

Список вопросов к экзамену по предмету Программирование на Джава зима 2020-2021 год

Тема 1. Особенности платформы Java. Синтаксис языка Java

1. К какому типу языков относится язык Джава

Java — строго типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (далее приобретенной Oracle). Каждая переменная и константа представляет определенный тип, который строго определен. Тип данных определен диапазоном значений, которые может хранить переменная или константа. Все выражения и параметры java проверяет на соответствие типов. Java - интерпретируемый язык, т.е. написанный код компилируется в байт-код Java, который передается в интерпретатор (JVM) Т.е. следует принципу "Писать один раз, запускать где угодно" - скомпилированный код Java может работать на всех платформах, поддерживающих Java, без необходимости перекомпиляции.

2. Особенности языка Джава

простота, объектная ориентированность и понятность;
надёжность и безопасность;
переносимость и независимость от платформы;
высокая производительность;
интрапретируемость, поточность и динамичность.

3. Класс Scanner и его использование для чтения стандартного потока вводы

Этот класс пригодится, если тебе нужно будет считывать данные, которые вводят юзеры. Класс - java.util.Scanner. Его функциональность очень проста. Он, словно настоящий сканер, считывает данные из источника, который ты для него укажешь. Например, из строки, из файла, из консоли. Далее он распознает эту информацию и обрабатывает нужным образом.

4. Класс Scanner, конструктор класса Scanner для чтения стандартного потока вводы 5.

**Методы класса Scanner nextLine(), nextInt(),
hasNextInt(), hasNextLine() и их использование
для чтения ввода пользователя с клавиатуры.**

Метод nextLine() обращается к источнику данных, находит там следующую строку, которую он еще не считывал и возвращает ее. После чего мы выводим ее на консоль. Метод nextInt() считывает и возвращает введенное число.

hasNextInt() — метод проверяет, является ли следующая порция введенных данных числом, или нет (возвращает, соответственно, true или false).

hasNextLine() — проверяет, является ли следующая порция данных строкой.

hasNextByte(), hasNextShort(), hasNextLong(), hasNextFloat(),

hasNextDouble() — все эти методы делают то же для остальных типов данных.

6. Примитивные типы данных, объявление и присваивание переменных

Byte, short, int, long, float, double, char, boolean

С помощью перегрузки метода out.println можно выводить в консоль все примитивные типы данных.

7. Условные операторы, полное и неполное ветвление в Джава, синтаксис

Условный оператор if позволяет задать условие, в соответствии с которым дальнейшая часть программы может быть выполнена.

Условный оператор if-else в языке означает «в ином случае». То есть если условие if не является истинным, выводим то, что в блоке else.

Вложенный if служит для проверки сразу на несколько условий, где используются разные действия.

Пример:

```
int a = 20;
int b = 5;
if(a == 20){
    System.out.println("a = 20");
    if(b == 5){
        System.out.println("b = 5");
    }
}
```

«Элвис» или оператор тернар - ?: . Получил прозвище за схожесть с причёской короля рок-н-ролла, он требует три операнда и позволяет писать меньше кода для простых условий.

Пример:

```
int a = 20;  
int b = 5;  
String answer = (a > b) ? "Условие верно" : "Условие ошибочно";  
System.out.println(answer);
```

Условный оператор или оператор множества switch позволяет сравнить переменную как с одним, так и с несколькими значениями.

Пример:

```
switch(выражение) {  
    case значение1:  
        // Блок кода 1  
        break;  
    case значение2:  
        // Блок кода 2  
        break;  
    case значениеN:  
        // Блок кода N  
        break;  
    default :  
        // Блок кода для default  
}
```

8. Оператор множественного выбора в Джава, синтаксис

Условный оператор или оператор множества switch позволяет сравнить переменную как с одним, так и с несколькими значениями.

Пример:

```
switch(выражение) {  
    case значение1:  
        // Блок кода 1  
        break;  
    case значение2:  
        // Блок кода 2  
        break;  
    case значениеN:  
        // Блок кода N  
        break;  
    default :  
        // Блок кода для default  
}
```

9. Класс System. Работа со стандартами потоками вывода

Системный является одним из базовых классов в Java и принадлежит пакету java.lang. Класс System является финальным и не предоставляет общедоступных конструкторов. Из-за этого все члены и методы, содержащиеся в этом классе, являются статическими по природе.

Таким образом, вы не можете наследовать этот класс для переопределения его методов. Поскольку класс System .имеет множество ограничений, существуют различные предварительно созданные поля и методы класса. Ниже я перечислил несколько важных функций, поддерживаемых этим классом:

Стандартный ввод и вывод.

Ошибки вывода потоков.

Доступ к внешним свойствам и переменным среды.

Встроенная утилита для копирования части массива.

Предоставляет средства для загрузки файлов и библиотек.

10.Перегруженные методы out.println() класса System и их использование для вывода в консоль

System – КЛАСС

out – ПОЛЕ КЛАССА

println() – МЕТОД КЛАССА

С помощью перегрузки метода out.println можно выводить в консоль все примитивные типы данных.

11.Константы в Джава: объявление константы

Для объявления констант в java используется ключевое слово final.

Константе, в отличии от переменной, значение может быть присвоено только один раз.

12. В результате выполнения этой строчки

13. Объявление и использование бестиповых переменных в Джава

Для объявления бестиповых данных используется ключевое слово var.

Компилятор сам подбирает тип переменных, опираясь на присвоенные им значения. При объявлении такой переменной сразу должна быть инициализация, иначе будет ошибка

14. Объявление переменных и инициализация типа класс

Для объявления переменной в Java используют следующий синтаксис: тип данных переменная [= значение], [переменная [= значение], ...] ; Идём дальше: если нужно объявить больше чем одну переменную указанного типа, допускается применение списка с запятыми:

```
int a, b, c; // объявление трёх целых переменных a, b и c
```

Инициализация полей класса значения по умолчанию

```
class Up {  
    static boolean b;  
    static byte by;  
    static char c;  
    static double d;
```

```
static float f;  
static int i;  
static long l;  
static short s;  
static String st;  
}
```

15.Арифметические операции, операции инкремента и декремента в Джава

«+» операция сложения двух чисел:

```
-----  
int a = 10;  
int b = 7;  
int c = a + b; // 17  
int d = 4 + b; // 11  
-----
```

«-» операция вычитания двух чисел:

```
-----  
int a = 10;  
int b = 7;  
int c = a - b; // 3  
int d = 4 - a; // -6  
-----
```

«*» операция умножения двух чисел

```
-----  
int a = 10;  
int b = 7;  
int c = a * b; // 70  
-----
```

```
int d = b * 5; // 35
```

«/» операция деления двух чисел:

```
int a = 20;  
int b = 5;  
int c = a / b; // 4  
double d = 22.5 / 4.5; // 5.0
```

«%» получение остатка от деления двух чисел:

```
int a = 33;  
int b = 5;  
int c = a % b; // 3  
int d = 22 % 4; // 2 (22 - 4*5 = 2)
```

«++» (префиксный инкремент) Предполагает увеличение переменной на единицу, например, $z=++y$ (вначале значение переменной y увеличивается на 1, а затем ее значение присваивается переменной z)

```
int a = 8;  
int b = ++a;  
System.out.println(a); // 9  
System.out.println(b); // 9
```

«++» (постфиксный инкремент) Também представляет увеличение переменной на единицу, например, $z=y++$ (вначале значение переменной y присваивается переменной z , а потом значение переменной y увеличивается на 1)

```
-----  
int a = 8;  
int b = a++;  
System.out.println(a); // 9  
System.out.println(b); // 8  
-----
```

«--» (префиксный декремент) Уменьшение переменной на единицу, например, $z=--y$ (вначале значение переменной y уменьшается на 1, а потом ее значение присваивается переменной z)

```
-----
```

```
int a = 8;  
int b = --a;  
System.out.println(a); // 7  
System.out.println(b); // 7  
-----
```

«--» (постфиксный декремент) $z=y--$ (сначала значение переменной y присваивается переменной z , а затем значение переменной y уменьшается на 1)

```
-----
```

```
int a = 8;  
int b = a--;  
System.out.println(a); // 7  
System.out.println(b); // 8  
-----
```

16. В результате выполнения фрагмента программы

17. Арифметические операции, приоритет выполнения операций

«+» операция сложения двух чисел:

```
----- int a = 10;  
int b = 7;  
int c = a + b; // 17  
int d = 4 + b; // 11
```

----- «-» операция вычитания двух чисел:

```
int a = 10;  
int b = 7;  
int c = a - b; // 3  
int d = 4 - a; // -6
```

----- «*» операция умножения двух чисел

```
int a = 10;  
int b = 7;  
int c = a * b; // 70  
int d = b * 5; // 35
```

----- «/» операция деления двух чисел:

```
int a = 20;
```

```
int b = 5;  
int c = a / b; // 4  
double d = 22.5 / 4.5; // 5.0
```

----- «%» получение остатка

от деления двух чисел: -----

```
int a = 33;  
int b = 5;  
int c = a % b; // 3  
int d = 22 % 4; // 2 (22 - 4*5 = 2)
```

Одни операции имеют больший приоритет, чем другие, и поэтому выполняются вначале. Операции в порядке уменьшения приоритета: ++ (постфиксный инкремент), -- (постфиксный декремент) ++ (префиксный инкремент), -- (префиксный декремент) * (умножение), / (деление), % (остаток от деления) + (сложение), - (вычитание)

18. Типы данных в языке Джава, классификация, примеры

boolean: хранит значение true или false

```
boolean isActive = false;  
boolean isAlive = true;
```

byte: хранит целое число от -128 до 127 и занимает 1 байт

```
byte a = 3;  
byte b = 8;
```

short: хранит целое число от -32768 до 32767 и занимает 2 байта

short a = 3;

short b = 8;

int: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта

int a = 4;

int b = 9;

long: хранит целое число от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт

long a = 5;

long b = 10;

double: хранит число с плавающей точкой от $\pm 4.9 \cdot 10^{-324}$ до $\pm 1.7976931348623157 \cdot 10^{308}$ и занимает 8 байт

double x = 8.5;

double y = 2.7;

В качестве разделителя целой и дробной части в дробных литералах используется точка.

float: хранит число с плавающей точкой от $-3.4 \cdot 10^{-38}$ до $3.4 \cdot 10^{38}$ и занимает 4 байта

float x = 8.5F;

float y = 2.7F;

19.Массивы в Джава, объявление и инициализация массивов, длина массива, получение доступа к элементу массива

a) Объявление:

1) typeData Name [] = new typeData[length]; - стиль Java

2) typeData []Name = new typeData[length]; - стиль из C++

б) Инициализация:

1) int c [] = new int[2];

c[0] = 1;

c[1] = 2;

2) int c [] = new int [] {1, 2, 3, 4};

3) int c [] = {1, 2, 3, 4};

Массив представляет набор однотипных значений. Объявление массива похоже на объявление обычной переменной, которая хранит одиночное значение, причем есть два способа объявления массива:

```
int nums[] = new int[4]; // массив из 4 чисел
int[] nums2 = new int[5]; // массив из 5 чисел
int[] nums2 = { 1, 2, 3, 5 };
int[][] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
int[][] nums = new int[3][];
nums[0] = new int[2];
nums[1] = new int[3];
nums[2] = new int[5];
```

20.Массивы в Джава, как объектные типы данных, контроль доступа за выход за границы массива

Выход за границы массива

За выходом за границы массивы следит интерпритатор.

Если случился выход за гранич. массива, то вылетит исключение `ArrayIndexOutOfBoundsException`.

Поскольку массив является **объектным** типом данных, его значения могут быть приведены к типу `Object` или, что то же самое, присвоены переменной типа `Object`. Например,

```
Object o = new int[4];
```

Это дает интересную возможность для массивов, основанных на типе `Object`, хранить в качестве элемента ссылку на самого себя:

```
Object arr[] = new Object[3];
arr[0]=new Object();
arr[1]=null;
arr[2]=arr; // Элемент ссылается
// на весь массив!
```

21.Операции над массивами, просмотр элементов массива, поиск по образцу, сортировка массива, сумма элементов массива

Операции над массивом

- a) Поиск - binarySearch Выполняет поиск с помощью алгоритма бинарного поиска. Имеет перегрузки.
- б) Сортировка - sort
- в) Сумма всех элементов - Arrays.stream(NameArray).sum()

Массивы в Java

- Некоторые методы класса **java.util.Arrays**
- **Методы сортировки:**
 - static void sort(type[] a) // type может быть byte, short, int, long, // char, float, double или тип Object
 - static void sort(type[] a, int from, int to)
 - static void sort(Object[] a, Comparator c)
 - static void sort(Object[] a, int from, int to, Comparator c)
- **Методы бинарного поиска:**
 - static int binarySearch(type[] a, type element)
 - static int binarySearch(Object[] a, Object element, Comparator c).
- **Методы заполнения массива:**
 - static void fill(type[], type value)
 - static void fill(type[], int from, int to, type value)
- **Методы сравнения массивов:**
 - static boolean equals(type[] a1, type[] a2)

11

22. В результате выполнения фрагмента программы

23. Операция конкатенации строк в Джава, ее обозначение и использование

В Java присутствует большое количество решений для работы со строками.

- а) Класс StringBuilder Пример:

```
StringBuilder stringBuilder = new StringBuilder(100);
stringBuilder.append("Baeldung");
```

```
stringBuilder.append(" is");
stringBuilder.append(" awesome");

assertEquals("Baeldung is awesome", stringBuilder.toString());
```

б) Оператор сложения "+". При компиляции символ "+" преобразуется в `StringBuilder.append()`. Использование "+" в цикле является плохой практикой, т.к каждый раз создаётся новый объект `String`, т.к. объект `String` явл. неизменяемым в Java. Пример:

```
String myString = "The " + "quick " + "brown " + "fox...";

assertEquals("The quick brown fox...", myString);
```

в) Строковые методы В классе `String` имеется ряд методов для объединения строк.

в.1) Метод `String.concat` Пример:

```
String myString = "Both".concat(" fickle")
.concat(" dwarves")
.concat(" jinx")
.concat(" my")
.concat(" pig")
.concat(" quiz");
```

```
println(myString);
```

в.2) Метод `String.format` Метод `String.format` позволяет преобразовывать различные объекты в строковый шаблон. В шаблоне находится символ "%"

для представления того, как должны быть расположены в шаблоне различные объекты Пример:

```
String myString = String.format("%s %s %.2f %s %s, %s...", "I",
"ate",
2.5056302,
"blueberry",
"pies",
"oops");

println(myString);
```

в.3) Метод String.Join В Java 8 и выше можно воспользоваться методом String.Join. С помощью него мы можем объединить массив строк с помощью общего разделителя.

Пример:

```
String[] strings = {"I'm", "running", "out", "of", "pangrams!"};
String myString = String.join(" ", strings);
```

```
println(myString);
```

г) Метод StringJoiner() Простой в использование класс. Констурктор принимает разделитель с необязательным префиксом и суффиксом.

Пример:

```
StringJoiner fruitJoiner = new StringJoiner(", ");
fruitJoiner.add("Apples");
fruitJoiner.add("Oranges");
fruitJoiner.add("Bananas");
```

```
println(myString);
```

д) Array.toString Класс Array содержит метод `toString`, который форматирует массив объектов в строку. Данный метод может вывести только строку в квадратных скобках.

