|  |
| --- |
| 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 №1-24**

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

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил:  Студент группы | « » 2023 г. | (подпись) |  |
|  |  |  |  |
| Принял:  Старший преподаватель кафедры ИиППО ИИТ | « » 2023 г. | (подпись) | Матчин В.Т. |

Москва 2023 г.

Оглавление

[Практическая работа № 1. Знакомство со средой разработки. Синтаксис и основные управляющие конструкции языка Джава 6](#_Toc146456804)

[Введение 7](#_Toc146456805)

[Практическая часть 8](#_Toc146456806)

[Вывод 14](#_Toc146456807)

[Практическая работа № 2. Объектно-ориентированное программирование в Джава. Классы в Джава 15](#_Toc146456808)

[Введение 17](#_Toc146456809)

[Практическая часть 18](#_Toc146456810)

[Вывод 25](#_Toc146456811)

[Практическая работа № 3. Классы Math и Random. Классы оболочки 26](#_Toc146456812)

[Введение 27](#_Toc146456813)

[Практическая часть 28](#_Toc146456814)

[Вывод 32](#_Toc146456815)

[Практическая работа № 4. Перечисления и их использование в Джава программах. 33](#_Toc146456816)

[Введение 36](#_Toc146456817)

[Практическая часть 37](#_Toc146456818)

[Вывод 53](#_Toc146456819)

[Практическая работа № 4.1. Наследование в Джава. Абстрактные классы. 54](#_Toc146456820)

[Введение 56](#_Toc146456821)

[Практическая часть 57](#_Toc146456822)

[Вывод 59](#_Toc146456823)

[Практическая работа № 5. Создание программ с графическим интерфейсом пользователя на языке Джава 60](#_Toc146456824)

[Введение 61](#_Toc146456825)

[Практическая часть 62](#_Toc146456826)

[Вывод 64](#_Toc146456827)

[Практическая работа № 6. Интерфейсы в Java 65](#_Toc146456828)

[Введение 66](#_Toc146456829)

[Практическая часть 67](#_Toc146456830)

[Вывод 69](#_Toc146456831)

[Практическая работа № 7. Реализация интерфейсов 70](#_Toc146456832)

[Введение 71](#_Toc146456833)

[Практическая часть 72](#_Toc146456834)

[Вывод 73](#_Toc146456835)

[Практическая работа № 8. Рекурсия Программирование рекурсии в Java. Решение задач на рекурсию 74](#_Toc146456836)

[Введение 75](#_Toc146456837)

[Практическая часть 76](#_Toc146456838)

[Вывод 78](#_Toc146456839)

[Практическая работа № 9. Использование полиморфизма при программировании при реализации алгоритмов сортировок и поиска 79](#_Toc146456840)

[Введение 80](#_Toc146456841)

[Практическая часть 81](#_Toc146456842)

[Вывод 84](#_Toc146456843)

[Практическая работа № 10. Стандартные интерфейсы Джава. Интерфейс Comparator 85](#_Toc146456844)

[Введение 87](#_Toc146456845)

[Практическая часть 88](#_Toc146456846)

[Вывод 92](#_Toc146456847)

[Практическая работа № 11. Работа с датой и временем. 93](#_Toc146456848)

[Введение 95](#_Toc146456849)

[Практическая часть 96](#_Toc146456850)

[Вывод 102](#_Toc146456851)

[Практическая работа № 12. Создание программ с графическим интерфейсом пользователя на языке Джава. Компоновка объектов с помощью Layout менеджеров. 103](#_Toc146456852)

[Введение 104](#_Toc146456853)

[Практическая часть 105](#_Toc146456854)

[Вывод 109](#_Toc146456855)

[Практическая работа № 13. Обработка строк в Java 110](#_Toc146456856)

[Введение 111](#_Toc146456857)

[Практическая часть 112](#_Toc146456858)

[Вывод 116](#_Toc146456859)

[Практическая работа № 14. Использование регулярных выражений в Джава приложениях 117](#_Toc146456860)

[Введение 119](#_Toc146456861)

[Практическая часть 120](#_Toc146456862)

[Вывод 124](#_Toc146456863)

[Практическая работа № 15. Вложенные и внутренние классы. Обработка событий в Джава программах с графическим интерфейсом пользователя 125](#_Toc146456864)

[Введение 126](#_Toc146456865)

[Практическая часть 127](#_Toc146456866)

[Вывод 131](#_Toc146456867)

[Практическая работа № 16. Обработка событий мыши и клавиатуры программах на Джава с графическим интерфейсом пользователя 132](#_Toc146456868)

[Введение 133](#_Toc146456869)

[Практическая часть 134](#_Toc146456870)

[Вывод 138](#_Toc146456871)

[Практическая работа № 17. Разработка интерактивных программ на языке Джава с использованием паттерна MVC 139](#_Toc146456872)

[Введение 140](#_Toc146456873)

[Практическая часть 141](#_Toc146456874)

[Вывод 146](#_Toc146456875)

[Практическая работа № 18. Исключения и работа с ними в Джава 147](#_Toc146456876)

[Введение 150](#_Toc146456877)

[Практическая часть 151](#_Toc146456878)

[Вывод 154](#_Toc146456879)

[Практическая работа № 19. Создание пользовательских исключений 155](#_Toc146456880)

[Введение 156](#_Toc146456881)

[Практическая часть 157](#_Toc146456882)

[Вывод 158](#_Toc146456883)

[Практическая работа № 20. Работа с дженериками. 159](#_Toc146456884)

[Введение 160](#_Toc146456885)

[Практическая часть 161](#_Toc146456886)

[Вывод 164](#_Toc146456887)

[Практическая работа № 21. Стирание типов в Джава и ваооывалоывало 165](#_Toc146456888)

[Введение 166](#_Toc146456889)

[Практическая часть 167](#_Toc146456890)

[Вывод 170](#_Toc146456891)

[Практическая работа № 22. Абстрактные типы данных. Стеки ваооывалоывало 171](#_Toc146456892)

[Введение 172](#_Toc146456893)

[Практическая часть 173](#_Toc146456894)

[Вывод 179](#_Toc146456895)

[Практическая работа № 23. Абстрактные типы данных. Очередь. 180](#_Toc146456896)

[Введение 182](#_Toc146456897)

[Практическая часть 183](#_Toc146456898)

[Вывод 190](#_Toc146456899)

[Практическая работа № 24. Паттерны проектирования. порождающие паттерны: абстрактная фабрика, фабричный метод 191](#_Toc146456900)

[Введение 192](#_Toc146456901)

[Практическая часть 193](#_Toc146456902)

[Вывод 195](#_Toc146456903)

# Практическая работа № 1. Знакомство со средой разработки. Синтаксис и основные управляющие конструкции языка Джава

**Цель:** введение в разработку программ на языке программирования

Джава.

**Задания на практическую работу № 1**

1. Создать проект в IntelliJ IDEA

2. Создать свой собственный Git репозитарий

3. Написать программу, в результате которой массив чисел создается с

помощью инициализации (как в Си) вводится и считается в цикле сумма

элементов целочисленного массива, а также среднее арифметическое его

элементов результат выводится на экран. Использовать цикл for.

4. Написать программу, в результате которой массив чисел вводится

пользователем с клавиатуры считается сумма элементов целочисленного

массива с помощью циклов do while, while, также необходимо найти

максимальный и минимальный элемент в массиве, результат выводится на

экран.

5. Написать программу, в результате которой выводятся на экран

аргументы командной строки в цикле for.

6. Написать программу, в результате работы которой выводятся на экран

первые 10 чисел гармонического ряда (форматировать вывод).

7. Написать программу, которая с помощью метода класса, вычисляет

факториал числа (использовать управляющую конструкцию цикла), проверить

работу метода.

8. Результаты выполнения практической работы залить через IDE в свой

репозитарий и продемонстрировать преподавателю.

## Введение

Среда разработки Java и понимание основных синтаксических и управляющих конструкций этого языка играют ключевую роль в разработке приложений на платформе Java. Практическая работа по данной теме представляет собой важное начало для тех, кто желает овладеть этим мощным языком программирования.

В этом отчете мы погрузимся в мир Java, начиная с основных концепций среды разработки и пошагово продвигаясь в изучении синтаксиса и основных управляющих конструкций языка. Мы ознакомимся с инструментами, необходимыми для создания Java-программ, а также углубимся в практические примеры, чтобы лучше понять, как эти концепции применяются на практике.

## Практическая часть



Листинг кода к заданию №3



Листинг кода к заданию №4 (1/2)



Листинг кода к заданию №4 (2/2)



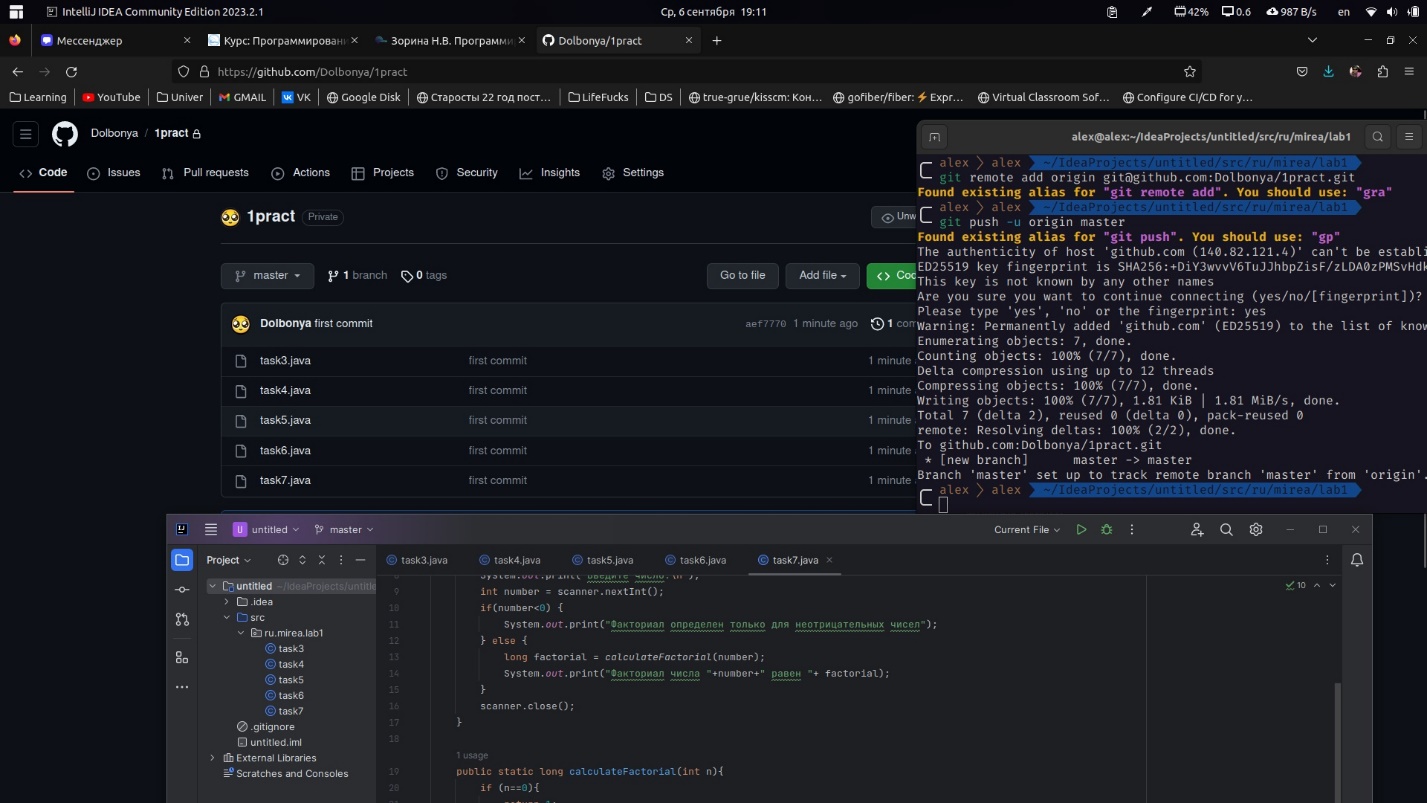
Листинг кода к заданию №5



Листинг кода к заданию №6



Листинг кода к заданию №7



Выполненное задание №8

## Вывод

В рамках данной практической работы мы изучили и попрактиковались в использовании различных ключевых аспектов языка программирования Java. Вот основные моменты, которые были рассмотрены:

1. **Методы**: Мы узнали, как создавать и использовать методы в Java. Методы позволяют упростить код, разделяя его на более мелкие и логически связанные части.
2. **Потоки ввода/вывода**: Мы изучили работу с классами Scanner и System.out для ввода данных с клавиатуры и вывода на экран. Эти классы позволяют взаимодействовать с пользователем и отображать результаты выполнения программы.
3. **Цикл for**: Мы освоили цикл for, который позволяет выполнять определенный набор инструкций заданное количество раз. Этот цикл особенно полезен при обработке массивов и коллекций.
4. **Цикл while**: Мы изучили цикл while, который выполняет набор инструкций до тех пор, пока условие истинно. Этот цикл полезен в случаях, когда количество итераций заранее неизвестно.
5. **Цикл do-while**: Мы также познакомились с циклом do-while, который подобен циклу while, но гарантирует выполнение хотя бы одной итерации перед проверкой условия.
6. **Условные конструкции**: Мы использовали условные операторы if, else if и else, чтобы выполнять различные части кода в зависимости от условий. Это позволяет создавать гибкие программы, которые принимают решения на основе данных.

В результате выполнения этой практической работы, мы разработали программы, включающие в себя все вышеупомянутые аспекты Java. Эти знания и навыки будут полезными при разработке более сложных приложений на языке Java и при работе с данными пользователя.

# Практическая работа № 2. Объектно-ориентированное программирование в Джава. Классы в Джава

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

**Задания на практическую работу № 2:**

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 1.

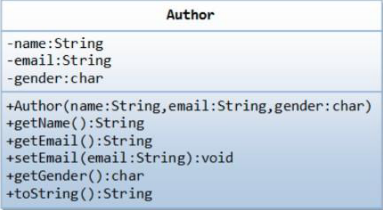


Рисунок 1 - Диаграмма класса Author.

2. Разработайте и реализуйте класс Dog (Собака), поля класса

описывают кличку и возраст собаки. Необходимо выполнить следующие

действия: определить конструктор собаки, чтобы принять и инициализировать данные экземпляра., включить стандартные методы (аксессоры) для получения и установки для имени и возраста, включить метод для перевода возраста собаки в “человеческий” возраст (возраст семь раз собаки), включите метод ToString, который возвращает описание экземпляра собаки в виде строки. Создание класса тестера под названием ПитомникСобак, реализует массив собак и основной метод этого класса позволяет добавить в него несколько объектов собаки.

3. Создать класс, описывающий модель окружности (Circle). В классе должны быть описаны нужные свойства окружности и методы для получения и изменения этих свойств. Добавить методы для расчета площади круга и длины окружности, а также метод позволяющий сравнивать две окружности. При помощи класса CircleTest, содержащего статический метод main(String[] args), протестировать работу класcа Circle.

## Введение

Объектно-ориентированное программирование (ООП) является одним из фундаментальных подходов к разработке программного обеспечения. В рамках данной практической работы мы глубоко погрузимся в мир объектно-ориентированного программирования, освоим ключевые концепции и принципы, а также изучим, как их применять на практике с использованием языка программирования Java.

Тема данной практической работы - "Объектно-ориентированное программирование в Java. Классы в Java." В ходе работы мы будем рассматривать основы создания классов, объектов и методов в Java, а также углубимся в понимание ключевых концепций ООП.

Понимание и применение ООП в Java играет критически важную роль в разработке программного обеспечения. Этот подход позволяет создавать более структурированный и модульный код, облегчает поддержку и расширение приложений, а также способствует повторному использованию кода.

## Практическая часть



Листинг файла Author.java к заданию №1



Листинг файла TestAuthor.java к заданию №1



Листинг файла Dog.java к заданию №2

package ru.mirea.lab2;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class DogKennel {  
 private List<Dog> dogs; // Список собак в питомнике  
  
 // Конструктор класса DogKennel для инициализации списка собак  
 public DogKennel() {  
 dogs = new ArrayList<>();  
 }  
  
  
 // Метод для добавления собаки в питомник  
 public void addDog(Dog *dog*) {  
 dogs.add(*dog*);  
 }  
  
 public List<Dog> getDogs() {  
 return dogs;  
 }  
  
 public static void main(String[] *args*) {  
 DogKennel kennel = new DogKennel();  
  
 // Создание нескольких собак и добавление их в питомник  
 Dog dog1 = new Dog("Buddy", 3);  
 Dog dog2 = new Dog("Max", 5);  
 Dog dog3 = new Dog("Charlie", 2);  
  
 kennel.addDog(dog1);  
 kennel.addDog(dog2);  
 kennel.addDog(dog3);  
  
 // Вывод информации о собаках в питомнике  
 List<Dog> kennelDogs = kennel.getDogs();  
 for (Dog dog : kennelDogs) {  
 System.out.println("Dog Name: " + dog.getName());  
 System.out.println("Dog Age: " + dog.getAge() + " years");  
 System.out.println("Human Age: " + dog.calculateHumanAge() + " years");  
 System.out.println(dog.toString());  
 }  
 }  
}

Листинг файла DogKennel.java к заданию №2



Листинг файла Circle.java к заданию №3 (1/2)



Листинг файла Circle.java к заданию №3 (2/2)



Листинг файла CircleTest.java к заданию №3

 Листинг файла CircleTest2.java к заданию №3

## Вывод

Практическая работа "Объектно-ориентированное программирование в Java: Классы в Java" важная часть обучения программированию на языке Java. В этой работе мы изучаем основы объектно-ориентированного программирования (ООП) и создаем собственные классы для представления объектов и их поведения. Вот ключевые аспекты, которые обычно рассматриваются в этой практической работе:

1. **Классы и объекты**: Мы изучаем, что такое классы и объекты, и как они используются в Java. Классы представляют собой шаблоны для создания объектов, а объекты - конкретные экземпляры этих классов.
2. **Поля и методы**: Мы определяем поля класса для хранения данных и методы для выполнения операций с этими данными. Эти методы могут быть геттерами и сеттерами для доступа к полям, а также другими методами для выполнения различных действий.
3. **Конструкторы**: Мы создаем конструкторы класса для инициализации объектов. Конструкторы позволяют устанавливать начальные значения полей при создании объекта.
4. **Переопределение методов**: Мы узнаем, как переопределять методы в подклассе, чтобы изменить их поведение. Для этого используется аннотация @Override.
5. **Тестирование классов**: Мы создаем тестовые классы и методы для проверки функциональности классов, используя библиотеку JUnit или другие средства тестирования.

После завершения этой практической работы, студенты обычно приобретают понимание основных концепций ООП и умение создавать собственные классы, используя их для решения различных задач и моделирования объектов в программировании на Java. Эти навыки являются фундаментальными для разработки сложных приложений на этом языке.

# Практическая работа № 3. Классы Math и Random. Классы оболочки

**Цель:** изучить работу с классами Math и Random основные концепции объектно-ориентированного программирования, научиться программировать математические вычисления с использованием этих классов, а также познакомиться с классами оболочками и их использованием в Джава программах и научиться форматировать вывод строк.

**Задания на практическую работу № 3:**

1. Создайте массив из 4 случайных целых чисел из отрезка [10;99], выведите его на экран в строку, далее определите и выведите на экран сообщение о том, является ли массив строго возрастающей последовательностью.\

2. Задание на форматирование строк вывода:

1. Создать класс конвертор валют.

3. Задание на классы Оболочки:

1. Создайте объекты класса Double, используя методы valueOf().
2. Преобразовать значение типа String к типу double. Используем метод Double.parseDouble().
3. Преобразовать объект класса Double ко всем примитивным типам.
4. Вывести значение объекта Double на консоль.
5. Преобразовать литерал типа double к строке: String d = Double.toString(3.14);

## Введение

В данной практической работе мы погрузимся в работу с основными классами Java, предназначенными для математических операций, генерации случайных чисел и работы с классами оболочками. Классы Math и Random предоставляют множество полезных методов для решения различных задач, связанных с математикой и генерацией случайных данных. Классы оболочки (Wrapper Classes) представляют удобные инструменты для работы с примитивными типами данных как объектами.

В ходе этой практической работы мы изучим базовые операции, доступные в классе Math, такие как вычисления степеней, извлечение корней, округление чисел и другие. Мы также познакомимся с классом Random, который позволяет генерировать случайные числа для различных целей.

Наконец, мы рассмотрим классы оболочки (Wrapper Classes), которые предоставляют методы и функционал для работы с примитивными типами данных как объектами, что может быть особенно полезным при работе с коллекциями и в других сценариях программирования.

Цель этой практической работы - освоить основные принципы использования классов Math, Random и классов оболочек в Java, а также научиться применять их для решения различных задач программирования.

## Практическая часть



Листинг файла task1 к заданию №1



Листинг файла CurrencyConverter к заданию №2 (1/2)



Листинг файла CurrencyConverter к заданию №2 (2/2)



Листинг файла task3 к заданию №3

## Вывод

В ходе практической работы № 3 по Java, посвященной классам Math и Random, а также классам оболочкам, были изучены основные методы и принципы работы с числовыми данными и случайными значениями.

Класс Math предоставляет множество статических методов для выполнения математических операций. Эти методы позволяют проводить различные вычисления, такие как вычисление квадратного корня, округление чисел, получение наибольшего и наименьшего значения, и др. Понимание и умение использовать методы класса Math позволяет эффективно работать с числовыми данными в Java.

Класс Random используется для генерации случайных чисел. Он предоставляет методы для генерации случайных чисел различных типов, таких как целочисленные и числа с плавающей запятой. Правильное применение класса Random позволяет создавать случайные значения, которые могут быть использованы в различных сценариях программы.

Классы-оболочки (wrapper classes) предоставляют объектное представление для примитивных типов данных. Они позволяют работать с примитивами как с объектами, что часто бывает полезно во многих ситуациях, например, при работе с коллекциями.

В результате выполнения практической работы, мы получили практические навыки по использованию классов Math и Random, а также освоили работу с классами-оболочками, что позволяет нам более эффективно и гибко оперировать числовыми данными в своих Java-программах.

# Практическая работа № 4. Перечисления и их использование в Джава программах.

**Цель:** познакомиться с новым ссылочным типом данных перечислением, научиться разрабатывать перечисления и использовать их в своих программах.

**Задания на практическую работу № 4:**

**Задание 1. Времена года**

Создать перечисление, содержащее названия времен года.

1) Создать переменную, содержащую ваше любимое время года и распечатать всю информацию о нем.

2) Создать метод, который принимает на вход переменную созданного вами enum типа. Если значение равно Лето, выводим на консоль “Я люблю лето” и так далее. Используем оператор switch.

3) Перечисление должно содержать переменную, содержащую среднюю температуру в каждом времени года.

4) Добавить конструктор, принимающий на вход среднюю температуру.

5) Создать метод getDescription, возвращающий строку “Холодное время года”. Переопределить метод getDescription - для константы Лето метод должен возвращать “Теплое время года”.

6) В цикле распечатать все времена года, среднюю температуру и описание времени года.

**Задание 2. Ателье**

Создать перечисление, содержащее размеры одежды (XXS, XS, S, M, L).

Перечисление содержит метод getDescription, возвращающий строку “Взрослый размер”. Переопределить метод getDescription - для константы XXS метод должен возвращать строку “Детский размер”. Также перечисление должно содержать числовое значение euroSize(32, 34, 36, 38, 40), соответствующее каждому размеру. Создать конструктор, принимающий на вход euroSize.

1) Создать интерфейсы MenClothing (мужская одежда) с методом dressMan() (одеть мужчину) и WomenClothing (женская одежда) с методом dressWomen() (одеть женщину).

2) Создать абстрактный класс Clothes (одежда), содержащий в качестве переменных класса - размер одежды, стоимость, цвет.

3) Создать классы наследники класса Clothes – класс TShirt (футболка) (реализует интерфейсы MenClothing и WomenClothing), класс Pants (штаны) (реализует интерфейсы MenClothing и WomenClothing ), класс Skirt (реализует интерфейсы WomenClothing), класс Tie (галстук ) (реализует интерфейсы MenClothing).

4) Создать массив, содержащий все типы одежды. Создать класс Atelier (Ателье), содержащий методы dressWomen, dressMan, на вход которых будет поступать массив, содержащий все типы одежды ( подумайте какой тип будет у массива). Переопределите метод dressWomen() для вывода на консоль всей информации о женской одежде. То же самое сделайте для метода dressMan().

**Задание 3. Интернет-магазин**

Создать мини приложение - интернет-магазин. Должны быть реализованы следующие возможности:

1) Аутентификация пользователя. Пользователь вводит логин и пароль с клавиатуры.

