|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ №1-4**

**по дисциплине «Программирование на языке Джава»**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил:  Студент группы Шмелев Д.В. | « » 2021г. | (подпись) | Шмелев Д.В. |
|  |  |  |  |
| Принял:  Ассистент кафедры ИиППО ИИТ | « » 2021г. | (подпись) | Литвинов В.В. |

Москва 2021 г.

Оглавление

[ПРАКТИЧЕСКАЯ РАБОТА №1. КЛАССЫ, КАК НОВЫЕ ТИПЫ ДАННЫХ. ПОЛЯ ДАННЫХ И МЕТОДЫ 4](#_Toc89947453)

[Цель работы 4](#_Toc89947454)

[Теоретическое введение 4](#_Toc89947455)

[Постановка задачи 5](#_Toc89947456)

[Программный код 6](#_Toc89947457)

[Вывод программы 7](#_Toc89947458)

[Вывод 7](#_Toc89947459)

[ПРАКТИЧЕСКАЯ РАБОТА №2. ИСПОЛЬЗОВАНИЕ UML ДИАГРАММ В ОБЪЕКТНООРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ 8](#_Toc89947460)

[Цель работы 8](#_Toc89947461)

[Теоретическое введение 8](#_Toc89947462)

[Постановка задачи 9](#_Toc89947463)

[Программный код 10](#_Toc89947464)

[Вывод программы 11](#_Toc89947465)

[Вывод 11](#_Toc89947466)

[ПРАКТИЧЕСКАЯ РАБОТА № 3. НАСЛЕДОВАНИЕ. АБСТРАКТНЫЕ СУПЕРКЛАССЫ И ИХ ПОДКЛАССЫ В JAVA. 12](#_Toc89947467)

[Цель работы 12](#_Toc89947468)

[Теоретическое введение 12](#_Toc89947469)

[Постановка задачи 12](#_Toc89947470)

[Программный код 14](#_Toc89947471)

[Вывод программы 16](#_Toc89947472)

[Вывод 16](#_Toc89947473)

[ПРАКТИЧЕСКАЯ РАБОТА № 4. СОЗДАНИЕ GUI. СОБЫТИЙНОЕ ПРОГРАММИРОВАНИЕ В JAVA. 17](#_Toc89947474)

[Цель работы 17](#_Toc89947475)

[Теоретическое введение 17](#_Toc89947476)

[Постановка задачи 18](#_Toc89947477)

[Программный код 20](#_Toc89947478)

[Вывод программы 22](#_Toc89947479)

[Вывод 23](#_Toc89947480)

[ПРАКТИЧЕСКАЯ РАБОТА № 5. ПРОГРАММИРОВАНИЕ РЕКУРСИИ В JAVA 24](#_Toc89947481)

[Цель работы 24](#_Toc89947482)

[Теоретическое введение 24](#_Toc89947483)

[Постановка задачи 25](#_Toc89947484)

[Программный код 27](#_Toc89947485)

[Вывод программы 33](#_Toc89947486)

[Вывод 33](#_Toc89947487)

[ПРАКТИЧЕСКАЯ РАБОТА № №6. ТЕХНИКИ СОРТИРОВКИ В JAVA 34](#_Toc89947488)

[Цель работы 34](#_Toc89947489)

[Теоретическое введение 34](#_Toc89947490)

[Постановка задачи 34](#_Toc89947491)

[Программный код 34](#_Toc89947492)

[**** 34](#_Toc89947493)

[Вывод программы 34](#_Toc89947494)

[**** 34](#_Toc89947495)

[Вывод 34](#_Toc89947496)

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

## Цель работы

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

## Теоретическое введение

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 диаграммы как прямоугольник в виде как трех секций, в котором присутствует секция наименования класса, секция инкапсуляции данных и методов (функций или операций) класса.

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 c 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.

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.

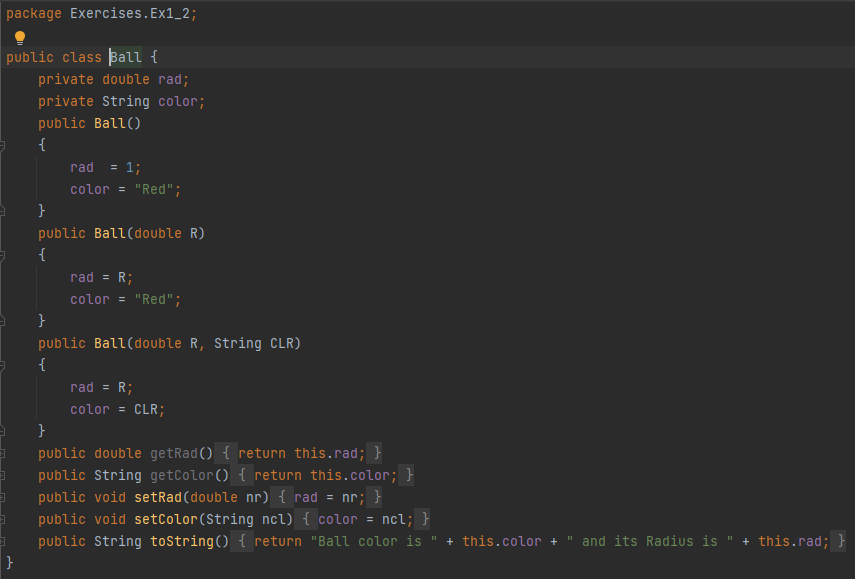
Экземпляр класса — это представление (или реализация) конкретного представителя класса.

## Постановка задачи

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 2. Реализуйте простейший класс «Мяч»

## Программный код



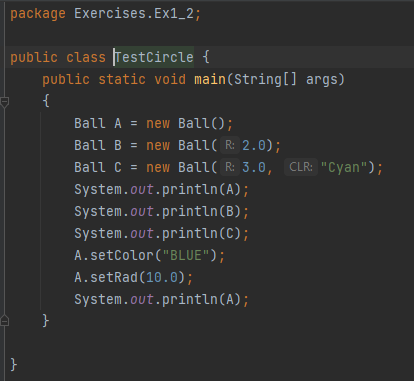


Рис. Код программы

## Вывод программы

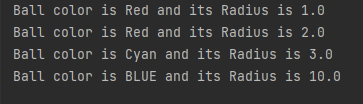


Рис. Вывод программы

## Вывод

В результате выполнения данной практической работы мы смогли освоить на практике работу с классами на Java, научились писать классы, методы, переменные и взаимодействовать с ними.

# ПРАКТИЧЕСКАЯ РАБОТА №2. ИСПОЛЬЗОВАНИЕ UML ДИАГРАММ В ОБЪЕКТНООРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

## Цель работы

Цель данной работы – это научиться работать с UML-диаграммами классов.

## Теоретическое введение

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).

Методы (или поведение, функции, работа c 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).

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).

## Постановка задачи

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

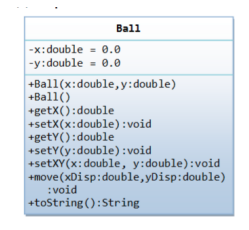


Рис. Пример UML диаграммы

Класс Ball моделирует движущийся мяч. В состав класса входят:

• Две переменные с модификатором private (поля данных класса): х, у, которые описывают положение мяча на поле.