Пример: -----

```
String[] myFavouriteLanguages = {"Java", "JavaScript", "Python"};
```

```
String toString = Arrays.toString(myFavouriteLanguages);
```

```
println(myString);
```

е) Collectors.joining Метод `Collectors.joining` позволяет направлять выходные данные потока в одну строку. Потоки имеют множество возможностей для редактирования строки.

Пример: -----

```
List awesomeAnimals = Arrays.asList("Shark", "Panda", "Armadillo");
```

```
String animalString = awesomeAnimals.stream().collect(Collectors.joining(",  
"));
```

```
println(myString);
```

Короче, можно просто строки соединять через “+”, но более быстрый способ (оптимизированный):



```
1 package ua.com.prologistic.stringutils;
2
3 public class StringBufferExample {
4
5     public static void main(String args[]){
6         StringBuffer stringBuffer = new StringBuffer("Prologistic");
7         // объединение строк с помощью StringBuffer
8         stringBuffer.append(".com.ua");
9     }
10 }
```

24. Циклы в Джава, цикл с предусловием, цикл с постусловием, пример записи и использование. Условие окончания цикла.

Циклы — это разновидность управляющих конструкций для организации многократного выполнения одного и того же участка кода.

while — цикл с предусловием;

do..while — цикл с постусловием;

for — цикл со счетчиком (цикл для);

for each.. — цикл “для каждого...” — разновидность for для перебора коллекции элементов.

while, do.. while и for можно использовать в качестве безусловных циклов.

Цикл **while** (с предусловием)

Этот цикл имеет следующую синтаксическую структуру:

while (<условие выполнения цикла>) { <тело цикла> }

В целом он не сильно отличается от цикла for — цикл while не имеет параметров <начальное действие> и <действие после итерации>, а

содержит лишь условие. Это позволяет выполнять <тело цикла> до тех пор, пока выражение в условии возвращает true перед каждой итерацией.

Цикл **do...while** (с постусловием)

Кроме цикла с предусловием while существует вариант, который выполняет хотя бы одну итерацию, а после этого проверяет условие. Это цикл do...while, который называется циклом с постусловием.

Синтаксис **do...while**:

do { <тело цикла> } while (<условие выполнения цикла>);

Сначала отрабатывает действие в <теле цикла>, а потом проверяется <условие выполнения цикла>. Если оно возвращает true, то цикл выполнит действие повторно.

25. Циклы в Джава, итерационный цикл for(), синтаксис, счетчик цикла, условие окончания цикла, модификация счетчика, пример использования

Цикл for. Разновидности

a) **for(<начальная точка>; <условие выхода>; <операторы счетчика>) {}**

Пример:

```
public void printAllElements(String[] stringArray) {  
    for(int i = 0; i < stringArray.length; i++) {
```

```
System.out.println(stringArray[i]);  
}  
}  
-----
```

б) `for(<Тип элемента> <Имя переменной, куда будет записан очередной элемент> :<Название массива>) {}`

Пример: -----

```
public void printAllElements(String[] stringArray) {  
    for(String s : stringArray) {  
        System.out.println(s);  
    }  
}
```

26. Способы объявления массивов в Джава, использование операции new для выделения памяти для элементов массива. Объявление с инициализацией, объявление массива определенного размера без инициализации.

а) Объявление:

- 1) `typeData Name [] = new typeData[length];` - стиль Java
- 2) `typeData []Name = new typeData[length];` - стиль из C++

б) Инициализация:

- 1) `int c [] = new int[2]; c[0] = 1;
c[1] = 2;`
- 2) `int c [] = new int [] {1, 2, 3, 4};`

3) int c [] = {1, 2, 3, 4};

Массив представляет набор однотипных значений. Объявление массива похоже на объявление обычной переменной, которая хранит одиночное значение, причем есть два способа объявления массива:

```
int nums[] = new int[4]; // массив из 4 чисел
int[] nums2 = new int[5]; // массив из 5 чисел
int[] nums2 = { 1, 2, 3, 5 };
int[][] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
int[][] nums = new int[3][];
nums[0] = new int[2];
nums[1] = new int[3];
nums[2] = new int[5];
```

Тема 2. Реализация ООП в Java.

27.Объявление класса на Джава, пример объявления

Модификатор доступа, слово class и название класса: public class Cat {
String name; int age; }

Все четыре уровня доступа имеют только элементы типов и конструкторы.
Это:

- * public;
- * private;
- * protected;
- * если не указан ни один из этих трех типов, то уровень доступа определяется по умолчанию (default).

Первые два из них уже были рассмотрены. Последний уровень (доступ по умолчанию) упоминался в прошлой лекции – он допускает обращения из

того же пакета, где объявлен и сам этот класс. По этой причине пакеты в Java являются не просто набором типов, а более структурированной единицей, так как типы внутри одного пакета могут больше взаимодействовать друг с другом, чем с типами из других пакетов.

Наконец, `protected` дает доступ наследникам класса. Понятно, что наследникам может потребоваться доступ к некоторым элементам родителя, с которыми не приходится иметь дело внешним классам.

Модификатор `private` дает гарантию, что никто напрямую этим полем не пользуется и изменение его типа было бы совсем несложной операцией, связанной с изменением только в одном классе.

Итак, модификаторы доступа упорядочиваются следующим образом (от менее открытых – к более открытым):

`private`

`(none) default`

`protected`

`public`

28. Использования `this` для доступа к компонентам класса.

Если класс содержит какие то свойства внутри себя, то получать доступ к ним можно с помощью `this`. Достаточно указать название объекта и через

точку добавить название свойства с которым хотим работать. А дальше можно присваивать значение переменной и тд.

Пример: Есть класс Animal который содержит свойства Age и Gender. При создании объекта может вызывать конструктор где может быть прописано присваивание значений с помощью this, например:

this.Age = 8 this.

Gender = "Male"

Говоря проще, слово this это указатель на объект класса с которым мы работаем.

Ключевое слово this представляет ссылку на текущий экземпляр класса. Через это ключевое слово мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы.

```
public class Program{  
  
    public static void main(String[] args) {  
  
        Person undef = new Person();  
        undef.displayInfo();  
  
        Person tom = new Person("Tom");  
        tom.displayInfo();  
  
        Person sam = new Person("Sam", 25);  
        sam.displayInfo();  
    }  
}  
class Person{  
  
    String name; // имя  
    int age; // возраст  
    Person()  
    {
```

29. Создание или инстанцирование объектов типа класс:

Класс — это, по сути, шаблон для объекта. Он определяет, как объект будет выглядеть и какими функциями обладать. Каждый объект является объектом какого-то класса.

Процесс создания объектов из классов называется инстанцированием. Объект - это экземпляр класса. Объект - это конкретный экземпляр той абстракции, которую представляет собой класс

Объявление метода с таким же именем как у класса называется конструктором. Конструктор выполняется при создании объекта из класса.

При создании объекта возвращается ссылочное значение (reference value). Ссылочное значение отсылает к объекту. Переменная метит место в памяти, куда может быть сохранено значение. Ссылка на объект (object reference) это переменная, которая хранит ссылочное значение. В Java объектами можно манипулировать только посредством их ссылочных значений или, что то же самое, посредством ссылок, которые содержат ссылочные значения.

Несколько ссылок могут ссылаться на один и тот же объект, т.е. они хранят ссылочное значение одного и того же объекта. Такие ссылки называются псевдонимами. Объектом можно управлять через любой из его псевдонимов.

30.Что такое класс в Java?

Класс - это шаблонная конструкция, которая позволяет описать в программе объект, его свойства (атрибуты или поля класса) и поведение (методы класса).

31.Модификатор доступа или видимости в Джава, виды и использование

Модификаторы доступа - ключевые слова, которые регулируют уровни доступа к разным частям программы

- а) Private Модификатор доступа private ограничивает доступ извне к св-вам и методам класса. Данные и методы с таким модификатором нельзя наследовать.
- б) Protected Поля и методы с модификатором доступа protected будут видны в пределах одного пакета и в пределах классов наследников
- в) package visible\default Если у переменной, метода, класса и т.д. нет никакого модификатора, Java по умолчанию присваивает им модификатор default
- г) Public Модификатор public разрешает доступ к полям и методам класса из вне.

	Доступ из...			
Модификаторы	Любого класса	Класса-наследника	Своего пакета	Своего класса
<code>public</code>	Есть	Есть	Есть	Есть
<code>protected</code>	Нет	Есть	Есть	Есть
без модификатора	Нет	Нет	Есть	Есть
<code>private</code>	Нет	Нет	Нет	Есть

32. Чем отличаются static-метод класса от обычного метода класса:

Обычные методы привязаны к объектам (экземплярам) класса и могут обращаться к обычным-переменным класса (а также к статическим переменным и методам). Статические же методы привязаны к статическому объекту класса и могут обращаться только к статическим переменным и/или другим статическим методам класса. Чтобы вызвать обычный метод у класса, сначала нужно создать объект этого класса, а только потом вызвать метод у объекта. Вызвать обычный метод не у объекта, а у класса нельзя. А чтобы вызвать статический метод, достаточно чтобы просто существовал статический объект класса (который всегда существует после загрузки класса в память). Именно поэтому метод `main()` — статический. Он привязан к статическому объекту класса, для его вызова не нужно создавать никакие объекты.

33. Для чего используется оператор new?

Оператор (операторная функция) new создаёт экземпляр объекта, встроенного или определённого пользователем, имеющего конструктор.

Point p = new Point();

34. Можно ли вызвать static-метод внутри обычного метода?

Статические методы можно вызывать откуда угодно, из любого места программы. А значит, их можно вызывать и из статических методов, и из обычных. Никаких ограничений тут нет

Да, мы могли бы назвать это так.

```
public class A{  
    public static void static_B(){  
    public void normal_C(){  
        A.static_B();  
        static_B();  
    }  
    public void static_C(){  
        A.static_B();  
        static_B();  
    }  
}
```

Приведенный выше код будет работать нормально.

35. Как вызвать обычный метод класса внутри static-метода?

Для того, чтобы вызывать не static метод внутри static метода, нужно создать объект класса внутри static метода и через него вызвать не static метод.

36. Для чего используется в Джава ключевое слово this?

Для доступа и/или обращения к свойствам и компонентам объекта, прописанным в классе данного объекта.

37. Объявление и использование методов, объявленных с модификатором public static.

public: публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором public, видны другим классам из текущего пакета и из внешних пакетов.

При создании объектов класса для каждого объекта создается своя копия нестатических обычных полей. А статические поля являются общими для всего класса. Поэтому они могут использоваться без создания объектов класса.

```
public class Program{  
  
    public static void main(String[] args) {  
  
        double radius = 60;  
  
        System.out.printf("Radius: %f \n", radius); // 60  
        System.out.printf("Area: %f \n", Math.PI * radius); // 188,4  
    }  
}  
class Math{  
    public static final double PI = 3.14;  
}
```

38. Синтаксис объявления методов, тип возвращаемого значения, формальные параметры и аргументы

приватность (static) (final) void/возвращаемое_значение
имя_метода(параметры) {}

метод может возвращать значение типа, указанного в объявлении

39. Методы с пустым списком параметров

Это метод, который не принимает никаких параметров

40. Стандартные методы класса сеттеры и геттеры, синтаксис и их назначение?

Геттеры при их вызове возвращают значение указанного в коде свойства или компонента объекта класса.

Сеттеры при их вызове меняют значение указанного в коде свойства или компонента на то, что пользователь передает при вызове в качестве параметра или на значение указанное в самом коде сеттера.

```
String GetName(){ return this.name; }
```

```
Void SetName(string name) { this.name = name; }
```

Сеттеры инициализируют значения полей класса, геттеры возвращают их

Суть сеттеров в том, чтобы не дать пользователю ввести некорректное значение в поле класса, а геттеров в том, что пользователь не имел явный доступ к приватным полям

41. Может ли быть поле данных класса объявлено как с модификатором static и final одновременно и что это означает?

Когда мы добавляем к полю\методу ключевое слово static, мы привязываем поле\метод к классу, а не к отдельному объекту этого класса. Это значит, что для обращения к статическому полю\методу не надо создавать объект класса. Достаточно написать имя класса и через точку обратиться на прямую к статическим свойствам или методам класса.

Да, может, например public static final float PI = 3,14...

Это статическая переменная, “привязанная к классу”, которую нельзя изменить

42.Методы класса конструкторы, синтаксис и назначение

Заполнение полей класса введенными в качестве параметров переменными.

приватность имя_класса(параметры){}

43.Может ли класс иметь в своем составе несколько конструкторов?

Да, может. Например при разном количестве подаваемых параметров при создании объекта класса.

да (будут называться перегруженными)

44.Может ли конструктор класса возвращать значение?

Конструктор не может возвращать значения (даже значение void). Если в конструкторе написать возвращение значения с помощью оператора return, то компилятор выдаст ошибку.

Тема 3. Реализация наследования в программах на Джаве

45.1)Наследование в Джава. Вид наследования и синтаксис Ключевое слово extends

```
1 class Employee extends Person{  
2     public Employee(String name){  
3         super(name); // если базовый класс определяет конструктор  
4             // то производный класс должен его вызвать  
5     }  
6 }
```

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово **extends**, после которого идет имя базового класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же поля и методы, которые есть в классе Person.

Если в базовом классе определены конструкторы, то в конструкторе производного класса необходимо вызвать один из конструкторов базового класса с помощью ключевого слова **super**. Например, класс Person имеет конструктор, который принимает один параметр. Поэтому в классе Employee в конструкторе нужно вызвать конструктор класса Person. То есть вызов `super(name)` будет представлять вызов конструктора класса Person.

При вызове конструктора после слова `super` в скобках идет перечисление передаваемых аргументов. При этом вызов конструктора базового класса должен идти в самом начале в конструкторе производного класса. Таким образом, установка имени сотрудника делегируется конструктору базового класса.

Причем даже если производный класс никакой другой работы не производит в конструкторе, как в примере выше, все равно необходимо вызвать конструктор базового класса.

46.Что означает перегрузка метода в Java (overload)?

В программировании, перегрузка метода означает использование одинакового имени метода с разными параметрами/ типами. Перегрузка методов — это приём программирования, который позволяет разработчику в одном классе для методов с разными параметрами использовать одно и то же имя. В этом случае мы говорим, что метод перегружен.

47.Что означает переопределение метода в Java (override)?

Переопределение метода (англ. Method overriding) — это возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, но предоставлял иное поведение, не вызывая коллизий при его использовании.

48.В чем разница между перегрузкой и переопределением методов, поясните

Перегрузка метода связана с понятием наличия двух или более методов в одном классе с одинаковым именем, но разными аргументами.

```
void foo(int a)
void foo(int a, float b)
```

Переопределение метода означает наличие двух методов с одинаковыми аргументами, но разных реализаций. Один из них будет существовать в родительском классе, в то время как другой будет находиться в производном или дочернем классе. `@Override`Аннотация, хотя и не

является обязательной, может быть полезна для обеспечения надлежащего переопределения метода во время компиляции.

```
class Parent {  
    void foo(double d) {  
        // do something  
    }  
}  
  
class Child extends Parent {  
    @Override void foo(double d){  
        // this method is overridden.  
    }  
}
```

49. Абстрактные классы в Джава и абстрактные методы класса

В Java абстрактный класс — это класс, который не может быть создан и который используется для обеспечения общей базы для подклассов. Абстрактный класс определяется с помощью ключевого слова `abstract` и может содержать как абстрактные, так и конкретные методы.