2) Просмотр списка каталогов товаров.

3) Просмотр списка товаров определенного каталога.

4) Выбор товара в корзину.

5) Покупка товаров, находящихся в корзине.

Для выполнения заданий необходимо создать перечисление согласно заданию, можете добавить свои операции или изменить что-то по своему усмотрению.

**Задание 4**

Создать класс, описывающий сущность компьютер (Computer). Для описания составных частей компьютера использовать отдельные классы (Processor, Memory, Monitor). Описать необходимые свойства и методы для каждого класса. Для названий марок компьютера используйте перечисления (enum)

## Введение

Перечисления (enum) представляют собой особый тип данных в Java, который позволяет определить набор констант с явными значениями. Этот механизм упрощает работу с ограниченным набором значений, что делает код более читаемым и поддерживаемым. В данной практической работе мы рассмотрим основы создания и использования перечислений в Java, а также узнаем, как они могут быть применены в реальных программных задачах.

## Практическая часть



Листинг файла Season к заданию №1





Листинг файла Atelier к заданию №2



Листинг файла Clothes к заданию №2

package ru.mirea.lab4.task2;  
  
// Пример использования  
public class Main {  
 public static void main(String[] *args*) {  
 // Создаем массивы с конкретными предметами одежды  
 MenClothing[] menClothes = {  
 new TShirt(Size.XS, 25.0, "Red"), // Мужская футболка, размер XS, стоимость 25.0, красная  
 new Pants(Size.M, 50.0, "Blue"), // Мужские штаны, размер M, стоимость 50.0, синие  
 new Tie(Size.L, 15.0, "Striped") // Галстук, размер L, стоимость 15.0, полосатый  
 };  
  
 WomenClothing[] womenClothes = {  
 new TShirt(Size.XS, 25.0, "Pink"), // Женская футболка, размер XS, стоимость 25.0, розовая  
 new Skirt(Size.S, 40.0, "Black") // Юбка, размер S, стоимость 40.0, черная  
 };  
  
 // Передаем массивы в методы ателье для одевания  
 Atelier.**dressMan**(menClothes); // Одеваем мужчин  
 Atelier.**dressWomen**(womenClothes); // Одеваем женщин  
 }  
}

Листинг файла Main к заданию №2

package ru.mirea.lab4.task2;  
  
interface MenClothing {  
 void dressMan();  
}

Листинг файла MenClothing к заданию №2

package ru.mirea.lab4.task2;  
  
import ru.mirea.lab4.task2.MenClothing;  
import ru.mirea.lab4.task2.Size;  
import ru.mirea.lab4.task2.WomenClothing;  
  
// Класс для представления штанов, который наследуется от абстрактного класса Clothes и реализует интерфейсы MenClothing и WomenClothing  
class Pants extends Clothes implements MenClothing, WomenClothing {  
  
 // Конструктор класса, вызывает конструктор суперкласса Clothes  
 public Pants(Size *size*, double *cost*, String *color*) {  
 super(*size*, *cost*, *color*);  
 }  
  
 // Реализация метода одевания мужчин  
 *@Override* public void dressMan() {  
 System.out.println("Мужчина одет в штаны");  
 }  
  
 // Реализация метода одевания женщин  
 *@Override* public void dressWomen() {  
 System.out.println("Женщина одета в штаны");  
 }  
  
 // Переопределение абстрактного метода для отображения информации о штанах  
 *@Override* public void displayInformation() {  
 System.out.println("Штаны: Размер - " + getSize() + ", Цвет - " + getColor() + ", Цена - " + getCost());  
 }  
}

Листинг файла Pants к заданию №2

package ru.mirea.lab4.task2;  
  
// Перечисление, представляющее размеры одежды  
public enum Size {  
 XXS(32), // Размер XXS с европейским размером 32  
 XS(34), // Размер XS с европейским размером 34  
 S(36), // Размер S с европейским размером 36  
 M(38), // Размер M с европейским размером 38  
 L(40); // Размер L с европейским размером 40  
  
 private int euroSize; // Европейский размер  
  
 // Конструктор перечисления с параметром (европейский размер)  
 Size(int *euroSize*) {  
 this.euroSize = *euroSize*;  
 }  
  
 // Метод для получения описания размера (детский или взрослый)  
 public String getDescription() {  
 return this == XXS ? "Детский размер" : "Взрослый размер";  
 }  
  
 // Метод для получения европейского размера  
 public int getEuroSize() {  
 return euroSize;  
 }  
}

Листинг файла Size к заданию №2

package ru.mirea.lab4.task2;  
  
import ru.mirea.lab4.task2.Size;  
  
// Класс для представления юбки, который наследуется от абстрактного класса Clothes и реализует интерфейс WomenClothing  
class Skirt extends Clothes implements WomenClothing {  
  
 // Конструктор класса, вызывает конструктор суперкласса Clothes  
 public Skirt(Size *size*, double *cost*, String *color*) {  
 super(*size*, *cost*, *color*);  
 }  
  
 // Реализация метода одевания женщин  
 *@Override* public void dressWomen() {  
 System.out.println("Женщина одета в юбку");  
 }  
  
 // Переопределение абстрактного метода для отображения информации о юбке  
 *@Override* public void displayInformation() {  
 System.out.println("Юбка: Размер - " + getSize() + ", Цвет - " + getColor() + ", Цена - " + getCost());  
 }  
}

Листинг файла Skirt к заданию №2

package ru.mirea.lab4.task2;  
  
import ru.mirea.lab4.task2.MenClothing;  
import ru.mirea.lab4.task2.Size;  
  
// Класс для представления галстука, который наследуется от абстрактного класса Clothes и реализует интерфейс MenClothing  
class Tie extends Clothes implements MenClothing {  
  
 // Конструктор класса, вызывает конструктор суперкласса Clothes  
 public Tie(Size *size*, double *cost*, String *color*) {  
 super(*size*, *cost*, *color*);  
 }  
  
 // Реализация метода одевания мужчин  
 *@Override* public void dressMan() {  
 System.out.println("Мужчина носит галстук");  
 }  
  
 // Переопределение абстрактного метода для отображения информации о галстуке  
 *@Override* public void displayInformation() {  
 System.out.println("Галстук: Размер - " + getSize() + ", Цвет - " + getColor() + ", Цена - " + getCost());  
 }  
}

Листинг файла Tie к заданию №2

package ru.mirea.lab4.task2;  
  
import ru.mirea.lab4.task2.MenClothing;  
import ru.mirea.lab4.task2.Size;  
import ru.mirea.lab4.task2.WomenClothing;  
  
// Класс для представления футболки, который наследуется от абстрактного класса Clothes и реализует интерфейсы MenClothing и WomenClothing  
class TShirt extends Clothes implements MenClothing, WomenClothing {  
  
 // Конструктор класса, вызывает конструктор суперкласса Clothes  
 public TShirt(Size *size*, double *cost*, String *color*) {  
 super(*size*, *cost*, *color*);  
 }  
  
 // Реализация метода одевания мужчин  
 *@Override* public void dressMan() {  
 System.out.println("Мужчина одет в футболку");  
 }  
  
 // Реализация метода одевания женщин  
 *@Override* public void dressWomen() {  
 System.out.println("Женщина одета в футболку");  
 }  
  
 // Переопределение абстрактного метода для отображения информации о футболке  
 *@Override* public void displayInformation() {  
 System.out.println("Футболка: Размер - " + getSize() + ", Цвет - " + getColor() + ", Цена - " + getCost());  
 }  
}

Листинг файла TShirt к заданию №2

package ru.mirea.lab4.task2;  
  
interface WomenClothing {  
 void dressWomen();  
}

Листинг файла WomenClothing к заданию №2

package ru.mirea.lab4.task3;  
  
enum Catalog {  
 ELECTRONICS,  
 CLOTHING,  
 BOOKS  
}

Листинг файла Catalog к заданию №3

package ru.mirea.lab4.task3;  
  
import java.util.\*;  
  
public class Main {  
 public static void main(String[] *args*) {  
 OnlineStore store = new OnlineStore(); // Создаем экземпляр интернет-магазина  
  
 Scanner scanner = new Scanner(System.in); // Создаем объект для ввода с клавиатуры  
  
 System.out.print("Введите имя пользователя: ");  
 String username = scanner.nextLine(); // Запрашиваем имя пользователя  
  
 System.out.print("Введите пароль: ");  
 String password = scanner.nextLine(); // Запрашиваем пароль  
  
 if (store.authenticate(username, password)) { // Проверяем аутентификацию пользователя  
 System.out.println("Авторизация успешна"); // Если успешно, выводим сообщение об успешной авторизации  
  
 while (true) { // Запускаем бесконечный цикл для обработки действий пользователя  
 System.out.println("Выберите действие:");  
 System.out.println("1. Посмотреть каталоги");  
 System.out.println("2. Выбрать каталог");  
 System.out.println("3. Посмотреть товары в каталоге");  
 System.out.println("4. Добавить товар в корзину");  
 System.out.println("5. Посмотреть корзину");  
 System.out.println("6. Купить товары");  
 System.out.println("0. Выйти");  
  
 int choice = scanner.nextInt(); // Получаем выбор пользователя  
 scanner.nextLine(); // Очищаем буфер  
  
 switch (choice) {  
 case 1:  
 store.viewCatalogs(); // Показываем каталоги товаров  
 break;  
 case 2:  
 System.out.print("Введите имя каталога: ");  
 String catalogName = scanner.nextLine(); // Запрашиваем у пользователя имя каталога  
 Catalog selectedCatalog = Catalog.**valueOf**(catalogName.toUpperCase()); // Преобразуем в enum  
 store.selectCatalog(selectedCatalog); // Выбираем каталог  
 break;  
 case 3:  
 store.viewProductsInSelectedCatalog(); // Показываем товары в выбранном каталоге  
 break;  
 case 4:  
 store.addToCartFromConsole(); // Добавляем товар в корзину  
 break;  
 case 5:  
 store.showShoppingCart(); // Показываем содержимое корзины  
 break;  
 case 6:  
 store.purchase(); // Покупаем товары  
 break;  
 case 0:  
 System.out.println("До свидания!"); // Выводим прощальное сообщение  
 return; // Завершаем выполнение программы  
 default:  
 System.out.println("Неверный выбор. Попробуйте снова."); // Сообщение о неверном выборе  
 }  
 }  
 } else {  
 System.out.println("Ошибка авторизации"); // Сообщение об ошибке авторизации  
 }  
 }  
}

Листинг файла Main к заданию №3

package ru.mirea.lab4.task3;  
  
import ru.mirea.lab4.task3.Product;  
import ru.mirea.lab4.task3.User;  
import ru.mirea.lab4.task3.Catalog;  
  
import java.util.\*;  
  
class OnlineStore {  
 private List<User> users; // Список пользователей  
 private Map<Catalog, List<Product>> products; // Словарь для хранения каталогов товаров  
 private List<Product> shoppingCart; // Список для хранения товаров в корзине  
 private Catalog selectedCatalog; // Выбранный каталог  
  
 // Конструктор для инициализации данных интернет-магазина  
 public OnlineStore() {  
 users = new ArrayList<>(); // Инициализация списка пользователей  
 products = new HashMap<>(); // Инициализация словаря с каталогами  
 shoppingCart = new ArrayList<>(); // Инициализация корзины  
  
 // Добавление пользователей и товаров  
 users.add(new User("user1", "password1"));  
 users.add(new User("user2", "password2"));  
  
 List<Product> electronics = new ArrayList<>();  
 electronics.add(new Product("Laptop", 1000.0));  
 electronics.add(new Product("Phone", 700.0));  
 products.put(Catalog.ELECTRONICS, electronics);  
  
 List<Product> clothing = new ArrayList<>();  
 clothing.add(new Product("T-Shirt", 20.0));  
 clothing.add(new Product("Jeans", 50.0));  
 products.put(Catalog.CLOTHING, clothing);  
  
 List<Product> books = new ArrayList<>();  
 books.add(new Product("Java Programming", 40.0));  
 books.add(new Product("History of Science", 30.0));  
 products.put(Catalog.BOOKS, books);  
 }  
  
 // Метод для аутентификации пользователя  
 public boolean authenticate(String *username*, String *password*) {  
 for (User user : users) {  
 if (user.getUsername().equals(*username*) && user.getPassword().equals(*password*)) {  
 return true; // Если пользователь найден, возвращаем true  
 }  
 }  
 return false; // Если пользователь не найден, возвращаем false  
 }  
  
 // Метод для просмотра списка каталогов  
 public void viewCatalogs() {  
 for (Catalog catalog : products.keySet()) {  
 System.out.println(catalog);  
 }  
 }  
  
 // Метод для выбора каталога  
 public void selectCatalog(Catalog *catalog*) {  
 if (products.containsKey(*catalog*)) {  
 selectedCatalog = *catalog*;  
 System.out.println("Выбранный каталог: " + *catalog*);  
 } else {  
 System.out.println("Каталог не найден");  
 }  
 }  
  
 // Метод для просмотра товаров в выбранном каталоге  
 public void viewProductsInSelectedCatalog() {  
 if (selectedCatalog != null) {  
 List<Product> productList = products.get(selectedCatalog);  
 for (Product product : productList) {  
 System.out.println(product);  
 }  
 } else {  
 System.out.println("Ни один каталог не выбран");  
 }  
 }  
  
 // Метод для добавления товара в корзину  
 public void addToCart(Product *product*) {  
 shoppingCart.add(*product*);  
 }  
  
 // Метод для совершения покупки  
 public void purchase() {  
 double total = 0.0;  
 for (Product product : shoppingCart) {  
 total += product.getPrice();  
 }  
 System.out.println("Общая стоимость вашей корзины: $" + total);  
 shoppingCart.clear(); // Очистка корзины после покупки  
 }  
  
 // Метод для отображения содержимого корзины  
 public void showShoppingCart() {  
 if (shoppingCart.isEmpty()) {  
 System.out.println("Корзина пуста");  
 } else {  
 System.out.println("Товары в корзине:");  
 for (Product product : shoppingCart) {  
 System.out.println(product);  
 }  
 }  
 }  
  
 // Метод для добавления товара в корзину через консольный ввод  
 public void addToCartFromConsole() {  
 Scanner scanner = new Scanner(System.in);  
  
 System.out.println("Выберите каталог и товар для добавления в корзину:");  
 viewCatalogs();  
  
 System.out.print("Введите имя каталога: ");  
 String catalogName = scanner.nextLine();  
  
 Catalog selectedCatalog = Catalog.**valueOf**(catalogName.toUpperCase());  
 selectCatalog(selectedCatalog);  
  
 viewProductsInSelectedCatalog();  
  
 System.out.print("Введите имя товара: ");  
 String productName = scanner.nextLine();  
  
 List<Product> productList = products.get(selectedCatalog);  
 boolean found = false;  
  
 for (Product product : productList) {  
 if (product.getName().equalsIgnoreCase(productName)) {  
 addToCart(product);  
 found = true;  
 break;  
 }  
 }  
  
 if (found) {  
 System.out.println("Товар добавлен в корзину");  
 } else {  
 System.out.println("Товар не найден");  
 }  
 }  
}

Листинг файла OnlineStore к заданию №3

package ru.mirea.lab4.task3;  
  
// Класс, представляющий продукт в интернет-магазине  
class Product {  
 private String name; // Наименование продукта  
 private double price; // Цена продукта  
  
 // Конструктор, инициализирующий имя и цену продукта  
 public Product(String *name*, double *price*) {  
 this.name = *name*;  
 this.price = *price*;  
 }  
  
 // Метод для получения имени продукта  
 public String getName() {  
 return name;  
 }  
  
 // Метод для получения цены продукта  
 public double getPrice() {  
 return price;  
 }  
  
 // Переопределение метода toString для красивого отображения продукта в виде строки  
 *@Override* public String toString() {  
 return name + " - $" + price;  
 }  
}

Листинг файла Product к заданию №3

package ru.mirea.lab4.task3;  
  
class User {  
 private String username;  
 private String password;  
  
 public User(String *username*, String *password*) {  
 this.username = *username*;  
 this.password = *password*;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
}

Листинг файла User к заданию №3

package ru.mirea.lab4.task4;  
  
enum Brand {  
 ASUS,  
 ACER,  
 DELL,  
 HP  
}

Листинг файла Brand к заданию №4

package ru.mirea.lab4.task4;  
  
// Класс, представляющий компьютер  
class Computer {  
 private Brand brand; // Марка компьютера  
 private Processor processor; // Процессор компьютера  
 private Memory memory; // Память компьютера  
 private Monitor monitor; // Монитор компьютера  
  
 // Конструктор для инициализации компонента компьютера  
 public Computer(Brand *brand*, Processor *processor*, Memory *memory*, Monitor *monitor*) {  
 this.brand = *brand*;  
 this.processor = *processor*;  
 this.memory = *memory*;  
 this.monitor = *monitor*;  
 }  
  
 // Метод для получения марки компьютера  
 public Brand getBrand() {  
 return brand;  
 }  
  
 // Метод для получения процессора компьютера  
 public Processor getProcessor() {  
 return processor;  
 }  
  
 // Метод для получения памяти компьютера  
 public Memory getMemory() {  
 return memory;  
 }  
  
 // Метод для получения монитора компьютера  
 public Monitor getMonitor() {  
 return monitor;  
 }  
}

Листинг файла Computer к заданию №4

package ru.mirea.lab4.task4;  
  
public class Main {  
 public static void main(String[] *args*) {  
 Processor intelProcessor = new Processor("Intel i7", 4);  
 Memory kingstonMemory = new Memory(8);  
 Monitor lgMonitor = new Monitor("LG UltraWide", 29);  
  
 Computer myComputer = new Computer(Brand.ASUS, intelProcessor, kingstonMemory, lgMonitor);  
  
 System.out.println("My Computer:");  
 System.out.println("Brand: " + myComputer.getBrand());  
 System.out.println("Processor: " + myComputer.getProcessor().getModel() + ", Cores: " + myComputer.getProcessor().getCores());  
 System.out.println("Memory: " + myComputer.getMemory().getSize() + "GB");  
 System.out.println("Monitor: " + myComputer.getMonitor().getModel() + ", Size: " + myComputer.getMonitor().getSizeInInches() + " inches");  
 }  
}

Листинг файла Main к заданию №4

package ru.mirea.lab4.task4;  
  
class Memory {  
 private int size; // в гигабайтах  
  
 public Memory(int *size*) {  
 this.size = *size*;  
 }  
  
 public int getSize() {  
 return size;  
 }  
}

Листинг файла Memory к заданию №4

package ru.mirea.lab4.task4;  
  
class Monitor {  
 private String model;  
 private int sizeInInches;  
  
 public Monitor(String *model*, int *sizeInInches*) {  
 this.model = *model*;  
 this.sizeInInches = *sizeInInches*;  
 }  
  
 public String getModel() {  
 return model;  
 }  
  
 public int getSizeInInches() {  
 return sizeInInches;  
 }  
}

Листинг файла Monitor к заданию №4

package ru.mirea.lab4.task4;  
  
class Processor {  
 private String model;  
 private int cores;  
  
 public Processor(String *model*, int *cores*) {  
 this.model = *model*;  
 this.cores = *cores*;  
 }  
  
 public String getModel() {  
 return model;  
 }  
  
 public int getCores() {  
 return cores;  
 }  
}

Листинг файла Processor к заданию №4

## Вывод

В данной практической работе мы изучили основы работы с перечислениями в Java и их практическое применение в программировании. Мы освоили создание перечислений, объявление своих методов и полей внутри них, а также использование перечислений в различных сценариях.

Перечисления предоставляют удобный способ организации данных, представляющих собой ограниченный набор значений. Они позволяют нам создавать более читаемый и понятный код, улучшая его структуру и обеспечивая безопасность типов.

Мы рассмотрели примеры применения перечислений в контексте реальных задач, таких как управление состояниями объектов и ограничение выбора определенных значений. Это дает нам возможность улучшать качество программ и сокращать количество ошибок, связанных с неверными значениями.

Таким образом, изучение перечислений в Java представляет собой важный элемент в освоении объектно-ориентированного программирования. Они являются мощным инструментом для создания чистого и структурированного кода, что облегчает разработку и поддержку программных продуктов.

# Практическая работа № 4.1. Наследование в Джава. Абстрактные классы.

**Цель:** познакомиться на практике с реализацией принципа ООП Наследование в Джава и освоить на практике работу с наследованием от абстрактных классов.

**Задание на практическую работу № 4.1:**

Создайте класс Phone, который содержит переменные number, model и weight.

1)Создайте три экземпляра этого класса.

2) Выведите на консоль значения их переменных.

3) Добавить в класс Phone методы: receiveCall, имеет один параметр – имя звонящего.

4)Выводит на консоль сообщение “Звонит {name}”.

5)Метод getNumber – возвращает номер телефона.

6) Вызвать эти методы для каждого из объектов. 7) Добавить конструктор в класс Phone, который принимает на вход три параметра для инициализации переменных класса - number, model и weight.

8)Добавить конструктор, который принимает на вход два параметра для инициализации переменных класса - number, model.

9)Добавить конструктор без параметров.

10)Вызвать из конструктора с тремя параметрами конструктор с двумя.

11)Добавьте перегруженный метод receiveCall, который принимает два параметра - имя звонящего и номер телефона звонящего.

12)Вызвать этот метод.

13)Создать метод sendMessage с аргументами переменной длины. Данный метод принимает на вход номера телефонов, которым будет отправлено сообщение.

14)Метод выводит на консоль номера этих телефонов.

## Введение

В этой практике мы будем изучать два важных концепта объектно-ориентированного программирования (ООП) в Java, которые позволяют создавать более гибкий и масштабируемый код.

**Наследование** — это механизм ООП, позволяющий одному классу (подклассу) наследовать свойства и методы другого класса (суперкласса). Это позволяет создавать иерархию классов, где подклассы могут расширять и изменять функциональность суперклассов.

**Абстрактные классы** — это особый тип классов, который не может быть инстанциирован непосредственно, а может содержать абстрактные методы (методы без реализации). Абстрактные классы предоставляют общий интерфейс для классов-потомков и определяют общие атрибуты и методы.

В ходе этой практической работы, мы будем:

1. Создавать иерархию классов с использованием наследования.
2. Исследовать принципы переопределения методов в подклассах.
3. Использовать абстрактные классы для создания общего интерфейса.

Эти концепции позволят нам создавать более эффективный и поддерживаемый код, особенно когда имеется несколько классов с общими характеристиками.

## Практическая часть

package ru.mirea.lab41;  
  
public class Phone {  
 private String number;  
 private String model;  
 private double weight;  
  
 // Конструктор с тремя параметрами  
 public Phone(String *number*, String *model*, double *weight*) {  
 this.number = *number*;  
 this.model = *model*;  
 this.weight = *weight*;  
 }  
  
 // Конструктор с двумя параметрами  
 public Phone(String *number*, String *model*) {  
 this(*number*, *model*, 0.0); // Вызываем конструктор с тремя параметрами  
 }  
  
 // Конструктор без параметров  
 public Phone() {  
 this("", "", 0.0); // Вызываем конструктор с тремя параметрами  
 }  
  
 // Метод receiveCall с одним параметром  
 public void receiveCall(String *callerName*) {  
 System.out.println("Звонит " + *callerName*);  
 }  
  
 // Перегруженный метод receiveCall с двумя параметрами  
 public void receiveCall(String *callerName*, String *callerNumber*) {  
 System.out.println("Звонит " + *callerName* + ", номер: " + *callerNumber*);  
 }  
  
 // Метод getNumber  
 public String getNumber() {  
 return number;  
 }  
  
 // Метод sendMessage с аргументами переменной длины  
 public void sendMessage(String... *numbers*) {  
 System.out.println("Отправить сообщение на следующие номера:");  
 for (String number : *numbers*) {  
 System.out.println(number);  
 }  
 }  
  
 public static void main(String[] *args*) {  
 // Создаем экземпляры класса Phone  
 Phone phone1 = new Phone("123456789", "Model1", 150);  
 Phone phone2 = new Phone("987654321", "Model2");  
 Phone phone3 = new Phone();  
  
 // Выводим значения переменных объектов  
 System.out.println("Phone 1: Number - " + phone1.getNumber() + ", Model - " + phone1.model + ", Weight - " + phone1.weight);  
 System.out.println("Phone 2: Number - " + phone2.getNumber() + ", Model - " + phone2.model + ", Weight - " + phone2.weight);  
 System.out.println("Phone 3: Number - " + phone3.getNumber() + ", Model - " + phone3.model + ", Weight - " + phone3.weight);  
  
 // Вызываем метод receiveCall для каждого объекта  
 phone1.receiveCall("John Doe");  
 phone2.receiveCall("Jane Doe");  
 phone3.receiveCall("Unknown Caller");  
  
 // Вызываем перегруженный метод receiveCall  
 phone1.receiveCall("James Smith", "555-1234");  
  
 // Вызываем метод sendMessage  
 phone1.sendMessage("555-1111", "555-2222", "555-3333");  
 }  
}

Листинг файла Phone к заданию №1

## Вывод

В ходе этой практической работы мы погрузились в мир наследования и абстрактных классов в Java. Эти мощные концепции ООП позволяют создавать гибкий и масштабируемый код.