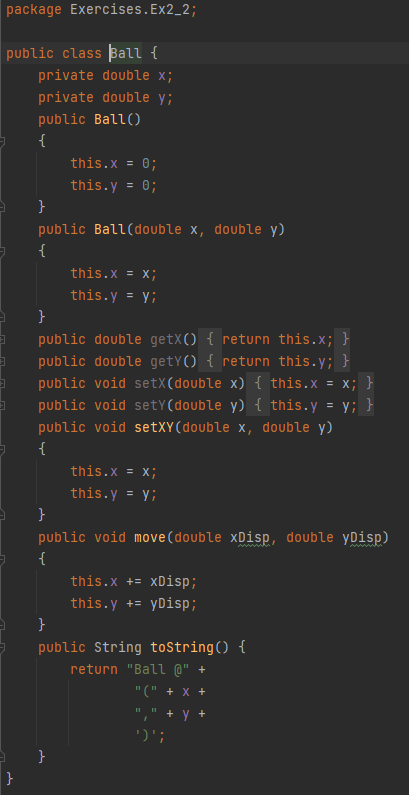
• Конструкторы, public методы получения и записи значений для private переменных.

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

• Метод move() , позволяет переместить мяч, так что что увеличивает х и у на данном участке на xDisp и yDisp, соответственно.

• Метод toString(), который возвращает "Ball @ (х , у) " .

## Программный код



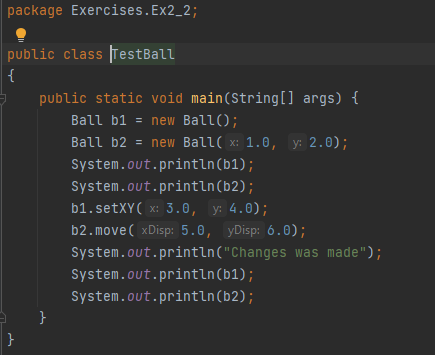


Рис. Код программы

## Вывод программы

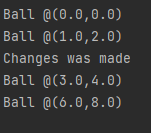


Рис. Вывод программы

## Вывод

В результате выполнения данной практической работы, мы научились создавать классы, основываясь на UML диаграммах

# ПРАКТИЧЕСКАЯ РАБОТА № 3. НАСЛЕДОВАНИЕ. АБСТРАКТНЫЕ СУПЕРКЛАССЫ И ИХ ПОДКЛАССЫ В JAVA.

## Цель работы

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

## Теоретическое введение

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.

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.

## Постановка задачи

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

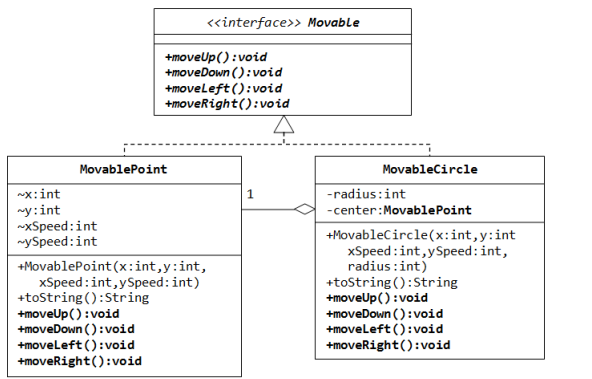


Рис. UML диаграмма задание

## Программный код

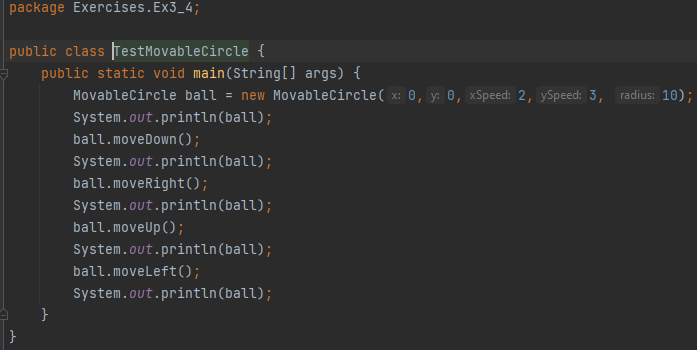
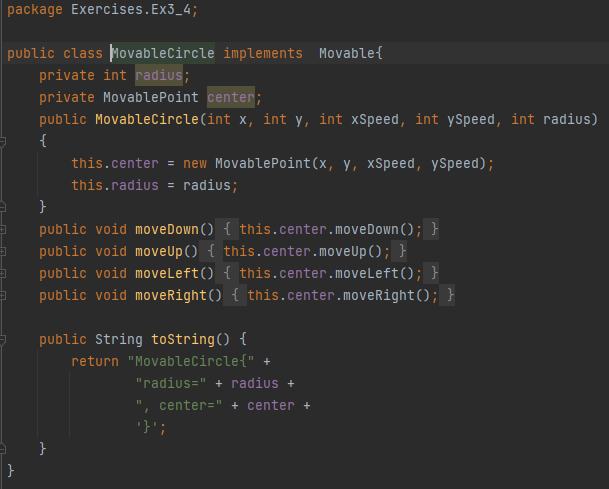
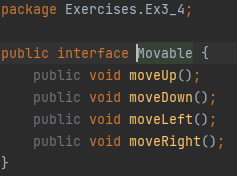


Рис. Код программы

## Вывод программы

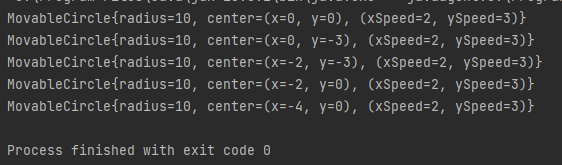


Рис. Вывод программы

## Вывод

В результате выполнения данной практической работы, мы научились работать и создавать абстрактные классы, изучили наследование, и научились создавать и реализовывать интерфейсы.

# ПРАКТИЧЕСКАЯ РАБОТА № 4. СОЗДАНИЕ GUI. СОБЫТИЙНОЕ ПРОГРАММИРОВАНИЕ В JAVA.

## Цель работы

Цель работы: введение в событийное программирование на языке Java.

## Теоретическое введение

Данная практическая работа посвящена закреплению практических навыков по созданию приложений на Java c 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.

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.

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.

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

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.

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

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

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)).

Добавление меню в программе Java проста. 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.

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

## Постановка задачи

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”

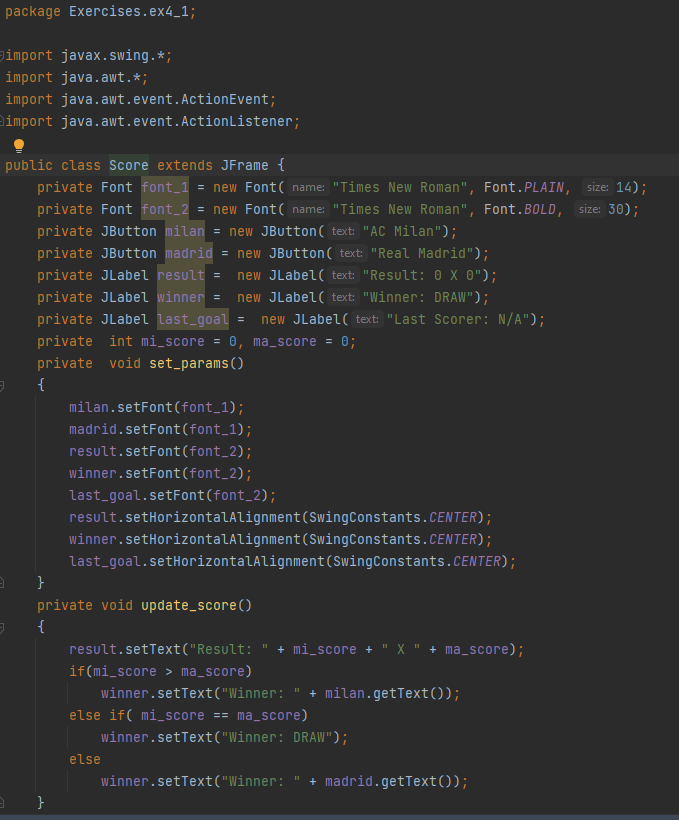
• другая JButton подписана “Real Madrid” • надпись JLabel содержит текст “Result: 0 X 0”

• 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.