Абстрактные методы — это методы, объявленные в абстрактном классе, но не имеющие реализации. Абстрактные методы используются для определения сигнатуры метода, но не реализации, и предназначены для реализации подклассами.

Абстрактные методы определяются с помощью ключевого слова `abstract` и не содержат тела метода. Абстрактные методы должны быть переопределены в подклассах, иначе подкласс должен быть объявлен абстрактным. Также абстрактные методы не могут быть объявлены как `final`, `static` или `private`.

50. Виды наследования в Джава, использование интерфейсов для реализации наследования

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово `extends`, после которого идет имя базового класса. Для класса `Employee` базовым является `Person`, и поэтому класс `Employee` наследует все те же поля и методы, которые есть в классе `Person`.

Производный класс имеет доступ ко всем методам и полям базового класса (даже если базовый класс находится в другом пакете) кроме тех, которые определены с модификатором `private`. При этом производный класс также может добавлять свои поля и методы.

51.Что наследуется при реализации наследования в Джава (какие компоненты класса), а что нет?

Правило 1. Наследуем только один класс.

Java не поддерживает наследование нескольких классов. Один класс - один родитель.

Обратите внимание - нельзя наследовать самого себя!

Правило 2. Наследуется все кроме приватных переменных и методов.

Выше мы говорили, что класс-наследник будет иметь доступ ко всем переменным и методам родителя. Это не совсем так.

На самом деле, все методы и переменные, помеченные модификатором `private`, не доступны классу-наследнику

52. К каким методам и полям базового класса производный класс имеет доступ (даже если базовый класс находится в другом пакете), а каким нет? Область видимости полей и данных из производного класса.

Производный класс имеет доступ ко всем методам и полям базового класса (даже если базовый класс находится в другом пакете) кроме тех, которые определены с модификатором `private`.

Тема 4. Полиморфизм в Джава. Работа со строками. Интерфейсы.

53. Объявление и инициализация переменных типа `String`

`String имя;`

Как и в случае с типами int и double, переменные типа String можно инициализировать сразу при создании. Это, кстати, можно делать вообще со всеми типами в Java.

```
String name = "Аня", city = "New York", message = "Hello!";
```

Тип String не является примитивным типом данных, однако это один из наиболее используемых типов в Java.

Тип String – означает, что переменная хранит строковый тип данных (в нее мы записываем текст).

```
String someText = "hi!"; //объявили и инициализировали переменную "someText" со значением "hi!"
```

Обратите внимание: когда мы инициализируем переменную числового типа - в качестве значения переменной мы указываем просто число. Когда мы инициализируем переменную строкового типа - мы пишем текст в двойных кавычках: "text". В Java строковые (String) данные всегда выделяются двойными кавычками, и только отдельные символы (char) - одинарными 'O'.

Еще один пример:

```
String s = "One more time"; //строковые данные в двойных кавычках
```

```
char sign = 'x'; //переменная "sign" может хранить исключительно символы; сейчас она хранит символ "x"
```

54. Операция конкатенации строк и ее использование

Конкатенация строк – это процесс объединения двух или нескольких строк в одну новую строку. Конкатенация выполняется с помощью оператора +.

Символ + также является оператором сложения в математических операциях. Для примера попробуйте объединить две короткие строки: "Sea" + "horse"; Seahorse. Конкатенация объединяет конец одной строки с началом другой строки, не вставляя пробелов.

Конкатенация – присоединение второй строки к концу первой. Используется для слияния двух строк. В Джаве обозначается символом +

Другой способ объединения строк представляет метод concat():

```
String str1 = "Java";
String str2 = "Hello";
str2 = str2.concat(str1); // HelloJava
```

Метод concat() принимает строку, с которой надо объединить вызывающую строку, и возвращает соединенную строку.

Еще один метод объединения - метод join() позволяет объединить строки с учетом разделителя. Например, выше две строки сливались в одно слово "HelloJava", но в идеале мы бы хотели, чтобы две подстроки были разделены пробелом. И для этого используем метод join():

```
String str1 = "Java";
String str2 = "Hello";
String str3 = String.join(" ", str2, str1); // Hello Java
```

55.Что означает утверждение, что объект класса String является неизменяемым

Методы могут только создавать и возвращать новые строки, в которых хранится результат операции.

Безопасность и String pool основные причины неизменяемости String в Java.

Безопасность объекта неизменяемого класса String обусловлена такими фактами:

- вы можете передавать строку между потоками и не беспокоиться что она будет изменена
- нет проблем с синхронизацией (не нужно синхронизировать операции со String)
- отсутствие утечек памяти
- в Java строки используются для передачи параметров для авторизации, открытия файлов и т.д. - неизменяемость позволяет избежать проблем с доступом
- возможность кэшировать hash code

String pool позволяет экономить память и не создавать новые объекты для каждой повторяющейся строки. В случае с изменяемыми строками - изменение одной приводило бы к изменению всех строк одинакового содержания.

56. При создании объектов строк с помощью класса StringBuffer, например StringBuffer
strBuffer = new StringBuffer(str) можно ли использовать операцию конкатенации строк или необходимо использовать методы класса StringBuffer

Нет, так как это разные классы.

необходимо использовать методы класса StringBuffer

Объекты String являются неизменяемыми, поэтому все операции, которые изменяют строки, фактически приводят к созданию новой строки, что сказывается на производительности приложения. Для решения этой проблемы, чтобы работа со строками проходила с меньшими издержками в Java были добавлены классы **StringBuffer** и **StringBuilder**. По сути они напоминают расширяемую строку, которую можно изменять без ущерба для производительности.

Конкатенация строк через StringBuffer в Java выполняется с помощью метода append.

57. Объявление и инициализация массива строк. Организация просмотра элементов массива

Общая форма объявления и выделение памяти для одномерного массива строк: `String[] arrayName = new String[size];` Инициализация одномерного массива строк точно такая же как инициализация одномерного массива любого другого типа.

```
String[] M = {  
    "Sunday",  
    "Monday",  
    "Tuesday",  
    "Wednesday",  
    "Thursday",  
    "Friday",  
    "Saturday"  
};  
  
String s;  
s = M[2]; // s = "Tuesday"  
s = M[4]; // s = "Thursday"
```

Любая строка в Java имеет тип `String`. Одномерный массив строк имеет тип `String[]`. Двумерный массив строк имеет тип `String[][]`.

Общая форма объявления и выделение памяти для одномерного массива строк

- `String[] arrayName = new String[size];`, где
 - `String` – встроенный в Java класс, который реализует строку символов. Объект типа `String` поддерживает большой набор операций;
 - `arrayName` – имя объекта (экземпляра) типа `String`. Фактически, `arrayName` есть ссылкой на объект типа `String`;
 - `size` – размер массива (количество строк, количество элементов типа `String`).

58. Понятие и объявление интерфейсов в Джава

Интерфейс в языке программирования Java - это абстрактный тип, который используется для указания поведения, которое должны реализовывать классы.

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

Чтобы класс применил интерфейс, надо использовать ключевое слово implements

59. Может ли один класс реализовывать несколько интерфейсов?

Мы можем наследовать только один класс, а реализовать интерфейсов — сколько угодно. Интерфейс может наследовать (extends) другой интерфейс/интерфейсы. Абстрактные классы используются, когда есть отношение "is-a", то есть класс-наследник расширяет базовый абстрактный класс, а интерфейсы могут быть реализованы разными классами, вовсе не связанными друг с другом.

```
1 usage 1 implementation
interface a {
    1 implementation
    void add();
}

1 usage 1 implementation
interface b {
    1 implementation
    void add();
}

class ab implements a, b {
    @Override
    public void add() {
    }
}
```

60.Что входит в состав интерфейса. (какие компоненты может содержать интерфейс)?

С точки зрения программного обеспечения в состав интерфейса входят два компонента: набор процессов ввода-вывода и процесс диалога.

Пользователь вычислительной системы взаимодействует с интерфейсом, через интерфейс он посыпает входные данные и принимает выходные.

Процессы по выполнению заданий вызываются интерфейсом в требуемые моменты времени. На теоретическом уровне интерфейс имеет три основных компонента: Способ общения машины с человеком-оператором Способ общения человека-оператора с машиной (компьютером) Способ пользовательского представления интерфейса.

Пример 1

```
/* File name : NameOfInterface.java */
import java.lang.*;
// Любое количество запросов импорта

public interface NameOfInterface { //создание интерфейса
    // Любое количество полей final и static
    // Любое количество объявлений абстрактных методов
}
```

61. Может ли интерфейс наследоваться от другого интерфейса?

Да.

Класс Java может распространять только один родительский класс. Множественное наследование не допускается. Однако интерфейсы не являются классами, а интерфейс может расширять несколько родительских интерфейсов.

62. Интерфейс Comparable, назначение, его методы и использование в Джава

```
class Person implements Comparable<Person>{

    private String name;
    Person(String name) {

        this.name = name;
    }
    String getName() {return name;}

    public int compareTo(Person p) {

        return name.compareTo(p.getName());
    }
}
```

Comparable используется для сравнения объектов, его основной метод - compareTo, он возвращает:

- 1 если первый объект больше
- 1 если первый объект меньше
- 0 если объекты равны

```
public class Student implements Comparable<Student> {
    private String name;
    private int age;
    private int rollNo;
    private String className;
    // getters and setters
    @Override
    public int compareTo(Student student) {
        return (this.rollNo - student.rollNo);
    }
}
```

63.Какое значение возвращает вызов метода object1.compareTo(object2), который сравнивает 2 объекта obj1 и obj2 в зависимости от объектов?

значение 0, если obj1 и obj2 объекты равны;

если аргумент меньше сравниваемого с ним объекта, значение меньше 0;

если аргумент obj2 больше, чем obj1, значение больше 0

64.Интерфейсные ссылки и их использование в Джава

При создании объектов класса в качестве типа объектной переменной может указываться имя реализованного в классе интерфейса. Другими словами, если класс реализует интерфейс, то ссылку на объект этого класса можно присвоить интерфейсной переменной — переменной, в качестве типа которой указано имя соответствующего интерфейса. Ситуация очень напоминает ту, что рассматривалась в предыдущей главе при наследовании, когда объектная переменная суперкласса ссылалась на объект подкласса. Как и в случае с объектными ссылками суперкласса, через интерфейсную ссылку можно сослаться не на все члены объекта

реализующего интерфейс класса. Доступны только те методы, которые объявлены в соответствующем интерфейсе. С учетом того, что класс может реализовать несколько интерфейсов, а один и тот же интерфейс может быть реализован в разных классах, ситуация представляется достаточно пикантной.

Тема 5. Основные принципы и типы исключительных ситуаций.

65. Понятие исключительной ситуации и ее обработка

Java, исключение – это специальный объект, описывающий исключительную ситуацию, которая возникла в некоторой части программного кода. Объект представляющий исключение, генерируется в момент возникновения исключительной ситуации. После возникновения критической (исключительной) ситуации исключение перехватывается и обрабатывается. Таким образом, возникает понятие генерирования исключения. Генерирование исключения – процесс создания объекта, который описывает данное исключение.

66. В каком случае программа должна использовать оператор throw?

Иногда метод, в котором может генерироваться исключение, сам не обрабатывает это исключение. В этом случае в объявлении метода используется оператор throws, который надо обработать при вызове этого метода. Например, у нас имеется метод вычисления факториала, и нам надо обработать ситуацию, если в метод передается число меньше 1:

```
public static int getFactorial(int num) throws Exception{
```

```
    if(num<1) throw new Exception("The number is less than 1 ");
    int result=1;
    for(int i=1; i<=num;i++){
        result*=i;
    }
    return result;
}
```

67. В Java все исключения делятся на два основных типа. Что это за типы и какие виды ошибок ни обрабатывают?

Все исключения в Java делятся на 2 категории — проверяемые (**checked**) и непроверяемые (**unchecked**). Все исключения, унаследованные от классов RuntimeException и Error, считаются **unchecked-исключениями**, все остальные — **checked-исключениями**.

68. Код ниже вызовет ошибку: Exception <...>
java.lang.ArrayIndexOutOfBoundsException: 4:
Что она означает?

Выход за рамки массива.

ArrayIndexOutOfBoundsException – это исключение, появляющееся во время выполнения. Оно возникает тогда, когда мы пытаемся обратиться к элементу массива по отрицательному или превышающему размер массива индексу.

Тип исключения	С чем связана ошибка
ArithmaticException	Выполнение арифметических операций (например, деление на ноль)
ArrayIndexOutOfBoundsException	Индекс массива выходит за допустимые границы
ArrayStoreException	Присвоение элементу массива значения несовместимого типа
ClassCastException	Попытка выполнить приведение несовместимых типов
IllegalArgumentException	Методу указан неправильный аргумент
IllegalMonitorStateException	Работа с монитором (относится к многопоточному программированию)
IllegalStateException	Ресурс находится в некорректном состоянии
IllegalThreadStateException	Предпринята попытка выполнить некорректную операцию на потоке
IndexOutOfBoundsException	Индекс выходит за допустимый диапазон значений
NegativeArraySizeException	Попытка создать массив отрицательного размера
NullPointerException	Некорректное использование ссылок (обычно когда объектная переменная содержит пустую ссылку)
NumberFormatException	Преобразование строки к числовому значению (когда в число преобразуется строка, содержащая некорректное текстовое представление числа)
SecurityException	Попытка нарушить режим защиты
StringIndexOutOfBoundsException	Неверное индексирование при работе с текстовой строкой
UnsupportedOperationException	Попытка выполнить некорректную операцию

69. Контролируемые исключения (checked)

1. **Checked** исключения, это те, которые должны обрабатываться блоком catch или описываться в сигнатуре метода. **Unchecked** могут не обрабатываться и не быть описаными.

Checked исключения отличаются от **Unchecked** исключения в Java, тем что:

1) Наличие\обработка **Checked** исключения проверяются на этапе компиляции. Наличие\обработка **Unchecked** исключения происходит на этапе выполнения.

70. Неконтролируемые исключения (unchecked) и ошибки, которые они обрабатывают

Собственно исключения наследуются от **класса Exception**. Среди этих исключений следует выделить класс RuntimeException. RuntimeException является базовым классом для так называемой группы непроверяемых исключений (unchecked exceptions) - компилятор не проверяет факт обработки таких исключений и их можно не указывать вместе с оператором throws в объявлении метода. Такие исключения являются следствием ошибок разработчика, например, неверное преобразование типов или выход за пределы массива.

Некоторые из классов непроверяемых исключений:

- `ArithmaticException`: исключение, возникающее при делении на ноль
- `IndexOutOfBoundsException`: индекс вне границ массива
- `IllegalArgumentException`: использование неверного аргумента при вызове метода
- `NullPointerException`: использование пустой ссылки
- `NumberFormatException`: ошибка преобразования строки в число

71. Как реализуются принципы ООП в Java при создании исключений?

Java является объектно-ориентированным языком. Это означает, что писать программы

на Java нужно с применением объектно-ориентированного стиля. И стиль этот основан

на использовании в программе объектов и классов.