Мы изучили:

1. Принципы наследования, который позволяет подклассам наследовать свойства и методы суперклассов, обеспечивая возможность создания иерархий классов.
2. Важность переопределения методов в подклассах для того, чтобы адаптировать функциональность суперклассов к уникальным требованиям подклассов.
3. Использование абстрактных классов, которые предоставляют общий интерфейс и реализуют общие атрибуты и методы для подклассов.

Эти знания помогут нам создавать более эффективный и поддерживаемый код в ваших Java-проектах. Наследование и абстрактные классы являются важной частью арсенала Java-разработчика, позволяя создавать программы, которые легко масштабировать и поддерживать в долгосрочной перспективе.

# Практическая работа № 5. Создание программ с графическим интерфейсом пользователя на языке Джава

**Цель:** научится разрабатывать программы на языке Джава с использованием графического интерфейса пользователя.

**Задание на практическую работу № 5:**

Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид.

Порядок работы:

1) Создайте пользовательское JFrame приложение, у которого есть следующие компоненты GUI:

* одна кнопка JButton подписана “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.

## Введение

Программы с графическим интерфейсом пользователя (GUI) предоставляют удобный способ взаимодействия пользователя с приложением. В этой практической работе по Java мы будем изучать создание GUI приложений с использованием библиотеки Swing.

Мы узнаем, как создавать окна, кнопки, текстовые поля, метки и другие компоненты, а также как обрабатывать события, возникающие при взаимодействии пользователя с приложением.

## Практическая часть

package ru.mirea.lab5;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class FootballMatchSimulator extends JFrame implements ActionListener {  
 // Переменные для хранения счета  
 private int milanScore = 0;  
 private int madridScore = 0;  
  
 // Компоненты GUI  
 private JButton milanButton;  
 private JButton madridButton;  
 private JLabel resultLabel;  
 private JLabel lastScorerLabel;  
 private JLabel winnerLabel;  
  
 // Конструктор класса  
 public FootballMatchSimulator() {  
 // Настройка фрейма  
 setTitle("Football Match Simulator");  
 setSize(300, 200);  
 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  
 // Создание компонентов  
 milanButton = new JButton("AC Milan");  
 madridButton = new JButton("Real Madrid");  
 resultLabel = new JLabel("Result: 0 X 0");  
 lastScorerLabel = new JLabel("Last Scorer: N/A");  
 winnerLabel = new JLabel("Winner: DRAW");  
  
 // Установка layout  
 setLayout(new GridLayout(5, 1));  
  
 // Добавление компонентов на панель  
 add(milanButton);  
 add(madridButton);  
 add(resultLabel);  
 add(lastScorerLabel);  
 add(winnerLabel);  
  
 // Добавление слушателей событий кнопок  
 milanButton.addActionListener(this);  
 madridButton.addActionListener(this);  
 }  
  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 // Обработка нажатий кнопок  
 if (*e*.getSource() == milanButton) {  
 milanScore++;  
 lastScorerLabel.setText("Last Scorer: AC Milan");  
 } else if (*e*.getSource() == madridButton) {  
 madridScore++;  
 lastScorerLabel.setText("Last Scorer: Real Madrid");  
 }  
  
 // Обновление отображаемого счета  
 resultLabel.setText("Result: " + milanScore + " X " + madridScore);  
  
 // Определение победителя  
 if (milanScore > madridScore) {  
 winnerLabel.setText("Winner: AC Milan");  
 } else if (madridScore > milanScore) {  
 winnerLabel.setText("Winner: Real Madrid");  
 } else {  
 winnerLabel.setText("Winner: DRAW");  
 }  
 }  
  
 public static void main(String[] *args*) {  
 // Запуск приложения  
 SwingUtilities.**invokeLater**(() -> {  
 FootballMatchSimulator simulator = new FootballMatchSimulator();  
 simulator.setVisible(true);  
 });  
 }  
}

Листинг файла FootballMatchSimulator к заданию №1

## Вывод

В ходе этой практической работы мы погрузились в мир создания программ с графическим интерфейсом на языке Java с использованием библиотеки Swing. Мы изучили основы создания окон, компонентов и их взаимодействие с пользователем.

Эти навыки открывают перед нами множество возможностей для разработки пользовательских приложений с удобным и интуитивно понятным интерфейсом. От простых калькуляторов до сложных приложений с множеством функциональных возможностей - Java Swing предоставляет все необходимое для создания разнообразных приложений с графическим интерфейсом.

Далее можно идти вперед и изучать более сложные компоненты Swing, а также применять полученные знания в собственных проектах для создания удобных и привлекательных пользовательских интерфейсов.

# Практическая работа № 6. Интерфейсы в Java

**Цель:** научится разрабатывать практике пользовательские интерфейсы, и применять их в программах на языке Джава.

**Задания на практическую работу №6:**

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.

## Введение

Интерфейсы представляют собой важный аспект объектно-ориентированного программирования в Java. Они позволяют определить контракт, который должны реализовать классы, что обеспечивает стандартизацию поведения в приложении.

В этой практической работе мы будем изучать создание интерфейсов, их назначение и применение в программах на языке Java. Мы также рассмотрим примеры использования интерфейсов для создания гибких и расширяемых систем.

## Практическая часть

package ru.mirea.lab6;  
  
// Интерфейс Nameable  
interface Nameable {  
 String getName(); // Метод возвращает имя объекта, реализующего интерфейс.  
}  
  
// Примеры классов, реализующих интерфейс Nameable  
class Planet implements Nameable {  
 private String name;  
  
 // Конструктор класса Planet  
 public Planet(String *name*) {  
 this.name = *name*;  
 }  
  
 *@Override* public String getName() {  
 return name; // Возвращает имя планеты  
 }  
}  
  
class Car implements Nameable {  
 private String brand;  
  
 // Конструктор класса Car  
 public Car(String *brand*) {  
 this.brand = *brand*;  
 }  
  
 *@Override* public String getName() {  
 return brand; // Возвращает марку автомобиля  
 }  
}  
  
class Animal implements Nameable {  
 private String species;  
  
 // Конструктор класса Animal  
 public Animal(String *species*) {  
 this.species = *species*;  
 }  
  
 *@Override* public String getName() {  
 return species; // Возвращает вид животного  
 }  
}  
  
public class Main {  
 public static void main(String[] *args*) {  
 // Создаем объекты классов, реализующих интерфейс Nameable  
 Nameable planet = new Planet("Earth");  
 Nameable car = new Car("Toyota");  
 Nameable animal = new Animal("Lion");  
  
 // Выводим наименования объектов  
 System.out.println("Planet name: " + planet.getName());  
 System.out.println("Car brand: " + car.getName());  
 System.out.println("Animal species: " + animal.getName());  
 }  
}

Листинг файла Main.java к заданию №1

package ru.mirea.lab6;  
  
// Интерфейс Priceable  
interface Priceable {  
 double getPrice(); // Метод, возвращающий цену объекта.  
}  
  
// Примеры классов, реализующих интерфейс Priceable  
class Product implements Priceable {  
 private double price;  
  
 // Конструктор класса Product  
 public Product(double *price*) {  
 this.price = *price*;  
 }  
  
 *@Override* public double getPrice() {  
 return price; // Возвращает цену продукта  
 }  
}  
  
class Service implements Priceable {  
 private double hourlyRate;  
 private int hours;  
  
 // Конструктор класса Service  
 public Service(double *hourlyRate*, int *hours*) {  
 this.hourlyRate = *hourlyRate*;  
 this.hours = *hours*;  
 }  
  
 *@Override* public double getPrice() {  
 return hourlyRate \* hours; // Возвращает стоимость услуги  
 }  
}  
  
public class Price {  
 public static void main(String[] *args*) {  
 // Создаем объекты классов, реализующих интерфейс Priceable  
 Priceable product = new Product(50.0);  
 Priceable service = new Service(20.0, 3);  
  
 // Выводим цены продукта и услуги  
 System.out.println("Product price: $" + product.getPrice());  
 System.out.println("Service price: $" + service.getPrice());  
 }  
}

Листинг файла Price.java к заданию №2

## Вывод

В ходе этой практической работы мы изучили важное понятие интерфейсов в Java. Интерфейсы предоставляют механизм стандартизации поведения объектов в приложении, что способствует гибкости и расширяемости кода.

Мы научились создавать интерфейсы, определять методы и их сигнатуры. Также мы познакомились с реализацией интерфейсов в классах, что позволяет объектам соответствовать определенному контракту.

Интерфейсы широко используются в реальных проектах для реализации абстракций и сокрытия деталей реализации. Они являются важным инструментом для создания чистого, гибкого и легко поддерживаемого кода.

С основами работы с интерфейсами в Java, теперь мы обладаем дополнительными знаниями, которые позволят нам создавать более гибкие и масштабируемые приложения.

# Практическая работа № 7. Реализация интерфейсов

**Цель:** научится разрабатывать практике пользовательские интерфейсы, и применять их в программах на языке Джава.

**Задание на практическую работу № 7:**

Разработайте интерфейс для работы со строками, который содержит:

а) функции подсчета символов в строке

б) функция возвращает строку, которая образовывает строку, состоящую из символов исходной строки s, которые размещены на нечетных позициях: 1, 3, 5, ...

в) функцию инвертирования строки

## Введение

Реализация интерфейсов является важным аспектом объектно-ориентированного программирования в Java. Интерфейсы предоставляют контракты, которые классы должны выполнить. Они позволяют нам определить набор методов, которые должны быть реализованы для обеспечения определенной функциональности.

В этой практической работе мы будем изучать, как создавать и реализовывать интерфейсы в Java. Мы также рассмотрим, как классы могут реализовать несколько интерфейсов одновременно, что позволяет им обладать различным поведением в зависимости от контрактов, которые они поддерживают.

## Практическая часть

package ru.mirea.lab7;  
  
// Интерфейс StringOperations  
interface StringOperations {  
 int countCharacters(String *s*); // Метод для подсчета символов в строке  
 String oddPositionChars(String *s*); // Метод для получения символов на нечетных позициях  
 String reverseString(String *s*); // Метод для инвертирования строки  
}  
  
// Класс, реализующий интерфейс StringOperations  
class StringProcessor implements StringOperations {  
  
 *@Override* public int countCharacters(String *s*) {  
 return *s*.length(); // Возвращает длину строки, то есть количество символов  
 }  
  
 *@Override* public String oddPositionChars(String *s*) {  
 StringBuilder result = new StringBuilder();  
  
 for (int i = 0; i < *s*.length(); i += 2) {  
 result.append(*s*.charAt(i)); // Добавляет символы на нечетных позициях в результирующую строку  
 }  
  
 return result.toString(); // Возвращает результат в виде строки  
 }  
  
 *@Override* public String reverseString(String *s*) {  
 return new StringBuilder(*s*).reverse().toString(); // Инвертирует строку и возвращает результат  
 }  
}  
  
public class Main {  
 public static void main(String[] *args*) {  
 StringOperations operations = new StringProcessor(); // Создаем объект класса StringProcessor, реализующего интерфейс StringOperations  
  
 String s = "Hello, World!";  
  
 // Подсчет символов  
 int charCount = operations.countCharacters(s);  
 System.out.println("Количество символов: " + charCount);  
  
 // Строка из символов на нечетных позициях  
 String oddChars = operations.oddPositionChars(s);  
 System.out.println("Символы на нечетных позициях: " + oddChars);  
  
 // Инвертированная строка  
 String reversed = operations.reverseString(s);  
 System.out.println("Инвертированная строка: " + reversed);  
 }  
}

Листинг файла Main.java к заданию №1

## Вывод

В ходе этой практической работы мы углубились в тему реализации интерфейсов в Java. Мы научились создавать интерфейсы, определять контракты методов и затем реализовывать эти интерфейсы в конкретных классах.

Одним из основных преимуществ использования интерфейсов является возможность реализации множества контрактов различных интерфейсов в одном классе. Это позволяет объектам обладать различными характеристиками и поведением, что способствует гибкости программного кода.

Теперь, имея понимание о реализации интерфейсов, мы готовы к созданию более гибких и масштабируемых приложений на Java. Интерфейсы являются мощным инструментом для определения общих стандартов и гарантирования соответствия контрактам в различных частях программы.

# Практическая работа № 8. Рекурсия Программирование рекурсии в Java. Решение задач на рекурсию

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

**Задания на практическую работу № 8:**

1. Задание 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.

2. Задание 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.

3. Задание 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.

## Введение

Рекурсия - это мощный и универсальный инструмент в программировании, позволяющий решать сложные задачи путем разбиения их на более простые подзадачи. В данной практической работе мы глубже погрузимся в мир рекурсивных алгоритмов, сфокусировавшись на их применении в языке программирования Java. Этот язык предоставляет нам богатый набор инструментов для работы с рекурсией, открывая перед нами возможности решения самых разнообразных задач.

Целью данной практической работы является ознакомление с основными концепциями рекурсивного программирования, а также развитие навыков применения этих знаний на практике. Мы рассмотрим различные примеры задач, подходы к их решению с использованием рекурсии и обсудим преимущества данного метода.

## Практическая часть

package ru.mirea.lab8; // Пакет, к которому принадлежит класс  
  
import java.util.Scanner; // Импорт класса Scanner из пакета java.util  
  
public class palindrome { // Объявление класса palindrome  
 public static void main(String[] *args*) { // Главный метод программы  
 Scanner scanner = new Scanner(System.in); // Создание объекта Scanner для чтения ввода пользователя  
 System.out.print("Введите слово: \n"); // Печать приглашения пользователю  
 String word = scanner.nextLine(); // Считывание введенного слова  
  
 if (**isPalindrome**(word)) { // Если введенное слово палиндром  
 System.out.println("YES"); // Печать "YES"  
 } else { // В противном случае  
 System.out.println("NO"); // Печать "NO"  
 }  
 }  
  
 public static boolean isPalindrome(String *word*) { // Метод для проверки, является ли слово палиндромом  
 if (*word*.length() <= 1) { // Если длина слова меньше или равна 1  
 return true; // Возвращаем true, так как односимвольное слово считается палиндромом  
 } else { // В противном случае  
 char firstChar = *word*.charAt(0); // Получаем первый символ слова  
 char lastChar = *word*.charAt(*word*.length() - 1); // Получаем последний символ слова  
 if (firstChar == lastChar) { // Если первый и последний символы совпадают  
 String subWord = *word*.substring(1, *word*.length() - 1); // Создаем подстроку без первого и последнего символов  
 return **isPalindrome**(subWord); // Рекурсивно вызываем isPalindrome для этой подстроки  
 } else { // Если первый и последний символы не совпадают  
 return false; // Возвращаем false, так как слово не является палиндромом  
 }  
 }  
 }  
}

Листинг файла palindrome.java к заданию №1

package ru.mirea.lab8; // Пакет, к которому принадлежит класс  
  
import java.util.Scanner; // Импорт класса Scanner из пакета java.util  
  
public class zero { // Объявление класса zero  
 public static void main(String[] *args*) { // Главный метод программы  
 Scanner scanner = new Scanner(System.in); // Создание объекта Scanner для чтения ввода пользователя  
  
 System.out.print("Введите количество нулей (a): "); // Печать приглашения пользователю для ввода количества нулей  
 int a = scanner.nextInt(); // Считывание количества нулей  
  
 System.out.print("Введите количество единиц (b): "); // Печать приглашения пользователю для ввода количества единиц  
 int b = scanner.nextInt(); // Считывание количества единиц  
  
 int[][] dp = new int[a+1][b+1]; // Инициализация двумерного массива dp  
  
 for (int i = 0; i <= a; i++) { // Заполнение базовых случаев, когда один из счетчиков равен 0  
 dp[i][0] = 1;  
 }  
 for (int j = 1; j <= b; j++) {  
 dp[0][j] = 1;  
 }  
  
 for (int i = 1; i <= a; i++) { // Заполнение dp таблицы используя рекуррентную формулу  
 for (int j = 1; j <= b; j++) {  
 dp[i][j] = dp[i-1][j-1] + dp[i][j-1];  
 }  
 }  
  
 System.out.println("Количество последовательностей: " + dp[a][b]); // Вывод результата  
 }  
}

Листинг файла zero.java к заданию №2

package ru.mirea.lab8;  
  
import java.util.Scanner;  
  
public class task3 {  
 public static void main(String[] *args*) {  
 Scanner scanner = new Scanner(System.in);  
  
 System.out.print("Введите число (десятичная запись без нулей): ");  
 int n = scanner.nextInt();  
  
 int reversed = **reverseNumber**(n);  
 System.out.println("Число с противоположным порядком цифр: " + reversed);  
 }  
  
 // Метод для переворачивания числа  
 public static int reverseNumber(int *n*) {  
 if (*n* < 10) { // Если число меньше 10, оно уже перевернуто  
 return *n*;  
 } else {  
 int lastDigit = *n* % 10; // Получаем последнюю цифру числа  
 int remainingDigits = *n* / 10; // Получаем все остальные цифры числа  
 int numberOfDigits = (int) Math.**log10**(remainingDigits) + 1; // Получаем количество цифр в оставшейся части числа  
 return lastDigit \* (int) Math.**pow**(10, numberOfDigits) + **reverseNumber**(remainingDigits); // Рекурсивно вызываем метод для оставшихся цифр  
 }  
 }  
}

Листинг файла task3.java к заданию №3

## Вывод

В ходе данной практической работы мы погрузились в мир рекурсивных алгоритмов и их применение в программировании на языке Java. Мы рассмотрели различные задачи, которые можно эффективно решать с использованием рекурсии.

Мы начали с базовых концепций рекурсии, разбирая простые примеры, такие как вычисление факториала и нахождение чисел Фибоначчи. Затем мы перешли к более сложным задачам, включая обходы деревьев и графов с использованием рекурсивных алгоритмов.

Важным моментом стало понимание того, когда применять рекурсию, а когда более прямолинейные подходы могут быть более эффективными. Рекурсия является мощным инструментом, но не всегда является наилучшим выбором в каждой ситуации.

# Практическая работа № 9. Использование полиморфизма при программировании при реализации алгоритмов сортировок и поиска

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

**Задания на практическую работу № 9:**

1. 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.

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

## Введение

Полиморфизм является одним из ключевых принципов объектно-ориентированного программирования (ООП), который позволяет использовать единый интерфейс для работы с различными типами данных. В рамках данной практической работы мы погрузимся в мир полиморфизма в Java, сосредотачиваясь на его применении при реализации алгоритмов сортировки и поиска.

Цель данной работы - продемонстрировать преимущества и гибкость полиморфизма в контексте разработки алгоритмов. Мы сосредоточимся на реализации различных видов сортировок и методов поиска, применяя полиморфизм для создания унифицированных интерфейсов, позволяющих работать с различными типами данных.

Эта практическая работа предоставит нам практический опыт использования полиморфизма в реальных алгоритмах, расширяя ваше понимание этого важного принципа ООП.

## Практическая часть

package ru.mirea.lab9;  
  
import java.util.Comparator;  
import java.util.Arrays;  
import java.util.List;  
  
// Класс, представляющий студента с итоговым баллом  
class Student {  
 private String name;  
 private double gpa;  
  
 public Student(String *name*, double *gpa*) {  
 this.name = *name*;  
 this.gpa = *gpa*;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getGpa() {  
 return gpa;  
 }  
}  
  
// Класс, реализующий Comparator для сортировки по итоговым баллам  
class SortingStudentsByGPA implements Comparator<Student> {  
  
 *@Override* public int compare(Student *student1*, Student *student2*) {  
 if (*student1*.getGpa() > *student2*.getGpa()) {  
 return -1; // Сортировка по убыванию  
 } else if (*student1*.getGpa() < *student2*.getGpa()) {  
 return 1;  
 } else {  
 return 0;  
 }  
 }  
}  
  
public class Main {  
 public static void main(String[] *args*) {  
 List<Student> students = Arrays.**asList**(  
 new Student("John Doe", 3.5),  
 new Student("Jane Doe", 4.0),  
 new Student("Bob Smith", 3.8)  
 );  
  
 // Сортировка с использованием SortingStudentsByGPA  
 students.sort(new SortingStudentsByGPA());  
  
 // Вывод отсортированного списка  
 for (Student student : students) {  
 System.out.println(student.getName() + ": " + student.getGpa());  
 }  
 }  
}

Листинг файла Main.java к заданию №1

package ru.mirea.lab9;  
  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
  
public class MergeSortExample {  
  
 public static void main(String[] *args*) {  
 // Создаем два списка данных о студентах  
 List<Student> list1 = new ArrayList<>();  
 list1.add(new Student("John", 3.8));  
 list1.add(new Student("Jane", 3.5));  
 list1.add(new Student("Bob", 4.0));  
  
 List<Student> list2 = new ArrayList<>();  
 list2.add(new Student("Alice", 3.9));  
 list2.add(new Student("Eve", 3.2));  
  
 // Объединяем и сортируем списки  
 List<Student> mergedList = **mergeSort**(list1, list2);  
  
 // Выводим результат  
 for (Student student : mergedList) {  
 System.out.println(student);  
 }  
 }  
  
 public static List<Student> mergeSort(List<Student> *list1*, List<Student> *list2*) {  
 List<Student> mergedList = new ArrayList<>();  
  
 int i = 0, j = 0;  
 while (i < *list1*.size() && j < *list2*.size()) {  
 if (*list1*.get(i).getGpa() > *list2*.get(j).getGpa()) {  
 mergedList.add(*list1*.get(i));  
 i++;  
 } else {  
 mergedList.add(*list2*.get(j));  
 j++;  
 }  
 }  
  
 while (i < *list1*.size()) {  
 mergedList.add(*list1*.get(i));  
 i++;  
 }  
  
 while (j < *list2*.size()) {  
 mergedList.add(*list2*.get(j));  
 j++;  
 }  
  
 // Используем компаратор для сортировки по убыванию  
 mergedList.sort(Comparator.**comparingDouble**(Student::getGpa).reversed());  
  
 return mergedList;  
 }  
  
 public static class Student {  
 private String name;  
 private double gpa;  
  
 public Student(String *name*, double *gpa*) {  
 this.name = *name*;  
 this.gpa = *gpa*;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getGpa() {  
 return gpa;  
 }  
  
 *@Override* public String toString() {  
 return "Student{" +  
 "name='" + name + '\'' +  
 ", gpa=" + gpa +  
 '}';  
 }  
 }  
}

Листинг файла MergeSortExample.java к заданию №2

## Вывод

В данной практической работе мы изучили и применили принципы использования полиморфизма при реализации алгоритмов сортировок и поиска в Java. Это позволяет создавать гибкие и читаемые программы, а также обеспечивает возможность замены алгоритмов без изменения основной логики приложения.

# Практическая работа № 10. Стандартные интерфейсы Джава. Интерфейс Comparator

**Цель:** закрепить знания в области использования стандартных интерфейсов языка Джава, научиться применять интерфейсы для разработки практических программ на Джаве.

**Задания на практическую работу № 10:**

Задание 1. (5%)

Создать свой класс Student со всеми переменными экземпляра, конструктором, включающим все переменные, предпочтительно использовать геттеры и сеттеры для каждой переменной. Класс студент имеет свойства: Имя, Фамилия, Специальность, Курс, Группа

Задание 2. (45%)

Напишите класс SortingStudentsByGPA (может у вас называться Tester или Main, так как содержит функцию main()) создайте поле как массив объектов Student с названием iDNumber, вы можете использовать как массив, так и и ArrayList или TreeSet для хранения данных о студентах

Добавьте методы класса:

1) заполнения массива setArray()

2) метод для сортировки по среднему баллу студентов quicksort() который реализует интерфейс Comparator таким образом, чтобы он сортировал студентов с их итоговым баллом в порядке убывания. В качестве алгоритма сортировки использовать методы сортировок: слиянием и быструю сортировку (добавьте в класс еще один метод).

3)метод для вывода массива студентов outArray()

4)Добавьте в класс возможность сортировать список студентов по другому полю

Задание 3. (50%)

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

## Введение

В мире программирования Java, интерфейсы предоставляют мощный инструмент для организации и управления кодом. Одним из наиболее важных стандартных интерфейсов является Comparator. Этот интерфейс позволяет нам определить способ сравнения объектов и, таким образом, управлять сортировкой данных в коллекциях.

Цель данной практической работы - глубже понять принципы работы интерфейса Comparator и научиться его применять в реальных проектах. В ходе выполнения заданий, вы познакомитесь с основами создания собственных компараторов, а также научитесь применять их для сортировки различных типов данных.