## Программный код



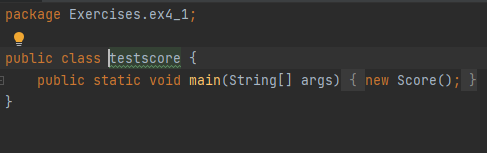


Рис. Код программы

## Вывод программы

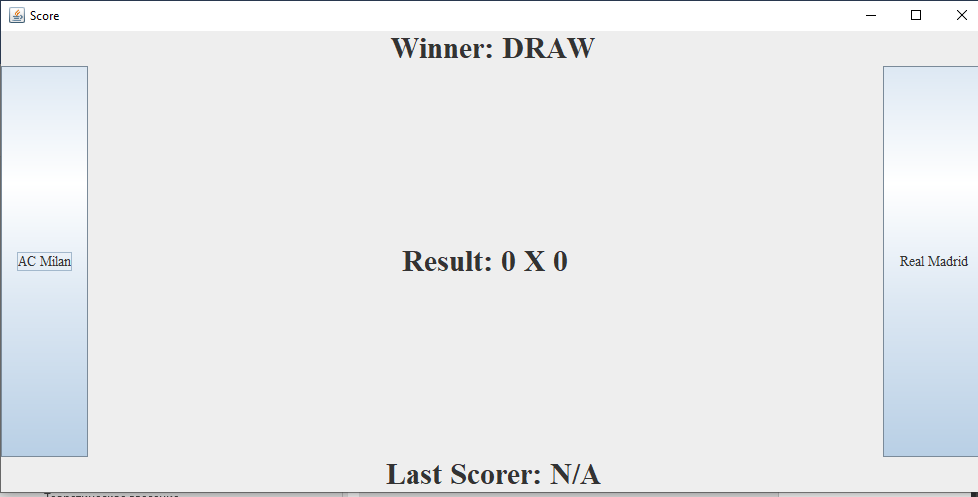


Рис. Приложение сделанное в коде

## Вывод

В результате выполнения данной практической работы, мы научились создавать базовые элементы приложения на Java, используя JPannel, JButton, Border Layout и другие.

# ПРАКТИЧЕСКАЯ РАБОТА № 5. ПРОГРАММИРОВАНИЕ РЕКУРСИИ В JAVA

## Цель работы

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

## Теоретическое введение

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

## Постановка задачи

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

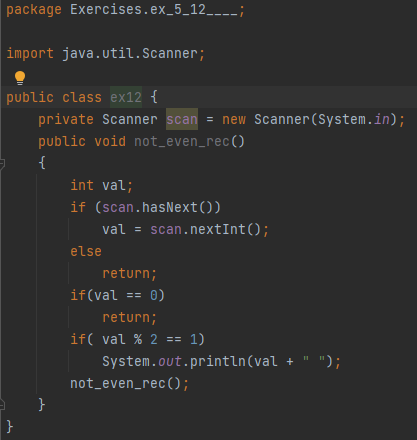
15. Цифры числа справа налево Дано натуральное число N. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами или новыми строками. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика.

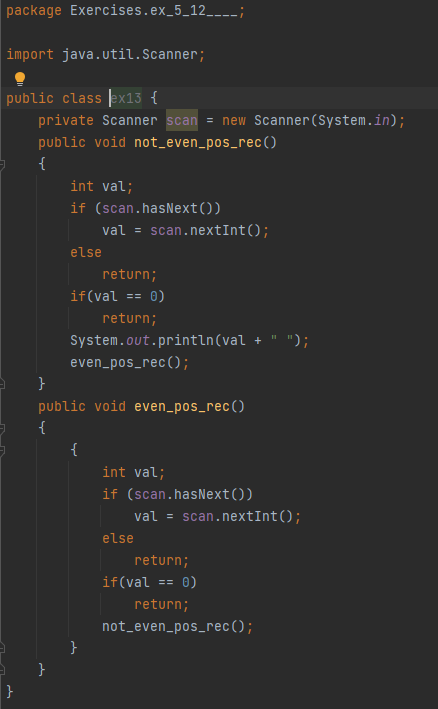
16. Количество элементов, равных максимуму Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Определите, какое количество элементов этой последовательности, равны ее наибольшему элементу.

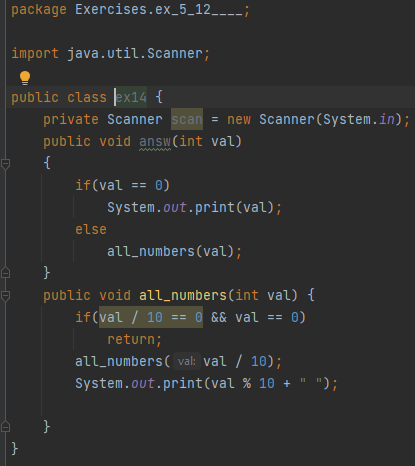
В этой задаче нельзя использовать глобальные переменные. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметра. В программе на языке Python функция возвращает результат в виде кортежа из нескольких чисел, и функция вообще не получает никаких параметров. В программе на языке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

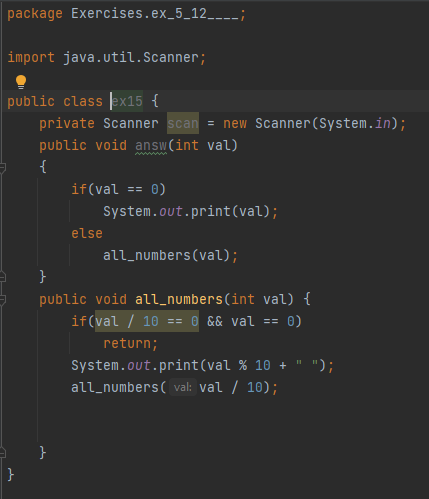
Гарантируется, что последовательность содержит хотя бы одно число (кроме нуля)

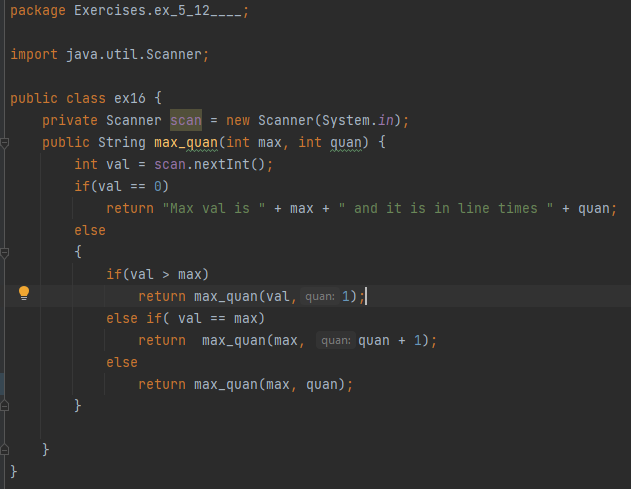
## Программный код











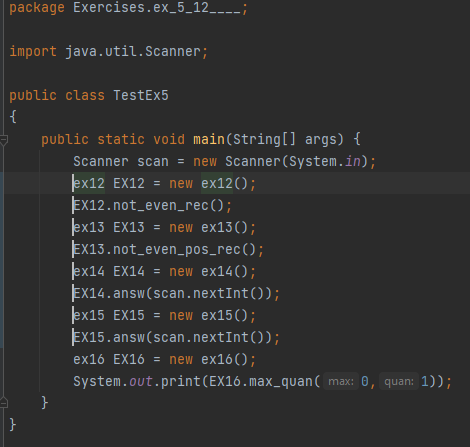
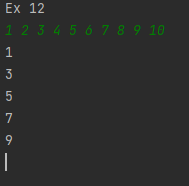
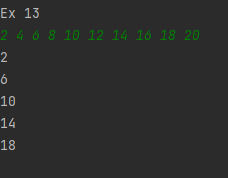
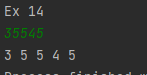


Рис. Код программы

## Вывод программы







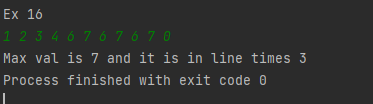
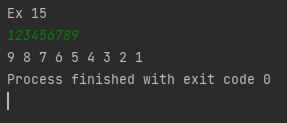


Рис. Вывод программы

## Вывод

В результате выполнения данной практической работы, мы научились программировать рекурсивные функции в Java. Узнали, что такое глубина, шаг и условие рекурсии. Также написали задачи на рекурсивные функции.