Основные принципы ООП:

- Абстракция

- Инкапсуляция
- Наследование
- Полиморфизм

Абстракцию в ООП можно также определить, как способ представления элементов задачи из реального мира в виде объектов в программе. Абстракция всегда связана с обобщением некоторой информации о свойствах предметов или объектов, поэтому главное — это отделить значимую информацию от незначимой в контексте решаемой задачи. При этом уровней абстракции может быть несколько.

```
public abstract class AbstractPhone {  
    private int year;  
  
    public AbstractPhone(int year) {  
        this.year = year;  
    }  
    public abstract void call(int outputNumber);  
    public abstract void ring (int inputNumber);  
}
```

Для исключения подобного вмешательства в конструкцию и работу объекта в ООП

используют принцип инкапсуляции — еще один базовый принцип ООП, при котором

атрибуты и поведение объекта объединяются в одном классе, внутренняя реализация

объекта скрывается от пользователя, а для работы с объектом предоставляется

открытый интерфейс.

```
public class SomePhone {  
  
    private int year;  
  
    private String company;  
  
    public SomePhone(int year, String company) {  
        this.year = year;  
  
        this.company = company;  
    }  
  
    private void openConnection(){  
        //findComutator  
  
        //openNewConnection...  
    }  
  
    public void call() {  
        openConnection();  
  
        System.out.println("Вызываю номер");  
  
    }  
  
    public void ring() {  
        System.out.println("Дзынь-дзынь");  
    }  
}
```

В программировании наследование заключается в использовании уже существующих

классов для описания новых.

```
public class Smartphone extends CellPhone {  
    private String operationSystem;  
    public Smartphone(int year, int hour, String operationSystem) {  
        super(year, hour);  
        this.operationSystem = operationSystem;  
    }  
    public void install(String program){  
        System.out.println("Устанавливаю " + program + " для " + operationSystem);  
    }  
}
```

Принцип в ООП, когда программа может использовать объекты с одинаковым

интерфейсом без информации о внутреннем устройстве объекта, называется

полиморфизмом.

```
AbstractPhone firstPhone = new ThomasEdisonPhone(1879);  
AbstractPhone phone = new Phone(1984);  
AbstractPhone videoPhone=new VideoPhone(2018);  
User user = new User("Андрей");  
user.callAnotherUser(224466,firstPhone);
```

```
// Вращайте ручку  
//Сообщите номер абонента, сэр  
user.callAnotherUser(224466,phone);  
//Вызываю номер 224466  
user.callAnotherUser(224466,videoPhone);  
//Подключаю видеоканал для абонента 224466
```

72. Какой оператор позволяет принудительно выбросить исключение?

Это оператор `throw: throw new Exception();`

В Java есть пять ключевых слов для работы с исключениями:

1. `try` — данное ключевое слово используется для отметки начала блока кода, который потенциально может привести к ошибке.
2. `catch` — ключевое слово для отметки начала блока кода, предназначенного для перехвата и обработки исключений.
3. `finally` — ключевое слово для отметки начала блока кода, которой является дополнительным. Этот блок помещается после последнего блока ‘`catch`’. Управление обычно передаётся в блок ‘`finally`’ в любом случае.
4. `throw` — служит для генерации исключений.
5. `throws` — ключевое слово, которое прописывается в сигнатуре метода, и обозначающее что метод потенциально может выбросить исключение с указанным типом.

Оператор `throw` используется для возбуждения исключения «вручную». Для того, чтобы сделать это, нужно иметь объект подкласса класса `Throwable`, который можно либо получить как параметр оператора `catch`, либо создать с помощью оператора `new`. Ниже приведена общая форма оператора `throw`: `throw [ОбъектТипаThrowable];`

При достижении этого оператора нормальное выполнение кода немедленно прекращается, так что следующий за ним оператор не выполняется. Ближайший окружающий блок `try` проверяется на наличие соответствующего возбужденному исключению обработчика `catch`. Если такой отыщется, управление передается ему.

Если нет, проверяется следующий из вложенных операторов `try`, и так до тех пор пока либо не будет найден подходящий раздел `catch`, либо обработчик исключений исполняющий системы Java не остановит программу, выведя при этом состояние стека вызовов.

Только подклассы класса `Throwable` могут быть возбуждены или перехвачены.

Простые типы (`int`, `char` и т.п.), а также классы, не являющиеся подклассами `Throwable` (например, `String` и `Object`) использоваться в качестве исключений не могут. Наиболее общий путь для использования исключений — создание своих собственных подклассов класса `Exception`.

При создании собственных классов исключений следует принимать во внимание следующие аспекты:

- Все исключения должны быть дочерними элементами `Throwable`.
- Если планируете создать контролируемое исключение, следует расширить класс `Exception`.
- Если хотите создать исключение на этапе выполнения, следует расширить класс `RuntimeException`.

73. Порядок выполнения операторов при обработке блока блока try...catch

При использовании блока try...catch вначале выполняются все инструкции между операторами try и catch. Если в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается и переходит к инструкции catch.

Конструкция try..catch состоит из двух основных блоков: try, и затем catch:

Работает она так:

1. Выполняется код внутри блока try.
2. Если в нём ошибок нет, то блок catch(err) игнорируется, то есть выполнение доходит до конца try и потом прыгает через catch.
3. Если в нём возникнет ошибка, то выполнение try на ней прерывается, и управление прыгает в начало блока catch(err).

При этом переменная err (можно выбрать и другое название) будет содержать объект ошибки с подробной информацией о произошедшем.

Таким образом, при ошибке в try скрипт не «падает», и мы получаем возможность обработать ошибку внутри catch

Тема 6. Джениерики и использование контейнерных классов в Джава

74. Абстрактный тип данных Stack (стек) в Java

Стек (Stack). Стековая память отвечает за хранение ссылок на объекты кучи и за хранение типов значений (также известных в Java как примитивные типы), которые содержат само значение, а не ссылку на объект из кучи. Кроме того, переменные в стеке имеют определенную видимость, также называемую областью видимости. Используются только объекты из активной области.

в которой вы можете выполнять базовые операции, такие как push, pop и т. д.

75. Универсальные типы или обобщенные типы данных, для чего создаются?

Универсальный тип является одиночным элементом программирования, который используется для выполнения одинаковой функциональности для различных типов данных. При определении универсальных классов или процедур не нужно определять отдельную версию для каждого типа данных, для которых может потребоваться выполнение этой функциональности.

76.Объявление обобщённого класса коллекции с параметризованным методом для обработки массива элементов коллекции на основе цикла for each (определение общего метода для отображения элементов массива)

77.Что представляет из себя класс ArrayList и в каком случае используется

ArrayList – это реализация динамического использования массива в Java. ArrayList следует использовать, когда в приоритете доступ по индексу, так как эти операции выполняются за константное время. Добавление в конец списка в среднем тоже выполняется за константное время.

При разработке часто бывает сложно предсказать, какого размера понадобятся

массивы. Поэтому функция динамического выделения памяти во время работы программы необходима каждому языку программирования. Динамическим называется массив, размер которого может измениться во время исполнения программы. В Java для такой цели существует класс ArrayList.

Что такое класс ArrayList?

ArrayList — реализация изменяемого массива интерфейса List, часть Collection

Framework, который отвечает за список (или динамический массив), расположенный в пакете `java.utils`. Этот класс реализует все необязательные операции со списком и предоставляет методы управления размером массива, который используется для хранения списка. В основе `ArrayList` лежит идея динамического массива. А именно, возможность добавлять и удалять элементы, при этом будет увеличиваться или уменьшаться по мере необходимости.

Что хранит `ArrayList`?

Только ссылочные типы, любые объекты, включая сторонние классы. Строки, потоки вывода, другие коллекции. Для хранения примитивных типов данных используются классы-обертки.

78. Класс `Pattern` и его использование

`Pattern Class` – объект класса `Pattern` представляет скомпилированное представление регулярного выражения. В классе `Pattern` публичный конструктор не предусмотрен. Для создания шаблона, вам сперва необходимо вызвать один из представленных публичных статических методов `compile()`, который далее произведет возврат объекта класса `Pattern`.

Большая часть функциональности по работе с регулярными выражениями в Java сосредоточена в пакете `java.util.regex`.

Само регулярное выражение представляет шаблон для поиска совпадений в строке. Для задания подобного шаблона и поиска подстрок в строке, которые удовлетворяют данному шаблону, в Java определены классы `Pattern` и `Matcher`.

Класс Java `Pattern` можно использовать двумя способами.

1. через метод `Pattern.matches()`, чтобы быстро проверить, соответствует ли текст (`String`) заданному выражению;
2. скомпилировать экземпляр `Pattern`, используя `Pattern.compile()`, который можно использовать несколько раз, чтобы сопоставить выражение с несколькими текстами.

79. Класс Math и его использование

Класс `Math` содержит методы для того, чтобы выполнить основные числовые операции, такие как элементарный экспоненциал, логарифм, квадратный корень, и тригонометрические функции. В отличие от некоторых из числовых методов класса `StrictMath`, все реализации эквивалентных функций класса `Math` не определяются, чтобы возвратить поразрядное те же самые результаты.

- Класс `Math` используется Вычисление абсолютных значений (значений по модулю)
- Вычисление значений тригонометрических функций (синусов, косинусов и т.д.)
- Возвведение в различные степени
- Извлечение корней различных степеней
- Генерация случайных чисел
- Округления

Для выполнения различных математических операций в Java в пакете *java.lang* определен класс Math. Рассмотрим его основные методы:

- `abs(double value)`: возвращает абсолютное значение для аргумента `value`
- `acos(double value)`: возвращает арккосинус `value`. Параметр `value` должен иметь значение от -1 до 1
- `asin(double value)`: возвращает арксинус `value`. Параметр `value` должен иметь значение от -1 до 1
- `atan(double value)`: возвращает арктангенс `value`
- `cbrt(double value)`: возвращает кубический корень числа `value`
- `ceil(double value)`: возвращает наименьшее целое число с плавающей точкой, которое не меньше `value`
- `cos(double d)`: возвращает косинус угла `d`

- `cosh(double d)`: возвращает гиперболический косинус угла d
 - `exp(double d)`: возвращает основание натурального логарифма, возведенное в степень d
 - `floor(double d)`: возвращает наибольшее целое число, которое не больше d
 - `floorDiv(int a, int b)`: возвращает целочисленный результат деления a на b
 - `log(double a)`: возвращает натуральный логарифм числа a
 - `log1p(double d)`: возвращает натуральный логарифм числа $(d + 1)$
 - `log10(double d)`: возвращает десятичный логарифм числа d
 - `max(double a, double b)`: возвращает максимальное число из a и b
 - `min(double a, double b)`: возвращает минимальное число из a и b
 - `pow(double a, double b)`: возвращает число a , возведенное в степень b
 - `random()`: возвращает случайное число от 0.0 до 1.0
 - `rint(double value)`: возвращает число `double`, которое представляет ближайшее к числу `value` целое число
- `round(double d)`: возвращает число d , округленное до ближайшего целого числа
- `scalb(double value, int factor)`: возвращает произведение числа `value` на 2 в степени `factor`
- `signum(double value)`: возвращает число 1, если число `value` положительное, и -1, если значение `value` отрицательное. Если `value` равно 0, то возвращает 0
- `sin(double value)`: возвращает синус угла `value`
- `sinh(double value)`: возвращает гиперболический синус угла `value`
 - `sqrt(double value)`: возвращает квадратный корень числа `value`
- `tan(double value)`: возвращает тангенс угла `value`
- `tanh(double value)`: возвращает гиперболический тангенс угла `value`
 - `toDegrees(double value)` переводит радианы в градусы и `toRadians(double value)` - градусы в радианы

Также класс `Math` определяет две константы: `Math.E` и `Math.PI`.

80.Как вызываются методы класса Math и что при этом происходит?

```
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Введите радиус круга: ");
        int radius = in.nextInt();

        long area = Math.round(Math.PI * Math.pow(radius, 2));
        System.out.printf("Площадь круга с радиусом %d равна %d\n", radius, area);
    }
}
```

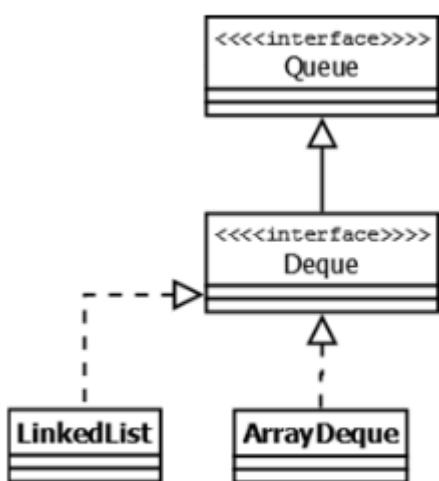
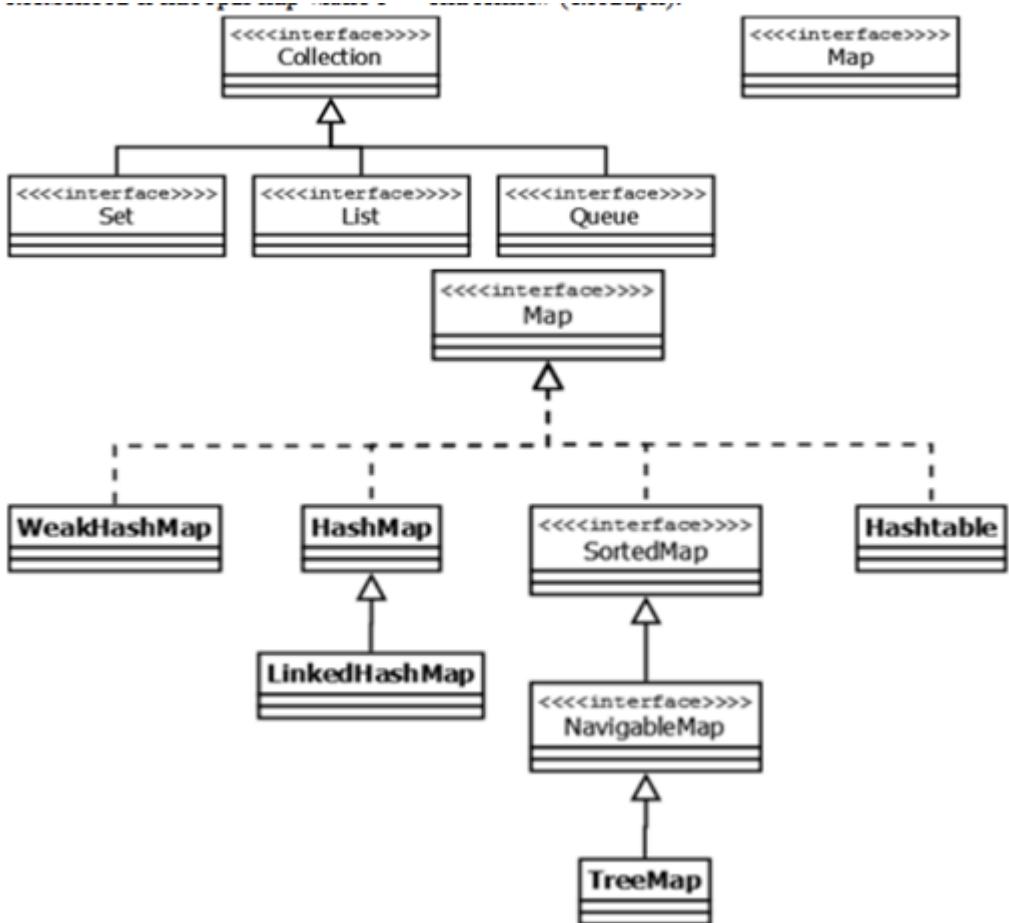
Тема 7. Java Core. Джениерики (продолжение) и использование контейнерных классов Java Framework Collection

81.Структура коллекций в Java Collection Framework. Иерархия интерфейсов

Java Collection Framework — иерархия интерфейсов и их реализаций, которая является частью JDK и позволяет разработчику пользоваться большим количеством структур данных из «коробки».