## Практическая часть

package ru.mirea.lab10;  
  
public class Student {  
 private String firstName;  
 private String lastName;  
 private String specialty;  
 private int course;  
 private String group;  
  
 private double gpa;  
  
 // Конструктор  
 public Student(String *firstName*, String *lastName*, String *specialty*, int *course*, String *group*,double *gpa*) {  
 this.firstName = *firstName*;  
 this.lastName = *lastName*;  
 this.specialty = *specialty*;  
 this.course = *course*;  
 this.group = *group*;  
 this.gpa=*gpa*;  
 }  
  
 // Геттеры  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public String getSpecialty() {  
 return specialty;  
 }  
  
 public int getCourse() {  
 return course;  
 }  
  
 public String getGroup() {  
 return group;  
 }  
  
 // Сеттеры  
 public void setFirstName(String *firstName*) {  
 this.firstName = *firstName*;  
 }  
  
 public void setLastName(String *lastName*) {  
 this.lastName = *lastName*;  
 }  
  
 public void setSpecialty(String *specialty*) {  
 this.specialty = *specialty*;  
 }  
  
 public void setCourse(int *course*) {  
 this.course = *course*;  
 }  
  
 public void setGroup(String *group*) {  
 this.group = *group*;  
 }  
  
  
 // Добавляем геттер и сеттер для GPA  
 public double getGPA() {  
 return gpa;  
 }  
  
 public void setGPA(double *gpa*) {  
 this.gpa = *gpa*;  
 }  
}

Листинг файла Student.java к заданиям №1,2,3

package ru.mirea.lab10;  
  
public class Main {  
 public static void main(String[] *args*) {  
 Student student = new Student("Иван", "Иванов", "Информатика", 2, "Группа-1",4.8);  
  
 // Получение значений с помощью геттеров  
 System.out.println("Имя: " + student.getFirstName());  
 System.out.println("Фамилия: " + student.getLastName());  
 System.out.println("Специальность: " + student.getSpecialty());  
 System.out.println("Курс: " + student.getCourse());  
 System.out.println("Группа: " + student.getGroup());  
  
 // Изменение значений с помощью сеттеров  
 student.setCourse(3);  
 student.setGroup("Группа-2");  
  
 // Печать измененных значений  
 System.out.println("Новый курс: " + student.getCourse());  
 System.out.println("Новая группа: " + student.getGroup());  
 }  
}

Листинг файла Main.java к заданиям №1,2,3

package ru.mirea.lab10;  
  
import java.util.Comparator;  
  
public class SortingStudentsByGPA {  
 private Student[] iDNumber;  
  
 // Конструктор, принимающий размер массива  
 public SortingStudentsByGPA(int *size*) {  
 iDNumber = new Student[*size*];  
 }  
  
 // Метод для заполнения массива  
 public void setArray(Student[] *students*) {  
 this.iDNumber = *students*;  
 }  
  
 // Метод для сортировки по среднему баллу с использованием быстрой сортировки  
 public void quicksort(Comparator<Student> *comparator*, int *low*, int *high*) {  
 if (*low* < *high*) {  
 int pi = partition(*comparator*, *low*, *high*);  
  
 quicksort(*comparator*, *low*, pi - 1);  
 quicksort(*comparator*, pi + 1, *high*);  
 }  
 }  
  
 private int partition(Comparator<Student> *comparator*, int *low*, int *high*) {  
 Student pivot = iDNumber[*high*];  
 int i = *low* - 1;  
  
 for (int j = *low*; j < *high*; j++) {  
 if (*comparator*.compare(iDNumber[j], pivot) > 0) {  
 i++;  
  
 Student temp = iDNumber[i];  
 iDNumber[i] = iDNumber[j];  
 iDNumber[j] = temp;  
 }  
 }  
  
 Student temp = iDNumber[i + 1];  
 iDNumber[i + 1] = iDNumber[*high*];  
 iDNumber[*high*] = temp;  
  
 return i + 1;  
 }  
  
 // Метод для вывода массива студентов  
 public void outArray() {  
 for (Student student : iDNumber) {  
 System.out.println(student.getFirstName() + " " + student.getLastName() + " { gpa: " + student.getGPA() +" , Курс: "+student.getCourse()+"}");  
 }  
 }  
  
 // Добавьте возможность сортировать список студентов по другому полю  
 public void sortByField(Comparator<Student> *comparator*) {  
 quicksort(*comparator*, 0, iDNumber.length - 1);  
 }  
  
 public static void main(String[] *args*) {  
 // Пример использования  
  
 // Создаем массив студентов  
 Student[] students = {  
 new Student("Иван", "Иванов", "Информатика", 2, "Группа-1", 4.5),  
 new Student("Петр", "Петров", "Математика", 3, "Группа-2", 4.8),  
 new Student("Мария", "Сидорова", "Физика", 1, "Группа-3", 4.2)  
 };  
  
 SortingStudentsByGPA sorter = new SortingStudentsByGPA(students.length);  
  
 // Заполняем массив студентами  
 sorter.setArray(students);  
  
 // Сортируем по среднему баллу  
 sorter.quicksort(Comparator.**comparing**(Student::getGPA).reversed(), 0, students.length - 1);  
  
 // Выводим отсортированный массив  
 System.out.println("Студенты, отсортированные по GPA:");  
 sorter.outArray();  
  
 // Сортируем по другому полю (например, по курсу)  
 sorter.sortByField(Comparator.**comparing**(Student::getCourse));  
  
 // Выводим отсортированный массив  
 System.out.println("\nСтуденты, отсортированные по курсу:");  
 sorter.outArray();  
 }  
}

Листинг файла SortingStudentByGPA.java к заданию №2

package ru.mirea.lab10;  
  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.List;  
  
public class MergeAndSortStudents {  
  
 public static void main(String[] *args*) {  
 List<Student> list1 = new ArrayList<>();  
 List<Student> list2 = new ArrayList<>();  
  
 // Добавляем студентов в первый список  
 list1.add(new Student("Иван", "Иванов", "Информатика", 2, "Группа-1", 4.5));  
 list1.add(new Student("Петр", "Петров", "Математика", 3, "Группа-2", 4.8));  
 list1.add(new Student("Мария", "Сидорова", "Физика", 1, "Группа-3", 4.2));  
  
 // Добавляем студентов во второй список  
 list2.add(new Student("Алексей", "Смирнов", "Информатика", 2, "Группа-1", 4.7));  
 list2.add(new Student("Екатерина", "Козлова", "Математика", 3, "Группа-2", 4.9));  
 list2.add(new Student("Анна", "Павлова", "Физика", 1, "Группа-3", 4.3));  
 // Объединяем списки  
 List<Student> mergedList = new ArrayList<>(list1);  
 mergedList.addAll(list2);  
  
 // Сортируем объединенный список по среднему баллу  
 Collections.**sort**(mergedList, Comparator.**comparing**(Student::getGPA).reversed());  
  
 // Выводим отсортированный список  
 for (Student student : mergedList) {  
 System.out.println(student.getFirstName() + " " + student.getLastName() + ": " + student.getGPA());  
 }  
 }  
}

Листинг файла MergeAndSortStudents.java к заданию №3

## Вывод

В данной практической работе мы изучили интерфейс Comparator и научились применять его для сортировки объектов в Java. Этот инструмент позволяет нам управлять порядком сортировки объектов в зависимости от наших потребностей.

# Практическая работа № 11. Работа с датой и временем.

**Цель:** научиться работать с датами и временем, применять методы класса Date и Calenadar, других классов для обработки строк.

**Задания на практическую работу № 11:**

**Задание 1.** (20%)

Написать программу, выводящую фамилию разработчика, дату и время получения задания, а также дату и время сдачи задания. Для получения последней даты и времени использовать класс Date из пакета java.util.\* (Объявление Dated=newDate() или метод System.currentTimeMillis().

**Задание 2.** (20%)

Приложение, сравнивающее текущую дату и дату, введенную пользователем c текущим системным временем

**Задание 3.** (20%)

Доработайте класс Student предусмотрите поле для хранения даты рождения, перепишите метод toString() таким образом, чтобы он разработайте метод, возвращал строковое представление даты рождения по вводимому в метод формату даты (например, короткий, средний и полный формат даты).

**Задание 4.** (10%)

Напишите пользовательский код, который формирует объекты Date и Calendar по следующим данным, вводимым пользователем: <Год><Месяц><Число> <Часы><минуты>

**Задание 5.** (30%)

Сравнить время выполнения кода в реализации кода в виде различных структур данных из предыдущих заданий (сравнить ArrayList и LinkedList по производительности – операции вставки, удаления, добавления и поиска по образцу)

## Введение

Работа с датой и временем - один из фундаментальных аспектов при разработке приложений, в которых необходимо управлять временными параметрами. Java предлагает мощные инструменты для работы с датами, что позволяет эффективно управлять временными интервалами, форматировать и анализировать даты, а также выполнять различные операции с временем.

Цель данной практической работы - ознакомиться с основными классами и методами, предоставляемыми Java для манипуляции датами и временем. Мы изучим классы Date, Calendar, SimpleDateFormat, а также более современные возможности, представленные в пакете java.time, включая LocalDate, LocalTime, и DateTimeFormatter.

## Практическая часть

package ru.mirea.lab11;  
  
import java.util.Date;  
import java.text.SimpleDateFormat;  
import java.util.Scanner;  
  
public class task1 {  
 public static class Main {  
 public static void main(String[] *args*) {  
 Scanner scanner = new Scanner(System.in);  
  
  
 //30-11-2022 13:40:20  
  
 // Фамилия разработчика  
 String surname = "Солобай";  
  
 // Получение текущей даты и времени  
 Date currentDate = new Date();  
  
 // Получение даты и времени задания от пользователя  
 System.out.print("Введите дату и время получения задания (в формате dd-MM-yyyy HH:mm:ss): ");  
 String receivedDateStr = scanner.nextLine();  
  
 // Преобразование строки в дату  
 SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
 Date taskReceivedDate = null;  
  
 try {  
 taskReceivedDate = dateFormat.parse(receivedDateStr);  
 } catch (Exception *e*) {  
 System.out.println("Ошибка при парсинге даты: " + *e*.getMessage());  
 return;  
 }  
  
 // Дата и время сдачи задания  
 System.out.print("Введите дату и время сдачи задания (в формате dd-MM-yyyy HH:mm:ss): ");  
 String submissionDateStr = scanner.nextLine();  
  
 Date taskSubmissionDate = null;  
  
 try {  
 taskSubmissionDate = dateFormat.parse(submissionDateStr);  
 } catch (Exception *e*) {  
 System.out.println("Ошибка при парсинге даты: " + *e*.getMessage());  
 return;  
 }  
  
 // Вывод информации  
 System.out.println("Фамилия разработчика: " + surname);  
 System.out.println("Дата и время получения задания: " + taskReceivedDate);  
 System.out.println("Дата и время сдачи задания: " + taskSubmissionDate);  
 }  
 }  
}

Листинг файла task1.java к заданию №1

package ru.mirea.lab11;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.Scanner;  
  
public class DateComparison {  
 public static void main(String[] *args*) {  
 // Получаем текущую дату и время  
 Date currentDate = new Date();  
  
 // Форматируем дату в строку  
 SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
 String currentDateString = dateFormat.format(currentDate);  
  
 System.out.println("Текущая дата и время: " + currentDateString);  
  
 // Получаем дату от пользователя  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введите дату в формате dd-MM-yyyy HH:mm:ss: ");  
 String userDateString = scanner.nextLine();  
  
 // Преобразуем строку пользователя в дату  
 Date userDate = null;  
 try {  
 userDate = dateFormat.parse(userDateString);  
 } catch (Exception *e*) {  
 System.out.println("Ошибка при парсинге даты: " + *e*.getMessage());  
 return;  
 }  
  
 // Сравниваем даты  
 if (currentDate.compareTo(userDate) > 0) {  
 System.out.println("Текущая дата и время позже введенной пользователем.");  
 } else if (currentDate.compareTo(userDate) < 0) {  
 System.out.println("Текущая дата и время раньше введенной пользователем.");  
 } else {  
 System.out.println("Введенная пользователем дата и текущая дата совпадают.");  
 }  
 }  
}

Листинг файла DateComparison.java к заданию №2

package ru.mirea.lab11;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Student {  
 private String name;  
 private Date birthDate;  
  
 public Student(String *name*, Date *birthDate*) {  
 this.name = *name*;  
 this.birthDate = *birthDate*;  
 }  
  
 // Геттеры и сеттеры для полей  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String *name*) {  
 this.name = *name*;  
 }  
  
 public Date getBirthDate() {  
 return birthDate;  
 }  
  
 public void setBirthDate(Date *birthDate*) {  
 this.birthDate = *birthDate*;  
 }  
  
 // Метод для получения строки с датой рождения в указанном формате  
 public String getFormattedBirthDate(String *format*) {  
 SimpleDateFormat dateFormat = new SimpleDateFormat(*format*);  
 return dateFormat.format(birthDate);  
 }  
  
 *@Override* public String toString() {  
 return "Student{" +  
 "name='" + name + '\'' +  
 ", birthDate=" + birthDate +  
 '}';  
 }  
}

Листинг файла Student.java к заданию №3

package ru.mirea.lab11;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] *args*) {  
 Scanner scanner = new Scanner(System.in);  
  
 System.out.print("Введите дату рождения (в формате dd-MM-yyyy): ");  
 String birthDateStr = scanner.nextLine();  
  
 SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");  
 Date birthDate;  
  
 try {  
 birthDate = dateFormat.parse(birthDateStr);  
 } catch (Exception *e*) {  
 System.out.println("Ошибка при парсинге даты: " + *e*.getMessage());  
 return;  
 }  
  
 Student student = new Student("Иванов", birthDate);  
  
 System.out.println("Дата рождения в коротком формате: " + student.getFormattedBirthDate("dd-MM-yyyy"));  
 System.out.println("Дата рождения в среднем формате: " + student.getFormattedBirthDate("dd-MMM-yyyy"));  
 System.out.println("Дата рождения в полном формате: " + student.getFormattedBirthDate("dd-MM-yyyy HH:mm:ss"));  
 }  
}

Листинг файла Main.java к заданию №3

package ru.mirea.lab11;  
  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
import java.util.Date;  
import java.util.Scanner;  
  
public class task4 {  
 public static void main(String[] *args*) {  
 Scanner scanner = new Scanner(System.in);  
  
 // Запрашиваем у пользователя данные  
 System.out.println("Введите дату (ГГГГММДД):");  
 String dateInput = scanner.nextLine();  
 System.out.println("Введите время (ЧЧмм):");  
 String timeInput = scanner.nextLine();  
  
 // Форматы для парсинга даты и времени  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");  
 SimpleDateFormat timeFormat = new SimpleDateFormat("HHmm");  
  
 try {  
 // Парсим введенные данные  
 Date date = dateFormat.parse(dateInput);  
 Date time = timeFormat.parse(timeInput);  
  
 // Создаем объекты Calendar и устанавливаем дату и время  
 Calendar calendar = Calendar.**getInstance**();  
 calendar.setTime(date);  
 Calendar timeCalendar = Calendar.**getInstance**();  
 timeCalendar.setTime(time);  
  
 // Устанавливаем часы и минуты  
 calendar.set(Calendar.HOUR\_OF\_DAY, timeCalendar.get(Calendar.HOUR\_OF\_DAY));  
 calendar.set(Calendar.MINUTE, timeCalendar.get(Calendar.MINUTE));  
  
 // Выводим результаты  
 System.out.println("Объект Date: " + date);  
 System.out.println("Объект Calendar: " + calendar.getTime());  
 } catch (ParseException *e*) {  
 *e*.printStackTrace();  
 System.out.println("Ошибка парсинга даты или времени.");  
 }  
 }  
}

Листинг файла task4.java к заданию №4

package ru.mirea.lab11;  
  
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;  
  
public class task5 {  
 public static void main(String[] *args*) {  
 final int SIZE = 100000; // Размер коллекции для тестирования  
  
 // Тестирование операций с ArrayList  
 List<Integer> arrayList = new ArrayList<>();  
 long startTime = System.**nanoTime**();  
  
 // Вставка в конец  
 for (int i = 0; i < SIZE; i++) {  
 arrayList.add(i);  
 }  
  
 // Удаление элемента  
 arrayList.remove(SIZE / 2);  
  
 // Вставка в середину  
 arrayList.add(SIZE / 2, 999);  
  
 // Поиск элемента  
 int index = arrayList.indexOf(999);  
  
 long endTime = System.**nanoTime**();  
 System.out.println("ArrayList: " + (endTime - startTime) + " наносекунд");  
  
 // Тестирование операций с LinkedList  
 List<Integer> linkedList = new LinkedList<>();  
 startTime = System.**nanoTime**();  
  
 // Вставка в конец  
 for (int i = 0; i < SIZE; i++) {  
 linkedList.add(i);  
 }  
  
 // Удаление элемента  
 linkedList.remove(SIZE / 2);  
  
 // Вставка в середину  
 linkedList.add(SIZE / 2, 999);  
  
 // Поиск элемента  
 index = linkedList.indexOf(999);  
  
 endTime = System.**nanoTime**();  
 System.out.println("LinkedList: " + (endTime - startTime) + " наносекунд");  
 }  
}

Листинг файла task5.java к заданию №5

## Вывод

В ходе этой практической работы мы глубоко погрузились в мир управления датой и временем с использованием Java. Мы изучили основные классы и методы, предоставляемые этим мощным языком программирования для работы с датами и временем. Начиная с класса Date и Calendar и заканчивая более современными классами LocalDate, LocalTime и DateTimeFormatter из пакета java.time, мы приобрели уверенность в обработке временных данных.

Мы освоили навыки создания, форматирования, сравнения и выполнения арифметических операций с датами и временем. Эти навыки являются фундаментальными для разработчика, работающего над приложениями, где управление временем играет важную роль.

Необходимо отметить, что правильная работа с датой и временем является неотъемлемой частью разработки надежных и эффективных приложений. Понимание особенностей времени в Java позволяет избежать множества ошибок, связанных с временными зонами, форматированием и арифметикой дат.

# Практическая работа № 12. Создание программ с графическим интерфейсом пользователя на языке Джава. Компоновка объектов с помощью Layout менеджеров.

**Цель:** научиться создавать графический интерфейс пользователя, освоить на практике работу с различными объектами для создания GUI, менеджерами размещения компонентов.

**Задания на практическую работу № 12:**

1. Создать окно, нарисовать в нем 20 случайных фигур, случайного цвета. Классы фигур должны наследоваться от абстрактного класса Shape, в котором описаны свойства фигуры: цвет, позиция.

2. Создать окно, отобразить в нем картинку, путь к которой указан в аргументах командной строки.

3. Создать окно, реализовать анимацию, с помощью картинки, состоящей из нескольких кадров. Сравнить время выполнения кода в реализации кода в виде различных структур данных из предыдущих заданий (сравнить ArrayList и LinkedList по производительности – операции вставки, удаления, добавления и поиска по образцу)

## Введение

В современном программировании создание приложений с интуитивно понятным и привлекательным пользовательским интерфейсом является одним из ключевых аспектов разработки. Язык программирования Java, благодаря своей платформонезависимости и обширной библиотеке Swing, предоставляет разработчикам мощный инструментарий для создания графических пользовательских интерфейсов (GUI).

Цель данной практической работы заключается в ознакомлении с основами создания программ с графическим интерфейсом пользователя на платформе Java. Особое внимание будет уделено принципам компоновки объектов на форме с использованием Layout менеджеров. Правильно выбранный Layout менеджер позволяет создавать адаптивные и эстетичные пользовательские интерфейсы, которые могут корректно отображаться на различных типах устройств и разрешениях экранов.

В ходе этой практической работы мы рассмотрим основные типы Layout менеджеров, изучим их принципы работы, а также научимся применять их для размещения компонентов в окне приложения. Благодаря этим знаниям, мы сможем разрабатывать функциональные и эргономичные программы с графическим интерфейсом на языке Java, что является важным навыком для любого Java-разработчика.

## Практическая часть

package ru.mirea.lab12;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.util.Random;  
  
// Абстрактный класс фигуры  
abstract class Shape {  
 protected Color color; // Цвет фигуры  
 protected Point position; // Позиция фигуры  
  
 // Конструктор абстрактной фигуры  
 public Shape(Color *color*, Point *position*) {  
 this.color = *color*;  
 this.position = *position*;  
 }  
  
 abstract void draw(Graphics *g*); // Абстрактный метод отрисовки фигуры  
}  
  
// Пример конкретной фигуры (круг)  
class Circle extends Shape {  
 private int radius; // Радиус круга  
  
 // Конструктор круга  
 public Circle(Color *color*, Point *position*, int *radius*) {  
 super(*color*, *position*);  
 this.radius = *radius*;  
 }  
  
 *@Override* void draw(Graphics *g*) {  
 *g*.setColor(color);  
 *g*.fillOval(position.x - radius, position.y - radius, 2 \* radius, 2 \* radius);  
 }  
}  
  
// Пример конкретной фигуры (прямоугольник)  
class Rectangle extends Shape {  
 private int width; // Ширина прямоугольника  
 private int height; // Высота прямоугольника  
  
 // Конструктор прямоугольника  
 public Rectangle(Color *color*, Point *position*, int *width*, int *height*) {  
 super(*color*, *position*);  
 this.width = *width*;  
 this.height = *height*;  
 }  
  
 *@Override* void draw(Graphics *g*) {  
 *g*.setColor(color);  
 *g*.fillRect(position.x, position.y, width, height);  
 }  
}  
  
public class RandomShapes extends JPanel {  
 private final int NUM\_SHAPES = 20; // Количество случайных фигур  
 private final Random random = new Random(); // Генератор случайных чисел  
  
 private Shape[] shapes; // Массив для хранения фигур  
  
 public RandomShapes() {  
 shapes = new Shape[NUM\_SHAPES];  
 for (int i = 0; i < NUM\_SHAPES; i++) {  
 int x = random.nextInt(600); // Рандомные координаты в пределах панели  
 int y = random.nextInt(400);  
 int red = random.nextInt(256); // Рандомный красный компонент цвета  
 int green = random.nextInt(256); // Рандомный зеленый компонент цвета  
 int blue = random.nextInt(256); // Рандомный синий компонент цвета  
 Color color = new Color(red, green, blue); // Случайный цвет  
  
 if (i % 2 == 0) {  
 int radius = random.nextInt(50); // Рандомный радиус для круга  
 shapes[i] = new Circle(color, new Point(x, y), radius);  
 } else {  
 int width = random.nextInt(100); // Рандомные ширина и высота для прямоугольника  
 int height = random.nextInt(100);  
 shapes[i] = new Rectangle(color, new Point(x, y), width, height);  
 }  
 }  
 }  
  
 *@Override* protected void paintComponent(Graphics *g*) {  
 super.paintComponent(*g*);  
 for (Shape shape : shapes) {  
 shape.draw(*g*);  
 }  
 }  
  
 public static void main(String[] *args*) {  
 JFrame frame = new JFrame("Random Shapes"); // Создание главного окна  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 frame.setSize(800, 600); // Установка размеров окна  
 frame.add(new RandomShapes()); // Добавление панели с фигурами в окно  
 frame.setVisible(true); // Отображение окна  
 }  
}

Листинг файла RandomShapes.java к заданию №1

package ru.mirea.lab12;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
// Класс для отображения изображения  
public class DisplayImage extends JPanel {  
 private ImageIcon imageIcon; // Объект для хранения изображения  
  
 // Конструктор, принимающий путь к изображению  
 public DisplayImage(String *imagePath*) {  
 imageIcon = new ImageIcon(*imagePath*); // Загрузка изображения по указанному пути  
 }  
  
 *@Override* protected void paintComponent(Graphics *g*) {  
 super.paintComponent(*g*);  
 Image image = imageIcon.getImage(); // Получение объекта Image из ImageIcon  
 *g*.drawImage(image, 0, 0, this); // Отображение изображения в панели  
 }  
  
 public static void main(String[] *args*) {  
 if (*args*.length != 1) {  
 System.out.println("Пожалуйста, укажите путь к изображению как аргумент командной строки.");  
 return;  
 }  
  
 String imagePath = *args*[0]; // Получение пути к изображению из аргумента командной строки  
  
 JFrame frame = new JFrame("Отображение изображения"); // Создание нового окна  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Установка действия при закрытии окна  
 frame.setSize(800, 600); // Установка размеров окна  
  
 DisplayImage displayImage = new DisplayImage(imagePath); // Создание объекта для отображения изображения  
 frame.add(displayImage); // Добавление объекта в окно  
  
 frame.setVisible(true); // Отображение окна  
 }  
}

Листинг файла DisplayImage.java к заданию №3

package ru.mirea.lab12;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.File;  
import java.io.IOException;  
import javax.imageio.ImageIO;  
import java.awt.image.BufferedImage;  
  
// Класс для анимированного окна  
public class AnimatedWindow extends JPanel implements ActionListener {  
 private Timer timer;  
 private int frameIndex;  
 private BufferedImage[] frames;  
  
 // Конструктор, принимающий путь к папке с изображениями  
 public AnimatedWindow(String *folderPath*) {  
 frames = new BufferedImage[3];  
 for (int i = 0; i < 3; i++) {  
 try {  
 frames[i] = ImageIO.**read**(new File(*folderPath* + "\\frame" + i + ".png"));  
 } catch (IOException *e*) {  
 *e*.printStackTrace();  
 }  
 }  
  
 frameIndex = 0;  
  
 // Создание таймера с интервалом 1000 миллисекунд (1 кадр в секунду)  
 timer = new Timer(1000, this);  
 timer.start();  
 }  
  
 // Метод для отрисовки компонента  
 *@Override* protected void paintComponent(Graphics *g*) {  
 super.paintComponent(*g*);  
 Image image = frames[frameIndex];  
 *g*.drawImage(image, 0, 0, this);  
 }  
  
 // Обработчик события таймера  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 frameIndex = (frameIndex + 1) % frames.length;  
 repaint();  
 }  
  
 // Метод main для запуска приложения  
 public static void main(String[] *args*) {  
 // Проверка наличия аргумента с путем к папке с изображениями  
 if (*args*.length < 1) {  
 System.out.println("Пожалуйста, укажите путь к папке с изображениями в аргументах командной строки.");  
 return;  
 }  
  
 String folderPath = *args*[0];  
  
 // Создание и настройка главного окна  
 JFrame frame = new JFrame("Анимированное окно");  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 frame.setSize(800, 600);  
  
 // Создание и добавление анимированного компонента в окно  
 AnimatedWindow animatedWindow = new AnimatedWindow(folderPath);  
 frame.add(animatedWindow);  
  
 // Отображение окна  
 frame.setVisible(true);  
 }  
}

Листинг файла AnimatedWindow.java к заданию №3

## Вывод

В ходе практической работы по созданию программ с графическим интерфейсом пользователя на языке Java и использованию Layout менеджеров были освоены важные аспекты разработки GUI-приложений. Эта работа предоставила ценный опыт и понимание того, как эффективно организовывать компоненты пользовательского интерфейса с помощью Layout менеджеров.