# ПРАКТИЧЕСКАЯ РАБОТА №6. ТЕХНИКИ СОРТИРОВКИ В JAVA

## Цель работы

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

## Теоретическое введение

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.

## Постановка задачи

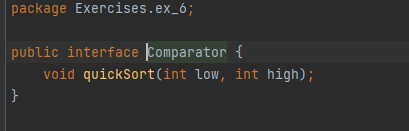
Упражнение 2.

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.

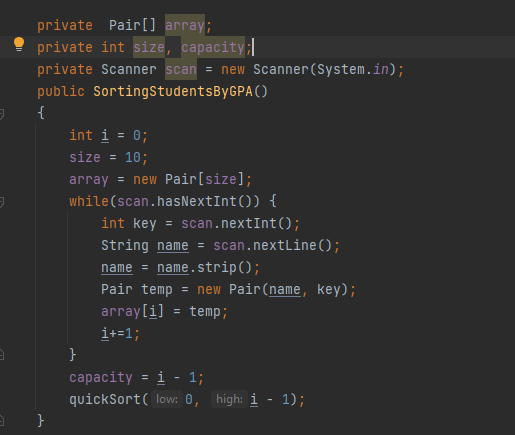
Упражнение 3.

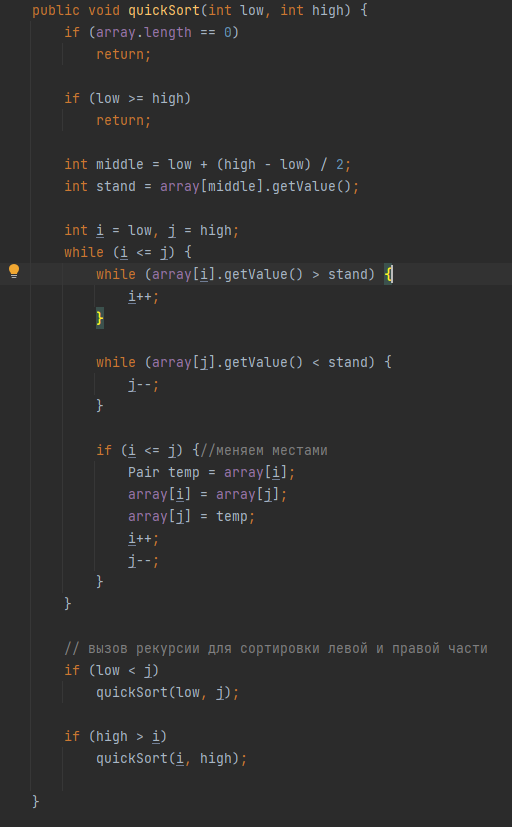
Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках с использованием алгоритма сортировки слиянием

