МИНИСТЕРСТВО ТРАНСПОРТА И КОММУНИКАЦИЙ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

Кафедра «Микропроцессорная техника и информационно-управляющие системы»

К.Ф ИЗМАЙЛОВ

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КРОСС-ПЛАТФОРМЕНЫЫХ ПРИЛОЖЕНИЙ

Учебно-методическое пособие по выполнению лабораторных работ для студентов электротехнического факультета

Гомель 2016

ВВЕДЕНИЕ

Основной задачей дисциплины «Программное обеспечение кросс-платформенных приложений» служит обучение студентов языку программирования Java. Предметной областью данной дисциплины является информационная технология, которая не может существовать без компьютерной техники, поэтому изучение этой дисциплины возможно только на современной компьютерной технике с использованием самых современных технологий.

В данном курсе рассматриваются методы разработки программного обеспечения универсального для работы на различных платформах. В современном мире вычислительных систем с каждым днем становится все больше. Целые классы объектов действительности, которые никогда не являлись вычислительными, получают возможность сбора и анализа данных, например часы или обувь. Зачастую они являются носимыми и имеют автономное питание. С другой стороны нарастает мощность встраиваемых и стационарных систем. От современного разработчика требуется понимание механизмов работы огромного числа устройств, зачастую никак не взаимодействующих друг с другом.

Очевидным является использование кроссплатформенных решений, позволяющих вести разработку программного обеспечения для широкого спектра устройств.

В данном пособии рассматриваются основные вопросы объектно-ориентированного программирования на языке программирования Java. Для изучения этих аспектов студентам предлагается выполнить ряд лабораторных работ.

1 ОСНОВЫ ПРОГРАММИРОВАНИЯ JAVA

1.1 Создание проекта в IntelliJ IDEA

1.1.1 Введение в IDE IntelliJ IDEA

IntelliJ IDEA — интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

IDE доступна в двух редакциях: коммерческая Ultimate и бесплатная Community Edition. Бесплатная версия среды разработки на основе открытого кода включает в себя:

- Поддержка языков Java SE, Groovy, Scala, Clojure и Erlang.
- Профессиональный набор инструментов для разработки Android-приложений.
- Поддержка JavaFX 2.0, интеграция с SceneBuilder; Дизайнер интерфейса для Swing.
- Интеграция с автоматизированными инструментами сборки и управления проектом, включая Maven, Gradle, Ant и другими.
- Инструменты для тестирования с поддержкой JUnit, TestNG, Spock, ScalaTest и spec2.
- Интеграция с системами управления версиями, включая Git, Subversion, Mercurial и CSV.

1.1.2 Порядок создания проекта консольного приложения.

Запустить IntelliJ IDEA Community Edition из меню пуск, или воспользовавшись ярлыком на рабочем столе.

Рисунок 1 – Окно приветствия

Запустить диалог создания нового проекта нажав кнопку Create New Project (Рисунок 1).

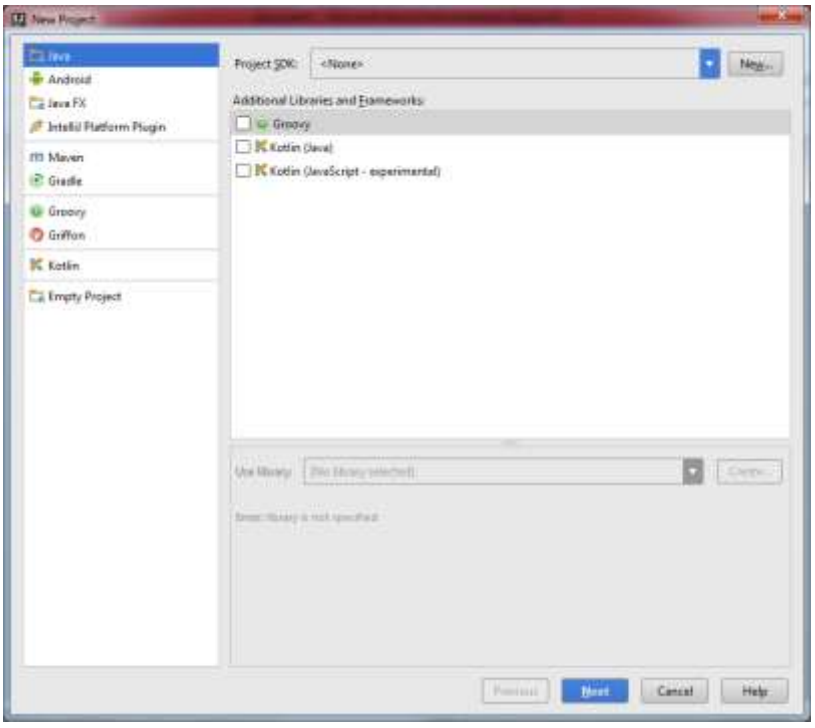

Рисунок 2 – Диалоговое окно создания проект

В открывшемся окне (Рисунок 2) выбрать тип проекта Java, затем необходимо выбрать версию SDK с которым мы будем работать.

Нажимаем кнопку New в строке Project SDK. В выпадающем списке (Рисунок 3) выбираем пункт JDK.

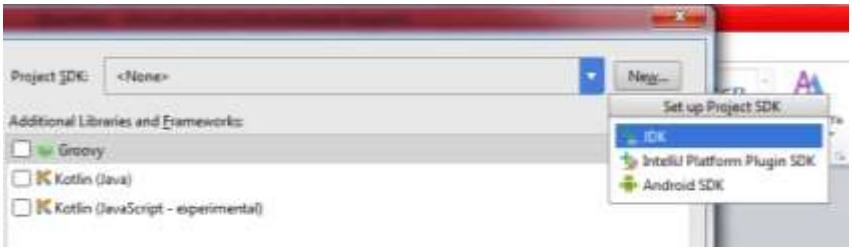

Рисунок 3 – Выпадающий список выбора SDK

Далее необходимо выбрать директорию с установленным Java SE Development Kit. Для версии Java 8 в операционной системе Windows путь по умолчанию к директории будет следующий: **C:\Program Files\Java\jdk1.8.0_72**, цифра 72 означает версию сборки и меняется в зависимости от установленной версии Java. Цифра 8 означает версию языка, например, для Java 7, директория будет называться jdk1.7.0_xx.

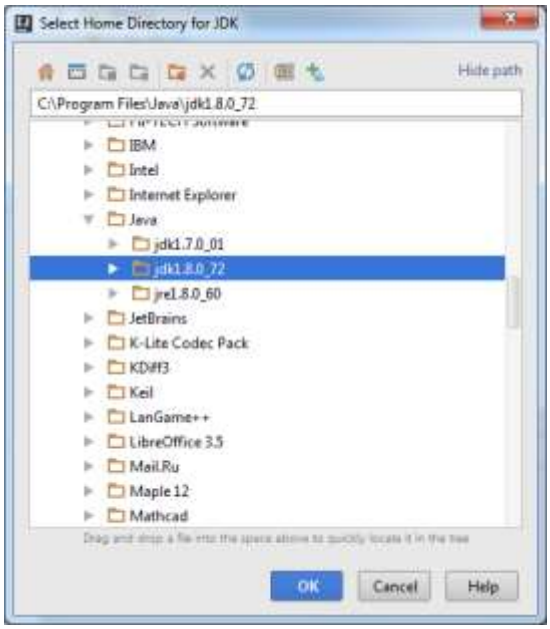

Рисунок 4 – Выбор директории с установленным JDK

Нажимаем OK.