В целом, данная практическая работа по созданию программ с графическим интерфейсом пользователя на языке Java с использованием Layout менеджеров была очень полезной. Она дала понимание того, как создавать интерактивные и адаптивные приложения, которые могут быть легко масштабированы и поддерживаемы в будущем. Этот опыт будет полезен при разработке будущих GUI-проектов на языке Java.

# Практическая работа № 13. Обработка строк в Java

**Цель:** закрепить знания в области обработки строк, научиться применять методы класса String и других классов для обработки строк

**Задания на практическую работу № 13:**

1. Реализуйте класс Shirt: Метод toString() выводит объяснение и значение полей построчно. Дан также строковый массив: shirts[0] = "S001,Black Polo Shirt,Black,XL"; shirts[1] = "S002,Black Polo Shirt,Black,L"; shirts[2] = "S003,Blue Polo Shirt,Blue,XL"; shirts[3] = "S004,Blue Polo Shirt,Blue,M"; shirts[4] = "S005,Tan Polo Shirt,Tan,XL"; shirts[5] = "S006,Black T-Shirt,Black,XL"; shirts[6] = "S007,White T-Shirt,White,XL"; shirts[7] = "S008,White T-Shirt,White,L"; shirts[8] = "S009,Green T Shirt,Green,S"; shirts[9] = "S010,Orange T-Shirt,Orange,S"; shirts[10] = "S011,Maroon Polo Shirt,Maroon,S"; Преобразуйте строковый массив в массив класса Shirt и выведите его на консоль.

2. Разработайте класс, который получает строковое представление телефонного номера в одном из двух возможных строковых форматов: +<Код страны><Номер 10 цифр>, например “+79175655655” или “+104289652211” или 8<Номер 10 цифр> для России, например “89175655655” и преобразует полученную строку в формат:   
+<Код страны><Три цифры>–<Три цифры>–<Четыре цифры>

3. В методе main считай с консоли имя файла, который содержит слова, разделенные пробелом. В методе getLine() используя StringBuilder расставьте все слова в таком порядке, чтобы последняя буква данного слова совпадала с первой буквой следующего не учитывая регистр. Каждое слово должно участвовать 1 раз.

## Введение

В современном программировании обработка текстовых данных является неотъемлемой частью множества приложений. Независимо от области применения - будь то разработка веб-приложений, анализ данных, или создание игр - эффективная манипуляция строками является важным навыком для каждого разработчика.

Язык программирования Java предоставляет богатый набор инструментов для работы с текстовыми данными. В этой практической работе мы погрузимся в мир обработки строк в Java, изучая различные методы и функции, предоставляемые этим языком. Мы научимся основам работы с символами, конкатенации, поиску подстрок и многим другим важным аспектам обработки текста.

## Практическая часть

// Объявление пакета, в котором находится класс Shirt  
package ru.mirea.lab13;  
  
// Определение класса Shirt  
class Shirt {  
 // Приватные поля класса Shirt  
 private String code;  
 private String description;  
 private String color;  
 private String size;  
  
 // Конструктор класса Shirt, принимающий параметры code, description, color, size  
 public Shirt(String *code*, String *description*, String *color*, String *size*) {  
 this.code = *code*;  
 this.description = *description*;  
 this.color = *color*;  
 this.size = *size*;  
 }  
  
 // Переопределение метода toString()  
 *@Override* public String toString() {  
 return "Code: " + code + "\nDescription: " + description + "\nColor: " + color + "\nSize: " + size + "\n";  
 }  
}  
  
// Определение класса task1  
public class task1 {  
 // Главный метод программы  
 public static void main(String[] *args*) {  
 // Массив строк с данными о рубашках  
 String[] shirtsData = {  
 "S001,Black Polo Shirt,Black,XL", "S002,Black Polo Shirt,Black,L",  
 "S003,Blue Polo Shirt,Blue,XL", "S004,Blue Polo Shirt,Blue,M",  
 "S005,Tan Polo Shirt,Tan,XL", "S006,Black T-Shirt,Black,XL",  
 "S007,White T-Shirt,White,XL", "S008,White T-Shirt,White,L",  
 "S009,Green T Shirt,Green,S", "S010,Orange T-Shirt,Orange,S",  
 "S011,Maroon Polo Shirt,Maroon,S"  
 };  
  
 // Создание массива объектов класса Shirt  
 Shirt[] shirts = new Shirt[shirtsData.length];  
  
 // Заполнение массива объектами Shirt, созданными на основе данных из строк shirtsData  
 for (int i = 0; i < shirtsData.length; i++) {  
 String[] shirtInfo = shirtsData[i].split(",");  
 shirts[i] = new Shirt(shirtInfo[0], shirtInfo[1], shirtInfo[2], shirtInfo[3]);  
 }  
  
 // Вывод информации о каждой рубашке  
 for (Shirt shirt : shirts) {  
 System.out.println(shirt);  
 }  
 }  
}

Листинг файла task1.java к заданию №1

package ru.mirea.lab13;  
  
public class PhoneNumberConverter {  
 private String phoneNumber;  
  
 // Конструктор класса, принимающий строковое представление телефонного номера  
 public PhoneNumberConverter(String *phoneNumber*) {  
 this.phoneNumber = *phoneNumber*;  
 }  
  
 // Метод для преобразования номера в нужный формат  
 public String convertToFormattedString() {  
 String formattedNumber;  
  
 // Убираем все нецифровые символы из строки  
 String digitsOnly = phoneNumber.replaceAll("\\D", "");  
  
 // Проверяем, если номер начинается с 8, то добавляем код страны России  
 if (digitsOnly.startsWith("8")) {  
 digitsOnly = "+7" + digitsOnly.substring(1);  
 } else if (!digitsOnly.startsWith("+")) {  
 digitsOnly = "+" + digitsOnly;  
 }  
  
 // Форматируем номер  
 formattedNumber = digitsOnly.substring(0, 5) + "-" + digitsOnly.substring(5, 8) + "-" + digitsOnly.substring(8);  
  
 return formattedNumber;  
 }  
  
 public static void main(String[] *args*) {  
 // Примеры использования  
  
 // Создаем объект с номером "+79175655655" и выводим его в нужном формате  
 PhoneNumberConverter converter1 = new PhoneNumberConverter("+79175655655");  
 System.out.println(converter1.convertToFormattedString());  
  
 // Создаем объект с номером "+104289652211" и выводим его в нужном формате  
 PhoneNumberConverter converter2 = new PhoneNumberConverter("+104289652211");  
 System.out.println(converter2.convertToFormattedString());  
  
 // Создаем объект с номером "89175655655" (российский формат) и выводим его в нужном формате  
 PhoneNumberConverter converter3 = new PhoneNumberConverter("89175655655");  
 System.out.println(converter3.convertToFormattedString());  
 }  
}

Листинг файла PhoneNumberConverter.java к заданию №2

package ru.mirea.lab13; // Объявление пакета  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
  
public class Main {  
  
 public static void main(String[] *args*) throws Exception {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  
 System.out.print("Введите абсолютный путь к файлу: ");  
 String fileName = reader.readLine(); // Чтение абсолютного пути к файлу из консоли  
 reader.close(); // Закрытие потока чтения  
  
 ArrayList<String> wordsList = **readWordsFromFile**(fileName); // Чтение слов из файла и сохранение в ArrayList  
 String result = **getLine**(wordsList.toArray(new String[0])); // Формирование результата перестановки слов  
 System.out.println(result); // Вывод результата на экран  
 }  
  
 public static ArrayList<String> readWordsFromFile(String *fileName*) throws Exception {  
 ArrayList<String> wordsList = new ArrayList<>(); // Создание ArrayList для хранения слов  
 BufferedReader fileReader = new BufferedReader(new FileReader(*fileName*)); // Открытие файла для чтения  
 String line;  
 while ((line = fileReader.readLine()) != null) { // Построчное чтение файла  
 String[] words = line.split(" "); // Разделение строки на слова  
 for (String word : words) {  
 if (!word.isEmpty()) {  
 wordsList.add(word); // Добавление непустых слов в ArrayList  
 }  
 }  
 }  
 fileReader.close(); // Закрытие потока чтения  
 return wordsList; // Возврат списка слов  
 }  
  
 public static String getLine(String... *words*) {  
 if (*words* == null || *words*.length == 0) {  
 return ""; // Возвращаем пустую строку, если не переданы слова  
 }  
  
 StringBuilder result = new StringBuilder(*words*[0]); // Инициализация результирующей строки первым словом  
 ArrayList<String> remainingWords = new ArrayList<>(); // Создание списка оставшихся слов  
  
 for (int i = 1; i < *words*.length; i++) {  
 remainingWords.add(*words*[i]); // Добавление оставшихся слов в список  
 }  
  
 while (true) {  
 if (remainingWords.isEmpty()) {  
 break; // Выход из цикла, если список оставшихся слов пуст  
 }  
  
 char lastChar = Character.**toLowerCase**(result.charAt(result.length() - 1)); // Получение последней буквы результирующей строки  
 boolean found = false;  
  
 for (int i = 0; i < remainingWords.size(); i++) {  
 String word = remainingWords.get(i); // Получение текущего слова из списка оставшихся  
 char firstChar = Character.**toLowerCase**(word.charAt(0)); // Получение первой буквы текущего слова  
  
 if (lastChar == firstChar) { // Проверка совпадения последней буквы результирующей строки и первой буквы текущего слова  
 result.append(" ").append(word); // Добавление слова к результирующей строке  
 remainingWords.remove(i); // Удаление слова из списка оставшихся  
 found = true;  
 break;  
 }  
 }  
  
 if (!found) {  
 break; // Выход из цикла, если не найдено подходящее слово  
 }  
 }  
  
 return result.toString(); // Возвращаем результирующую строку  
 }  
}  
  
//C:\Users\Jeka\Desktop\3sem\ЖАБА\praca1\src\ru\mirea\lab13\input.txt

Листинг файла Main.java к заданию №3

## Вывод

В ходе выполнения данной практической работы мы изучили основные методы и приемы обработки строк в Java. Мы ознакомились с операциями конкатенации, сравнения и разделения строк, а также научились применять регулярные выражения для более сложных операций над строками. В процессе работы мы также освоили работу с классами StringBuilder и StringBuffer для эффективного построения и редактирования строк.

Понимание работы с строками в Java представляет собой важный навык для разработчика, поскольку строки являются неотъемлемой частью многих программ. Наша практическая работа позволила нам углубить свои знания в этой области и освоить различные методы манипуляции строками. Эти навыки пригодятся нам при разработке программ, где требуется эффективная и надежная обработка текстовых данных.

# Практическая работа № 14. Использование регулярных выражений в Джава приложениях

**Цель:** понять особенности использования регулярных выражений в Java, научиться работать с строками и применять регулярные выражения для обработки строк в программах.

**Задания на практическую работу № 14:**

1. Написать регулярное выражение, определяющее является ли данная строчка датой в формате dd/mm/yyyy. Начиная с 1900 года до 9999 года.

a) пример правильных выражений: 29/02/2000, 30/04/2003,01/01/2003. b) пример неправильных выражений: 29/02/2001, 30-04-2003, 1/1/1899.

2. Написать регулярное выражение, определяющее является ли данная строчка допустимым (корректным) е-mail адресом согласно RFC под номером 2822. a) пример правильных выражений: user@example.com, root@localhost

b) пример неправильных выражений: myhost@@@com.ru, @my.ru, Julia String.

3. Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов. Где символом может быть цифр, английская буква, и знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру.

a) пример правильных выражений: F032\_Password, TrySpy1.

b) пример неправильных выражений: smart\_pass, A007.

4. Напишите метод filter, который принимает на вход массив (любого типа) и реализацию интерфейса Filter c методом apply(Object o), чтобы убрать из массива лишнее. Проверьте как он работает на строках или других объектах.

5. Постройте частотный словарь букв русского (или английского) алфавита. Здесь не имеет значения проблема выбора и анализа корпуса языка, достаточно будет взять текст небольшой длины).

## Введение

Регулярные выражения – это мощный инструмент, который позволяет осуществлять эффективный поиск, анализ и манипуляцию текстовой информации. В мире программирования, особенно в языке Java, они играют значительную роль в обработке строковых данных. Использование регулярных выражений в Java приложениях позволяет разработчикам создавать гибкие и эффективные алгоритмы для работы с текстовой информацией.

Мы рассмотрим синтаксис и основные методы, предоставляемые Java для работы с регулярными выражениями. Кроме того, мы рассмотрим примеры применения регулярных выражений для различных задач, таких как поиск, замена, разделение и валидация текстовых данных.

В ходе этой работы, мы погрузимся в увлекательный мир регулярных выражений, и научимся применять их для эффективной обработки строк в наших Java приложениях. Научившись использовать регулярные выражения, мы сможем существенно улучшить качество и функциональность своих программ, сократив при этом затраты времени на обработку текстовой информации.

## Практическая часть

package ru.mirea.lab14; // Объявление пакета  
  
import java.util.regex.\*; // Импорт классов для работы с регулярными выражениями  
  
public class task1 { // Объявление публичного класса с именем task1  
  
 public static void main(String[] *args*) { // Основной метод программы  
 String regex = "^(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[0-2])/(19\\d{2}|[2-9]\\d{3})$";  
 // Регулярное выражение для проверки дат в формате dd/mm/yyyy, с ограничением с 1900 до 9999 года  
  
 String[] dates = { // Массив строк с датами для проверки  
 "29/02/2000",  
 "30/04/2003",  
 "01/01/2003",  
 "29/02/2001",  
 "30-04-2003",  
 "1/1/1899"  
 };  
 Pattern pattern = Pattern.**compile**(regex); // Создание объекта Pattern для компиляции регулярного выражения  
  
 for (String date : dates) { // Цикл для перебора всех дат из массива  
 Matcher matcher = pattern.matcher(date); // Создание объекта Matcher для сопоставления с регулярным выражением  
 if (matcher.matches()) { // Проверка, соответствует ли строка регулярному выражению  
 System.out.println(date + " - правильный формат"); // Если соответствует - выводим сообщение о правильном формате  
 } else {  
 System.out.println(date + " - неправильный формат"); // Если не соответствует - выводим сообщение о неправильном формате  
 }  
 }  
 }  
}

Листинг файла task1.java к заданию №1

package ru.mirea.lab14; // Объявление пакета  
  
import java.util.regex.\*; // Импорт классов для работы с регулярными выражениями  
  
public class task2 { // Объявление публичного класса с именем task2  
  
 public static void main(String[] *args*) { // Основной метод программы  
 String regex = "^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$"; // Регулярное выражение для проверки email-адресов  
 String[] emails = { // Массив строк с email-адресами для проверки  
 "user@example.com",  
 "root@localhost",  
 "myhost@@@com.ru",  
 "@my.ru",  
 "Julia String"  
 };  
 Pattern pattern = Pattern.**compile**(regex); // Создание объекта Pattern для компиляции регулярного выражения  
  
 for (String email : emails) { // Цикл для перебора всех email-адресов из массива  
 Matcher matcher = pattern.matcher(email); // Создание объекта Matcher для сопоставления с регулярным выражением  
 if (matcher.matches()) { // Проверка, соответствует ли строка регулярному выражению  
 System.out.println(email + " - правильный формат"); // Если соответствует - выводим сообщение о правильном формате  
 } else {  
 System.out.println(email + " - неправильный формат"); // Если не соответствует - выводим сообщение о неправильном формате  
 }  
 }  
 }  
}

Листинг файла task2.java к заданию №2

package ru.mirea.lab14;  
  
import java.util.regex.\*; // Импорт классов для работы с регулярными выражениями  
  
public class task3 { // Объявление публичного класса с именем PasswordChecker  
  
 public static void main(String[] *args*) { // Основной метод программы  
 // Регулярное выражение для проверки надежности пароля  
 String regex = "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)[a-zA-Z\\d\_]{8,}$";  
  
 String[] passwords = { // Массив паролей для проверки  
 "F032\_Password",  
 "TrySpy1",  
 "smart\_pass",  
 "A007"  
 };  
  
 Pattern pattern = Pattern.**compile**(regex); // Создание объекта Pattern для компиляции регулярного выражения  
  
 for (String password : passwords) { // Цикл для перебора всех паролей из массива  
 Matcher matcher = pattern.matcher(password); // Создание объекта Matcher для сопоставления с регулярным выражением  
 if (matcher.matches()) { // Проверка, соответствует ли пароль регулярному выражению  
 System.out.println(password + " - надежный пароль"); // Если соответствует - выводим сообщение о надежности пароля  
 } else {  
 System.out.println(password + " - ненадежный пароль"); // Если не соответствует - выводим сообщение о ненадежности пароля  
 }  
 }  
 }  
}

Листинг файла task3.java к заданию №3

package ru.mirea.lab14;  
  
interface Filter {  
 boolean apply(Object *o*); // Интерфейс с методом apply, который определяет условие фильтрации.  
}  
  
public class task4 {  
 public static Object[] filter(Object[] *array*, Filter *filter*) {  
 int newSize = 0;  
  
 // Подсчет количества элементов, которые проходят фильтрацию.  
 for (Object element : *array*) {  
 if (*filter*.apply(element)) {  
 newSize++;  
 }  
 }  
  
 Object[] result = new Object[newSize]; // Создание нового массива для хранения отфильтрованных элементов.  
 int index = 0;  
  
 // Заполнение нового массива отфильтрованными элементами.  
 for (Object element : *array*) {  
 if (*filter*.apply(element)) {  
 result[index] = element;  
 index++;  
 }  
 }  
  
 return result; // Возвращение нового массива с отфильтрованными элементами.  
 }  
  
 public static void main(String[] *args*) {  
 String[] strings = {"apple", "banana", "cherry", "date"};  
  
 // Создание объекта фильтра для фильтрации строк длиной более 5 символов.  
 Filter stringLengthFilter = new Filter() {  
 *@Override* public boolean apply(Object *o*) {  
 return ((String) *o*).length() > 5;  
 }  
 };  
  
 Object[] filteredStrings = **filter**(strings, stringLengthFilter); // Применение фильтра к массиву строк.  
  
 System.out.println("Filtered Strings:");  
 for (Object str : filteredStrings) {  
 System.out.println(str); // Вывод отфильтрованных строк.  
 }  
 }  
}

Листинг файла task4.java к заданию №4

package ru.mirea.lab14;  
  
import java.util.HashMap; // Импорт класса HashMap для создания словаря  
import java.util.Map; // Импорт интерфейса Map для работы с коллекцией Map  
  
public class task5 { // Объявление публичного класса с именем FrequencyDictionary  
  
 public static void main(String[] *args*) { // Основной метод программы  
 String text = "Этот текст предназначен для анализа частоты букв в русском алфавите.";  
  
 // Убираем все пробелы и приводим к нижнему регистру  
 text = text.replaceAll("\\s+", "").toLowerCase();  
 // "\\s+" - регулярное выражение для одного или более пробелов  
  
 // Создаем словарь для хранения частот  
 Map<Character, Integer> frequencyDictionary = new HashMap<>();  
 // Инициализация словаря, где ключ - символ, значение - частота  
  
 // Перебираем символы в тексте  
 for (char symbol : text.toCharArray()) {  
 // Преобразование строки в массив символов и итерация по символам  
 // в тексте  
  
 // Проверяем, является ли символ буквой русского алфавита  
 if (Character.**isLetter**(symbol) && symbol >= 'а' && symbol <= 'я') {  
 // Проверка на то, что символ - буква и принадлежит русскому алфавиту  
  
 // Если символ подходит, увеличиваем частоту  
 frequencyDictionary.put(symbol, frequencyDictionary.getOrDefault(symbol, 0) + 1);  
 // Увеличиваем частоту данного символа на 1, или 0 если символ встречается впервые  
 }  
 }  
  
 // Выводим результат  
 for (Map.Entry<Character, Integer> entry : frequencyDictionary.entrySet()) {  
 System.out.println(entry.getKey() + ": " + entry.getValue());  
 // Перебор и вывод всех элементов словаря (символ:частота)  
 }  
 }  
}

Листинг файла task5.java к заданию №5

## Вывод

В ходе данной практической работы было изучено применение регулярных выражений в Java приложениях. Регулярные выражения предоставляют мощный инструмент для работы с текстовыми данными, позволяя осуществлять поиск, сопоставление и манипуляции с паттернами.

В ходе работы были рассмотрены основные концепции и методы работы с регулярными выражениями в Java, такие как использование класса Pattern для компиляции регулярного выражения, а также класса Matcher для выполнения операций с текстом.

Были изучены различные метасимволы и конструкции, позволяющие задавать сложные шаблоны для поиска и анализа данных. Кроме того, были рассмотрены методы для замены подстрок и разделения текста с использованием регулярных выражений.

Эта практическая работа позволила научиться эффективно применять регулярные выражения в Java приложениях для решения различных задач обработки текстовых данных. Приобретенные знания и навыки могут быть полезны при разработке приложений, работающих с большим объемом текстовой информации и требующих точной обработки по определенным шаблонам.

# Практическая работа № 15. Вложенные и внутренние классы. Обработка событий в Джава программах с графическим интерфейсом пользователя

**Цель:** изучить использование анонимных и внутренних классов, научиться разрабатывать интерактивные программы на языке Джава с использованием графического интерфейса пользователя.

**Задания на практическую работу № 15:**

1. Разработайте программу с меню, двумя кнопками и текстовым полем ввода. В этой программе у вас должны быть разные настройки в меню. Должно быть меню «Файл», которое включает в себя подменю «Сохранить», «Выйти» и «Правка», включая подменю «Копировать, вырезать, вставить» и меню «Справка».

2. Напишите программу калькулятор, используя пример в листинге 15.6. Реализуйте помимо сложения вычитание, деление и умножение для двух чисел, которые вводятся с клавиатуры.

## Введение

В современном программировании создание сложных и функциональных приложений часто требует применения различных техник и инструментов. Одним из ключевых аспектов разработки в прикладной области является использование объектно-ориентированного подхода и работа с графическим пользовательским интерфейсом. В данной практической работе мы рассмотрим две важные темы: вложенные и внутренние классы, а также обработку событий в Java-программах с GUI.

Вложенные и внутренние классы представляют собой мощный механизм для организации кода и решения сложных задач. Они позволяют создавать более чистый и модульный код, а также реализовывать различные паттерны проектирования.

Обработка событий в программах с графическим интерфейсом — это неотъемлемая часть разработки приложений для взаимодействия с пользователем. Правильная обработка событий обеспечивает отзывчивость интерфейса и позволяет создавать удобные и интуитивно понятные приложения.