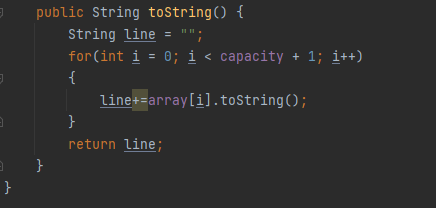
## Программный код

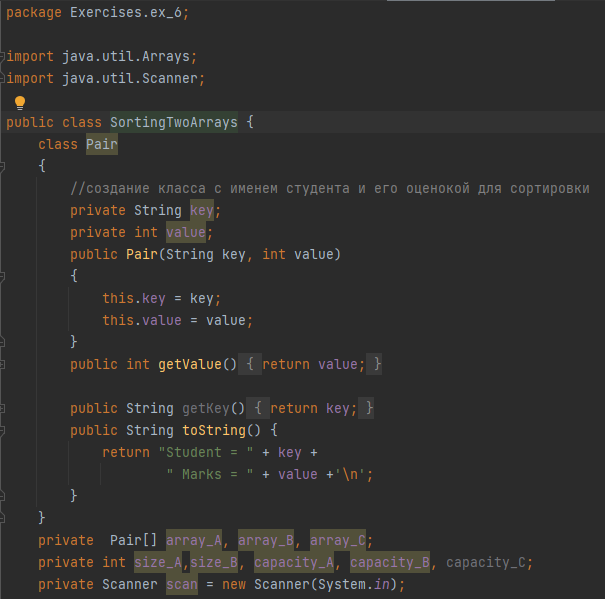




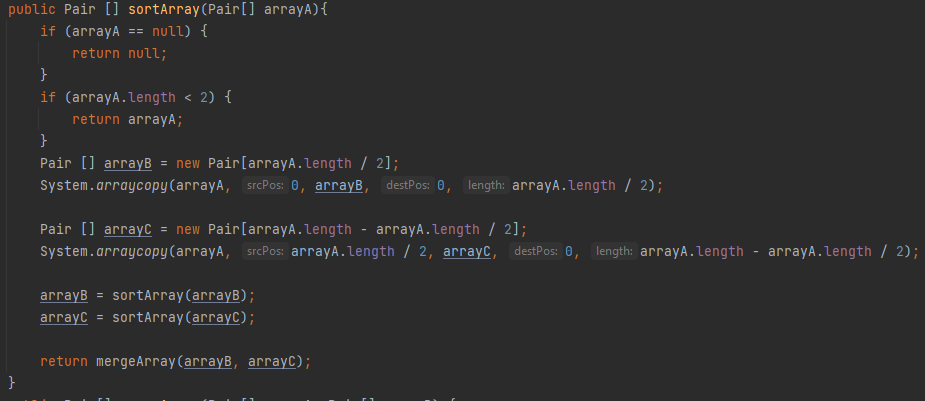


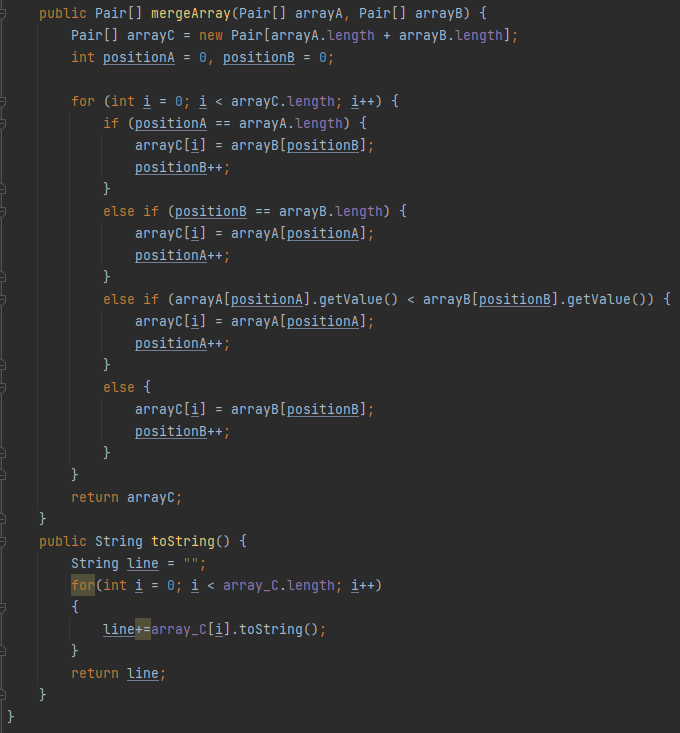












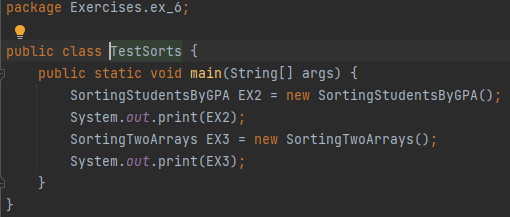
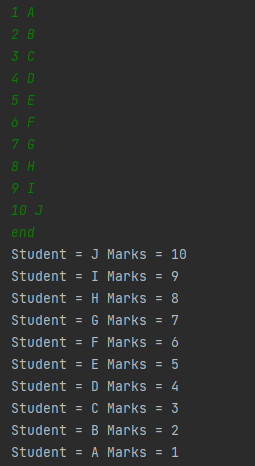


Рис. Код программы

## Вывод программы



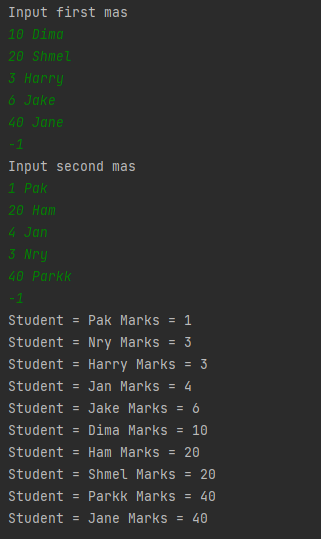


Рис. Вывод программы

## Вывод

В результате выполнения данной практической работы мы изучили и научились разным алгоритмам сортировки. А также научились реализовывать из в Java.

# ПРАКТИЧЕСКАЯ РАБОТА № 7. ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ КОНТЕЙНЕРНЫХ КЛАССОВ ПРИ ПРОГРАММИРОВАНИИ НА JAVA

## Цель работы

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

## Теоретическое введение

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).

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.

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

## Постановка задачи

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

1 3 5 7 9

2 4 6 8 0

Пример вывода

second 5

Упражнение 1

Используйте для организации хранения структуру данных Stack.

Упражнение 2

Используйте для организации хранения структуру данных Queue.

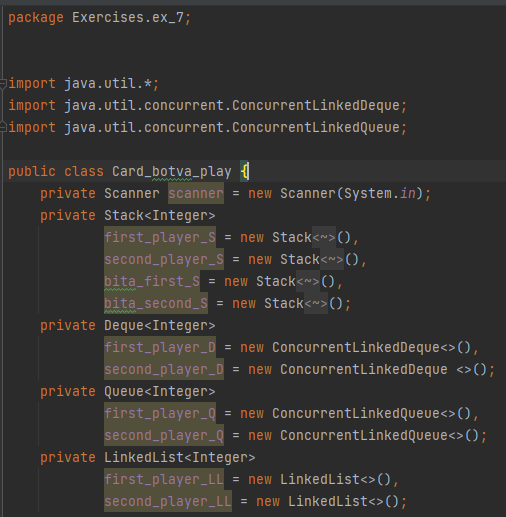
Упражнение 3

Используйте для организации хранения структуру данных Dequeue

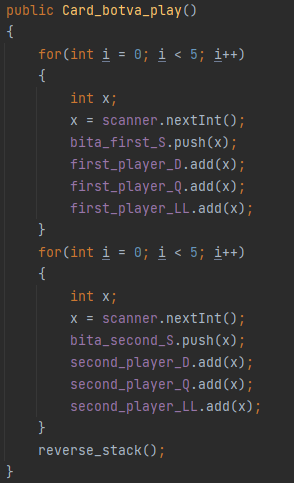
Упражнение 4

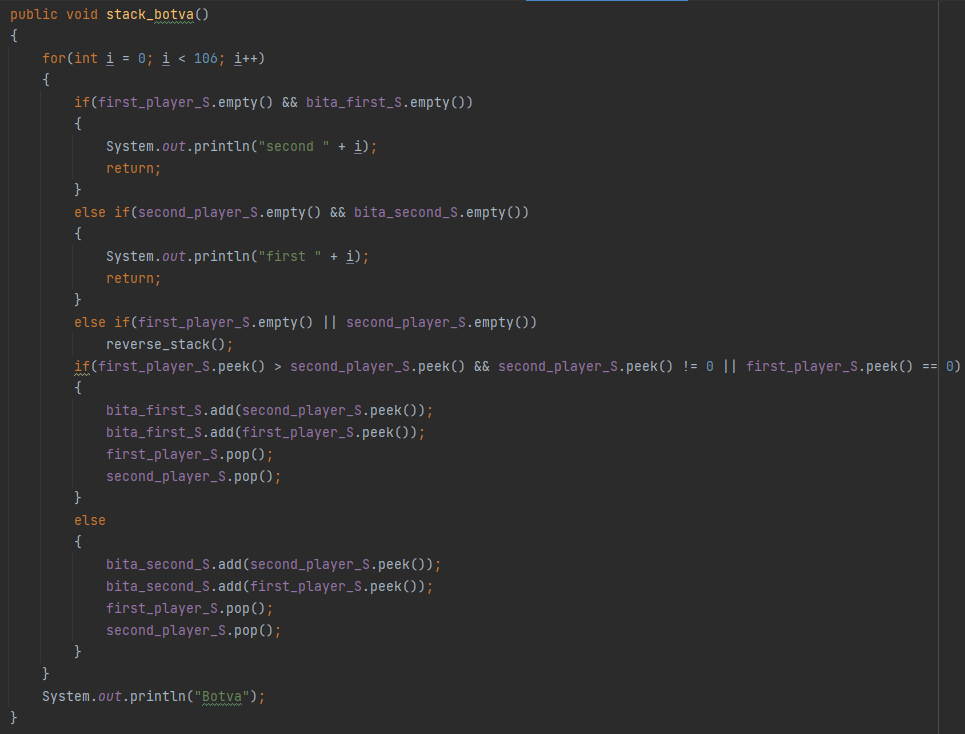
Используйте для организации хранения структуру данных DoubleList.