После того как в поле Project SDK появится выбранная нами версия JDK нажимаем Next.

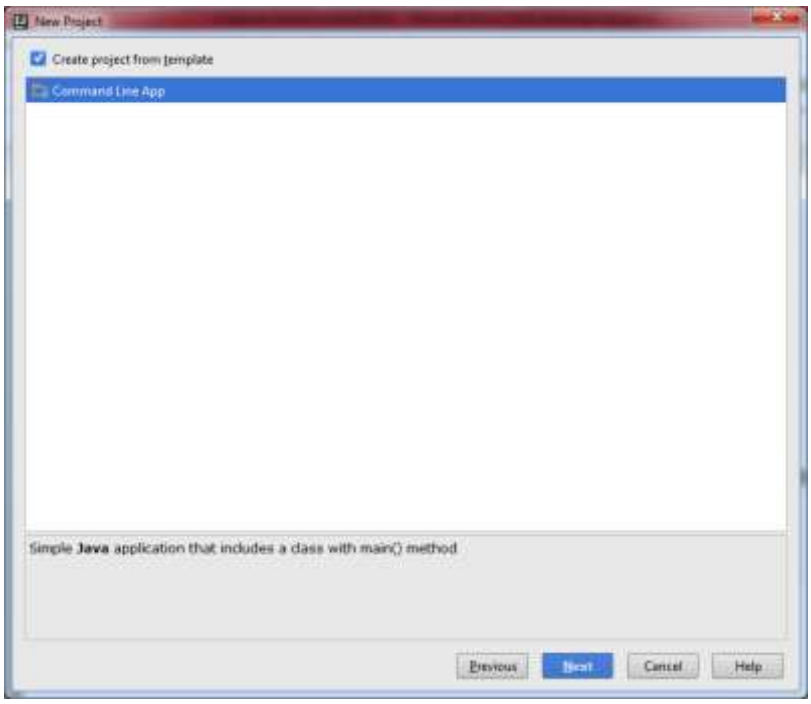

Рисунок 5 – Окно выбора шаблона

В появившемся окне (Рисунок 5) ставим галочку Create project from template и выбираем Command Line App (консольное приложение) и жмем Next.

В следующем окне (Рисунок 6) задаем название проекта и место его сохранения.

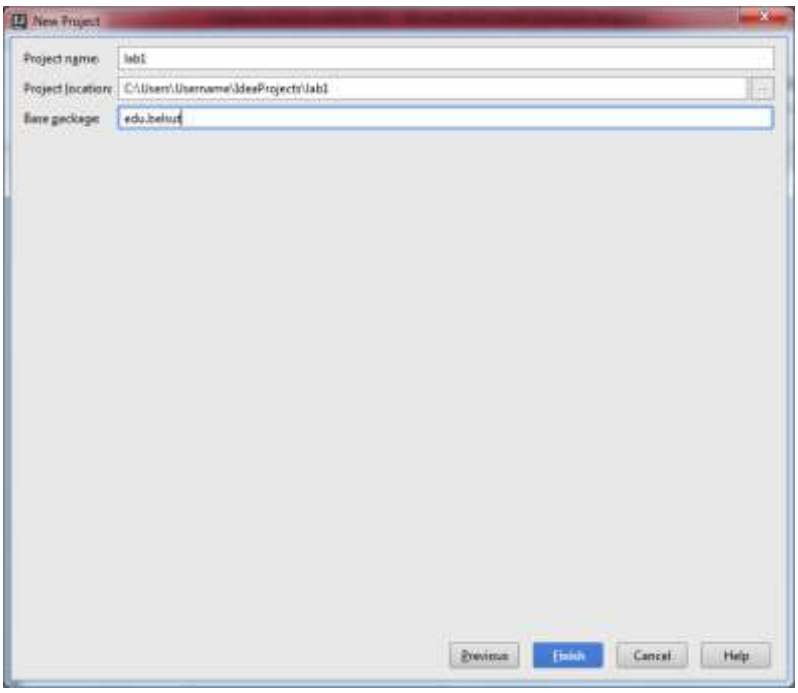

Рисунок 6 – Задание названия и места сохранения проекта

Кроме названия проекта и места его сохранения на диске необходимо указать название пакета. Пакет (package) — это некий контейнер, который используется для того, чтобы изолировать имена классов. Например, вы можете создать класс List, заключить его в пакет и не думать после этого о возможных конфликтах, которые могли бы возникнуть, если бы кто-нибудь еще создал класс с именем List. Обычно в одном пакете рсполагают классы, которые взаимосвязаны или имеют одного и того же автора. Каждый пакет, как и папка, имеет свое имя, притом в Java реально создаются папки под ваши пакеты с совпадающим именем. Так как пакеты полностью ассоциируются с папками, то у них существует вложенность, она показывается с помощью точки, например «main.slave» будет означать, что пакет с именем slave находится в пакете main, то же самое произойдет и с соответствующими папками.

В Java принято именовать пакеты, которые вы разрабатываете, также как ваш личный сайт или сайт фирмы, в которой вы работаете, записанный задом наперед.

Далее нам могут предложить создать директорию под проект, соглашаемся, нажав OK.

После этого мы увидим основное окно IntelliJ IDEA. Некоторое время займет процесс индексирования системой файлов вашего проекта для последующего быстрого поиска по проекту.