## Практическая часть

package ru.mirea.lab15; // Пакет, в котором находится класс  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
  
public class task1 {  
 private static JTextField **textField**; // Объявление текстового поля  
  
 public static void main(String[] *args*) {  
 JFrame frame = new JFrame("Моя программа"); // Создание главного окна  
 frame.setSize(300, 200); // Установка размеров окна  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Установка операции по закрытию  
  
 JMenuBar menuBar = new JMenuBar(); // Создание строки меню  
  
 // Меню "Файл"  
 JMenu fileMenu = new JMenu("Файл"); // Создание меню "Файл"  
 JMenuItem saveItem = new JMenuItem("Сохранить"); // Пункт меню "Сохранить"  
 saveItem.addActionListener(new ActionListener() {  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 // Реализация сохранения файла  
 JFileChooser fileChooser = new JFileChooser();  
 int returnValue = fileChooser.showSaveDialog(null);  
 if (returnValue == JFileChooser.APPROVE\_OPTION) {  
 String filename = fileChooser.getSelectedFile().getName();  
 JOptionPane.**showMessageDialog**(frame, "Файл '" + filename + "' сохранен");  
 }  
 }  
 });  
  
 JMenuItem exitItem = new JMenuItem("Выйти"); // Пункт меню "Выйти"  
 exitItem.addActionListener(new ActionListener() {  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 System.**exit**(0); // Выход из программы  
 }  
 });  
  
 fileMenu.add(saveItem); // Добавление пункта "Сохранить" в меню "Файл"  
 fileMenu.add(exitItem); // Добавление пункта "Выйти" в меню "Файл"  
  
 // Меню "Правка"  
 JMenu editMenu = new JMenu("Правка"); // Создание меню "Правка"  
 JMenuItem copyItem = new JMenuItem("Копировать"); // Пункт меню "Копировать"  
 JMenuItem cutItem = new JMenuItem("Вырезать"); // Пункт меню "Вырезать"  
 JMenuItem pasteItem = new JMenuItem("Вставить"); // Пункт меню "Вставить"  
  
 copyItem.addActionListener(new ActionListener() {  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 // Реализация копирования  
 **textField**.copy(); // Копирование текста из текстового поля  
 }  
 });  
  
 cutItem.addActionListener(new ActionListener() {  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 // Реализация вырезания  
 **textField**.cut(); // Вырезание текста из текстового поля  
 }  
 });  
  
 pasteItem.addActionListener(new ActionListener() {  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 // Реализация вставки  
 **textField**.paste(); // Вставка текста в текстовое поле  
 }  
 });  
  
 editMenu.add(copyItem); // Добавление пункта "Копировать" в меню "Правка"  
 editMenu.add(cutItem); // Добавление пункта "Вырезать" в меню "Правка"  
 editMenu.add(pasteItem); // Добавление пункта "Вставить" в меню "Правка"  
  
 JMenu helpMenu = new JMenu("Справка"); // Создание меню "Справка"  
  
 menuBar.add(fileMenu); // Добавление меню "Файл" в строку меню  
 menuBar.add(editMenu); // Добавление меню "Правка" в строку меню  
 menuBar.add(helpMenu); // Добавление меню "Справка" в строку меню  
  
 frame.setJMenuBar(menuBar); // Установка строки меню в главном окне  
  
 JPanel panel = new JPanel(); // Создание панели  
 JButton button1 = new JButton("Кнопка 1"); // Создание кнопки "Кнопка 1"  
 JButton button2 = new JButton("Кнопка 2"); // Создание кнопки "Кнопка 2"  
 **textField** = new JTextField(10); // Создание текстового поля  
  
 panel.add(button1); // Добавление кнопки "Кнопка 1" на панель  
 panel.add(button2); // Добавление кнопки "Кнопка 2" на панель  
 panel.add(**textField**); // Добавление текстового поля на панель  
  
 button1.addActionListener(new ActionListener() {  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 // Обработчик для кнопки 1 (добавьте свой функционал здесь)  
 }  
 });  
  
 button2.addActionListener(new ActionListener() {  
 *@Override* public void actionPerformed(ActionEvent *e*) {  
 // Обработчик для кнопки 2 (добавьте свой функционал здесь)  
 }  
 });  
  
 frame.add(panel); // Добавление панели в главное окно  
  
 frame.setVisible(true); // Установка видимости главного окна  
 }  
}

Листинг файла task1.java к заданию №1

package ru.mirea.lab15;  
  
import java.awt.\*;  
import java.awt.event.\*;  
import javax.swing.\*;  
  
// Создаем класс-наследник JFrame для построения GUI  
public class Calculator extends JFrame {  
 // Создание текстовых полей и кнопок  
 JTextField jta1 = new JTextField(10); // Поле для первого числа  
 JTextField jta2 = new JTextField(10); // Поле для второго числа  
 JButton addButton = new JButton("Add"); // Кнопка для сложения  
 JButton subButton = new JButton("Subtract"); // Кнопка для вычитания  
 JButton mulButton = new JButton("Multiply"); // Кнопка для умножения  
 JButton divButton = new JButton("Divide"); // Кнопка для деления  
  
 // Конструктор класса  
 Calculator() {  
 super("Calculator"); // Устанавливаем заголовок окна  
 setLayout(new FlowLayout()); // Устанавливаем менеджер компоновки  
 setSize(250, 250); // Устанавливаем размеры окна  
  
 // Добавляем компоненты на форму  
 add(new JLabel("1st Number")); // Метка для первого числа  
 add(jta1); // Поле ввода для первого числа  
 add(new JLabel("2nd Number")); // Метка для второго числа  
 add(jta2); // Поле ввода для второго числа  
  
 add(addButton); // Кнопка сложения  
 add(subButton); // Кнопка вычитания  
 add(mulButton); // Кнопка умножения  
 add(divButton); // Кнопка деления  
  
 // Настраиваем обработчики событий для кнопок  
 addButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent *ae*) {  
 calculate('+'); // Вызов метода calculate с оператором сложения  
 }  
 });  
  
 subButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent *ae*) {  
 calculate('-'); // Вызов метода calculate с оператором вычитания  
 }  
 });  
  
 mulButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent *ae*) {  
 calculate('\*'); // Вызов метода calculate с оператором умножения  
 }  
 });  
  
 divButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent *ae*) {  
 calculate('/'); // Вызов метода calculate с оператором деления  
 }  
 });  
  
 setVisible(true); // Делаем окно видимым  
 }  
  
 // Метод для выполнения арифметических операций  
 public void calculate(char *operator*) {  
 try {  
 double x1 = Double.**parseDouble**(jta1.getText().trim()); // Получаем первое число  
 double x2 = Double.**parseDouble**(jta2.getText().trim()); // Получаем второе число  
 double result = 0; // Переменная для результата вычислений  
  
 switch (*operator*) {  
 case '+':  
 result = x1 + x2; // Сложение  
 break;  
 case '-':  
 result = x1 - x2; // Вычитание  
 break;  
 case '\*':  
 result = x1 \* x2; // Умножение  
 break;  
 case '/':  
 if (x2 != 0) {  
 result = x1 / x2; // Деление (проверка деления на ноль)  
 } else {  
 JOptionPane.**showMessageDialog**(null, "Error: Division by zero!", "Alert", JOptionPane.ERROR\_MESSAGE);  
 return;  
 }  
 break;  
 }  
  
 JOptionPane.**showMessageDialog**(null, "Result = " + result, "Alert", JOptionPane.INFORMATION\_MESSAGE); // Вывод результата  
  
 } catch (Exception *e*) {  
 JOptionPane.**showMessageDialog**(null, "Error in Numbers!", "Alert", JOptionPane.ERROR\_MESSAGE); // Обработка ошибок  
 }  
 }  
  
 // Метод main для запуска программы  
 public static void main(String[] *args*) {  
 new Calculator(); // Создаем объект класса Calculator  
 }  
}

Листинг файла Calculator.java к заданию №2

## Вывод

В результате выполнения данной практической работы мы ознакомились с важными аспектами Java-программирования: вложенными и внутренними классами, а также обработкой событий в графическом интерфейсе. Понимание и умение применять эти концепции позволяет разработчикам создавать более эффективный и чистый код, а также строить удобные и интуитивно понятные пользовательские интерфейсы.

Вложенные и внутренние классы предоставляют мощный инструмент для организации кода, позволяя группировать связанные элементы и сокращать объем кода.

Обработка событий в GUI приложениях является неотъемлемой частью разработки. Это позволяет создавать интерфейсы, реагирующие на действия пользователя, и обеспечивает плавное и отзывчивое взаимодействие.

Полученные знания и навыки могут быть применены в различных областях разработки, от создания десктопных приложений до мобильных приложений с графическим интерфейсом. Они представляют собой важный шаг к созданию качественного и функционального программного продукта.

# Практическая работа № 16. Обработка событий мыши и клавиатуры программах на Джава с графическим интерфейсом пользователя

**Цель:** научиться обрабатывать различные события мыши и клавиатуры для разных компонентов.

**Задания на практическую работу № 16:**

1. Реализуйте программу Проверка пароля на Джава с использованием Layout менеджеров компоновки. Окно программы должно иметь вид как на рис.1

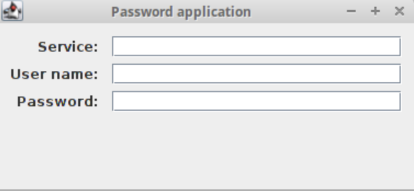


Рисунок 1 Окно программы Проверка пароля

2. Реализуйте программу на Джава с использованием JTextArea и двумя следующего меню выбора: a) Цвет: который имеет возможность выбора из три возможных: синий, красный и черный

b) Шрифт: три вида: “Times New Roman”, “MS Sans Serif”, “Courier New”.

## Введение

В современных приложениях с графическим интерфейсом пользовательский взаимодействие играет ключевую роль. Эффективная обработка событий мыши и клавиатуры является неотъемлемой частью разработки программ, обеспечивающих интуитивно-понятный и отзывчивый пользовательский интерфейс. Язык Java предоставляет богатые инструменты для работы с событиями, что позволяет разработчикам создавать интерактивные и функциональные приложения.

В данной практической работе мы будем исследовать методы обработки событий мыши и клавиатуры в Java, используя графический пользовательский интерфейс. Мы рассмотрим основные принципы работы с событиями, а также научимся реализовывать функционал, реагирующий на действия пользователя.

## Практическая часть

package ru.mirea.lab16;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class TextEditorApp extends JFrame {  
 private JTextArea textArea;  
 private JMenu colorMenu;  
 private JMenu fontMenu;  
  
 public TextEditorApp() {  
 *// Настройка окна приложения* setTitle("Текстовый редактор");  
 setSize(600, 400);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 *// Создание текстовой области* textArea = new JTextArea();  
 textArea.setFont(new Font("Times New Roman", Font.*PLAIN*, 12));  
  
 *// Создание меню выбора цвета* colorMenu = new JMenu("Цвет");  
 JMenuItem blueItem = new JMenuItem("Синий");  
 JMenuItem redItem = new JMenuItem("Красный");  
 JMenuItem blackItem = new JMenuItem("Черный");  
  
 blueItem.addActionListener(new ColorMenuItemListener(Color.*BLUE*));  
 redItem.addActionListener(new ColorMenuItemListener(Color.*RED*));  
 blackItem.addActionListener(new ColorMenuItemListener(Color.*BLACK*));  
  
 colorMenu.add(blueItem);  
 colorMenu.add(redItem);  
 colorMenu.add(blackItem);  
  
 *// Создание меню выбора шрифта* fontMenu = new JMenu("Шрифт");  
 JMenuItem timesNewRomanItem = new JMenuItem("Times New Roman");  
 JMenuItem msSansSerifItem = new JMenuItem("MS Sans Serif");  
 JMenuItem courierNewItem = new JMenuItem("Courier New");  
  
 timesNewRomanItem.addActionListener(new FontMenuItemListener("Times New Roman"));  
 msSansSerifItem.addActionListener(new FontMenuItemListener("MS Sans Serif"));  
 courierNewItem.addActionListener(new FontMenuItemListener("Courier New"));  
  
 fontMenu.add(timesNewRomanItem);  
 fontMenu.add(msSansSerifItem);  
 fontMenu.add(courierNewItem);  
  
 *// Создание строки меню* JMenuBar menuBar = new JMenuBar();  
 menuBar.add(colorMenu);  
 menuBar.add(fontMenu);  
  
 *// Установка строки меню в окно приложения* setJMenuBar(menuBar);  
  
 *// Добавление текстовой области в окно* getContentPane().add(new JScrollPane(textArea));  
  
 *// Отображение окна* setVisible(true);  
 }  
  
 *// Обработчик событий для выбора цвета* private class ColorMenuItemListener implements ActionListener {  
 private Color color;  
  
 public ColorMenuItemListener(Color color) {  
 this.color = color;  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 textArea.setForeground(color);  
 }  
 }  
  
 *// Обработчик событий для выбора шрифта* private class FontMenuItemListener implements ActionListener {  
 private String fontName;  
  
 public FontMenuItemListener(String fontName) {  
 this.fontName = fontName;  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 textArea.setFont(new Font(fontName, Font.*PLAIN*, 18));  
 }  
 }  
  
 public static void main(String[] args) {  
 new TextEditorApp();  
 }  
}

}  
}

Листинг файла TextEditorApp.java к заданию №2

package ru.mirea.lab16;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.\*;  
import java.util.Scanner;  
  
public class PasswordCheckerApp extends JFrame {  
 private JTextField serviceField, usernameField;  
 private JPasswordField passwordField;  
 private JButton checkButton;  
  
 public PasswordCheckerApp() {  
 *// Настройка окна приложения* setTitle("Проверка пароля");  
 setSize(300, 150);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 *// Создание компонентов* JLabel serviceLabel = new JLabel("Service:");  
 JLabel usernameLabel = new JLabel("Username:");  
 JLabel passwordLabel = new JLabel("Password:");  
  
 serviceField = new JTextField(20);  
 usernameField = new JTextField(20);  
 passwordField = new JPasswordField(20);  
  
 checkButton = new JButton("Проверить");  
 checkButton.addActionListener(e -> checkPassword());  
  
 *// Настройка Layout менеджера* setLayout(new GridLayout(4, 2));  
  
 *// Добавление компонентов в окно* add(serviceLabel);  
 add(serviceField);  
 add(usernameLabel);  
 add(usernameField);  
 add(passwordLabel);  
 add(passwordField);  
 add(new JLabel()); *// Пустая ячейка для разделения* add(checkButton);  
  
 *// Отображение окна* setVisible(true);  
 }  
  
 private void checkPassword() {  
 String service = serviceField.getText();  
 String username = usernameField.getText();  
 String password = new String(passwordField.getPassword());  
  
 *// Загрузка данных из файла* try {  
 File file = new File("C:\\Users\\Alex\\Desktop\\3sem\\javaprac\\src\\ru\\mirea\\lab16\\passwords.csv");  
 Scanner scanner = new Scanner(file);  
  
 while (scanner.hasNextLine()) {  
 String line = scanner.nextLine();  
 String[] parts = line.split(",");  
 String storedService = parts[0];  
 String storedUsername = parts[1];  
 String storedPassword = parts[2];  
  
 if (service.equals(storedService) && username.equals(storedUsername) && password.equals(storedPassword)) {  
 JOptionPane.showMessageDialog(this, "Пароль верный!");  
 return;  
 }  
 }  
  
 JOptionPane.showMessageDialog(this, "Пароль неверный.");  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }

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

Листинг файла PasswordCheckerApp.java к заданию №1

## Вывод

В ходе этой практической работы мы изучили важные аспекты обработки событий мыши и клавиатуры в приложениях, разработанных на Java с использованием графического интерфейса пользователя. Мы освоили методы подключения обработчиков событий, создания интерактивных элементов и реализации функционала, реагирующего на действия пользователя.

Эти навыки являются фундаментальными для создания пользовательских интерфейсов, которые предоставляют удобство и эффективность взаимодействия с приложением. Понимание обработки событий позволяет разработчикам создавать интуитивно-понятные и отзывчивые приложения, удовлетворяющие потребности пользователей.

# Практическая работа № 17. Разработка интерактивных программ на языке Джава с использованием паттерна MVC

**Цель:** введение в разработку программ c с использованием событийного программирования на языке программирования Джава с использованием паттерна MVC.

**Задания на практическую работу № 17:**

1. Напишите реализацию программного кода, с использованием паттерна MVC для расчета заработной платы сотрудника предприятия. Предлагается использовать следующие классы.

Класс Employee – сотрудник будет выступать в качестве слоя модели

Класс EmployeeView будет действовать как слой представления.

Класс EmployeeContoller будет действовать как уровень контроллера.

2. Вы можете написать программную реализацию, используя собственную идею, реализуя паттерн MVC. Выполнение задания предполагает создание GUI.

## Введение

В современной разработке программного обеспечения важным аспектом является создание приложений, способных обеспечивать высокую интерактивность и удобство использования для пользователей. В этом контексте паттерн MVC (Model-View-Controller) представляет собой эффективный подход, который позволяет разделить логику приложения на три основных компонента: модель, представление и контроллер. Этот паттерн обеспечивает более четкую организацию кода, улучшает его поддержку и расширяемость.

## Практическая часть

package ru.mirea.lab17;  
  
*// Класс Employee (Модель)*public class Employee {  
 private String name; *// Имя сотрудника* private double hourlyRate; *// Почасовая ставка* private int hoursWorked; *// Количество отработанных часов  
  
 // Конструктор класса* public Employee(String name, double hourlyRate, int hoursWorked) {  
 this.name = name;  
 this.hourlyRate = hourlyRate;  
 this.hoursWorked = hoursWorked;  
 }  
  
 *// Метод для получения имени сотрудника* public String getName() {  
 return name;  
 }  
  
 *// Метод для получения почасовой ставки* public double getHourlyRate() {  
 return hourlyRate;  
 }  
  
 *// Метод для получения количества отработанных часов* public int getHoursWorked() {  
 return hoursWorked;  
 }  
  
 *// Метод для расчета заработной платы* public double calculateSalary() {  
 return hourlyRate \* hoursWorked;  
 }  
}

Листинг файла Employee.java к заданию №1

package ru.mirea.lab17;  
  
*// Класс EmployeeController (Контроллер)*public class EmployeeController {  
 private Employee model;  
 private EmployeeView view;  
  
 public EmployeeController(Employee model, EmployeeView view) {  
 this.model = model;  
 this.view = view;  
 }  
  
 public void updateView() {  
 view.printEmployeeDetails(model.getName(), model.getHourlyRate(), model.getHoursWorked(), model.calculateSalary());  
 }  
}

Листинг файла EmployeeController.java к заданию №1

package ru.mirea.lab17;  
  
*// Класс EmployeeView (Представление)*public class EmployeeView {  
 public void printEmployeeDetails(String name, double hourlyRate, int hoursWorked, double salary) {  
 System.*out*.println("Имя: " + name);  
 System.*out*.println("Часовая ставка: " + hourlyRate);  
 System.*out*.println("Отработано часов: " + hoursWorked);  
 System.*out*.println("Заработная плата: " + salary);  
 }  
}

Листинг файла EmployeeView.java к заданию №1

package ru.mirea.lab17;  
  
*// Пример использования*public class Main {  
 public static void main(String[] args) {  
 *// Создаем сотрудника* Employee employee = new Employee("Иван Иванов", 10.0, 40);  
  
 *// Создаем представление и контроллер* EmployeeView view = new EmployeeView();  
 EmployeeController controller = new EmployeeController(employee, view);  
  
 *// Выводим информацию о сотруднике* controller.updateView();  
 }  
}

Листинг файла Main.java к заданию №1

package ru.mirea.lab17;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class CalculatorController {  
 private CalculatorView view;  
 private CalculatorModel model;  
  
 public CalculatorController(CalculatorView view, CalculatorModel model) {  
 this.view = view;  
 this.model = model;  
  
 this.view.addAdditionListener(new AdditionListener());  
 this.view.addSubtractionListener(new SubtractionListener());  
 this.view.addMultiplicationListener(new MultiplicationListener());  
 this.view.addDivisionListener(new DivisionListener());  
 }  
  
 class AdditionListener implements ActionListener {  
 public void actionPerformed(ActionEvent event) {  
 double firstNumber = view.getFirstNumber();  
 double secondNumber = view.getSecondNumber();  
 model.add(firstNumber, secondNumber);  
 view.setResult(model.getResult());  
 }  
 }  
  
 class SubtractionListener implements ActionListener {  
 public void actionPerformed(ActionEvent event) {  
 double firstNumber = view.getFirstNumber();  
 double secondNumber = view.getSecondNumber();  
 model.subtract(firstNumber, secondNumber);  
 view.setResult(model.getResult());  
 }  
 }  
  
 class MultiplicationListener implements ActionListener {  
 public void actionPerformed(ActionEvent event) {  
 double firstNumber = view.getFirstNumber();  
 double secondNumber = view.getSecondNumber();  
 model.multiply(firstNumber, secondNumber);  
 view.setResult(model.getResult());  
 }  
 }  
  
 class DivisionListener implements ActionListener {  
 public void actionPerformed(ActionEvent event) {  
 double firstNumber = view.getFirstNumber();  
 double secondNumber = view.getSecondNumber();  
 model.divide(firstNumber, secondNumber);  
 view.setResult(model.getResult());  
 }  
 }  
}

Листинг файла CalculatorController.java к заданию №2

package ru.mirea.lab17;  
  
public class CalculatorModel {  
 private double result;  
  
 public void add(double x, double y) {  
 result = x + y;  
 }  
  
 public void subtract(double x, double y) {  
 result = x - y;  
 }  
  
 public void multiply(double x, double y) {  
 result = x \* y;  
 }  
  
 public void divide(double x, double y) {  
 if (y != 0) {  
 result = x / y;  
 } else {  
 result = Double.*NaN*; *// Обработка деления на ноль* }  
 }  
  
 public double getResult() {  
 return result;  
 }  
}

Листинг файла CalculatorModel.java к заданию №2

package ru.mirea.lab17;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class CalculatorView {  
 private JTextField firstNumber = new JTextField(10);  
 private JTextField secondNumber = new JTextField(10);  
 private JButton addButton = new JButton("+");  
 private JButton subButton = new JButton("-");  
 private JButton mulButton = new JButton("\*");  
 private JButton divButton = new JButton("/");  
 private JTextField resultField = new JTextField(10);  
  
 public CalculatorView() {  
 JFrame frame = new JFrame("Простой калькулятор");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 JPanel panel = new JPanel();  
  
 panel.add(firstNumber);  
 panel.add(secondNumber);  
 panel.add(addButton);  
 panel.add(subButton);  
 panel.add(mulButton);  
 panel.add(divButton);  
 panel.add(resultField);  
  
 frame.getContentPane().add(BorderLayout.*CENTER*, panel);  
 frame.setSize(300, 200);  
 frame.setVisible(true);  
 }  
  
 public double getFirstNumber() {  
 return Double.parseDouble(firstNumber.getText());  
 }  
  
 public double getSecondNumber() {  
 return Double.parseDouble(secondNumber.getText());  
 }  
  
 public void setResult(double result) {  
 resultField.setText(String.valueOf(result));  
 }  
  
 public void addAdditionListener(ActionListener listener) {  
 addButton.addActionListener(listener);  
 }  
  
 public void addSubtractionListener(ActionListener listener) {  
 subButton.addActionListener(listener);  
 }  
  
 public void addMultiplicationListener(ActionListener listener) {  
 mulButton.addActionListener(listener);  
 }

public void addDivisionListener(ActionListener listener) {  
 divButton.addActionListener(listener);  
 }  
}

Листинг файла CalculatorView.java к заданию №2

package ru.mirea.lab17;  
  
public class Maincalc {  
 public static void main(String[] args) {  
 CalculatorModel model = new CalculatorModel();  
 CalculatorView view = new CalculatorView();  
 CalculatorController controller = new CalculatorController(view, model);  
 }  
}

Листинг файла Maincalc.java к заданию №2

## Вывод

В результате выполнения данной практической работы, вы приобретете практические навыки работы с паттерном MVC в контексте разработки приложений на языке Java. Этот опыт будет полезен при создании более сложных и масштабных проектов, где четкое разделение логики приложения на модель, представление и контроллер позволяет обеспечить высокую поддерживаемость и расширяемость кода. Кроме того, вы сможете создавать более интерактивные и удобные в использовании приложения для конечных пользователей.