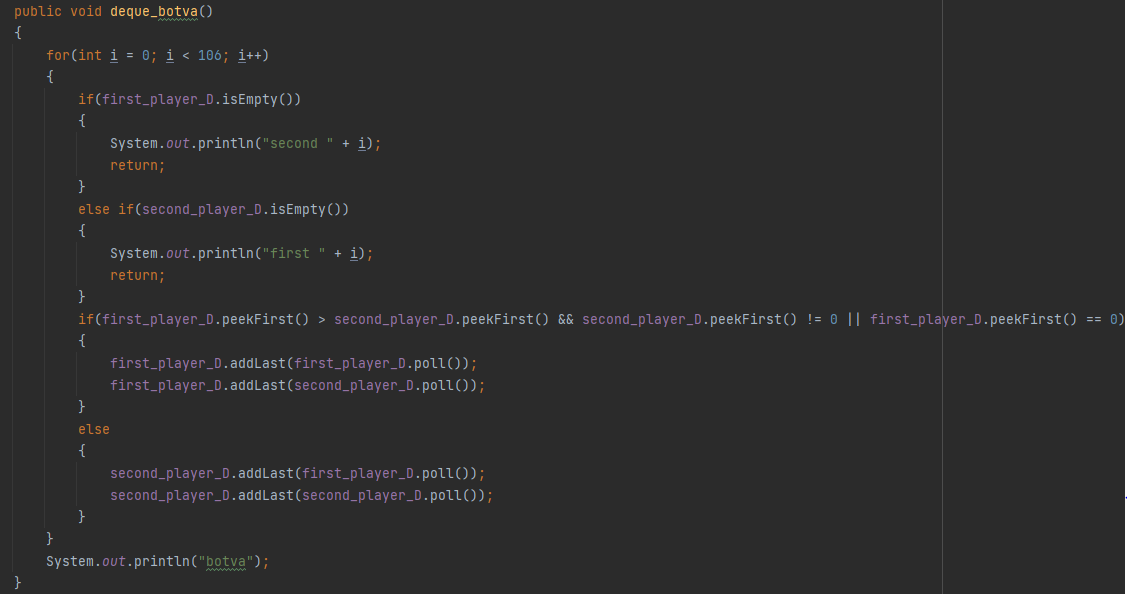
## Программный код

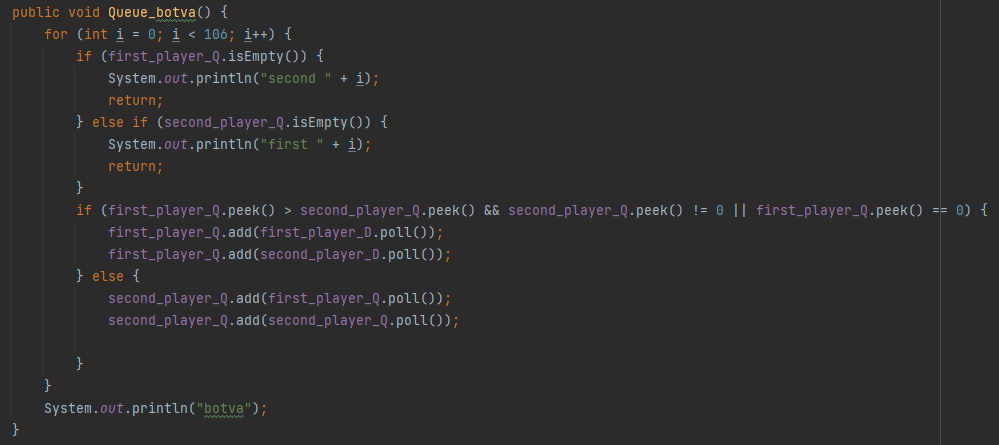


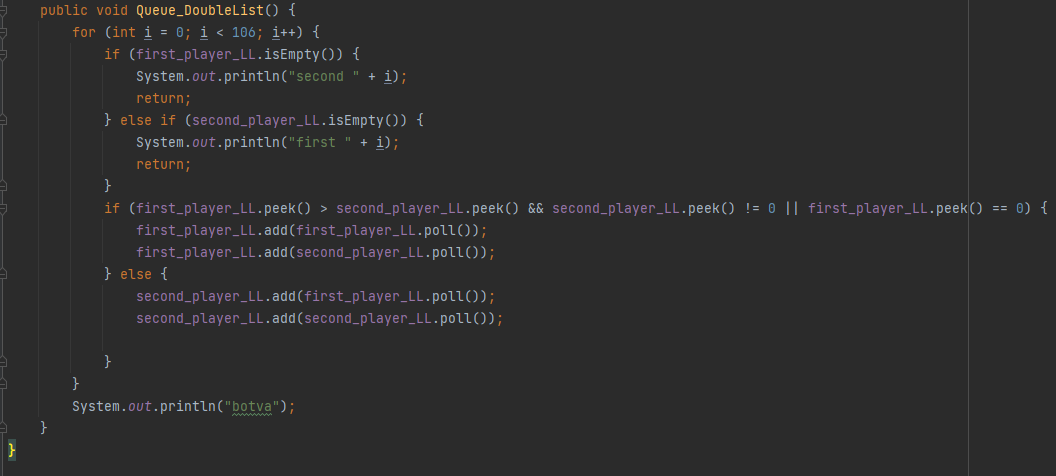












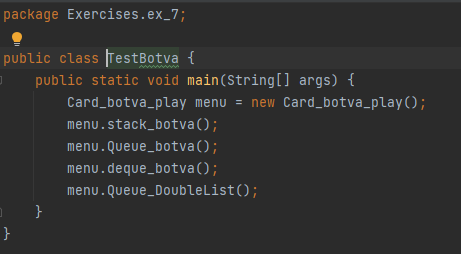


Рис. Код программы

## Вывод программы

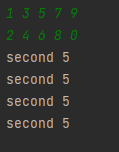


Рис. Вывод программы

## Вывод

В результате выполнения данной практической работы мы изучили контейнеры в Java, как их реализовывать и использовать. Узнали их различия и процесс работы.

# ПРАКТИЧЕСКАЯ РАБОТА № 8 РАЗЛИЧНЫЕ ВИДЫ СПИСКОВ ОЖИДАНИЯ

## Постановка задачи

Вначале, рассмотрите класс, представленный на диаграмме - общий класс список ожидания. Допустим мы решили, что нам понадобиться еще два вида списков ожидания:

• BoundedWaitList: Этот список ожидания имеет ограниченную емкость, указываемую в момент создания. Он не принимает более элементов, чем заранее задано (возможное количество потенциальных элементов в списке ожидания).

• UnfairWaitList: В этом списке ожидания, можно удалить элемент, который не является первым в очереди - и помните он не может вернуться обратно! (Возможны различные реализации, но в вашей реализации необходимо удалить первое вхождение данного элемента.) Также возможно, чтобы например, первый элемент будет отправлен обратно в конец списка.

После описания всей задачи в целом, мы сможем решить, что мы нам нужен интерфейс IWaitList, и затем нужно создать три разных класса для трех списков ожидания. Также предполагается, что один из списков ожидания должен быть супер классом для двух других списков ожидания.

Задание

Исследуйте UML диаграмму классов на рисунке и понаблюдайте, как она выражает то, что мы говорили выше в словах. Убедитесь, что вы понимаете все аспекты диаграммы.

Расширить и модифицировать исходный код WaitList, как необходимо, чтобы полностью реализовать всю схему UML. Включить комментарии Javadoc. Обратите внимание на переключение ролей после реализации каждого интерфейса / класса!

Изучение работу метода main(), которая использует ваши новые классы и интерфейс.