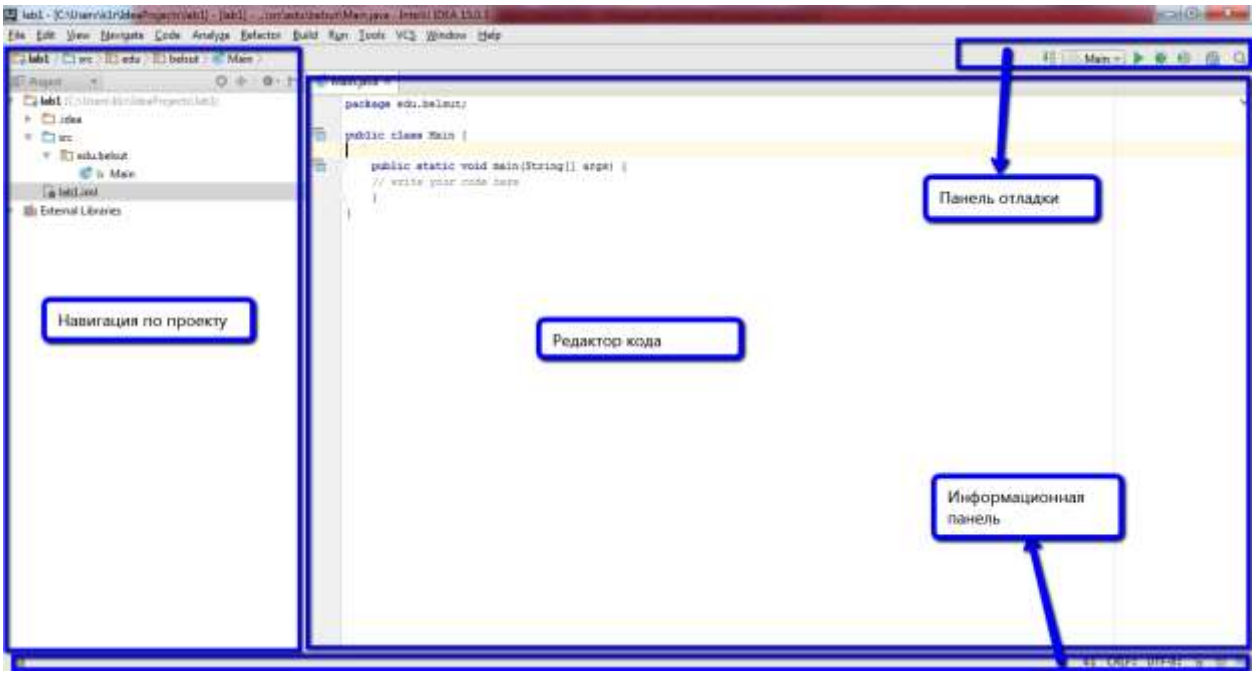

Рисунок 7 – Основное окно IntelliJ IDEA

IntelliJ IDEA достаточно гибкая среда разработки с точки зрения организации рабочего места пользователя. Опытный разработчик может настроить расположение всех частей программы для себя.

По умолчанию пользователю отображается определенный набор окон (Рисунок 7).

1.2 Создание нового класса в Intellij idea

Для создания нового класса воспользуемся встроенными диалоговыми возможностями среды. Нажмем правой кнопкой мыши на интересующем нас пакете (новый класс будет создан именно в нем) и выберем пункт New, а затем Java Class (Рисунок 8).

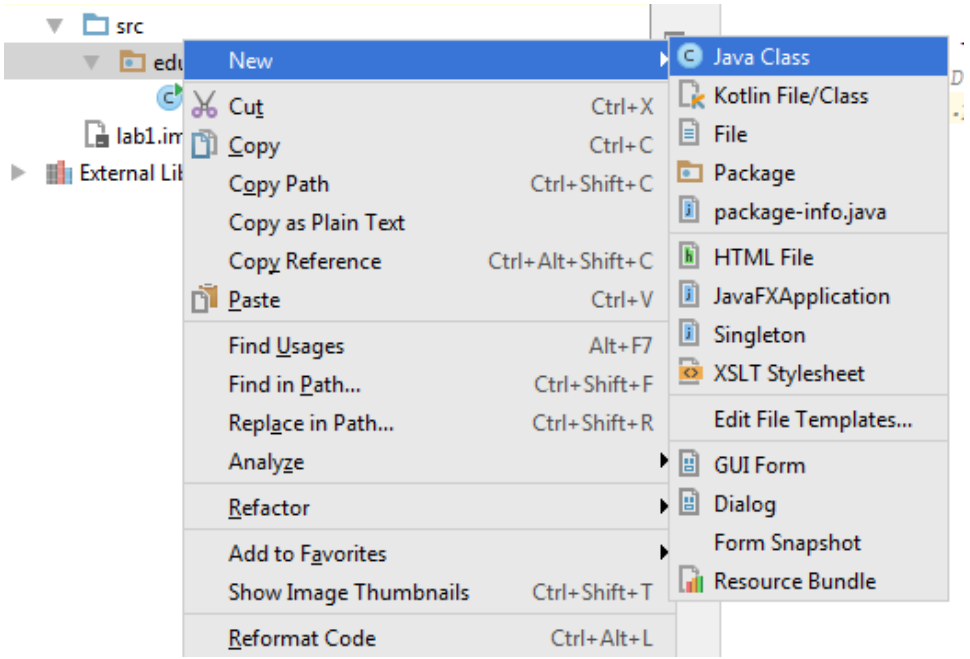

Рисунок 8 – Создание нового класса в пакете

Далее появится диалоговое окно, в котором необходимо ввести имя класса. В Java существуют достаточно строгие предписания по оформлению кода, в том числе и названия классов, полей и методов классов.

Название класса должно начинаться с заглавной буквы, например класс Developer. Если название класса, поля либо метода класса состоит из нескольких слов, необходимо использовать так называемый **верблюжий регистр** (Camel Case).

Camel Case, он же — Верблюжий Регистр, он же — Горбатый Стиль – стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово пишется с заглавной буквы. Стиль получил название Camel Case, поскольку заглавные буквы внутри слова напоминают горбы верблюда.

Для названия классов используют UpperCamelCase (первая буква заглавная), для методов и объектов класса — lowerCamelCase (первая буква прописная).

Далее создадим поля класса. Поля класса это переменные хранящие все данные объекта класса. Поля класса объявляются в нем следующим образом:

- ✓ спецификатор тип имя;

Спецификаторы доступа являются важнейшим инструментом для обеспечения инкапсуляции в Java. В Java существуют несколько видов спецификаторов доступа. К ним относятся:

public(общедоступный) – метод, поле или класс, перед названием которого стоит этот спецификатор доступа будет доступен с любой точки проекта, как для классов находящихся в этом же пакете(директории), так и в другом.

private (приватный) – метод или переменная (не класс, класс не может иметь спецификатор доступа private) могут использоваться только в пределах объекта класса, в котором они были объявлены

protected (защищенный) – метод или переменная (не класс) доступны всем классам пакета (директории) и всем наследникам класса, в котором они объявлены, даже если те находятся в других пакетах.

В языке Java могут использоваться статические переменные класса, объявленные один раз для всего класса со спецификатором **static** и одинаковые для всех экземпляров (объектов) класса, или переменные экземпляра класса, создаваемые для каждого объекта класса. Переменные со спецификатором **final** являются константами. Константное поле должно использовать в названии все заглавные буквы, если же название состоит из нескольких слов, они должны разделяться символом нижнего подчеркивания.

Например:

```
public class Student {
    private final static String GROUP_NAME = "EM-41";
    private final static int COURSE = 4;
}
```

Создадим поля для хранения информации о студенте, его фамилии, имени и отчества.

```
public class Student {
    private final static String GROUP_NAME = "EM-41";
    private final static int COURSE = 4;

    private String name;
    private String lastName;
    private String secondName;
}
```

При появлении полей класса необходимо добавить конструктор, который будет инициализировать эти поля нужными нам значениями. IntelliJ IDEA может сгенерировать конструктор вместо нас. Для этого в главном меню выберем Code->Generate либо в необходимом месте нажмем комбинацию клавиш Alt + Insert. Появится выпадающее меню (Рисунок 9).