Базовые понятия

На вершине иерархии в Java Collection Framework располагаются 2 интерфейса: Collection и Map. Эти интерфейсы разделяют все коллекции, входящие во фреймворк на две части по типу хранения данных: простые последовательные наборы элементов и наборы пар «ключ — значение» (словари).



82. Коллекция HashMap, создание и методы работы с ней

HashMap - хранит значения в произвольном порядке, но позволяет быстро искать элементы карты. Позволяет задавать ключ или значение ключевым словом **null**.

Сегодня мы поговорим о еще одной структуре данных — **Map**. Ее официальное русское название — “ассоциативный массив”, но его используют нечасто. Более распространены варианты “словарь”, “карта”, или (чаще всего) — сленговый англизм “мапа” :) Внутри **Map** данные хранятся в формате “ключ”-“значение”, то есть по парам. И в качестве ключей, и в качестве значений могут выступать любые объекты — числа, строки или объекты других классов. Ранее мы разбирали структуры данных, где элементы хранятся сами по себе. В массиве, или списке `ArrayList/LinkedList` мы храним какое-то количество элементов.

Создание HashMap в Java и работа с классом

Создается данная реализация очень просто:

```
public static void main(String[] args) {
```

```
    HashMap<Integer, String> passportsAndNames = new HashMap<>();  
    passportsAndNames.put(212133, "Лидия Аркадьевна Бубликова");  
    passportsAndNames.put(162348, "Иван Михайлович Серебряков");  
    passportsAndNames.put(8082771, "Дональд Джон Трамп");  
    System.out.println(passportsAndNames);
```

```
}
```

83. Чем является класс LinkedList

Обобщенный класс `LinkedList` представляет структуру данных в виде связанного списка. Он наследуется от класса `AbstractSequentialList` и реализует интерфейсы `List`, `Dequeue` и `Queue`. То есть он соединяет функциональность работы со списком и функциональность очереди

84. Одним из ключевых методов интерфейса Collection является метод Iterator iterator(). Что возвращает это метод?

Одним из ключевых методов интерфейса Collection является метод Iterator<E> iterator(). Он возвращает итератор - то есть объект, реализующий интерфейс Iterator.

Интерфейс Iterator имеет следующее определение:

Реализация интерфейса предполагает, что с помощью вызова метода next() можно получить следующий элемент. С помощью метода hasNext() можно узнать, есть ли следующий элемент, и не достигнут ли конец коллекции. И если элементы еще имеются, то hasNext() вернет значение true. Метод hasNext() следует вызывать перед методом next(), так как при достижении конца коллекции метод next() выбрасывает исключение NoSuchElementException. И метод remove() удаляет текущий элемент, который был получен последним вызовом next().

```
public interface Iterator <E>{  
    E next();  
    boolean hasNext();
```

```
void remove();
}

import java.util.*;

public class Program {

    public static void main(String[] args) {

        ArrayList<String> states = new ArrayList<String>();
        states.add("Germany");
        states.add("France");
        states.add("Italy");
        states.add("Spain");

        Iterator<String> iter = states.iterator();
        while(iter.hasNext()){

            System.out.println(iter.next());
        }
    }
}
```

Интерфейс **Iterator** предоставляет ограниченный функционал. Гораздо больший набор методов предоставляет другой итератор - интерфейс **ListIterator**. Данный итератор используется классами, реализующими интерфейс **List**, то есть классами **LinkedList**, **ArrayList** и др.

85.Что возвращает метод next()

Метод **next()** возвращает следующий (очередной) элемент коллекции

Е **next()**: возвращает текущий элемент и переходит к следующему, если такого нет, то генерируется исключение **NoSuchElementException**

86. Что возвращает метод hasNext()

Метод hasNext() используется, чтобы проверять, есть ли еще элементы.

True или False

boolean hasNext(): возвращает true, если в коллекции имеется следующий элемент, иначе возвращает false

87. Обобщенный класс HashSet класс коллекция, наследует свой функционал от класса AbstractSet, а также реализует интерфейс Set. Что он себя представляет?

Интерфейс Set расширяет интерфейс Collection и представляет набор уникальных элементов. Set не добавляет новых методов, только вносит изменения унаследованные. В частности, метод add() добавляет элемент в коллекцию и возвращает true, если в коллекции еще нет такого элемента.

Обобщенный класс HashSet представляет хеш-таблицу. Он наследует свой функционал от класса AbstractSet, а также реализует интерфейс Set.

Хеш-таблица представляет такую структуру данных, в которой все объекты имеют уникальный ключ или хеш-код. Данный ключ позволяет уникально идентифицировать объект в таблице.

Для создания объекта HashSet можно воспользоваться одним из следующих конструкторов:

- HashSet(): создает пустой список
- HashSet(Collection<? extends E> col): создает хеш-таблицу, в которую добавляет все элементы коллекции col
- HashSet(int capacity): параметр capacity указывает начальную емкость таблицы, которая по умолчанию равна 16
- HashSet(int capacity, float koef): параметр koef или коэффициент заполнения, значение которого должно быть в пределах от 0.0 до 1.0, указывает, насколько должна быть заполнена емкость объектами прежде чем произойдет ее расширение. Например, коэффициент 0.75 указывает, что при заполнении емкости на 3/4 произойдет ее расширение.

Класс HashSet не добавляет новых методов, реализуя лишь те, что объявлены в родительских классах и применяемых интерфейсах:

88. Обобщенный класс HashMap класс коллекция, которая реализует интерфейс Map для хранения пар ключ-значение. Что он себя представляет?

Интерфейс Map<K, V> представляет отображение или иначе говоря словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта Map. Такие коллекции облегчают поиск элемента, если нам известен ключ - уникальный идентификатор объекта.

Следует отметить, что в отличие от других интерфейсов, которые представляют коллекции, интерфейс Map НЕ расширяет интерфейс Collection.

Среди методов интерфейса Map можно выделить следующие:

Операции с Map

Ниже мы рассмотрим 6 операций с Map:

1. `put(K key, V value)` - добавляет элемент в карту;
2. `get(Object key)` - ищет значение по его ключу;
3. `remove(Object key)` - удаляет значение по его ключу;
4. `containsKey(Object key)` - спрашивает, есть ли в карте заданный ключ;
5. `containsValue(Object value)` - спрашивает есть ли в карте заданное значение;
6. `size()` - возвращает размер карты (количество пар "ключ-значение").

Среди методов интерфейса Map можно выделить следующие:
(<https://metanit.com/java/tutorial/5.8.php>)

`void clear():` очищает коллекцию

boolean containsKey(Object k): возвращает true, если коллекция содержит ключ k

boolean containsValue(Object v): возвращает true, если коллекция содержит значение v

Set<Map.Entry<K, V>> entrySet(): возвращает набор элементов коллекции. Все элементы представляют объект Map.Entry

boolean equals(Object obj): возвращает true, если коллекция идентична коллекции, передаваемой через параметр obj

boolean isEmpty: возвращает true, если коллекция пуста

V get(Object k): возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение null

V getOrDefault(Object k, V defaultValue): возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение defaultVlue

V put(K k, V v): помещает в коллекцию новый объект с ключом k и значением v. Если в коллекции уже есть объект с подобным ключом, то он перезаписывается. После добавления возвращает предыдущее значение для ключа k,

если он уже был в коллекции. Если же ключа еще не было в коллекции, то возвращается значение null

V putIfAbsent(K k, V v): помещает в коллекцию новый объект с ключом k и значением v, если в коллекции еще нет элемента с подобным ключом.

Set<K> keySet(): возвращает набор всех ключей отображения

Collection<V> values(): возвращает набор всех значений отображения

void putAll(Map<? extends K, ? extends V> map): добавляет в коллекцию все объекты из отображения map

V remove(Object k): удаляет объект с ключом k

int size(): возвращает количество элементов коллекции

Чтобы положить объект в коллекцию, используется метод put, а чтобы получить по ключу - метод get. Реализация интерфейса Map также позволяет получить наборы как ключей, так и значений. А метод entrySet() возвращает набор всех элементов в виде объектов Map.Entry<K, V>.

Обобщенный интерфейс Map.Entry<K, V> представляет объект с ключом типа K и значением типа V и определяет следующие методы:

boolean equals(Object obj): возвращает true, если объект obj, представляющий интерфейс Map.Entry, идентичен текущему

K getKey(): возвращает ключ объекта отображения

V getValue(): возвращает значение объекта отображения

V setValue(V v): устанавливает для текущего объекта значение v

int hashCode(): возвращает хеш-код данного объекта

При переборе объектов отображения мы будем оперировать этими методами для работы с ключами и значениями объектов.

Базовым классом для всех отображений является абстрактный класс AbstractMap, который реализует большую часть методов интерфейса Map. Наиболее распространенным классом отображений является HashMap, который реализует интерфейс Map и наследуется от класса AbstractMap.

Пример использования класса:

```
Map<Integer, String> states = new HashMap<Integer, String>();  
  
states.put(1, "Germany");  
  
states.put(2, "Spain");  
  
states.put(4, "France");
```

```
states.put(3, "Italy");
```

Тема 8. Стандартные потоки ввода-вывода. Сериализация.

89.Стандартные потоки ввода-вывода, предоставляемые Java

Для ввода данных используется класс Scanner из библиотеки пакетов Java. Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Java представлен объектом — System.in. А стандартный поток вывода (дисплей) — уже знакомым вам объектом System.out. Есть ещё стандартный поток для вывода ошибок — System.err, но работа с ним выходит за рамки нашего курса.

Для ввода данных используется класс Scanner из библиотеки пакетов Java. Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Java представлен объектом — System.in. А стандартный поток вывода (дисплей) — уже знакомым вам объектом System.out. Есть ещё стандартный поток для вывода ошибок — System.err, но работа с ним выходит за рамки нашего курса.

InputStream	OutputStream	Reader	Writer
FileInputStream	FileOutputStream	FileReader	FileWriter
BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
DataInputStream	DataOutputStream		
ObjectInputStream	ObjectOutputStream		

90. Понятие сериализации, интерфейс Serializable

Сериализация — это процесс сохранения состояния объекта в последовательность байт. Serializable - это маркерный интерфейс (не имеет элемента данных и метода). Он используется для “маркировки” классов java, чтобы объекты этих классов могли получить определенные возможности.

91. Какие объекты можно сериализовать?

Сразу надо сказать, что сериализовать можно только те объекты, которые реализуют интерфейс Serializable. Этот интерфейс не определяет никаких методов, просто он служит указателем системе, что объект, реализующий его, может быть сериализован. Сериализация. Класс ObjectOutputStream.

92. Какие методы определяет интерфейс Serializable?

Интерфейс Serializable обеспечивает автоматическую сериализацию, используя инструменты Java Object Serialization. Serializable не объявляет методов: он действует как маркер, сообщая инструментам Object Serialization, что класс Бина является сериализуемым. Пометка класса Serializable означает, что JVM сообщают, что уверены, что класс будет работать с сериализацией по умолчанию.

Сериализация. Класс ObjectOutputStream

Для сериализации объектов в поток используется класс **ObjectOutputStream**. Он записывает данные в поток.

Для создания объекта ObjectOutputStream в конструктор передается поток, в который производится запись:

```
1 | ObjectOutputStream(OutputStream out)
```

Для записи данных ObjectOutputStream использует ряд методов, среди которых можно выделить следующие:

- **void close():** закрывает поток
- **void flush():** очищает буфер и сбрасывает его содержимое в выходной поток
- **void write(byte[] buf):** записывает в поток массив байтов
- **void write(int val):** записывает в поток один младший байт из val
- **void writeBoolean(boolean val):** записывает в поток значение boolean
- **void writeByte(int val):** записывает в поток один младший байт из val
- **void writeChar(int val):** записывает в поток значение типа char, представленное целочисленным значением
- **void writeDouble(double val):** записывает в поток значение типа double
- **void writeFloat(float val):** записывает в поток значение типа float
- **void writeInt(int val):** записывает целочисленное значение int
- **void writeLong(long val):** записывает значение типа long
- **void writeShort(int val):** записывает значение типа short
- **void writeUTF(String str):** записывает в поток строку в кодировке UTF-8
- **void writeObject(Object obj):** записывает в поток отдельный объект

93. Что означает понятие десериализация?

Сериализация — это процесс сохранения состояния объекта в последовательность байт.

Десериализация — это процесс восстановления объекта из этих байт. Любой Java-объект преобразуется в последовательность байт

94. Класс File, определенный в пакете java.io, не работает напрямую с потоками. В чем состоит его задача?

Его задачей является управление информацией о файлах и каталогах.

В Java есть специальный класс (File), с помощью которого можно управлять файлами на диске компьютера. Читать директории, к примеру.

95. При работе с объектом класса FileOutputStream происходит вызов метода FileOutputStream.write(), что в результате этого происходит?

Главное назначение класса FileOutputStream — запись байтов в файл

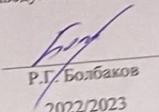
- void write(int b) записывает один байт в выходной поток. Аргумент этого метода имеет тип int, что позволяет вызывать write, передавая ему выражение, при этом не нужно выполнять приведение его типа к byte.
- void write(byte b[]) записывает в выходной поток весь указанный массив байтов.

- void write(byte b[], int off, int len) записывает в поток len байтов массива, начиная с элемента b[off].

*Наследование, this, super.

*Параметризованные классы и методы

ДЖАВА БИЛЕТ № 1

<p>МИНОБРНАУКИ РОССИИ Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА – Российский технологический университет» Институт информационных технологий Кафедра инструментального и прикладного программного обеспечения</p>	<p>ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 1 Дисциплина: Программирование на языке Джава</p> <p>Форма обучения: очная Курс 2 Семестр 3</p>	<p>Утверждено на заседании кафедры (протокол № 1 от «24» августа 2022 г.) Заведующий кафедрой  R.G. Болбаков 2022/2023 учебный год</p>
<p>1. Парадигма объектно-ориентированного программирование. Основные принципы ООП и их реализация в языке программирования Java и C++</p> <p>2. Понятие структуры данных списков. Линейный список. Виды списков и их реализация на Java. Доступ к элементу структуры данных списков. Использование списков. Трудоемкость операций со списками.</p> <p>3. Напишите метод под названием alternate, который принимает два списка целых чисел в качестве параметров и возвращает новый список, содержащий чередующиеся элементы из двух списков, в следующем порядке:</p> <ul style="list-style-type: none"> • Первый элемент из первого списка • Первый элемент из второго списка • Второй элемент из первого списка • Второй элемент из второго списка • Третий элемент из первого списка • Третий элемент из второго списка <p>Если списки не содержат одинаковое количество элементов, оставшиеся элементы из более длинного списка должны быть расположены последовательно в конце. Например, для первого списка (1, 2, 3, 4, 5) и второго списка (6, 7, 8, 9, 10, 11, 12) вызов alternate (list1, list2) должен вернуть список, содержащий (1, 6, 2, 7, 3, 8, 4, 9, 5, 10, 11, 12). Не изменяйте передаваемые списки параметров.</p>		

1. Парадигма объектно-ориентированного программирование. Основные принципы ООП и их реализация в языке программирования Java и C++.