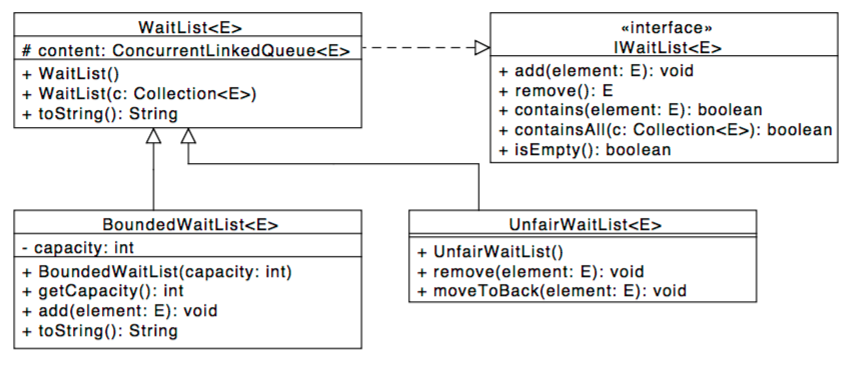
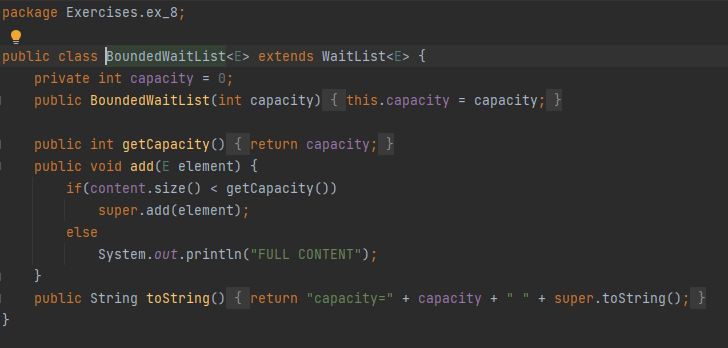
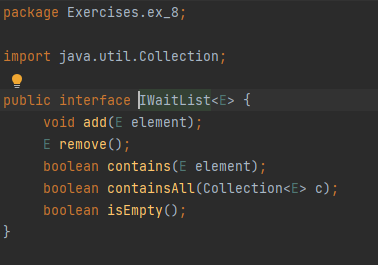
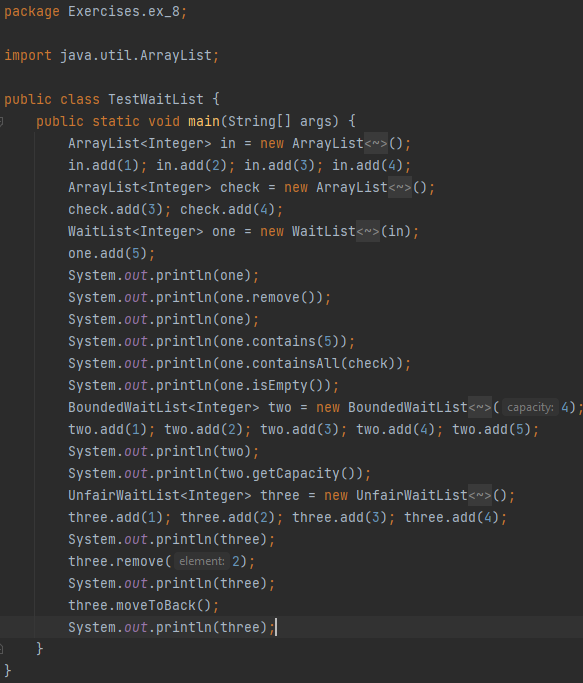


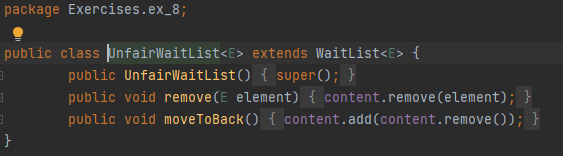
Рис. UML диаграмма задания

## Программный код









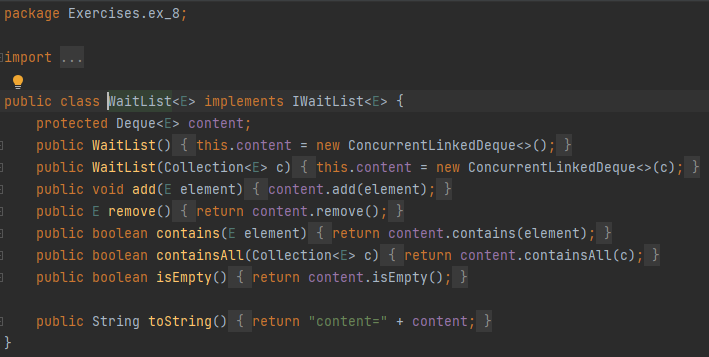


Рис. Код программы

## Вывод программы

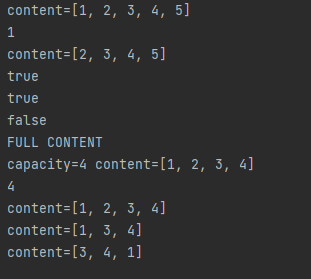


Рис. Вывод программы

## Вывод

В результате выполнения данной практической работы мы познакомились с WaitList также научились описывать собственные структуры, в которые мы можем записывать и извлекать данные, с описанными нами условиями.

# ПРАКТИЧЕСКАЯ РАБОТА №9 СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ИСКЛЮЧЕНИЙ

## Цель работы

Цель данной практической работы – научиться создавать собственные исключения.

## Теоретическое введение

Язык Java предоставляет исчерпывающий набор классов исключений, но иногда при разработке программ вам потребуется создавать новые – свои собственные исключения, которые являются специфическими для потребностей именно вашего приложения. В этой практической работе вы научитесь создавать свои собственные пользовательские классы исключений. Как вы уже знаете, в Java есть два вида исключений- проверяемы и непроверяемые. Для начала рассмотрим создание пользовательских проверяемых исключений.

Создание проверяемых пользовательских исключений

Проверяемые исключения — это исключения, которые необходимо обрабатывать явно. Рассмотрим пример кода:

try (Scanner file = new Scanner(new File(fileName)))

if (file.hasNextLine()) return file.nextLine(); }

catch(FileNotFoundException e) {

// Logging, etc

}

Приведенный выше код является классическим способом обработки проверяемых исключений на Java. Хотя код выдает исключение FileNotFoundException, но в целом неясно, какова точная причина ошибки – не такого файла нет или же имя файла является недопустимым.

Чтобы создать собственное пользовательское исключение, мы будем наследоваться от класса java.lang.Exception. Давайте рассмотрим пример как это реализуется на практике и создадим собственный класс для проверяемого исключения с именем BadFileNameException:

public class BadFileNameException extends Exception {

public BadFileNameException(String errorMessage) {

super(errorMessage);

}

}

Обратите внимание, что мы также должны написать конструктор в нашем классе, который принимает параметр типа String в качестве сообщения об ошибке, в котором вызывается конструктор родительского класса. Фактически это все, что нам нужно сделать, чтобы определить свое собственное пользовательское исключение.

Далее, давайте посмотрим, как мы можем использовать пользовательское исключение в программе:

try (Scanner file = new Scanner(new File(fileName))) {

if (file.hasNextLine())

return file.nextLine();

}

catch (FileNotFoundException e) {

if (!isCorrectFileName(fileName))

{

throw new BadFileNameException("Bad filename : " + fileName );

} //...

}

Мы создали и использовали свое собственное пользовательское исключение, теперь в случае ошибки, можно понять, что произошло, и какое именно исключение сработало. Как вы думаете, этого достаточно? Если ваш ответ да, то мы не узнаем основную причину, по которой сработало исключения. Как исправить программу. Для этого мы также можем добавить параметр java.lang.Throwable в конструктор. Таким образом, мы можем передать родительское исключение во время вызова метода:

public BadFileNameException(String errorMessage, Throwable err) {

super(errorMessage, err);

}

Теперь мы связали BadFileNameException с основной причиной возникновения данного исключения, например:

try (Scanner file = new Scanner(new File(fileName))) {

if (file.hasNextLine())

{

return file.nextLine();

}

}

catch (FileNotFoundException err) {

if (!isCorrectFileName(fileName)) {

throw new BadFileNameException( "Bad filename: " + fileName, err);

}

// ...

}

Мы рассмотрели, как мы можем использовать пользовательские исключения в программах, учитывая их связь с причинами по которым они могут возникать.

Создание непроверяемых пользовательских исключений

В том же примере, который мы рассматривали выше предположим, что нам нужно такое пользовательское исключение, в котором обрабатывается ошибка, если файла не содержит расширения.