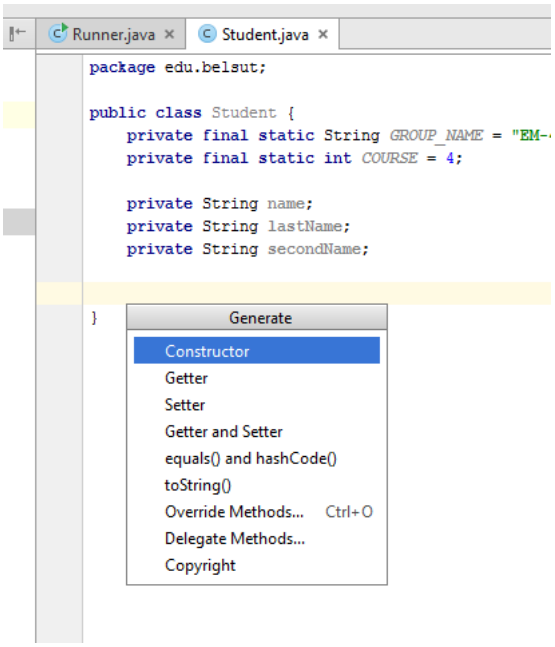

Рисунок 9 – Генерация конструктора

Выберем пункт Constructor. Далее нам предлагается выбрать необходимое количество параметров для генерации (Рисунок 10).

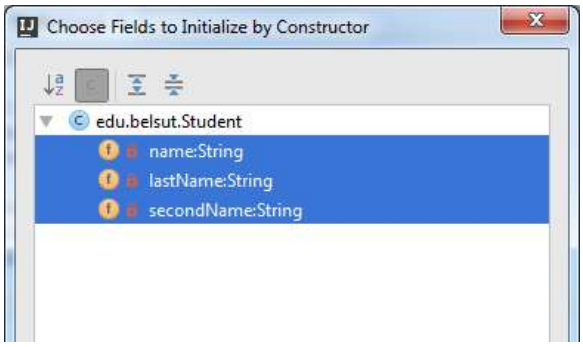

Рисунок 10 – Создание конструктора с тремя параметрами

Выбираем все 3 параметра (Рисунок 10) и нажимаем OK.

```
public class Student {
    private final static String GROUP_NAME = "EM-41";
    private final static int COURSE = 4;

    private String name;
    private String lastName;
    private String secondName;
```

```
        public Student(String name, String lastName, String secondName) {
            this.name = name;
            this.lastName = lastName;
            this.secondName = secondName;
        }
    }
```

Ключевое слово **this** всегда служит ссылкой на объект, для которого был вызван метод.

Сгенерируем также так называемые сеттеры и геттеры. В главном меню выберем Code->Generate либо в необходимом месте нажмем комбинацию клавиш Alt + Insert и выберем пункт Getter and Setter.

```
public class Student {
    private final static String GROUP_NAME = "EM-41";
    private final static int COURSE = 4;

    private String name;
    private String lastName;
    private String secondName;

    public Student(String name, String lastName, String secondName) {
        this.name = name;
        this.lastName = lastName;
        this.secondName = secondName;
    }

    public String getName() {
        return name;
    }

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

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getSecondName() {
        return secondName;
    }

    public void setSecondName(String secondName) {
        this.secondName = secondName;
    }
}
```

1.3 Вывод на экран. Работа со строками

1.3.1 Вывод на экран

Откроем проект консольного приложения в IntelliJ IDEA.

Напишем код, который выводит на экран фразу Hello, world!. Для этого используем стандартный поток вывода.

Стандартный поток ввода (клавиатура) в Java представлен объектом – System.in. А стандартный поток вывода (дисплей) – объектом System.out. Есть ещё стандартный поток для вывода ошибок – System.err.

System.out содержит набор методов для вывода данных в окно консоли. Метод println используется для вывода строки символов.

Рассмотрим следующий код:

```
package edu.belsut;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello, world!");ъ
    }
}
```

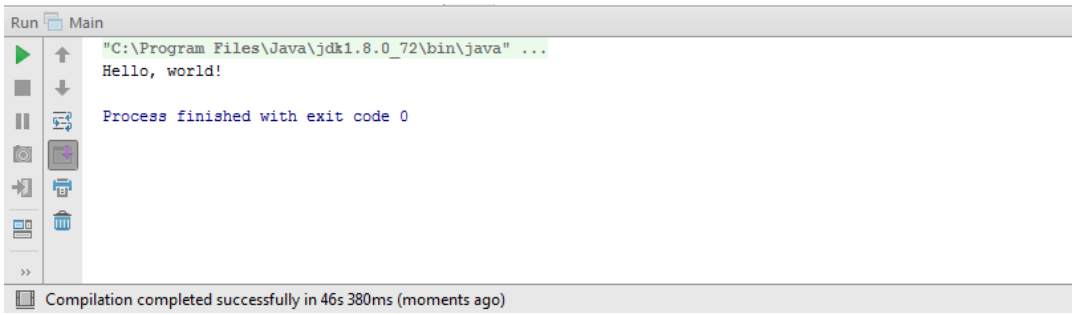

Рисунок 11 – Результат выполнения кода

1.3.2 Работа со строками

Идентичным вышенаписанному коду будет код:

```
package edu.belsut;

public class Main {

    public static void main(String[] args) {
        String result = "Hello, world!";
        System.out.println(result);
    }
}
```

В этом коде видно, что на экран выводится объект класса String.

В пакет java.lang встроен класс, инкапсулирующий структуру данных, соответствующую строке символов. Этот класс, называемый String, не что иное, как объектное представление неизменяемого символьного массива. В

этом классе есть методы, которые позволяют сравнивать строки, осуществлять в них поиск и извлекать определенные символы и подстроки.

Следует помнить, что объекты класса String являются неизменяемыми. Поэтому, когда вам кажется, что вы меняете строку, то на самом деле вы создаёте новую строку. В Java есть специальные классы StringBuffer и StringBuilder, который допускают изменения в строке.

Рассмотрим основные методы класса String:

- String concat(String s) или «+» — слияние строк;

```
public class Main {

    public static void main(String[] args) {
        String first = "Hello";
        String second = "world";
        String result = first.concat(second);
        System.out.println(result);
        System.out.println(first + "," + second);
    }
}
```

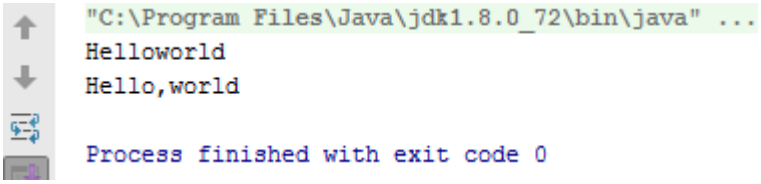

Рисунок 12 – Результат выполнения кода

- boolean equals(Object ob) и equalsIgnoreCase(String s) — сравнение строк с учетом и без учета регистра соответственно;

```
public class Main {

    public static void main(String[] args) {
        String first = "World";
        String second = "world";
        if(first.equals(second)){
            System.out.println("Строки равны.");
        }else{
            System.out.println("Строки не равны.");
        }
        if(first.equalsIgnoreCase(second)){
            System.out.println("Строки равны.");
        }else{
            System.out.println("Строки не равны.");
        }
    }
}
```