# Практическая работа № 18. Исключения и работа с ними в Джава

**Цель:** 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. Ключевые слова: try, catch, finally, throw, throws

**Задания на практическую работу № 18:**

1.Измените код в листинге 18.3 на следующий: Листинг 18.4 Пример программы

public class Exception2 {

public void exceptionDemo() {

Scanner myScanner = new Scanner( System.in);

System.out.print( "Enter an integer ");

String intString = myScanner.next();

int i = Integer.parseInt(intString);

System.out.println( 2/i );

}

}

Шаг 2. Запустите эту программу со следующими выводами: Qwerty 0 1.2

1. Посмотрите на вывод.

Объясните какие исключения выбрасываются?

Шаг 3. Измените код, добавив блоки try – catch, чтобы иметь дело с определяемыми исключениями. Объясните поведение программы

Листинг 18.7 класс ThrowsDemo

public class ThrowsDemo {

public void getKey() {

Scanner myScanner = new Scanner( System.in );

String key = myScanner.next();

printDetails( key );

}

public void printDetails(String key) {

try { String message = getDetails(key);

System.out.println( message );

}catch ( Exception e){

throw e;

}

}

private String getDetails(String key) {

if(key == "") {

throw new Exception( "Key set to

empty string" );

}

return "data for " + key; }

}

2. Создайте следующий класс ( листинг 18.7) и попытайтесь его скомпилировать. При попытке компиляции вы получите следующий синтаксис ошибки: Исключение Unreported java.lang.Exception В результате успешного пробрасывания исключение должен быть поймано или объявлено. Объясните причину. Ответ. Причиной полученной ошибки является выражение throw е. Пояснение. В данном случае метод printDetails () решил, что он не может иметь дело с исключением и проходит все дерево его вызовов. Поскольку метод getKey() не имеет блока try-catch для обработки исключений, то Java становится перед выбором, что в таком случае делать.

Проблему можно решить несколькими способами:

1) Добавьте соответствующие try-catch блоки таким образом, чтобы в конечном итоге один из них обрабатывал исключение;

2) Удалите блоки try-catch для всех методов, кроме одного, который обрабатывает исключение. Добавьте throws, котрый бросает исключение методу, который проходит исключение без обработки.

3. Измените код из предыдущего примера следующим образом:

1. Удалите throws Exception из метода getKey().

2. Измените метод getKey(), добавив try-catch блок для обработки исключений.

3. Добавьте цикл к getKey() таким образом, чтобы пользователь получил еще один шанс на ввод значение ключа

## Введение

Исключения представляют собой важную часть программирования, которая позволяет эффективно управлять ошибками и нестандартными ситуациями в Java. В данной практической работе мы будем изучать, как обрабатывать исключения, а также как создавать собственные пользовательские исключения. Умение правильно обрабатывать исключения является неотъемлемой частью профессионального программирования, позволяя создавать надежные и устойчивые к ошибкам приложения.

## Практическая часть

package ru.mirea.lab18;  
  
import java.util.Scanner;  
  
public class Exception2 {  
 public void exceptionDemo() {  
 Scanner myScanner = new Scanner(System.*in*);  
 System.*out*.print("Enter an integer ");  
 String intString = myScanner.next();  
 int i = Integer.parseInt(intString);  
 System.*out*.println(2/i);  
 }  
}

public static void main(String[] args) {  
 Exception2 exception2 = new Exception2();  
 exception2.exceptionDemo();  
 }  
}

Листинг файла Exception2.java к заданию №1

Запустим программу с различными входными данными: "Qwerty", "0", "1.2". Посмотрим на вывод:

Если вводится "Qwerty", то программа выбросит NumberFormatException, так как "Qwerty" нельзя преобразовать в целое число.

Если вводится "0", то программа выбросит ArithmeticException, так как деление на ноль невозможно.

Если вводится "1.2", то программа также выбросит NumberFormatException, так как "1.2" не является целым числом.

public class Exception2 {  
 public void exceptionDemo() {  
 Scanner myScanner = new Scanner(System.*in*);  
 System.*out*.print("Enter an integer ");  
 try {  
 String intString = myScanner.next();  
 int i = Integer.parseInt(intString);  
 System.*out*.println(2/i);  
 } catch (NumberFormatException e) {  
 System.*out*.println("Invalid input. Please enter an integer.");  
 } catch (ArithmeticException e) {  
 System.*out*.println("Attempted division by zero");  
 }  
 }  
  
 public static void main(String[] args) {  
 Exception2 exception2 = new Exception2();  
 exception2.exceptionDemo();  
 }  
}

Листинг файла Exception2.java к заданию №1(Измененный код)

package ru.mirea.lab18;  
  
import java.util.Scanner;  
  
public class ThrowsDemo {  
 *// Метод getKey может выбросить исключение типа Exception* public void getKey() throws Exception {  
 Scanner myScanner = new Scanner(System.*in*);  
 int attempts = 2;  
  
 while (attempts > 0) {  
 String key = myScanner.next();  
 try {  
 printDetails(key);  
 return; *// Если успешно, выходим из цикла* } catch (Exception e) {  
 System.*out*.println("Поймано исключение: " + e.getMessage());  
 attempts--;  
 if (attempts == 0) {  
 System.*out*.println("Исчерпаны попытки. Завершение работы.");  
 } else {  
 System.*out*.println("У вас осталось " + attempts + " попыток. Пожалуйста, введите корректный ключ:");  
 }  
 }  
 }  
 }  
  
 *// Метод printDetails может выбросить исключение типа Exception* public void printDetails(String key) throws Exception {  
 String message = getDetails(key);  
 System.*out*.println(message);  
 }  
  
 *// Метод getDetails может выбросить исключение типа Exception* private String getDetails(String key) throws Exception {  
 if (key.equals("")) {  
 throw new Exception("Ключ установлен как пустая строка");  
 }  
 return "данные для " + key;  
 }  
  
 *// Метод manuallyTriggerException вызывает исключение типа Exception вручную* public void manuallyTriggerException() {  
 try {  
 throw new Exception("Исключение вызвано вручную");  
 } catch (Exception e) {  
 System.*out*.println("Поймано исключение: " + e.getMessage());  
 }  
 }  
}

Листинг файла ThrowsDemo.java к заданию №2

package ru.mirea.lab18;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 ThrowsDemo demo = new ThrowsDemo();  
 try {  
 demo.getKey(); *// Можно использовать этот метод, как ранее  
 //demo.manuallyTriggerException(); // Вызываем исключение вручную* } catch (Exception e) {  
 System.*out*.println("Поймано исключение: " + e.getMessage());  
 }  
 }  
}

Листинг файла Main.java к заданиям №2-3

## Вывод

Фактически при работе с исключениями весь материал делится на две части: синтаксис (ответ на вопрос, что компилятор пропустит, а что нет) и семантика (вопрос, как лучше делать) исключений. В отличие от вариантов с for, while, switch, использование исключений – более сложный механизм. Но он сложен не синтаксически, а семантически, по своему подходу. То есть при генерации исключений нужно думать о том - не как правильно его использовать, и с каким умыслом его использовать. То есть вопрос стоит так, в каких ситуациях стоит ли ломать систему и где, а в каких ситуациях ее восстанавливать. В хорошей инженерной системе каждый любой модуль всегда проверяет все входные данные.

# Практическая работа № 19. Создание пользовательских исключений

**Цель работы:** научиться создавать собственные исключения.

**Задания на практическую работу № 19:**

1. 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.

## Введение

В современном программировании, обработка ошибок является неотъемлемой частью разработки надежных и устойчивых программных систем. Однако, иногда стандартные исключения не могут полностью охватить все возможные сценарии ошибок. Именно здесь на помощь приходят пользовательские исключения.

## Практическая часть

package ru.mirea.lab19;  
  
*// Создаем пользовательское исключение для недействительного ИНН*class InvalidINNException extends Exception {  
 public InvalidINNException(String message) {  
 super(message);  
 }  
}  
  
*// Класс, представляющий клиента*class Client {  
 private String fullName;  
 private String INN;  
  
 public Client(String fullName, String INN) {  
 this.fullName = fullName;  
 this.INN = INN;  
 }  
  
 *// Метод для проверки действительности ИНН* public void validateINN() throws InvalidINNException {  
 *// В данном примере, просто проверяем длину ИНН (допустимо, если это не реальная проверка)* if (INN.length() != 10 && INN.length() != 12) {  
 throw new InvalidINNException("Недействительный ИНН");  
 }  
 }  
}  
  
*// Пример использования*public class Main {  
 public static void main(String[] args) {  
 try {  
 *// Пример создания клиента с ИНН* Client client = new Client("Иванов Иван Иванович", "123456789");  
  
 *// Пытаемся проверить ИНН* client.validateINN();  
 System.*out*.println("ИНН действителен");  
  
 } catch (InvalidINNException e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

Листинг файла Main.java к заданию №1

## Вывод

В результате выполнения данной практической работы, мы овладели навыками создания и использования пользовательских исключений в языке Java. Этот инструмент позволяет разработчикам более гибко управлять ситуациями ошибок, улучшая читаемость кода и обеспечивая более высокую надежность программного продукта.

Знание работы с пользовательскими исключениями открывает новые возможности для создания более гибких и надежных программных систем, что является важным аспектом в профессиональной разработке программного обеспечения.

# Практическая работа № 20. Работа с дженериками.

**Цель работы:** научиться работать с обобщенными типами в Java и применять их в программах.

**Задания на практическую работу № 20:**

1. Создать обобщенный класс с тремя параметрами (T, V, K).

2. Класс содержит три переменные типа (T, V, K), конструктор, принимающий на вход параметры типа (T, V, K), методы возвращающие значения трех переменных. Создать метод, выводящий на консоль имена классов для трех переменных класса.

3. Наложить ограничения на параметры типа: T должен реализовать интерфейс Comparable (классы оболочки, String), V должен реализовать интерфейс Serializable и расширять класс Animal, K.

4. Написать класс Matrix, на основе обобщенного типа, реализовать операции с матрицами.

## Введение

В мире современного программирования, работа с коллекциями данных представляет собой неотъемлемую часть разработки. Дженерики в языке программирования Java предоставляют мощный инструмент для создания универсальных и типобезопасных структур данных и алгоритмов.

## Практическая часть

package ru.mirea.lab20;  
  
import java.io.Serializable;  
  
*// Класс Cat наследуется от Animal и реализует интерфейс Serializable*class Cat extends Animal implements Serializable {  
 private static final String *species* = "";  
  
 *// Конструктор по умолчанию* public Cat() {  
 super(*species*);  
 }  
}  
  
*// Интерфейс для объектов, которые имеют имя*interface Nameable {  
 String getName(); *// Метод возвращает имя объекта, реализующего интерфейс.*}  
  
*// Класс Animal реализует интерфейс Nameable*class Animal implements Nameable {  
 private String species;  
  
 *// Конструктор класса Animal* public Animal(String species) {  
 this.species = species;  
 }  
  
 @Override  
 public String getName() {  
 return species; *// Возвращает вид животного* }  
}  
  
*// Параметризированный класс с тремя типами*class CustomClass<T, V extends Animal & Serializable, K extends Comparable<K>> {  
 private T t;  
 private V v;  
 private K k;  
  
 *// Конструктор класса CustomClass* public CustomClass(T t, V v, K k) {  
 this.t = t;  
 this.v = v;  
 this.k = k;  
 }  
  
 *// Геттеры* public T getT() {  
 return t;  
 }  
  
 public V getV() {  
 return v;  
 }  
  
 public K getK() {  
 return k;  
 }  
  
 *// Метод выводит типы объектов* public void printClassNames() {  
 System.*out*.println("Type of T: " + t.getClass().getName());  
 System.*out*.println("Type of V: " + v.getClass().getName());  
 System.*out*.println("Type of K: " + k.getClass().getName());  
 }  
}  
  
*// Параметризированный класс для работы с матрицами*public class Matrix<T extends Number> {  
 private T[][] data;  
  
 *// Конструктор класса Matrix* public Matrix(T[][] data) {  
 this.data = data;  
 }  
  
 *// Геттеры и сеттеры* public T[][] getData() {  
 return data;  
 }  
  
 public void setData(T[][] data) {  
 this.data = data;  
 }  
  
 *// Метод для печати матрицы* public void printMatrix() {  
 for (T[] row : data) {  
 for (T element : row) {  
 System.*out*.print(element + " ");  
 }  
 System.*out*.println();  
 }  
 }  
  
 *// Метод для сложения матриц* public Matrix<T> add(Matrix<T> other) {  
 if (data.length != other.data.length || data[0].length != other.data[0].length) {  
 throw new IllegalArgumentException("Matrices must have the same dimensions for addition.");  
 }  
  
 *// Создаем новую матрицу для результата* T[][] result = (T[][]) new Number[data.length][data[0].length];  
  
 *// Складываем соответствующие элементы* for (int i = 0; i < data.length; i++) {  
 for (int j = 0; j < data[0].length; j++) {  
 result[i][j] = (T) (Double) (data[i][j].doubleValue() + other.data[i][j].doubleValue());  
 }  
 }  
  
 return new Matrix<>(result);  
 }  
}

Листинг файла Matrix.java к заданиям №1-4

package ru.mirea.lab20;  
  
public class Main {  
 public static void main(String[] args) {  
 *// Пример использования CustomClass* CustomClass<Integer, Cat, String> customObject = new CustomClass<>(10, new Cat(), "Hello");  
 System.*out*.println("Value of T: " + customObject.getT());  
 System.*out*.println("Value of V: " + customObject.getV());  
 System.*out*.println("Value of K: " + customObject.getK());  
 customObject.printClassNames();  
  
 *// Пример использования Matrix* Integer[][] data1 = {{1, 2}, {3, 4}};  
 Integer[][] data2 = {{5, 6}, {7, 8}};  
  
 Matrix<Integer> matrix1 = new Matrix<>(data1);  
 Matrix<Integer> matrix2 = new Matrix<>(data2);  
  
 System.*out*.println("Matrix 1:");  
 matrix1.printMatrix();  
  
 System.*out*.println("Matrix 2:");  
 matrix2.printMatrix();  
  
 Matrix<Integer> sumMatrix = matrix1.add(matrix2);  
  
 System.*out*.println("Sum Matrix:");  
 sumMatrix.printMatrix();  
 }  
}

Листинг файла Main.java к заданиям №1-4

## Вывод

В данной практической работе мы изучали работу с дженериками в Java. Дженерики предоставляют средства для создания обобщенных типов, что позволяет писать код, не зависящий от конкретного типа данных.

Мы создали несколько классов и интерфейсов, используя дженерики, и рассмотрели применение:

* Cat и Animal: Классы, использующие дженерики, позволяющие работать с объектами различных типов.
* Nameable: Интерфейс, который требует, чтобы объекты имели метод getName(), что позволяет получать имя объекта.
* CustomClass: Параметризированный класс, принимающий три разных типа данных. Это демонстрирует гибкость и универсальность дженериков.
* Matrix: Параметризированный класс для работы с матрицами, демонстрирующий применение дженериков в реальной задаче.

Мы также реализовали методы для работы с объектами, проверяя их типы и выполняя операции согласно заданию.

Использование дженериков делает код более универсальным, позволяя работать с различными типами данных, без необходимости дублирования кода.

В результате этой практической работы, мы получили хороший опыт в работе с дженериками в Java, что позволит нам эффективно применять их в реальных проектах.

# Практическая работа № 21. Стирание типов в Джава и ваооывалоывало

**Цель работы:** научиться работать с обобщенными типами в Java и применять прием стирание типов разработке программ на Джава.

**Задания на практическую работу № 21:**

1. Написать метод для конвертации массива строк/чисел в список.

2. Написать класс, который умеет хранить в себе массив любых типов данных (int, long etc.).

3. Реализовать метод, который возвращает любой элемент массива по индексу

## Введение

В современном программировании, особенно в языках с сильной типизацией, важно понимать, как работают механизмы обработки типов данных. Одним из ключевых аспектов в Java является стирание типов, который иногда может вызывать некоторые недопонимания.

В данной практической работе мы глубоко погрузимся в концепцию стирания типов в Java. Мы исследуем, как компилятор Java управляет типами во время компиляции и исполнения программы, а также рассмотрим практические примеры и сценарии, где стирание типов играет важную роль.

## Практическая часть

package ru.mirea.lab21;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class ArrayToListConverter {  
 public static <T> List<T> convertArrayToList(T[] array) {  
 List<T> list = new ArrayList<>();  
 for (T element : array) {  
 list.add(element);  
 }  
 return list;  
 }  
  
 public static void main(String[] args) {  
 String[] stringArray = {"Hello", "World", "Java"};  
 List<String> stringList = convertArrayToList(stringArray);  
 System.*out*.println(stringList);  
  
 Integer[] intArray = {1, 2, 3, 4, 5};  
 List<Integer> intList = convertArrayToList(intArray);  
 System.*out*.println(intList);  
 }  
}

Листинг файла ArrayToListConverter.java к заданию №1

package ru.mirea.lab21;  
  
public class GenericArray<T> {  
 private T[] array;  
  
 public GenericArray(T[] array) {  
 this.array = array;  
 }  
  
 public T[] getArray() {  
 return array;  
 }  
  
 public void setArray(T[] array) {  
 this.array = array;  
 }  
}

Листинг файла GenericArray.java к заданию №2

package ru.mirea.lab21;  
  
public class ArrayElementGetter {  
 public static <T> T getElement(T[] array, int index) {  
 if (index >= 0 && index < array.length) {  
 return array[index];  
 } else {  
 throw new IndexOutOfBoundsException("Index out of bounds");  
 }  
 }  
}

Листинг файла ArrayElementGetter.java к заданию №3

package ru.mirea.lab21; *// Объявление пакета  
  
// Импорт классов из других файлов*import ru.mirea.lab21.ArrayElementGetter;  
import ru.mirea.lab21.ArrayToListConverter;  
import ru.mirea.lab21.GenericArray;  
  
import java.util.List; *// Импорт класса List из стандартной библиотеки Java*public class Main {  
 public static void main(String[] args) {  
 *// Проверка ArrayToListConverter  
  
 // Создание массива строк* String[] stringArray = {"Hello", "World", "Java"};  
  
 *// Конвертация массива строк в список* List<String> stringList = ArrayToListConverter.convertArrayToList(stringArray);  
  
 *// Вывод результата* System.*out*.println("ArrayToListConverter result:");  
 System.*out*.println(stringList);  
  
 *// Создание массива целых чисел* Integer[] intArray = {1, 2, 3, 4, 5};  
  
 *// Конвертация массива целых чисел в список* List<Integer> intList = ArrayToListConverter.convertArrayToList(intArray);  
  
 *// Вывод результата* System.*out*.println(intList);  
  
 *// Проверка GenericArray  
  
 // Создание массива целых чисел* Integer[] integerArray = {10, 20, 30};  
  
 *// Создание объекта GenericArray для хранения массива целых чисел* GenericArray<Integer> genericIntArray = new GenericArray<>(integerArray);  
  
 *// Вывод результата* System.*out*.println("\nGenericArray result:");  
 Integer[] resultIntArray = genericIntArray.getArray();  
 for (Integer num : resultIntArray) {  
 System.*out*.print(num + " ");  
 }  
  
 *// Создание массива строк* String[] strArray = {"One", "Two", "Three"};  
  
 *// Создание объекта GenericArray для хранения массива строк* GenericArray<String> genericStringArray = new GenericArray<>(strArray);  
  
 *// Вывод результата* System.*out*.println("\nGenericArray result:");  
 String[] resultStringArray = genericStringArray.getArray();  
 for (String str : resultStringArray) {  
 System.*out*.print(str + " ");  
 }  
  
 *// Проверка ArrayElementGetter  
  
 // Создание массива целых чисел* Integer[] intArray2 = {100, 200, 300, 400, 500};  
  
 *// Выбор элемента из массива по индексу* int index = 2;  
 try {  
 Integer element = ArrayElementGetter.getElement(intArray2, index);  
  
 *// Вывод результата* System.*out*.println("\n\nArrayElementGetter result:");  
 System.*out*.println("Element at index " + index + ": " + element);  
 } catch (IndexOutOfBoundsException e) {  
 *// Вывод ошибки, если индекс выходит за пределы массива* System.*out*.println(e.getMessage());  
 }  
 }  
}

Листинг файла Main.java к заданиям №1-3

## Вывод

В результате выполнения практической работы по теме "Стирание типов в Java" мы приобрели глубокое понимание того, как работает механизм стирания типов в этом языке программирования. Мы изучили, как компилятор Java преобразует обобщенные типы в их необобщенные аналоги и как это может влиять на работу программы.

Это знание окажется весьма полезным при разработке сложных приложений, где эффективная работа с типами данных является критически важным аспектом. Понимание стирания типов поможет нам избегать ошибок и написать более надежный и производительный код на Java.

# Практическая работа № 22. Абстрактные типы данных. Стеки ваооывалоывало

**Цель работы:** научиться разрабатывать программы с абстрактными типами данных на языке Джава и применять паттерн MVC при разработке программ.

**Задания на практическую работу № 22:**

1. Напишите графический интерфейс для калькулятора, используя знания полученные ранее при програмиировании GUI с использованием SWING и AWT. Используйте паттерн проектирования MVC.

2. Постройте систему тестов и проверьте, что ваш калькулятор успешно проходит все тесты и «защищён от дурака» (как дурака-пользователя программы, так и дурака-программиста, использующего ваш стек и калькулятор). Например, если вводится выражение, в котором число операций превосходит число помещенных в стек элементов (например 1 2 + \*), то программа не допустит уменьшения переменной sp до отрицательных значений, а выдаст предупреждение «Невозможно выполнить POP для пустого стека».

## Введение

В современном программировании абстрактные типы данных играют важную роль, позволяя разработчикам эффективно управлять и организовывать данные. Один из наиболее распространенных абстрактных типов данных — это стек. Стек представляет собой коллекцию элементов, упорядоченных по принципу "последним вошел - первым вышел" (LIFO). В этой практической работе мы будем изучать основы работы со стеком в языке Java, рассматривая его реализацию и применение.