ООП - методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Эта концепция предоставила программистам возможность

делить программу на кучу «модулей»-классов, каждый из которых делает свою часть работы. Все объекты, взаимодействуя между собой, образуют работу нашей программы.

Кроме того, написанный нами код можно повторно использовать в другом месте программы, что также экономит большое количество времени.

Есть 4 принципа, которые составляют парадигму ООП: наследование, абстракция, инкапсуляция, полиморфизм.

Наследование — механизм, который позволяет описать новый класс на основе существующего (родительского). При этом свойства и функциональность родительского класса заимствуются новым классом. Ключевое слово “extends” используется для наследования классов

Существует несколько видов **наследования**:

- Одиночное **наследование** (single inheritance). При этом класс может наследовать только один суперкласс.
Синтаксис:

```
class SubClass extends SuperClass {  
    // тело класса  
}
```

- Множественное **наследование**. В Java множественное **наследование** не поддерживается. Однако стоит упомянуть, что класс может реализовывать несколько интерфейсов.
- Иерархическое **наследование**. При этом класс может наследовать суперкласс, который в свою очередь может наследовать другой суперкласс и т.д. Синтаксис:

```
class SubClass extends SuperClass {  
    // тело класса  
}  
class SubSubClass extends SubClass {  
    // тело класса  
}
```

Абстракция - это выделение главных, наиболее значимых характеристик предмета и наоборот — отbrasывание второстепенных, незначительных.

Инкапсуляция - метод, используемый для реализации абстракции в ООП. Она ограничивает доступ к членам и методам класса. Для инкапсуляции в ООП применяются ключи модификаторов доступа. Например, в языке java инкапсуляция достигается с помощью ключевых слов private, protected и public,default.

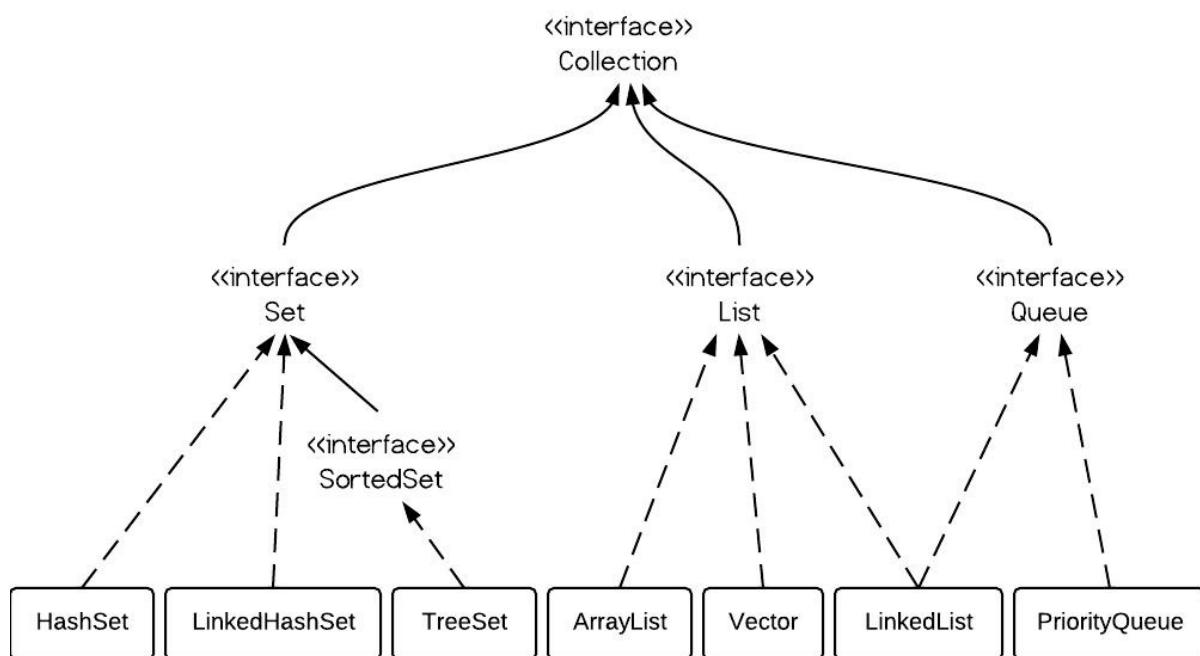
Полиморфизм — это возможность работать с несколькими типами так, будто это один и тот же тип. При этом поведение объектов будет разным в зависимости от типа, к которому они принадлежат. (Есть класс Animal с

методом voice, класс Dog и Cat наследуются от Animal, метод voice у них будет другой)

<https://javarush.com/groups/posts/1966-principih-obhhektno-orientirovannogo-programmирования>

2. Понятие структуры данных список. Линейный список. Виды списков и их реализация на Java. Доступ к элементу структуры данных список. Использование списков. Трудоемкость операций со списками.

Структура данных — это контейнер, который хранит данные в определенном макете. Этот «макет» позволяет структуре данных быть эффективной в некоторых операциях и неэффективной в других. **Линейные**, элементы образуют последовательность или линейный список, обход узлов линеен. Примеры: Массивы. Связанный список, стеки и очереди.



3. Напишите метод под названием alternate, который принимает два списка целых чисел в качестве параметров и возвращает новый список, содержащий чередующиеся элементы из двух списков, в следующем порядке:

- Первый элемент из первого списка
- Первый элемент из второго списка
- Второй элемент из первого списка
- Второй элемент из второго списка
- Третий элемент из первого списка
- Третий элемент из второго списка

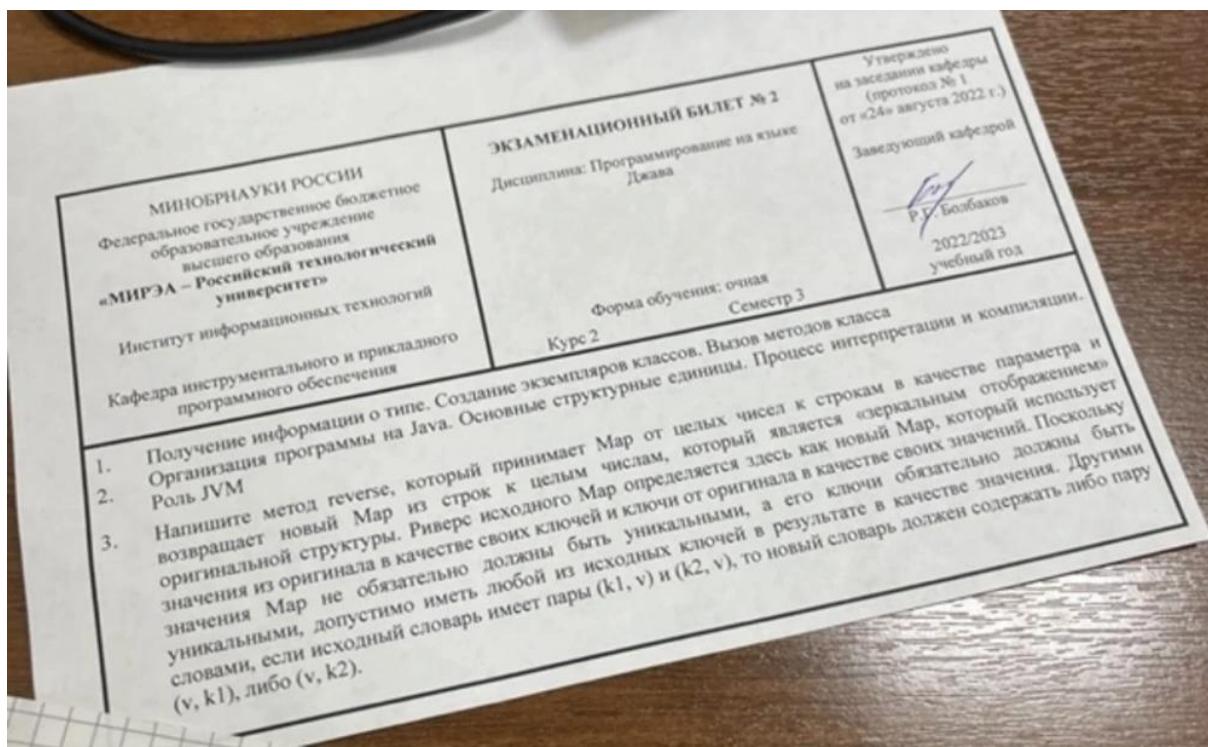
Если списки не содержат одинаковое количество элементов, оставшиеся элементы из более длинного списка должны быть расположены последовательно в конце. Например, для первого списка (1, 2, 3, 4, 5) и второго списка (6, 7, 8, 9, 10, 11, 12) вызов alternate (list1, list2) должен вернуть список, содержащий (1, 6, 2, 7, 3, 8, 4, 9, 5, 10, 11, 12). Не изменяйте передаваемые списки параметров.

```
import java.util.ArrayList;
import java.util.List;

public class Main
{
    public static List alternate(List first, List second)
    {
        int sizea=first.size();
        int sizeb=second.size();
        List new_list=new ArrayList();
        int size;
        if(sizea>sizeb)
        {
            size=sizeb;
        }
        else if(sizeb>sizea)
        {
            size=sizea;
        }
        else
        {
            size=sizea;
        }
        for(int i=0;i<size;i++)
        {
            new_list.add(first.get(i));
            new_list.add(second.get(i));
        }
        if(size<sizea)
        {
            for(int i=size;i<sizea;i++)
            {
                new_list.add(first.get(i));
            }
        }
    }
}
```

```
        new_list.add(first.get(i));
    }
}
else if(size<sizeb)
{
    for(int i=size;i<sizeb;i++)
    {
        new_list.add(second.get(i));
    }
}
return new_list;
}
public static void main(String[] args)
{
    List first_test=new ArrayList();
    List second_test=new ArrayList();
    first_test.add(1);
    first_test.add(2);
    first_test.add(3);
    first_test.add(4);
    first_test.add(5);
    second_test.add(6);
    second_test.add(7);
    second_test.add(8);
    second_test.add(9);
    second_test.add(10);
    second_test.add(11);
    second_test.add(12);
    System.out.println(alternate(first_test,second_test));
}
}
```

БИЛЕТ № 2



1. Получение информации о типе. Создание экземпляров классов. Вызов методов класса.

В Java класс — это шаблон для создания объектов. Класс определяет данные и поведение (методы) типа объекта.

Чтобы объявить класс в Java, используется ключевое слово `class`, за которым следует имя класса. Тело класса заключено в фигурные скобки {}.

```
class MyClass {  
    // class body  
}
```

Пример класса с методом:

```
class Point {  
    private double x;  
    private double y;  
    void setLocation(double x, double y) {  
        this.x = x; // refers to the "x" field of the current object  
        this.y = y; // refers to the "y" field of the current object  
    }  
}
```

Чтобы создать объект класса, необходимо в программе написать [ИмяКласса] [ИмяОбъекта] = new [ИмяКласса](Параметры);

Для вызова метода в программе нужно написать [ИмяОбъекта].[ИмяМетода](Параметры);

Пример:

```
Point point1 = new Point();
point1.setLocation(i, j);
```

2. Организация программы на Java. Основные структурные единицы. Процесс интерпретации и компиляции. Роль JVM

Организация программы на Java: Java является объектно-ориентированным языком, поэтому всю программу можно представить как набор взаимодействующих между собой классов и объектов.

```
1 public class Program{
2
3     public static void main (String args[]){
4
5         System.out.println("Hello Java!");
6     }
7 }
```

Основу программы составляет класс Program, или же любое другое имя класса. При определении класса вначале идет модификатор доступа public, который указывает, что данный класс будет доступен всем, то есть мы сможем его запустить из командной строки. Далее идет ключевое слово class, а затем название класса. После названия класса идет блок кода, в котором расположено содержимое класса.

Входной точкой в программу на языке Java является метод main, который определен в классе Program. Именно с него начинается выполнение программы. Он обязательно должен присутствовать в программе

Основные структурные единицы: массив, переменная, класс

Процесс интерпретации и компиляции: Java интерпретируемый и компилируемый одновременно.

Роль JVM: Java-программы компилируются в байт-код, который можно запускать на любом устройстве, на котором установлена виртуальная машина Java (JVM). Это означает, что программы Java являются переносимыми, то есть их можно запускать на любом устройстве, поддерживающем Java, независимо от используемого оборудования и операционной системы.

Обратная сторона медали JVM это то, что Java программы работают намного медленнее чем такие же на Си, но для большинства приложений это не существенное замедление. В качестве основных причин медленной работы Java можно назвать:

1) динамическое связывание: в отличие от программ на языке Си, связывание выполняется во время выполнения, каждый раз, когда программа запускается на Java.

2) работа интерпретатора времени выполнения: преобразование байткода в собственный машинный код выполняется во время выполнения программы на Java, что еще больше снижает скорость