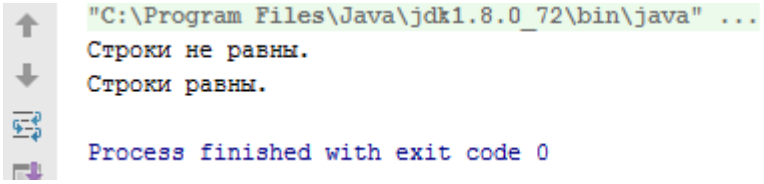

Рисунок 13 – Результат выполнения кода

- String substring(int n, int m) — извлечение из строки подстроки длины m-n, начиная с позиции n. Нумерация символов в строке начинается с нуля;
- String substring(int n) — извлечение из строки подстроки, начиная с позиции n;

```
public class Main {

    public static void main(String[] args) {
        String result = "Hello, world";
        System.out.println(result.substring(7));
        System.out.println(result.substring(3, 9));
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_72\bin\java" ...
world
lo, wo

Process finished with exit code 0
```

Рисунок 14 – Результат выполнения кода

- int length() — определение длины строки;

```
public class Main {

    public static void main(String[] args) {
        String result = "Hello, world";
        System.out.println(result.length());
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_72\bin\java" ...
12

Process finished with exit code 0
```

Рисунок 15 – Результат выполнения кода

- int indexOf(char ch) — определение позиции символа в строке;
- String toUpperCase()/toLowerCase() — преобразование всех символов вызывающей строки в верхний/нижний регистр;

```
public class Main {

    public static void main(String[] args) {
        String result = "Hello, world";
```

```
        System.out.println(result.toUpperCase());
        System.out.println(result.toLowerCase());
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_72\bin\java" ...
HELLO, WORLD
hello, world

Process finished with exit code 0
```

Рисунок 16 – Результат выполнения кода

- String replace(char c1, char c2) — замена в строке всех вхождений первого символа вторым символом;
- String intern() — заносит строку в «пул» литералов и возвращает ее объектную ссылку;
- String trim() — удаление всех пробелов в начале и конце строки;
- char charAt(int position) — возвращение символа из указанной позиции (нумерация с нуля);
- boolean isEmpty() — возвращает true, если длина строки равна 0;
- char[] getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) — извлечение символов строки в массив символов;
- static String format(String format, Object… args), format(Locale l, String format, Object… args) — генерирует форматированную строку, полученную с использованием формата, интернационализации и др.;
- String[] split(String regex), String[] split(String regex, int limit) — поиск вхождения в строку заданного регулярного выражения (разделителя) и деление исходной строки в соответствии с этим на массив строк.

```
public class Main {

    public static void main(String[] args) {
        String result = " Hello, world    ";
        String[] array = result.trim().split(" ");
        System.out.println(array[0]);
        System.out.println(array[1]);
    }
}
```

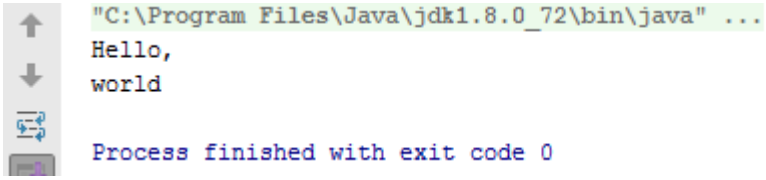

Рисунок 17 – Результат выполнения кода

1.4 Коллекции Java

1.4.1 Понятие коллекции

Коллекция – это хранилище, поддерживающее различные способы накопления и упорядочения объектов с целью обеспечения возможностей эффективного доступа к ним. Коллекции представляют собой реализацию абстрактных типов (структур) данных, поддерживающих три основные операции:

- добавление нового элемента в коллекцию;
- удаление элемента из коллекции;
- изменение элемента в коллекции.

Интерфейсы коллекций:

- **Map**<K,V> – карта отображения вида “ключ-значение”;
- **Collection**<E> – вершина иерархии остальных коллекций;
- **List**<E> – специализирует коллекции для обработки списков;
- **Set**<E> – специализирует коллекции для обработки множеств, содержащих уникальные элементы.

Все классы коллекций реализуют также интерфейсы **Serializable**, **Cloneable** (кроме WeakHashMap). Кроме того, классы, реализующие интерфейсы List<E> и Set<E>, реализуют также интерфейс **Iterable**<E>.

1.4.2 Методы коллекций

В интерфейсе Collection<E> определены методы, которые работают на всех коллекциях:

boolean **add**(E obj) – добавляет obj к вызывающей коллекции и возвращает true, если объект добавлен, и false, если obj уже элемент коллекции;

boolean **addAll**(Collection<? extends E> c) – добавляет все элементы коллекции к вызывающей коллекции;

void **clear**() – удаляет все элементы из коллекции;

boolean **contains**(Object obj) – возвращает true, если вызывающая коллекция содержит элемент obj;

boolean **equals**(Object obj) – возвращает true, если коллекции эквивалентны;

boolean **isEmpty**() – возвращает true, если коллекция пуста;

Iterator<E> **iterator**() – извлекает итератор;

boolean **remove**(Object obj) – удаляет obj из коллекции;

int **size**() – возвращает количество элементов в коллекции;

Object[] **toArray**() – копирует элементы коллекции в массив объектов;

<T> T[] **toArray**(T a[]) – копирует элементы коллекции в массив объектов определенного типа.

1.4.3 Интерфейс Map

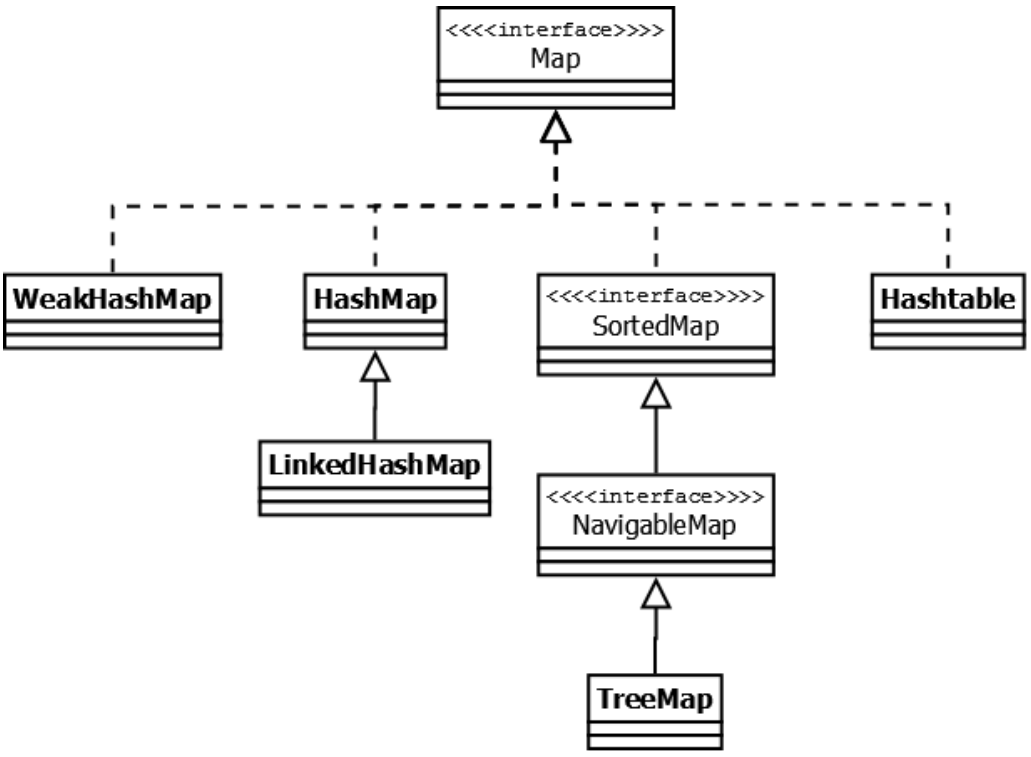

Рисунок 18 – Иерархия коллекций интерфейса Map

Hashtable — реализация такой структуры данных, как хэш-таблица. Она не позволяет использовать null в качестве значения или ключа. Эта