## Практическая часть

package ru.mirea.lab22;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class Calculator implements ActionListener {  
 JFrame frame; *// Главное окно приложения* JTextField TextField; *// Поле для ввода/вывода чисел и результата* JTextField historyField; *// Поле для отображения истории операций* JButton[] numberButtons = new JButton[10]; *// Массив кнопок с цифрами* JButton[] functionButtons = new JButton[9]; *// Массив кнопок с операциями* JButton addButton,subButton,mulButton,divButton; *// Кнопки арифметических операций* JButton decButton, equButton, delButton, clrButton, negButton; *// Кнопки для десятичной точки, равно, удаления, очистки и смены знака* JPanel panel; *// Панель для размещения кнопок* Font myFont = new Font("Times New Roman", Font.*BOLD*, 30); *// Настройки шрифта* double num1=0,num2=0,result=0; *// Переменные для хранения чисел и результата* char operator; *// Переменная для хранения оператора* Calculator() {  
  
 *// Настройки главного окна приложения* frame = new JFrame("Calculator");  
 frame.setSize(420, 550);  
 frame.setLayout(null);  
 frame.setVisible(true);  
 frame.setResizable(false);  
  
 *// Поле для ввода/вывода чисел и результата* TextField = new JTextField();  
 TextField.setBounds(50, 45, 300, 50);  
 TextField.setFont(myFont);  
 TextField.setEditable(false); *// Не редактируется пользователем* TextField.setHorizontalAlignment(SwingConstants.*RIGHT*); *// Выравнивание текста по правому краю  
  
 // Поле для отображения истории операций* historyField = new JTextField();  
 historyField.setBounds(50, 15, 300, 25);  
 historyField.setEditable(false); *// Не редактируется пользователем* historyField.setBorder(null); *// Убрать границу поля* historyField.setHorizontalAlignment(SwingConstants.*RIGHT*); *// Выравнивание текста по правому краю* historyField.setFont(new Font("Times New Roman", Font.*PLAIN*, 20));  
 historyField.setForeground(Color.*DARK\_GRAY*);  
  
 *// Создание кнопок для операций* addButton = new JButton("+");  
 subButton = new JButton("-");  
 mulButton = new JButton("\*");  
 divButton = new JButton("/");  
 decButton = new JButton(".");  
 equButton = new JButton("=");  
 delButton = new JButton("X");  
 clrButton = new JButton("C");  
 negButton = new JButton("(-)");  
  
 *// Сохранение кнопок операций в массиве для удобства обработки* functionButtons[0] = addButton;  
 functionButtons[1] = subButton;  
 functionButtons[2] = mulButton;  
 functionButtons[3] = divButton;  
 functionButtons[4] = decButton;  
 functionButtons[5] = equButton;  
 functionButtons[6] = delButton;  
 functionButtons[7] = clrButton;  
 functionButtons[8] = negButton;  
  
 *// Настройка свойств кнопок операций* for (int i = 0; i < 9; i++) {  
 functionButtons[i].addActionListener(this);  
 functionButtons[i].setFont(myFont);  
 functionButtons[i].setFocusable(false); *// Убрать фокус (не подсвечиваться)* }  
  
 *// Создание кнопок с цифрами и настройка их свойств* for (int i = 0; i < 10; i++) {  
 numberButtons[i] = new JButton(String.valueOf(i));  
 numberButtons[i].addActionListener(this);  
 numberButtons[i].setFont(myFont);  
 numberButtons[i].setFocusable(false); *// Убрать фокус (не подсвечиваться)* }  
  
 *// Расположение кнопок на панели* negButton.setBounds(50, 430, 100, 50);  
 delButton.setBounds(150, 430, 100, 50);  
 clrButton.setBounds(250, 430, 100, 50);  
  
 *// Создание панели для кнопок и настройка её расположения* panel = new JPanel();  
 panel.setBounds(50, 100, 300, 300);  
 panel.setLayout(new GridLayout(4,4,10,10));  
  
 *// Добавление кнопок и полей на панель* panel.add(numberButtons[1]);  
 panel.add(numberButtons[2]);  
 panel.add(numberButtons[3]);  
 panel.add(addButton);  
 panel.add(numberButtons[4]);  
 panel.add(numberButtons[5]);  
 panel.add(numberButtons[6]);  
 panel.add(subButton);  
 panel.add(numberButtons[7]);  
 panel.add(numberButtons[8]);  
 panel.add(numberButtons[9]);  
 panel.add(mulButton);  
 panel.add(decButton);  
 panel.add(numberButtons[0]);  
 panel.add(equButton);  
 panel.add(divButton);  
  
 *// Установка расположения компонентов в окне* frame.setLocationRelativeTo(null);  
 frame.add(panel);  
 frame.add(negButton);  
 frame.add(delButton);  
 frame.add(clrButton);  
 frame.add(TextField);  
 frame.add(historyField);  
 frame.setVisible(true);  
  
 frame.repaint(); *// Перерисовать окно* }  
  
 public static void main(String[] args) {  
 Calculator calc = new Calculator();  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 *// Обработка нажатий на кнопки с цифрами* for (int i = 0; i < 10; i++) {  
 if (e.getSource() == numberButtons[i]) {  
 TextField.setText(TextField.getText().concat(String.valueOf(i)));  
 }  
 }  
  
 *// Обработка нажатия на кнопку с десятичной точкой* if (e.getSource() == decButton) {  
 String currentText = TextField.getText();  
 if (!currentText.contains(".")) {  
 TextField.setText(currentText + ".");  
 }  
 }  
  
 *// Обработка нажатия на кнопку сложения* if (e.getSource() == addButton) {  
 num1 = Double.parseDouble(TextField.getText());  
 operator = '+';  
 TextField.setText("");  
 historyField.setText(num1 + " + ");  
 }  
  
 *// Обработка нажатия на кнопку вычитания* if (e.getSource() == subButton) {  
 num1 = Double.parseDouble(TextField.getText());  
 operator = '-';  
 TextField.setText("");  
 historyField.setText(num1 + " - ");  
 }  
  
 *// Обработка нажатия на кнопку умножения* if (e.getSource() == mulButton) {  
 num1 = Double.parseDouble(TextField.getText());  
 operator = '\*';  
 TextField.setText("");  
 historyField.setText(num1 + " \* ");  
 }  
  
 *// Обработка нажатия на кнопку деления* if (e.getSource() == divButton) {  
 num1 = Double.parseDouble(TextField.getText());  
 operator = '/';  
 TextField.setText("");  
 historyField.setText(num1 + " / ");  
 }  
  
 *// Обработка нажатия на кнопку равенства* if (e.getSource() == equButton) {  
 if (TextField.getText().isEmpty()) {  
 historyField.setText("");  
 } else {  
 num2 = Double.parseDouble(TextField.getText());  
 *// Выполнение операции в зависимости от выбранного оператора* switch (operator) {  
 case '+' -> result = num1 + num2;  
 case '-' -> result = num1 - num2;  
 case '\*' -> result = num1 \* num2;  
 case '/' -> result = num1 / num2;  
 }  
  
 *// Вывод результата истории операций* TextField.setText(String.valueOf(result));  
 historyField.setText(historyField.getText().concat(String.valueOf(num2 + " =")));  
 num1 = result; *// Сохранение результата для дальнейших вычислений* }  
 }  
  
 *// Обработка нажатия на кнопку очистки* if (e.getSource() == clrButton) {  
 TextField.setText("");  
 historyField.setText("");  
 }  
  
 *// Обработка нажатия на кнопку удаления последнего символа* if (e.getSource() == delButton) {  
 String string = TextField.getText();  
 TextField.setText("");  
 for (int i = 0; i < string.length() - 1; i++) {  
 TextField.setText(TextField.getText() + string.charAt(i));  
 }  
 }  
  
 *// Обработка нажатия на кнопку смены знака* if (e.getSource() == negButton) {  
 if (TextField.getText().length() == 0) {  
 TextField.setText("-");  
 } else {  
 double temp = Double.parseDouble(TextField.getText());  
 temp \*= -1;  
 TextField.setText(String.valueOf(temp));  
 }  
 }  
 }  
}

Листинг файла Calculator.java к заданиям №1-2

package ru.mirea.lab22;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class CalculatorTest {  
  
 @Test  
 public void testAddition() {  
 Calculator calc = new Calculator();  
 calc.TextField.setText("5");  
 calc.addButton.doClick();  
 calc.TextField.setText("3");  
 calc.equButton.doClick();  
 assertEquals("8.0", calc.TextField.getText());  
 }  
  
 @Test  
 public void testSubtraction() {  
 Calculator calc = new Calculator();  
 calc.TextField.setText("10");  
 calc.subButton.doClick();  
 calc.TextField.setText("5");  
 calc.equButton.doClick();  
 assertEquals("5.0", calc.TextField.getText());  
 }  
  
 @Test  
 public void testMultiplication() {  
 Calculator calc = new Calculator();  
 calc.TextField.setText("4");  
 calc.mulButton.doClick();  
 calc.TextField.setText("2");  
 calc.equButton.doClick();  
 assertEquals("8.0", calc.TextField.getText());  
 }  
  
 @Test  
 public void testDivision() {  
 Calculator calc = new Calculator();  
 calc.TextField.setText("10");  
 calc.divButton.doClick();  
 calc.TextField.setText("2");  
 calc.equButton.doClick();  
 assertEquals("5.0", calc.TextField.getText());  
 }  
  
 @Test  
 public void testInvalidInput() {  
 Calculator calc = new Calculator();  
 calc.TextField.setText("5");  
 calc.addButton.doClick();  
 calc.equButton.doClick();  
 assertEquals("", calc.TextField.getText());  
 }  
  
 @Test  
 public void testClearButton() {  
 Calculator calc = new Calculator();  
 calc.TextField.setText("5");  
 calc.addButton.doClick();  
 calc.TextField.setText("3");  
 calc.clrButton.doClick();  
 assertEquals("", calc.TextField.getText());  
 }  
  
 @Test  
 public void testNegationButton() {  
 Calculator calc = new Calculator();  
 calc.TextField.setText("5");  
 calc.negButton.doClick();  
 assertEquals("-5.0", calc.TextField.getText());  
 }  
  
 @Test  
 public void testDecimalPoint() {  
 Calculator calc = new Calculator();  
 calc.decButton.doClick();  
 calc.TextField.setText("3");  
 calc.decButton.doClick();  
 calc.TextField.setText("1.5");  
 calc.decButton.doClick();  
 assertEquals("1.5", calc.TextField.getText());  
 }  
}

Листинг файла CalculatorTest.java к заданиям №1-2

## Вывод

В ходе выполнения данной практической работы мы углубили свои знания в области абстрактных типов данных, сосредотачиваясь на стеке. Мы изучили принципы его работы, основные операции и методы, поняли применение стека в реальных сценариях программирования.

Знание работы с абстрактными типами данных, такими как стек, является важным компонентом навыков любого разработчика. Стек находит свое применение во многих областях программирования, от алгоритмов до структур данных.

# Практическая работа № 23. Абстрактные типы данных. Очередь.

**Цель работы:** научиться разрабатывать программы с абстрактными типами данных на языке Джава

**Задания на практическую работу № 23:**

Найдите инвариант структуры данных «очередь». Определите функции, которые необходимы для реализации очереди. Найдите их пред- и постусловия.

Реализуйте классы, представляющие циклическую очередь с применением массива.

Класс ArrayQueueModule должен реализовывать один экземпляр очереди с использованием переменных класса.

Класс ArrayQueueADT должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).

Класс ArrayQueue должен реализовывать очередь в виде класса (с

неявной передачей ссылки на экземпляр очереди).

Должны быть реализованы следующие функции(процедуры)/методы:

enqueue – добавить элемент в очередь;

element – первый элемент в очереди;

dequeue – удалить и вернуть первый элемент в очереди;

size – текущий размер очереди;

isEmpty – является ли очередь пустой;

clear – удалить все элементы из очереди.

Обратите внимание на инкапсуляцию данных и кода во всех трех реализациях.

Напишите тесты реализованным классам.

## Введение

Абстрактные типы данных (АТД) представляют собой основополагающий концептуальный инструмент в программировании, позволяющий абстрагироваться от деталей реализации и сосредотачиваться на операциях, выполняемых с данными. Одним из важнейших АТД является структура данных "Очередь". Очередь представляет собой коллекцию элементов, в которой новые элементы добавляются в конец, а извлекаются из начала, работая по принципу "первым пришёл — первым вышел" (FIFO).

## Практическая часть

package ru.mirea.lab23;  
  
public class ArrayQueue {  
 private int[] queue;  
 private int front;  
 private int rear;  
 private int size;  
  
 *// Создать новую очередь* public ArrayQueue() {  
 queue = new int[10];  
 front = 0;  
 rear = -1;  
 size = 0;  
 }  
  
 *// Добавить элемент в очередь* public void enqueue(int element) {  
 if (size == queue.length)  
 resize();  
 rear = (rear + 1) % queue.length;  
 queue[rear] = element;  
 size++;  
 }  
  
 *// Получить первый элемент в очереди* public int element() {  
 if (isEmpty())  
 throw new IllegalStateException("Queue is empty");  
 return queue[front];  
 }  
  
 *// Удалить и вернуть первый элемент в очереди* public int dequeue() {  
 if (isEmpty())  
 throw new IllegalStateException("Queue is empty");  
 int removedElement = queue[front];  
 front = (front + 1) % queue.length;  
 size--;  
 return removedElement;  
 }  
  
 *// Получить текущий размер очереди* public int size() {  
 return size;  
 }  
  
 *// Проверить, является ли очередь пустой* public boolean isEmpty() {  
 return size == 0;  
 }  
  
 *// Очистить все элементы из очереди* public void clear() {  
 queue = new int[10];  
 front = 0;  
 rear = -1;  
 size = 0;  
 }  
  
 *// Увеличить размер массива при необходимости* private void resize() {  
 int[] newQueue = new int[queue.length \* 2];  
 int index = 0;  
 for (int i = front; i <= rear; i++) {  
 newQueue[index++] = queue[i % queue.length];  
 }  
 queue = newQueue;  
 front = 0;  
 rear = size - 1;  
 }  
}

Листинг файла ArrayQueue.java к заданию №1

package ru.mirea.lab23;  
  
public class ArrayQueueADT {  
 private int[] queue;  
 private int front;  
 private int rear;  
 private int size;  
  
 *// Создать новую очередь* public static ArrayQueueADT createQueue() {  
 ArrayQueueADT newQueue = new ArrayQueueADT();  
 newQueue.queue = new int[10];  
 newQueue.front = 0;  
 newQueue.rear = -1;  
 newQueue.size = 0;  
 return newQueue;  
 }  
  
 *// Добавить элемент в очередь* public void enqueue(int element) {  
 if (size == queue.length)  
 resize();  
 rear = (rear + 1) % queue.length;  
 queue[rear] = element;  
 size++;  
 }  
  
 *// Получить первый элемент в очереди* public int element() {  
 if (isEmpty())  
 throw new IllegalStateException("Queue is empty");  
 return queue[front];  
 }  
  
 *// Удалить и вернуть первый элемент в очереди* public int dequeue() {  
 if (isEmpty())  
 throw new IllegalStateException("Queue is empty");  
 int removedElement = queue[front];  
 front = (front + 1) % queue.length;  
 size--;  
 return removedElement;  
 }  
  
 *// Получить текущий размер очереди* public int size() {  
 return size;  
 }  
  
 *// Проверить, является ли очередь пустой* public boolean isEmpty() {  
 return size == 0;  
 }  
  
 *// Очистить все элементы из очереди* public void clear() {  
 queue = new int[10];  
 front = 0;  
 rear = -1;  
 size = 0;  
 }  
  
 *// Увеличить размер массива при необходимости* private void resize() {  
 int[] newQueue = new int[queue.length \* 2];  
 int index = 0;  
 for (int i = front; i <= rear; i++) {  
 newQueue[index++] = queue[i % queue.length];  
 }  
 queue = newQueue;  
 front = 0;  
 rear = size - 1;  
 }  
}

Листинг файла ArrayQueueADT.java к заданию №1

package ru.mirea.lab23;  
  
import java.util.Arrays;  
  
public class ArrayQueueModule {  
 private static int[] *queue* = new int[10];  
 private static int *front* = 0;  
 private static int *rear* = -1;  
 private static int *size* = 0;  
  
 *// Добавить элемент в очередь* public static void enqueue(int element) {  
 if (*size* == *queue*.length)  
 resize();  
 *rear* = (*rear* + 1) % *queue*.length;  
 *queue*[*rear*] = element;  
 *size*++;  
 }  
  
 *// Получить первый элемент в очереди* public static int element() {  
 if (isEmpty())  
 throw new IllegalStateException("Queue is empty");  
 return *queue*[*front*];  
 }  
  
 *// Удалить и вернуть первый элемент в очереди* public static int dequeue() {  
 if (isEmpty())  
 throw new IllegalStateException("Queue is empty");  
 int removedElement = *queue*[*front*];  
 *front* = (*front* + 1) % *queue*.length;  
 *size*--;  
 return removedElement;  
 }  
  
 *// Получить текущий размер очереди* public static int size() {  
 return *size*;  
 }  
  
 *// Проверить, является ли очередь пустой* public static boolean isEmpty() {  
 return *size* == 0;  
 }  
  
 *// Очистить все элементы из очереди* public static void clear() {  
 *queue* = new int[10];  
 *front* = 0;  
 *rear* = -1;  
 *size* = 0;  
 }  
  
 *// Увеличить размер массива при необходимости* private static void resize() {  
 int[] newQueue = new int[*queue*.length \* 2];  
 int index = 0;  
 for (int i = *front*; i <= *rear*; i++) {  
 newQueue[index++] = *queue*[i % *queue*.length];  
 }  
 *queue* = newQueue;  
 *front* = 0;  
 *rear* = *size* - 1;  
 }  
}

Листинг файла ArrayQueueModule.java к заданию №1

package ru.mirea.lab23;  
  
import static org.junit.Assert.\*;  
import org.junit.\*;  
  
public class TestArrayQueue {  
  
 private ArrayQueue queue;  
  
 @Before  
 public void setUp() {  
 queue = new ArrayQueue();  
 }  
  
 @Test  
 public void testEnqueueAndSize() {  
 for (int i = 0; i < 5; i++) {  
 queue.enqueue(i);  
 }  
 assertEquals(5, queue.size());  
 }  
  
 @Test  
 public void testDequeue() {  
 queue.enqueue(42);  
 int result = queue.dequeue();  
 assertEquals(42, result);  
 assertTrue(queue.isEmpty());  
 }  
  
 @Test(expected = IllegalStateException.class)  
 public void testDequeueOnEmptyQueue() {  
 queue.dequeue();  
 }  
  
 @Test  
 public void testElement() {  
 queue.enqueue(42);  
 int result = queue.element();  
 assertEquals(42, result);  
 assertEquals(1, queue.size());  
 }  
  
 @Test(expected = IllegalStateException.class)  
 public void testElementOnEmptyQueue() {  
 queue.element();  
 }  
  
 @Test  
 public void testClear() {  
 for (int i = 0; i < 5; i++) {  
 queue.enqueue(i);  
 }  
 queue.clear();  
 assertTrue(queue.isEmpty());  
 }  
  
 @Test  
 public void testIsEmpty() {  
 assertTrue(queue.isEmpty());  
 queue.enqueue(42);  
 assertFalse(queue.isEmpty());  
 }  
}

Листинг файла TestArrayQueue.java к заданию №1

package ru.mirea.lab23;  
  
import static org.junit.Assert.\*;  
import org.junit.\*;  
  
public class TestArrayQueueADT {  
  
 private ArrayQueueADT queue;  
  
 @Before  
 public void setUp() {  
 queue = ArrayQueueADT.createQueue();  
 }  
  
 @Test  
 public void testEnqueueAndSize() {  
 for (int i = 0; i < 5; i++) {  
 queue.enqueue(i);  
 }  
 assertEquals(5, queue.size());  
 }  
  
 @Test  
 public void testDequeue() {  
 queue.enqueue(42);  
 int result = queue.dequeue();  
 assertEquals(42, result);  
 assertTrue(queue.isEmpty());  
 }  
  
 @Test(expected = IllegalStateException.class)  
 public void testDequeueOnEmptyQueue() {  
 queue.dequeue();  
 }  
  
 @Test  
 public void testElement() {  
 queue.enqueue(42);  
 int result = queue.element();  
 assertEquals(42, result);  
 assertEquals(1, queue.size());  
 }  
  
 @Test(expected = IllegalStateException.class)  
 public void testElementOnEmptyQueue() {  
 queue.element();  
 }  
  
 @Test  
 public void testClear() {  
 for (int i = 0; i < 5; i++) {  
 queue.enqueue(i);  
 }  
 queue.clear();  
 assertTrue(queue.isEmpty());  
 }  
  
 @Test  
 public void testIsEmpty() {  
 assertTrue(queue.isEmpty());  
 queue.enqueue(42);  
 assertFalse(queue.isEmpty());  
 }  
}

Листинг файла TestArrayQueueADT.java к заданию №1

package ru.mirea.lab23;  
  
import static org.junit.Assert.\*;  
import org.junit.\*;  
  
public class TestArrayQueueModule {  
  
 @Before  
 public void setUp() {  
 ArrayQueueModule.clear();  
 }  
  
 @Test  
 public void testEnqueueAndSize() {  
 for (int i = 0; i < 5; i++) {  
 ArrayQueueModule.enqueue(i);  
 }  
 assertEquals(5, ArrayQueueModule.size());  
 }  
  
 @Test  
 public void testDequeue() {  
 ArrayQueueModule.enqueue(42);  
 int result = ArrayQueueModule.dequeue();  
 assertEquals(42, result);  
 assertTrue(ArrayQueueModule.isEmpty());  
 }  
  
 @Test(expected = IllegalStateException.class)  
 public void testDequeueOnEmptyQueue() {  
 ArrayQueueModule.dequeue();  
 }  
  
 @Test  
 public void testElement() {  
 ArrayQueueModule.enqueue(42);  
 int result = ArrayQueueModule.element();  
 assertEquals(42, result);  
 assertEquals(1, ArrayQueueModule.size());  
 }  
  
 @Test(expected = IllegalStateException.class)  
 public void testElementOnEmptyQueue() {  
 ArrayQueueModule.element();  
 }  
  
 @Test  
 public void testClear() {  
 for (int i = 0; i < 5; i++) {  
 ArrayQueueModule.enqueue(i);  
 }  
 ArrayQueueModule.clear();  
 assertTrue(ArrayQueueModule.isEmpty());  
 }  
  
 @Test  
 public void testIsEmpty() {  
 assertTrue(ArrayQueueModule.isEmpty());  
 ArrayQueueModule.enqueue(42);  
 assertFalse(ArrayQueueModule.isEmpty());  
 }  
}

Листинг файла TestArrayQueueModule.java к заданию №1

## Вывод

В ходе данной практической работы мы подробно рассмотрели абстрактные типы данных и сфокусировали внимание на структуре данных "Очередь". Мы изучили основные операции с этим типом данных, такие как добавление элемента (enqueue) и извлечение элемента (dequeue), а также рассмотрели специфичные случаи работы с пустой и полной очередью.

Этот опыт позволил нам лучше понять принципы работы с абстрактными типами данных и их важность в разработке эффективных программ. Знание структур данных позволяет выбирать наиболее подходящий инструмент для решения конкретных задач, что существенно повышает эффективность и читаемость кода.

# Практическая работа № 24. Паттерны проектирования. порождающие паттерны: абстрактная фабрика, фабричный метод

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

**Задания на практическую работу № 24:**

1. 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).

## Введение

В мире программирования существует бесчисленное множество проблем, которые разработчики решают каждый день. Однако, иногда для эффективного решения сложных задач требуется более высокий уровень абстракции. В таких случаях на помощь приходят паттерны проектирования - bewährte (проверенные временем) архитектурные решения, позволяющие создавать гибкий, расширяемый и поддерживаемый код.

В данной практической работе мы сосредотачиваем внимание на двух порождающих паттернах: абстрактная фабрика и фабричный метод. Эти паттерны обеспечивают эффективный способ создания объектов, позволяя абстрагироваться от деталей их конкретной реализации. Рассмотрим основные принципы и применение этих паттернов, а также рассмотрим конкретные примеры их использования на языке Java.

## Практическая часть

package ru.mirea.lab24;  
  
public abstract class AbstractChairFactory {  
 public abstract Chair createVictorianChair();  
  
 public abstract Chair createMultifunctionalChair();  
  
 public abstract Chair createMagicChair();  
}

Листинг файла AbstractChairFactory.java к заданию №1

package ru.mirea.lab24;  
  
public interface Chair {  
 void sit();  
}

Листинг файла Chair.java к заданию №1

package ru.mirea.lab24; *// Определение пакета ru.mirea.lab24.*public class Client {  
 *// Объявление класса Client.* public void Sit(Chair chair) {  
 *// Метод Sit принимает объект стула в качестве параметра.* chair.sit();  
 *// Вызываем метод sit() для сидения на переданном стуле.* }  
}

Листинг файла Client.java к заданию №1

package ru.mirea.lab24;  
  
public class MagicChair implements Chair {  
 @Override  
 public void sit() {  
 System.*out*.println("Вы сели на магический стул");  
 }  
}

Листинг файла MagicChair.java к заданию №1

package ru.mirea.lab24;  
  
public class ModernChairFactory extends AbstractChairFactory {  
 @Override  
 public Chair createVictorianChair() {  
 return new VictorianChair();  
 }  
  
 @Override  
 public Chair createMultifunctionalChair() {  
 return new MultifunctionalChair();  
 }  
  
 @Override  
 public Chair createMagicChair() {  
 return new MagicChair();  
 }  
}

Листинг файла ModernChairFactory.java к заданию №1

package ru.mirea.lab24;  
  
public class MultifunctionalChair implements Chair {  
 @Override  
 public void sit() {  
 System.*out*.println("Вы сели на многофункциональный стул");  
 }  
}

Листинг файла MultifunctionalChair.java к заданию №1

package ru.mirea.lab24;  
  
public class VictorianChair implements Chair {  
 @Override  
 public void sit() {  
 System.*out*.println("Вы сели на викторианский стул");  
 }  
}

Листинг файла VictorianChair.java к заданию №1

package ru.mirea.lab24;  
  
*// Импортируем пакет ru.mirea.lab24 для доступа к классам этого пакета.*public class Main {  
 public static void main(String[] args) {  
 *// Создаем экземпляр фабрики стульев современного стиля.* AbstractChairFactory factory = new ModernChairFactory();  
  
 *// Создаем стулья разных типов с помощью фабрики.* Chair victorianChair = factory.createVictorianChair();  
 Chair multifunctionalChair = factory.createMultifunctionalChair();  
 Chair magicChair = factory.createMagicChair();  
  
 *// Создаем клиента.* Client client = new Client();  
  
 *// Клиент садится на каждый из стульев.* client.Sit(victorianChair);  
 client.Sit(multifunctionalChair);  
 client.Sit(magicChair);  
 }  
}

Листинг файла Main.java к заданию №1

## Вывод

В ходе данной практической работы мы подробно изучили два важнейших порождающих паттерна: абстрактную фабрику и фабричный метод. Они предоставляют элегантные решения для создания объектов в зависимости от контекста и требований приложения.

Абстрактная фабрика позволяет создавать семейства взаимосвязанных объектов, обеспечивая высокую степень абстракции и уровень гибкости. Этот паттерн особенно полезен в ситуациях, где необходимо поддерживать разные варианты системы.

Фабричный метод, в свою очередь, делегирует ответственность за создание объектов подклассам, позволяя им самим определять конкретные типы создаваемых объектов. Это способствует повышению гибкости и расширяемости кода.

Понимание и умение применять эти паттерны является важным инструментом для разработчика, позволяя создавать высококачественные и поддерживаемые приложения.