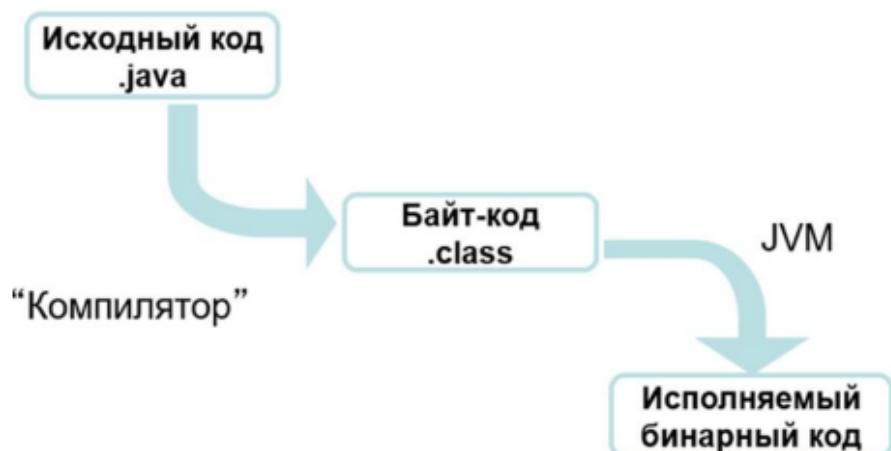


Рисунок 1.7. Технологическая схема обработки кода на Java

3. Напишите метод `reverse`, который принимает `Map` от целых чисел к строкам в качестве параметра и возвращает новый `Map` из строк к целым числам, который является «зеркальным отображением» оригинальной структуры. Риверс исходного `Map` определяется здесь как новый `Map`, который использует значения из оригинала в качестве своих ключей и ключи от оригинала в качестве своих значений. Поскольку значения `Map` не обязательно должны быть уникальными, а его ключи обязательно должны быть уникальными, допустимо иметь любой из исходных ключей в результате в качестве значения. Другими словами, если исходный словарь имеет пары (`k1, v`) и (`k2, v`), то новый словарь должен содержать либо пару (`v, k1`), либо (`v, k2`).

```
import java.util.HashMap;
import java.util.Map;
```

```
public class Second {

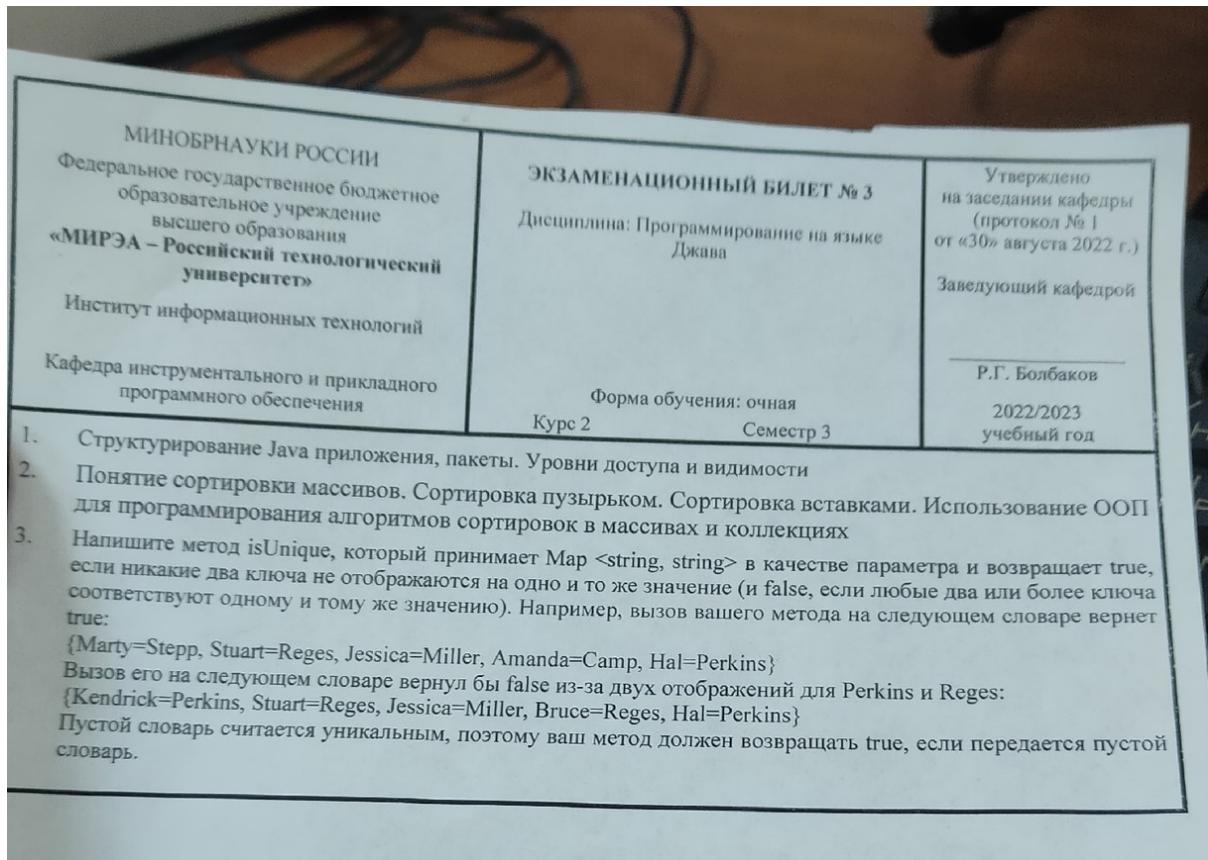
    public static Map reverse(Map<Integer, String> map){
```

```
Map<String, Integer> new_map = new HashMap<>();
for(Map.Entry<Integer, String> entry : map.entrySet()){
    new_map.put(entry.getValue(), entry.getKey());
}

return new_map;
}

public static void main(String[] args) {
    Map<Integer, String> map = new HashMap<>();
    map.put(1, "cat");
    map.put(2, "wulf");
    map.put(3, "rat");
    map.put(4, "dog");
    System.out.println(map);
    System.out.println(reverse(map));
}
}
```

БИЛЕТ № 3



1. Структурирование Джава приложения, пакеты. Уровни доступа и видимости

Все члены класса в языке Java - поля и методы - имеют модификаторы доступа. Модификаторы доступа позволяют задать допустимую область видимости для членов класса, то есть контекст, в котором можно употреблять данную переменную или метод.

В Java используются следующие модификаторы доступа:

- `public`: публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором `public`, видны другим классам из текущего пакета и из внешних пакетов.
- `private`: закрытый класс или член класса, противоположность модификатору `public`. Закрытый класс или член класса доступен только из кода в том же классе.
- `protected`: такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах
- Модификатор по умолчанию. Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.

2. Понятие сортировки массивов. Сортировка пузырьком. Сортировка вставками. Использование ООП для программирования алгоритмов сортировок в массивах и коллекциях.

Для сортировки массива можно использовать один из алгоритмов сортировки, предоставляемых стандартной библиотекой Java, например `Arrays.sort()`. Например:

```
int[] numbers = {4, 3, 2, 1, 5};  
Arrays.sort(numbers);
```

Сортировка Пузырьком:

```
import java.util.Arrays;  
  
public class Main {  
    public static void main(String[] args) {  
        int [] mas = {11, 3, 14, 16, 7};  
  
        boolean isSorted = false;  
        int buf;  
        while(!isSorted) {  
            isSorted = true;  
            for (int i = 0; i < mas.length-1; i++) {  
                if(mas[i] > mas[i+1]){  
                    isSorted = false;  
  
                    buf = mas[i];  
                    mas[i] = mas[i+1];  
                    mas[i+1] = buf;  
                }  
            }  
        }  
        System.out.println(Arrays.toString(mas));  
    }  
}
```

Сортировка вставками:

```
public static void insertionSort(int[] sortArr) {  
    int j;  
    //сортировку начинаем со второго элемента, т.к. считается, что первый элемент  
    уже отсортирован  
    for (int i = 1; i < sortArr.length; i++) {  
        //сохраняем ссылку на индекс предыдущего элемента  
        int swap = sortArr[i];  
        for (j = i; j > 0 && swap < sortArr[j - 1]; j--) {  
            //элементы отсортированного сегмента перемещаем вперёд, если они  
            больше элемента для вставки  
            sortArr[j] = sortArr[j - 1];  
        }  
        sortArr[j] = swap;  
    }  
  
    public static void main(String args[]) {  
        int[] sortArr = {12, 6, 4, 1, 15, 10};  
    }
```

```
insertionSort(sortArr);
for(int i = 0; i < sortArr.length; i++){
    System.out.print(sortArr[i] + "\n");
}
}
```

БИЛЕТ № 6

1. Оператор new. Понятие ссылки и указателя на объект. Реализация в C++ и Java. Время жизни объекта.

Оператор new: Оператор new - создает экземпляр указанного класса и возвращает ссылку на вновь созданный объект.

Понятие ссылки и указателя на объект: Ссылка - это объект, указывающий на определенные данные, но не хранящий их. Получение объекта по ссылке называется *разыменованием*.

Указатель хранит адрес ячейки памяти

Выделять память физически не требуется (здесь нет работы с адресной арифметикой как в языке Си или С++), также отсутствуют такие типы данных как указатели.

Ссылка this - иногда будет требоваться, чтобы метод ссылался на вызвавший его объект. Чтобы это было возможно, в Java определено ключевое слово this. Оно может использоваться внутри любого метода для ссылки на текущий объект. То есть this всегда служит ссылкой на объект, для которого был вызван метод. Ключевое слово this можно использовать везде, где допускается ссылка на объект типа текущего класса.

Реализация в C++ и Java:

C++:

Java:

```
13 int main()
14 {
15     int t = 237; // Простая переменная
16     int *p; // Создание указателя, который принимает лишь адрес другой переменной
17     p = &t; // Устанавливаем адрес нашей первой переменной
18     cout << p << endl;
19     cout << *p << endl;
20     return 0;
21 }
22
```

```
0x7ffed869a81c           input
237
```

```
Car car = new Car();
```

Переменная «car»

ссылается

Объект
«Car»

Время жизни объекта (или жизненный цикл объекта):



2. Организация работы с файлами в Java

FileInputStream и FileOutputStream - это классы, которые позволяют читать и записывать данные в файлы.

```
1 public class Main {  
2  
3     public static void main(String[] args) throws IOException {  
4  
5         File file = new File("C:\\\\Users\\\\Username\\\\Desktop\\\\test.txt");  
6         FileOutputStream fileOutputStream = new FileOutputStream(file);  
7  
8         String greetings = "Привет! Добро пожаловать на JavaRush - лучший сайт для тех, кто хочет стать  
9             программистом!";  
10        fileOutputStream.write(greetings.getBytes());  
11        fileOutputStream.close();  
12    }  
13 }  
14 }  
15 }
```

Вот как будет выглядеть реализация чтения данных из файла при помощи [FileInputStream](#):

```
1 public class Main {  
2  
3     public static void main(String[] args) throws IOException {  
4  
5         FileInputStream fileInputStream = new FileInputStream("C:\\\\Users\\\\Username\\\\Desktop\\\\test.txt")  
6  
7         int i;  
8  
9         while((i=fileInputStream.read())!= -1){  
10            System.out.print((char)i);  
11        }  
12    }  
13 }  
14 }
```

Мы считываем из файла по одному байту, преобразуем считанные байты в символы и выводим их в консоль.

А вот и результат в консоли:

```
So close no matter how far  
Couldn't be much more from the heart  
Forever trusting who we are  
And nothing else matters
```

Класс `File` определяет объекты, которые представляют собой файлы и каталоги. Класс `File` не работает напрямую с потоками, он используется для создания объектов, которые могут быть использованы для создания ПОТОКОВ ВВОДА-ВЫВОДА.

Основные методы класса `File`:

- `createNewFile()` - создает новый файл, если файл с таким именем уже существует, то метод вернет `false`;
- `delete()` - удаляет файл;
- `exists()` - проверяет существует ли файл;
- `isDirectory()` - проверяет является ли файл каталогом;
- `isFile()` - проверяет является ли файл обычным файлом;
- `isHidden()` - проверяет является ли файл скрытым;
- `length()` - возвращает размер файла в байтах;
- `list()` - возвращает массив строк, содержащий имена файлов и каталогов, которые находятся в каталоге;
- `mkdir()` - создает каталог;
- `renameTo(File dest)` - переименовывает файл.
- `toPath()` - возвращает объект типа `Path`, который представляет собой путь к файлу.

3. Напишите метод с именем `guavaSort`, который принимает массив строк в качестве параметра и упорядочивает строки в массиве в отсортированном порядке возрастания. В частности, ваш алгоритм сортировки должен использовать JFC для Multiset или Multimap для реализации варианта алг

оритма блочной сортировки, который будет работать со строками. Используйте коллекцию JFC для подсчета вхождений строк, аналогично тому, как это делается в блочной сортировке, а затем поместите эти строки обратно в массив в отсортированном порядке. Например, предположим, что нашему методу передается следующий массив:

[Farm, Zoo, Car, Apple, Bee, Golf, Bee, Dog, Golf, Zoo, Zoo, Bee, Bee, Apple]

Ваша коллекция должна хранить следующие вхождения строк:

[Apple × 2, Bee × 4, Car, Dog, Farm, Golf × 2, Zoo × 3]

Что вы должны использовать, чтобы поместить строки обратно в массив в отсортированном порядке:

[Apple, Apple, Bee, Bee, Bee, Bee, Car, Dog, Farm, Golf, Golf, Zoo, Zoo, Zoo]

Ваш код должен выполняться за время $O(N \log N)$ и использовать память $O(N)$, где N - количество элементов в массиве. Вы можете предположить, что переданный массив и все строки в нем не равны нулю. Не используйте никаких других вспомогательных коллекций, кроме одного Multiset или Multimap.

```
import java.util.Arrays;
import java.util.HashMap;

public class Program {
    public static void main(String[] args) {
        String[] arr = {"Farm", "Zoo", "Car", "Apple", "Bee",
"Golf", "Bee", "Dog", "Golf", "Zoo", "Zoo", "Bee", "Bee",
"Apple"};
        guavaSort(arr);
    }
}
```

```
    }
    private static void guavaSort(String[] arr) {
        HashMap<String, Integer> matchesMap = new HashMap<>();
        for (int i = 0; i != arr.length; i++) {
            if (matchesMap.containsKey(arr[i]))
                matchesMap.replace(arr[i],
matchesMap.get(arr[i]) + 1);
            else
                matchesMap.put(arr[i], 1);
        }
        System.out.println(matchesMap);
        Arrays.sort(arr);
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

БИЛЕТ № 7

МИНИСТЕРСТВО РОССИИ Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА – Российский технологический университет» Институт информационных технологий Кафедра инструментального и прикладного программного обеспечения	ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 7 Дисциплина: Программирование на языке Джава	Утверждено на заседании кафедры (протокол № 1 от «24» августа 2022 г.) Заведующий кафедрой Р. Г. Болбаков 2022/2023 учебный год
	Форма обучения: очная Курс 2 Семестр 3	

1. Переопределение методов в Java, абстрактные методы.

2. Иерархия классов ввода вывода. Работа с файлами в Java. Работа с файлами. СерIALIZАЦИЯ объектов

3. Напишите метод removeAll, который можно добавить в класс LinkedList. Метод должен эффективно удалить из отсортированного списка целых чисел все значения, появляющиеся во втором отсортированном списке целых чисел. Например, предположим, что переменные LinkedList list1 и list2 ссылается на следующие списки:

list1: [1, 3, 5, 7]
list2: [1, 2, 3, 4, 5]

Если была вызвана list1.removeAll(list2); то, списки должны хранить следующие значения после вызова:

list1: [7]
list2: [1, 2, 3, 4, 5]

Обратите внимание, что все значения из list1, которые появляются в list2, были удалены, а list2 не изменился. Если бы вместо этого был вызов list2.removeAll(list1); списки будут иметь следующие значения:

list1: [1, 3, 5, 7]
list2: [2, 4]

Оба списка гарантированно находятся в отсортированном (неубывающем) порядке, хотя в любом списке могут быть дубликаты. Поскольку списки отсортированы, вы можете решить эту проблему очень эффективно за один проход данных. Ваше решение должно выполняться за время $O(M + N)$, где M и N - длины двух списков. Предположим, что мы добавляем этот метод в класс LinkedList, как показано ниже. Вы не можете вызывать какие-либо другие методы класса для решения этой задачи, вы не можете создавать новые узлы и не можете использовать какие-либо вспомогательные структуры данных для решения этой проблемы (например, массив, ArrayList, Queue, String и т. д.). Вы также не можете изменять какие-либо поля данных узлов. Вы должны решить эту задачу, переставив ссылки на списки.