коллекция была реализована раньше, чем Java Collection Framework, но в последствии была включена в его состав. Как и другие коллекции из Java 1.0, Hashtable является синхронизированной (почти все методы помечены как synchronized). Из-за этой особенности у неё имеются существенные проблемы с производительностью и, начиная с Java 1.2, в большинстве случаев рекомендуется использовать другие реализации интерфейса Map ввиду отсутствия у них синхронизации.

HashMap — коллекция является альтернативой Hashtable. Двумя основными отличиями от Hashtable являются то, что HashMap не синхронизирована и HashMap позволяет использовать null как в качестве ключа, так и значения. Так же как и Hashtable, данная коллекция не является упорядоченной: порядок хранения элементов зависит от хэш-функции. Добавление элемента выполняется за константное время O(1), но время удаления, получения зависит от распределения хэш-функции. В идеале является константным, но может быть и линейным O(n).

LinkedHashMap — это упорядоченная реализация хэш-таблицы. Здесь, в отличие от HashMap, порядок итерирования равен порядку добавления элементов. Данная особенность достигается благодаря двунаправленным связям между элементами (аналогично LinkedList). Но это преимущество имеет также и недостаток — увеличение памяти, которое занимает коллекция.

TreeMap — реализация Map основанная на красно-чёрных деревьях. Как и LinkedHashMap является упорядоченной. По умолчанию, коллекция сортируется по ключам с использованием принципа "natural ordering", но это поведение может быть настроено под конкретную задачу при помощи объекта Comparator, которые указывается в качестве параметра при создании объекта TreeMap.

WeakHashMap — реализация хэш-таблицы, которая организована с использованием weak references. Другими словами, Garbage Collector

автоматически удалит элемент из коллекции при следующей сборке мусора, если на ключ этого элемента нет жёстких ссылок.

1.4.4 Интерфейс List

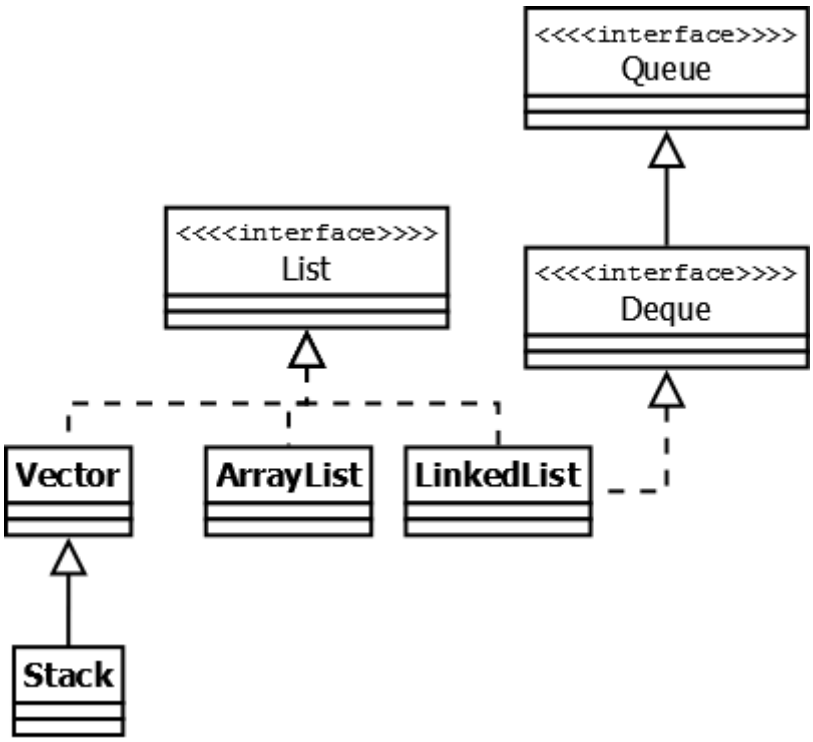

Рисунок 19 – Иерархия коллекций интерфейса List

Реализации этого интерфейса представляют собой упорядоченные коллекции. Кроме того, разработчику предоставляется возможность доступа к элементам коллекции по индексу и по значению (так как реализации позволяют хранить дубликаты, результатом поиска по значению будет первое найденное вхождение).

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

Stack — данная коллекция является расширением коллекции Vector. Была добавлена в Java 1.0 как реализация стека LIFO (last-in-first-out).

Является частично синхронизированной коллекцией (кроме метода добавления push()). После добавления в Java 1.6 интерфейса Deque, рекомендуется использовать именно реализации этого интерфейса, например ArrayDeque.

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

LinkedList — ещё одна реализация List. Позволяет хранить любые данные, включая null. Особенностью реализации данной коллекции является то, что в её основе лежит двунаправленный связный список (каждый элемент имеет ссылку на предыдущий и следующий). Благодаря этому, добавление и удаление из середины, доступ по индексу, значению происходит за линейное время O(n), а из начала и конца за константное O(1). Так же, ввиду реализации, данную коллекцию можно использовать как стек или очередь. Для этого в ней реализованы соответствующие методы.

1.4.5 Интерфейс Set

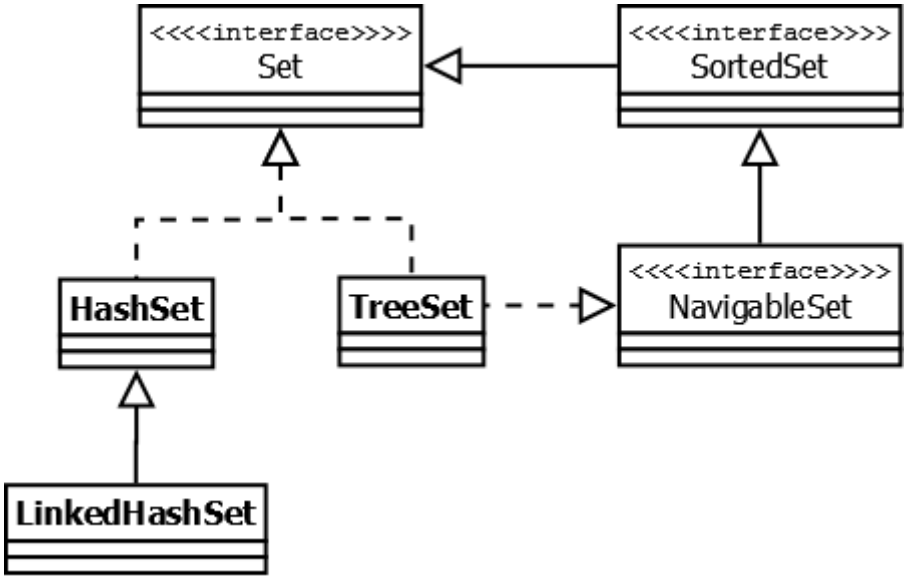

Рисунок 20 – Иерархия интерфейсов Set

Представляет собой неупорядоченную коллекцию, которая не может содержать дублирующиеся данные. Является программной моделью математического понятия «множество».