В этом случае нам как раз понадобится создать пользовательское непроверяемое исключение, похожее на предыдущее, потому что данная ошибка будет обнаружена только во время выполнения участка кода. Чтобы создать собственное непроверяемое исключение, нам нужно наследоваться от класса java.lang.RuntimeException:

public class BadFileExtensionException extends RuntimeException { public BadFileExtensionException(String errorMessage, Throwable err) { super(errorMessage, err); } }

Теперь, мы можем использовать это нестандартное исключение в рассматриваемом нами выше примере:

try (Scanner file = new Scanner(new File(fileName))) { if (file.hasNextLine()) { return file.nextLine(); } else { throw new IllegalArgumentException("Non readable file"); } } catch (FileNotFoundException err) { if (!isCorrectFileName(fileName)) { throw new BadFileNameException( "Bad filename: " + fileName , err); } //... } catch(IllegalArgumentException err) { if(!containsExtension(fileName)) { throw new BadFileExtensionException( "Filename does not contain extension: " + fileName, err); } //... }

Заключение В приведенных выше примерах мы рассмотрели основные особенности обработки исключений.

## Постановка задачи

Клиент совершает покупку онлайн. При оформлении заказа у пользователя запрашивается фио и номер ИНН. В программе проверяется, действителен ли номер ИНН для такого клиента. Исключение будет выдано в том случае, если введен недействительный ИНН.

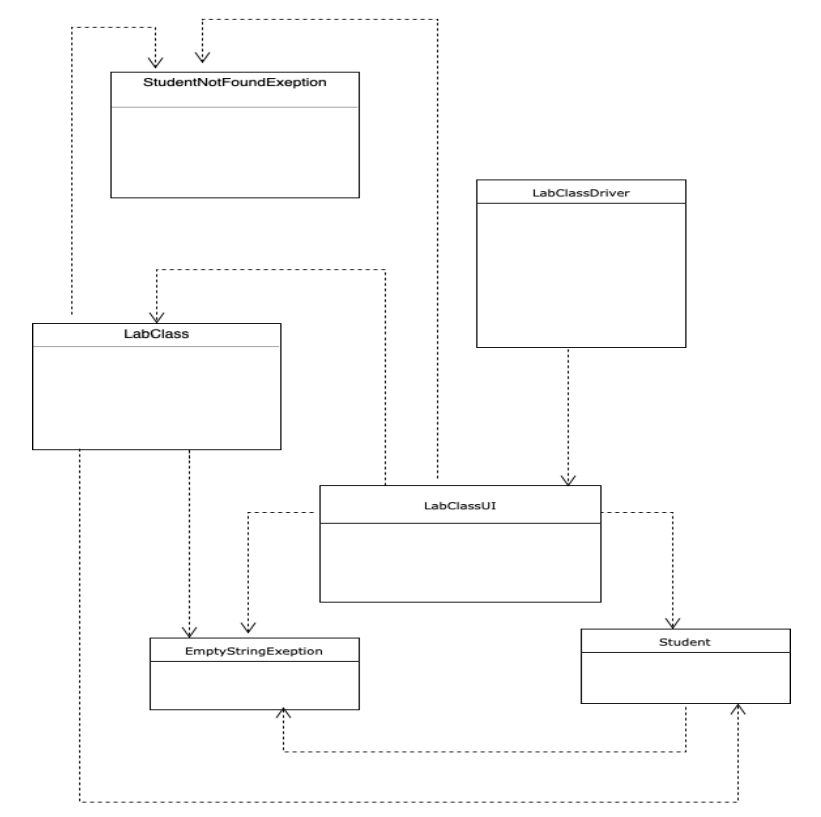


Рис. Диаграмма логики исключений

## Программный код

## Вывод программы

## 

## Вывод

В результате выполнения данной практической работы мы изучили исключения в Java. Научились их использовать создавать собственные, обрабатывать и т.д. Также реализовали программу, где мы их прописываем и обрабатываем.

# ПРАКТИЧЕСКАЯ РАБОТА № 10 ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ. ПОРОЖДАЮЩИЕ ПАТТЕРНЫ: АБСТРАКТНАЯ ФАБРИКА, ФАБРИЧНЫЙ МЕТОД

## Цель работы

Цель данной практической работы – научиться применять порождающие паттерны при разработке программ на Java. В данной практической работе рекомендуется использовать следующие паттерны: Абстрактная фабрика и фабричный метод.

## Теоретическое введение

Понятие паттерна.

Паттерны (или шаблоны) проектирования описывают типичные способы решения часто встречающихся проблем при проектировании программ.

## Постановка задачи

Упражнение 1

Реализовать класс Абстрактная фабрика для комплексных чисел

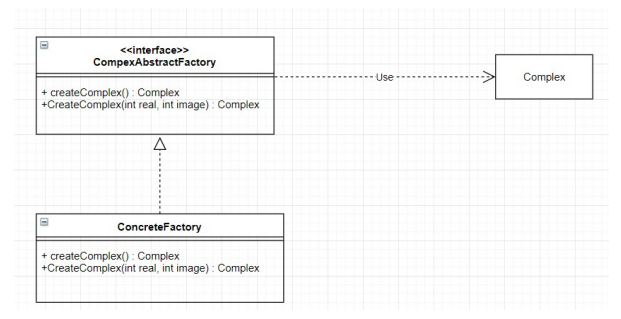


Рис. UML диаграмма задания

Упражнение 2

Реализовать класс Абстрактная фабрика для различных типов стульев: Викторианский стул, Многофункциональный стул, Магический стул, а также интерфейс Стул, от которого наследуются все классы стульев, и класс Клиент, который использует интерфейс стул в своем методе Sit (Chair chair).

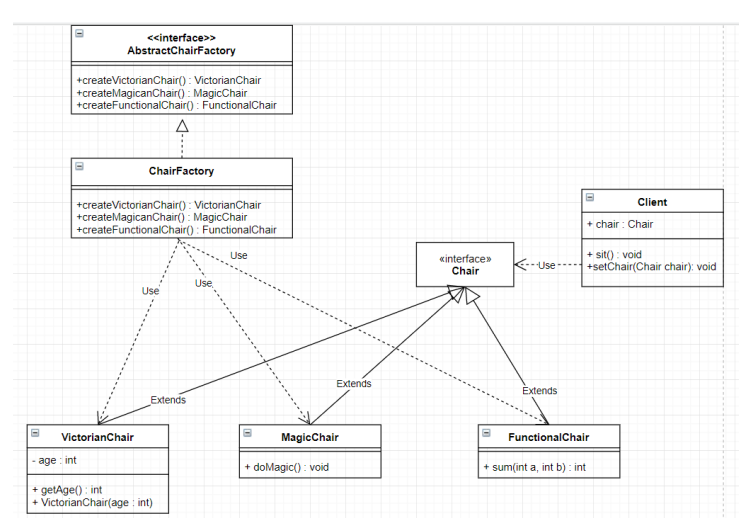


Рис. UML диаграмма задания

## Программный код

## 

## Вывод программы

## 

## Вывод

В результате выполнения данной практической работы мы изучили метод паттерного программирования абстрактная фабрика. Также научились реализовывать и работать с ним.

# ПРАКТИЧЕСКАЯ РАБОТА № №11, 12 ОБРАБОТКА СТРОК.

## Цель работы

Цель работы:

## Теоретическое введение

## Постановка задачи

## Программный код

## 

## Вывод программы

## 

## Вывод

В результате выполнения данной практической работы

# ПРАКТИЧЕСКАЯ РАБОТА №

## Цель работы

Цель работы:

## Теоретическое введение

## Постановка задачи

## Программный код

## 

## Вывод программы

## 

## Вывод

В результате выполнения данной практической работы

# ПРАКТИЧЕСКАЯ РАБОТА №

## Цель работы

Цель работы:

## Теоретическое введение

## Постановка задачи

## Программный код

## 

## Вывод программы

## 

## Вывод

В результате выполнения данной практической работы