся указатель, а не сам объект. Объект создается только на этапе выполнения специальной операцией **new**. Память под сам указатель выделяется статически компилятором, а память под объект выделяется динамически средой выполнения.

2) *В Java используется автоматическое управление памятью*. Сборщик мусора Java работает в фоновом режиме, отслеживая объекты, в которых приложение больше не нуждается, и освобождая от них память. В связи с тем, что управление памятью в Java выполняется автоматически, здесь на уровне языка отсутствуют явные указатели и адресная арифметика, а также операции взятия адреса, разыменование указателя, обращение к элементу класса по ссылке. При этом объявления переменных объектных типов оформлять как указатели не надо. Здесь

также нет деструкторов и операции **delete**. Аналогом деструкторов в Java можно считать метод-финализатор, который вызывается автоматически сборщиком мусора перед освобождением памяти занятой объектом.

3) *Классы в Java являются основным средством конструирования новых типов данных.* Здесь отсутствуют структуры и объединения. Классы в Java используются и как средства структурирования программ. Здесь отсутствует понятие модуля, как физического контейнера классов. Это означает, что минимальной единицей компиляции является класс, а любая программная система на Java есть система классов. Классы состоят из полей и методов.

4) Ключевое отличие реализации ООП в Java от C++ — это *отсутствие множественного наследования* и связанных с ним сложностей, как для программиста, так и для разработчика компилятора.

5) В Java нет невиртуальных методов — это означает, что *все публичные методы классов являются виртуальными*, т.е. на этапе выполнения для них осуществляется динамическая диспетчеризация вызовов, которая предусматривает, что вызывается метод того класса, с объектом которого программа работает в данный момент, а не метод того класса, который использовался при объявлении переменной объектного типа. Такое поведение является основой полиморфизма: переменные базового типа могут ссылаться на объекты производных типов.

6) *Класс описывается в одном файле*. Один файл может содержать описание нескольких классов. В этом случае лишь один из них должен быть публичным. Имя файла должно в точности соответствовать имени публичного класса.

4. ВАРИАНТЫ ЗАДАНИЙ

Таблица 4.1 Варианты заданий

№	Количество буферов (N)	Типы элементов буфера (T)	Число элементов в буферах (L)	Сортировка (S)	Вычисление (F)	Сохранение (O)
1	3	Integer	50	Пузырька	Min	SaveOneLine
2	4	Long	60	Выборки	Max	SaveSeparateLines
3	5	Float	70	Шелла	Sum	SaveOneLine
4	6	Double	80	Быстрая	Min	SaveSeparateLines
5	7	Long	90	Пузырька	Max	SaveOneLine
6	3	Float	80	Выборки	Sum	SaveSeparateLines
7	4	Double	70	Шелла	Min	SaveOneLine

8	5	Integer	60	Быстрая	Max	SaveSeparateLines
9	6	Float	50	Пузырька	Sum	SaveOneLine
10	7	Double	40	Выборки	Min	SaveSeparateLines
11	3	Long	90	Пузырька	Max	SaveOneLine
12	4	Integer	70	Выборки	Sum	SaveSeparateLines
13	5	Double	50	Шелла	Min	SaveOneLine
14	6	Long	30	Быстрая	Max	SaveSeparateLines
15	7	Integer	60	Пузырька	Sum	SaveOneLine

5. СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

Титульный лист, цель работы, постановку задачи, вариант задания, текст программы с комментариями, скриншоты выполнения и описание тестовых примеров, выводы по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Поясните понятия *класс* и *объект* и особенности их реализации на языке Java?
2. Как осуществляется наследование на языке Java?
3. Что такое инкапсуляция и какие уровни инкапсуляции можно реализовать на Java?
4. Что такое переопределение?
5. Поясните понятие виртуального метода и как сделать метод виртуальным при написании Java программ?
6. Поясните действие спецификатора **static**?
7. Поясните действие спецификатора **final**?
8. Что такое абстрактный класс?
9. Объясните проблемы множественного наследования?
10. Что такое интерфейсы и как они объявляются и реализуются на языке Java ?
11. Почему на Java отсутствует понятие деструктор**?**

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ноутон, П. Java™ 2 [Текст] : пер. с англ. / П. Ноутон, Г. Шилдт. - СПб. : БХВ – Петербург, 2007. - 1050 с.

2. Шилдт, Г. Искусство программирования на Java [Текст] : пер. с англ. / Г. Шилдт, Д. Холмс. - М. ; СПб. ; К. : Вильямс, 2005. - 334 с

3. Хабибуллин, И. Ш. Java 2 [Текст] : самоучитель / И. Ш. Хабибуллин. - СПб. : БХВ - Петербург, 2005. - 720 с.

Заказ № _____ от «_____» _____2015г. Тираж _____экз.
Изд-во СевГУ