HashSet — реализация интерфейса Set, базирующаяся на HashMap. Внутри использует объект HashMap для хранения данных. В качестве ключа используется добавляемый элемент, а в качестве значения — объект-пустышка (new Object()). Из-за особенностей реализации порядок элементов не гарантируется при добавлении.

LinkedHashSet — отличается от HashSet только тем, что в основе лежит LinkedHashMap вместо HashSet. Благодаря этому отличию порядок элементов при обходе коллекции является идентичным порядку добавления элементов.

TreeSet — аналогично другим классам-реализациям интерфейса Set содержит в себе объект NavigableMap, что и обуславливает его поведение. Предоставляет возможность управлять порядком элементов в коллекции при помощи объекта Comparator, либо сохраняет элементы с использованием "natural ordering".

1.4.6 Интерфейс Queue

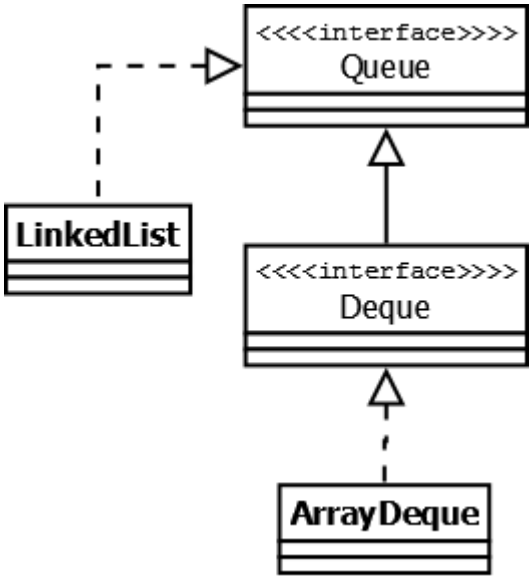

Рисунок 21 – Иерархия интерфейсов Queue

Этот интерфейс описывает коллекции с предопределённым способом вставки и извлечения элементов, а именно — очереди FIFO (first-in-first-out). Помимо методов, определённых в интерфейсе Collection, определяет дополнительные методы для извлечения и добавления элементов в очередь. Большинство реализаций данного интерфейса находится в пакете java.util.concurrent.

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

ArrayDeque — реализация интерфейса Deque, который расширяет интерфейс Queue методами, позволяющими реализовать конструкцию вида LIFO (last-in-first-out). Интерфейс Deque и реализация ArrayDeque были добавлены в Java 1.6. Эта коллекция представляет собой реализацию с

использованием массивов, подобно ArrayList, но не позволяет обращаться к элементам по индексу и хранение null. Как заявлено в документации, коллекция работает быстрее, чем Stack, если используется как LIFO коллекция, а также быстрее, чем LinkedList, если используется как FIFO.

Лабораторная работа №1

РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

Цель работы: научиться создавать консольные приложения на языке Java с использованием объектно-ориентированного программирования. Научиться работать со строками.

Часть 1. Создание проекта в IDE IntelliJ IDEA. Работа со строками.

Порядок работы:

1. Создать консольное приложение в среде IntelliJ IDEA.
2. Создать строку, состоящую из фамилии, имени и отчества студента.
3. Вывести на экран количество символов в фамилии, имени и отчестве студента, а также инициалы.
4. Вывести на экран повторяющиеся символы и их количество.

Часть 2. Разработка класса.

Порядок работы:

1. Разработать класс для хранения данных по индивидуальному заданию.
2. Сделать метод вывода на экран методом спроектированного класса. При необходимости изменить метод для соответствия индивидуальному заданию.
3. Перенести все методы из первой части задания в класс по индивидуальному заданию.
4. Вызвать все методы спроектированного класса из класса, содержащего функцию main.
5. Оформить отчет по лабораторной работе.

Содержание отчета:

1. Название лабораторной работы.
2. Цель работы.
3. Индивидуальное задание по варианту.
4. Листинг всех классов.

5 Скриншот окна консоли с выведенным результатом.

6 Вывод.

Таблица 1 – Индивидуальное задание для выполнения лабораторной работы №1

Номер варианта	Класс для проектирования	Метод класса
1	Автобусная остановка	Поиск по части названия
2	Улица города	Поиск по части названия
3	Товар в магазине	Вывод цены и цены со скидкой
4	Мобильная игра	Вывод количества скачиваний
5	Дисциплина в ВУЗе	Поиск по фамилии преподавателя
6	Дом	Вывод улицы и номера
7	Автобусный маршрут	Вывод названия конечных остановок
8	Страна	Вывод названия и количества пользователей
9	Операционная система	Вывод названия и разрядности
10	Валюта	Вывод полного и краткого названий
11	Мобильный телефон	Вывод модели и производителя
12	Билет на экзамене	Вывод текста двух вопросов
13	Футбольный клуб	Вывод названия и бюджета
14	Планета	Радиус и название

Лабораторная работа №2

КОЛЛЕКЦИИ И ИНТЕРФЕЙСЫ

Цель работы: научиться работать с коллекциями и интерфейсами языка Java.

Порядок работы:

1. Создать класс для работы с коллекцией классов по индивидуальному заданию из первой лабораторной работы.
2. В этом классе добавить методы добавления элементов в конец коллекции, в указанную позицию коллекции.
3. Изучить понятия коллекций и интерфейсов языка Java.
4. Добавить в класс, работающий с коллекцией, метод из индивидуального задания. Тип коллекции также должен соответствовать заданию.
5. Добавить метод вывода коллекции в консоль.
6. Произвести функциональное тестирование всех методов спроектированного класса из класса Main(Runner).

Содержание отчета:

1. Название лабораторной работы.
2. Цель работы.
3. Индивидуальное задание по варианту.
4. Листинг всех классов.
5. Скриншот окна консоли с выведенным результатом.
6. Вывод.

Таблица 2 – Индивидуальное задание для выполнения лабораторной работы №2

Номер варианта	Класс для проектирования	Тип коллекции	Метод коллекции
1	Автобусная остановка	ArrayList	Сортировка по названию
2	Улица города	LinkedList	Сохранение коллекции в файл
3	Товар в магазине	HashSet	Подсчет суммы цен всех товаров
4	Мобильная игра	HashSet	Подсчет общего числа скачиваний
5	Дисциплина в ВУЗе	HashMap (ключ - дисциплина)	Вывод всех дисциплин преподавателя
6	Дом	HashMap (ключ - дом)	Вывод номеров всех домов на улице
7	Автобусный маршрут	LinkedList	Вывод названия конечных остановок
8	Страна	ArrayList	Сортировка по названию
9	Операционная система	ArrayList	Сортировка по названию
10	Валюта	ArrayList	Сортировка по полному названию
11	Мобильный телефон	HashMap (ключ - модель)	Вывод на экран всех моделей заданного производителя
12	Билет на экзамене	ArrayList	Выбор случайного билета
13	Футбольный клуб	HashSet	Подсчет общего бюджета
14	Планета	ArrayList	Сортировка по радиусу (диаметру)

Лабораторная работа №3

НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ

Цель работы: изучить понятия наследование и полиморфизм и научиться применять их для написания программ на языке программирования Java.

Порядок работы:

1. Изучить парадигмы ООП наследование и полиморфизм, для этого воспользоваться рекомендуемой литературой.
2. Создать новый консольный проект. Создать в нем иерархию классов, необходимых для выполнения лабораторной работы.
3. Реализовать индивидуальное задание. Реализовать метод по индивидуальному заданию с использованием полиморфизма.
4. Оформить отчет по лабораторной работе.

Задание на лабораторную работу:

1. Создать абстрактный класс «Простой калькулятор» и реализовать в нем базовую функциональность: операции сложения, вычитания, умножения и деления.
2. Наследовать от класса «Простой калькулятор» класс «Калькулятор с памятью». В данном классе реализовать ячейку памяти с возможностью ее очистки, увеличения и уменьшения на заданное число. Обязательно наличие конструктора с одним параметром – значение ячейки памяти.
3. Наследовать от класса «Калькулятор с памятью» новый класс и реализовать в нем операцию по индивидуальному заданию.
4. Произвести вызов всех методов полученного класса.

Содержание отчета:

1. Название лабораторной работы.
2. Цель работы.
3. Индивидуальное задание по варианту.
4. Листинг всех классов.

5 Скриншот окна консоли с выведенными результатами работы всех методов спроектированного класса.

6 Вывод.

Таблица 3 – Индивидуальное задание для выполнения лабораторной работы №3

Номер варианта	Метод по индивидуальному заданию
1	Возведение в степень
2	Извлечение квадратного корня
3	Синус
4	Косинус
5	E в степени
6	10 в степени
7	Факториал числа
8	Перевод из градусов в радианы
9	Тангенс
10	1/x
11	Возведение в куб
12	Извлечение кубического корня
13	Остаток от деления
14	Целочисленное деление

Лабораторная работа №4

ИСКЛЮЧЕНИЯ. JAVADOC

Цель работы: изучить понятия исключения научиться применять их для написания программ на языке программирования Java. Изучить принципы создания документации к программному коду с использованием системы документирования Javadoc.

Порядок работы:

1. Изучить понятие исключений и процесс генерации исключительных ситуаций. Для этого воспользоваться рекомендуемой литературой.
2. Для выполнения работы использовать готовый код лабораторной работы №3.
3. Добавить в спроектированные методы проверку на ошибки и генерацию ошибочной ситуации в случае их наличия.
4. Для каждой исключительной ситуации спроектировать собственный класс исключений, наследуемый от суперкласса Exception.
5. Вызвать все методы класса, в которых может проявляться исключительная ситуация как с корректными параметрами, так и с ошибочными, выводить в консоль различные сообщения в зависимости от типа ошибочной ситуации.
6. Оформить документацию Javadoc.
7. Оформить отчет по лабораторной работе.

Содержание отчета:

1. Название лабораторной работы.
2. Цель работы.
3. Индивидуальное задание по варианту.
4. Листинг всех классов.
5. Скриншоты окна консоли с выведенными результатами работы всех методов спроектированного класса.
6. Вывод.

Лабораторная работа №5

РАБОТА С ФАЙЛОВОЙ СИСТЕМОЙ. (4 ЧАСА)

Цель работы: изучить основные классы позволяющие работать с файлами в языке Java.

Порядок работы:

1. Модифицировать исходный код лабораторной работы №4, добавив ввод данных для калькулятора из файла "in.txt", например функция sin(2*10 + 5) - 1 будет иметь следующий вид:

   2

   *

   10

   +

   5

   sin

   -

   1
2. Обязательно использовать в формуле функцию из индивидуального задания к лабораторной работе №3.
3. Результат вычислений сохранять в файл "out.txt".
4. Продемонстрировать работу как минимум с тремя вариантами входных данных.
5. Оформить документацию Javadoc.
6. Оформить отчет по лабораторной работе.

Содержание отчета:

1. Название лабораторной работы.
2. Цель работы.
3. Индивидуальное задание по варианту.
4. Листинг всех классов.
5. Скриншоты окна консоли с выведенными результатами работы всех методов спроектированного класса.

6 Содержимое файлов in.txt и out.txt.

7 Вывод.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Блинов, И.Н. Java. Промышленное программирование / И.Н. Блинов, В.С. Романчик. – Мн.: Универсал Пресс, 2007. – 124 с
2. Эккель, Б. Философия Java. 4-е издание / Б. Эккель. – СПб: Питер, 2009. – 638с.
3. Шилдт, Г. Java. Полное руководство, 8-е издание / Г. Шилдт. – СПб.: Вильямс, 2012. – 1104 с.
4. Фримен, Э. Паттерны проектирования / Э. Фримен, [и др.] – СПб.: Питер, 2011. – 456 с.
5. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха – СПб:Питер, 2007. – 544с.
6. Тейт, Б. Горький вкус Java / Б. Тейт – СПб.: Питер, 2003. – 334 с.
7. Леоненков, А. Самоучитель UML. Эффективный инструмент моделирования информационных систем / А.Леоненков – СПб: BHV, 2001. – 304с.
8. Макконнел, С. Совершенный код. Мастер-класс/ С.Макконнел – М.: «Русская редакция»; СПб.: Питер, 2007. – 896 с.
9. Блох, Д. Java. Эффективное программирование / Д. Блох – М.: Лори, 2002. – 224 с.
10. Буч, Г. Язык UML. Руководство пользователя / Г.Буч, Дж.Рамбо, А. Джекобсон – М.: ДМК, 2000. – 432 с.
11. Burnette, E. Eclipse IDE Pocket Guide / E. Burnette – O'Relly, 2005. – 127 p.

СОДЕРЖАНИЕ

Введение 2

1 Основы программирования Java 3

1.1 Создание проекта в IntelliJ IDEA 3

1.1.1 Введение в IDE IntelliJ IDEA 3

1.1.2 Порядок создания проекта консольного приложения. 3

1.2 Создание нового класса в Intellij idea 8

1.3 Вывод на экран. Работа со строками 12

1.3.1 Вывод на экран 12

1.3.2 Работа со строками 13

1.4 Коллекции Java 17

1.4.1 Понятие коллекции 17

1.4.2 Методы коллекций 17

1.4.3 Интерфейс Map 18

1.4.4 Интерфейс List 20

1.4.5 Интерфейс Set 22

1.4.6 Интерфейс Queue 23

Лабораторная работа №1 Разработка консольного приложения 25

Лабораторная работа №2 Коллекции и интерфейсы 27

Лабораторная работа №3 Наследование и полиморфизм 29

Лабораторная работа №4 Исключения. Javadoc 30

Лабораторная работа №5 Работа с файловой системой. 32

Список рекомендуемой литературы 34