```
public class LinkedList {  
    private ListNode front;  
    ...  
  
    public class ListNode {  
        public int data;  
        public ListNode next;  
        ...  
    }  
}
```

1. Переопределение методов в Java, абстрактные методы.

Переопределение метода - это возможность создавать методы с одинаковым именем и параметрами, но с разным телом. При этом методы должны быть объявлены в одном классе, но в разных подклассах.

Пример:

```
public class calc {  
    public int add(int a, int b) {  
        return a + b;  
    }  
}  
public class calc2 extends calc {  
    @Override  
    public int add(int a, int b) {
```

```
        return a + b + 1;
    }
}
```

Строка `@override` не является обязательной, но ее использование позволяет избежать ошибок при переопределении методов.

Стоит отметить, что переопределение методов не является полной заменой перегрузки методов. Перегрузка методов используется для создания различных вариантов одного и того же метода, а переопределение методов используется для изменения поведения метода в подклассе.

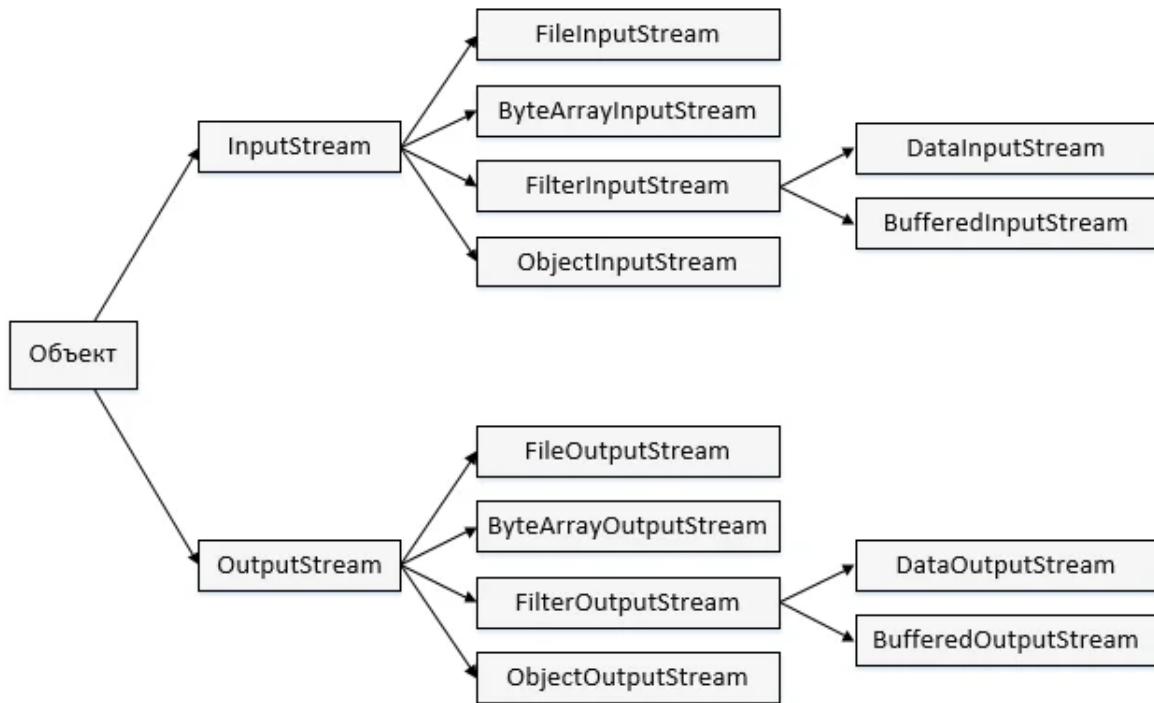
Абстрактные методы — это методы, объявленные в абстрактном классе, но не имеющие реализации. Абстрактные методы используются для определения сигнатуры метода, но не реализации, и предназначены для реализации подклассами.

Абстрактные методы определяются с помощью ключевого слова `abstract` и не содержат тела метода. Абстрактные методы должны быть переопределены в подклассах, иначе подкласс должен быть объявлен абстрактным. Также абстрактные методы не могут быть объявлены как `final`, `static` или `private`.

Стоит отметить, что абстрактные классы могут содержать как абстрактные, так и конкретные методы. Конкретные методы — это методы, которые имеют реализацию. Конкретные методы могут быть объявлены как `final`, `static` или `private`, но не могут быть объявлены как `abstract`.

```
abstract class Shape {
    abstract void draw();
    void msg() {
        System.out.println("This is a shape");
    }
}
```

2. Иерархия классов ввода вывода. Работа с файлами в Java. Работа с файлами. СерIALIZАЦИЯ объектов



Класс Scanner — это класс Java, который используется для чтения входных данных из различных источников, таких как файлы, сетевые сокеты и стандартный поток ввода (stdin). Он является частью пакета java.util и может использоваться для чтения.

Например:

```
Scanner scanner = new Scanner(System.in);
```

После того, как вы создали объект `Scanner`, вы можете использовать его различные методы для чтения **ввода** из стандартного потока **ввода**. Например, чтобы прочитать целое число из стандартного потока **ввода**, вы можете использовать метод `nextInt()`:

```
int i = scanner.nextInt();
```

и другие методы (см IntelliJ)

Класс System — это встроенный класс Java, предоставляющий доступ к различным функциям и переменным системного уровня. Одной из функций, предоставляемых классом `System`, является возможность работы со стандартным потоком вывода, представляющим собой поток, который используется для вывода данных на консоль или в окно терминала.

Для вывода данных в стандартный поток вывода в Java можно использовать статический метод `println()` объекта `System.out`. Этот метод принимает один аргумент, то есть данные, которые вы хотите вывести, и выводит данные на консоль, за которыми следует символ новой строки. Например:

Например, чтобы вывести строку «Hello, world!» в консоль можно использовать следующий код:

```
System.out.println("Hello, world!");
```

и другие методы (см IntelliJ)

[FileInputStream](#) и [OutputStream](#) - это классы, которые позволяют читать и записывать данные в файлы.

Сериализация объектов:

91. Какие объекты можно сериализовать?

В Java можно сериализовать только объекты, которые реализуют интерфейс `Serializable`. Если класс не реализует этот интерфейс, то при попытке сериализовать объект этого класса будет выброшено исключение `NotSerializableException`.

Все поля класса должны быть сериализуемыми. Если поле не сериализуемое, то при сериализации объекта будет выброшено исключение `NotSerializableException`.

92. Какие методы определяет интерфейс Serializable?

Интерфейс `Serializable` не содержит никаких методов, он просто помечает классы, которые могут быть сериализованы.

93. Что означает понятие десериализация?

Десериализация - это процесс преобразования последовательности байтов в объект. Это обратный процесс сериализации.

В Java десериализация объектов реализуется с классом `ObjectInputStream`.

Пример десериализации объекта:

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class Main {
    public static void main(String[] args) {
        try (ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream("file.txt"))) {
            Person person = (Person) objectInputStream.readObject();
            System.out.println(person);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

3. Напишите метод removeAll, который можно добавить в класс LinkedList. Метод должен эффективно удалить из отсортированного списка целых чисел все значения, появляющиеся во втором отсортированном списке целых чисел. Например, предположим, что переменные LinkedList list1 и list2 ссылаются на следующие списки:

list1: [1, 3, 5, 7]

list2: [1, 2, 3, 4, 5]

Если была вызвана list1.removeAll(list2); то, списки должны хранить следующие значения после вызова:

list1: [7]

list2: [1, 2, 3, 4, 5]

Обратите внимание, что все значения из list1, которые появляются в list2, были удалены, а list2 не изменился. Если бы вместо этого был вызов list2.removeAll(list1); списки будут иметь следующие значения:

list1: [1, 3, 5, 7]

list2: [2, 4]

Оба списка гарантированно находятся в отсортированном (неубывающем) порядке, хотя в любом списке могут быть дубликаты. Поскольку списки отсортированы, вы можете решить эту проблему очень эффективно за один проход данных. Ваше решение должно выполняться за время $O(M + N)$, где M и N - длины двух списков. Предположим, что мы добавляем этот метод в класс LinkedList, как показано ниже. Вы не можете вызывать какие-либо другие методы класса для решения этой задачи, вы не можете создавать новые узлы и не можете использовать какие-либо вспомогательные структуры данных для

решения этой проблемы (например, массив, ArrayList, Queue, String и т. д.). Вы также не можете изменять какие-либо поля данных узлов. Вы должны решить эту задачу, переставив ссылки на списки.

```
public class Main {  
    public static void main(String[] args) {  
        LinkedList list1 = new LinkedList();  
        LinkedList list2 = new LinkedList();  
        list1.add(new int[]{1, 3, 5, 7});  
        System.out.println("list1: " + list1);  
        list2.add(new int[]{1, 2, 3, 4, 5});  
        System.out.println("list2: " + list2);  
        list2.removeAll(list1);  
    }  
}
```

```

        System.out.println("list1: " + list2);
    }

}

class LinkedIntList {
    static class ListNode {
        protected int data;
        protected ListNode next;

        ListNode(int a) {
            data = a;
            next = null;
        }
    }

    private ListNode head, last;

    public void add(int value) {
        ListNode newNode = new ListNode(value);
        if (head == null) {
            head = newNode;
            last = head;
        }
        last.next = newNode;
        last = newNode;
    }

    public void add(int[] arr) {
        for (int i : arr) {
            add(i);
        }
    }

    @Override
    public String toString() {
        ListNode tmp = head;
        String s = "";
        while (tmp != null) {
            s += tmp.data + " ";
            tmp = tmp.next;
        }
        return s;
    }

    public void removeAll(LinkedIntList list) {
        ListNode tmp = head, obs = list.head, slow = head;
        while (tmp != null && obs != null) {
            if (tmp.data < obs.data) {
                slow = tmp;
                tmp = tmp.next;
                continue;
            }
            if (tmp.data >= obs.data) {
                if (obs.next == null) {
                    obs.next = tmp;
                    obs = tmp;
                } else {
                    obs = obs.next;
                }
            }
        }
    }
}

```

```
        }
        if (tmp.data > obs.data) {
            obs = obs.next;
            continue;
        }
        if (tmp == head) {
            head = head.next;
            slow = head;
            tmp = head;
            continue;
        }
        slow.next = tmp.next;
        tmp = slow.next;
    }
}
```

БИЛЕТ № 9

1. Статические поля и методы. Класс Math, его основные методы.

В Java статический метод — это метод, связанный с классом, а не с экземпляром (объектом) класса. Статический метод можно вызывать для самого класса, а не для объекта класса.

Обычный (нестатический) метод — это метод, связанный с экземпляром (объектом) класса. Обычный метод может быть вызван для объекта класса и может получить доступ к полям и методам объекта.

Методы с модификатором static не могут использовать ключевое слово this.

Статический метод можно вызвать из обычного (нестатического) метода в Java.

Чтобы вызвать статический метод из обычного метода, можно использовать имя класса, за которым следует точка и имя метода.

Статическое поле данных — это поле данных, которое принадлежит классу, а не объекту. Статическое поле данных может быть объявлено как final, что означает, что его значение не может быть изменено после инициализации.

Класс Math — это класс стандартной библиотеки Java. Он предоставляет методы для выполнения основных математических операций, таких как возведение в степень, вычисление квадратного корня, тригонометрические функции и т.д.

```
import java.lang.Math;
public class Main {
    public static void main(String[] args) {
        System.out.println(Math.abs(-10)); // 10
        System.out.println(Math.pow(2, 3)); // 8.0
        System.out.println(Math.sqrt(9)); // 3.0
        System.out.println(Math.sin(Math.PI / 2)); // 1.0
    }
}
```

Класс Math содержит множество статических методов, которые можно использовать для выполнения различных математических операций. Например, метод abs() возвращает абсолютное значение числа, метод pow() возвращает значение числа, возведенного в степень, метод sqrt() возвращает квадратный корень числа, а метод sin() возвращает синус угла в радианах.

Конструктор класса Math объявлен как приватный, поэтому его нельзя использовать для создания объектов. Все методы класса Math также объявлены как статические, поэтому они могут быть вызваны без создания объекта класса Math.

2. Обобщенное программирование. Понятие и использование дженериков в Java.

В Java универсальные типы или обобщенные типы данных создаются для того, чтобы создавать классы, интерфейсы или методы, которые могут работать с различными типами данных. Они используются для более гибкого и многократно используемого кода, а также для обеспечения безопасности типов.

Универсальные типы или обобщенные типы данных в Java определяются с помощью параметра типа, который указывается в объявлении класса, интерфейса или метода. Параметр типа указывается в угловых скобках <> после имени класса, интерфейса или метода. Параметр типа может быть любым допустимым идентификатором, но принято использовать одну букву из английского алфавита. Например, T, E, K, V и т.д.

Обозначения стандартных параметров типа:

T - type. Обозначает тип данных.

E - element. Обозначает элемент.

K - key. Обозначает ключ.

V - value. Обозначает значение.

N - number. Обозначает число.

S, U, V - second, third, fourth типы.

Синтаксис объявления универсального типа или обобщенного типа данных в Java:

```
public class ClassName<T, U> {  
    private T t;  
    private U u;  
    public ClassName(T t, U u) {  
        this.t = t;  
        this.u = u;  
    }  
    public T getT() {  
        return t;  
    }  
    public U getU() {  
        return u;  
    }  
}
```

```
    }  
}
```

Основное назначение использования дженериков — это абстракция работы над типами при работе с коллекциями («Java Collection Framework»).

3. Напишите метод `removeDuplicates`, который можно добавить в класс `LinkedList`. Метод должен удалить любые дубликаты из связанного списка целых чисел. Результирующий список должен иметь значения в том же относительном порядке, что и их первое вхождение в исходном списке. Другими словами, значение `i` должно появляться перед значением `z` в окончательном списке тогда и только тогда, когда первое вхождение `i` появилось до первого появления `j` в исходном списке. Например, если переменная с именем `list` хранит следующий список: [14, 8, 14, 12, 1, 14, 11, 8, 8, 10, 4, 9, 1, 2, 5, 2, 4, 12, 12]

После вызова `list.removeDuplicates ()`; список должен хранить эти значения в таком виде: [14, 8, 12, 1, 11, 10, 4, 9, 2, 5]

Предположим, что мы добавляем этот метод в класс `LinkedList`, как показано ниже. Вы не можете вызывать какие-либо другие методы класса для решения этой задачи, вы не можете создавать новые узлы и не можете использовать какие-либо вспомогательные структуры данных для решения этой проблемы (например, массив, `ArrayList`, `Queue`, `String` и т. д.). Вы также не можете изменять какие-либо поля данных узлов. Вы должны решить эту задачу, переставив ссылки в списке.

```
public class Program {  
    public static void main(String[] args) {  
        LinkedList list1 = new LinkedList();  
        list1.add(new int[]{14, 8, 14, 12, 1, 14, 11, 8, 8, 10, 4, 9, 1,  
2, 5, 2, 4, 12, 12});  
        System.out.println(list1);  
        list1.removeDuplicates();  
        System.out.println(list1);  
    }  
}  
  
class LinkedList {  
    static class ListNode {  
        protected int data;  
        protected ListNode next;  
  
        ListNode(int a) {  
            data = a;  
            next = null;  
        }  
    }
```

```
private ListNode head, last;
public int length = 0;

public void add(int value) {
    ListNode newNode = new ListNode(value);
    if (head == null) {
        head = newNode;
        last = head;
    }
    last.next = newNode;
    last = newNode;
    length++;
}

public void add(int[] arr) {
    for (int i : arr) {
        add(i);
    }
}

@Override
public String toString() {
    StringBuilder result = new StringBuilder();
    result.append(head == null ? 0 : head.data);
    ListNode tmp = head == null ? null : head.next;
    while (tmp != null) {
        result.append(" ").append(tmp.data);
        tmp = tmp.next;
    }
    return result.toString();
}

public void removeDuplicates() {
    ListNode current = head, duplicate;
    while (current != null) {
        duplicate = current;
        while (duplicate.next != null) {
            if (duplicate.next.data == current.data) {
                duplicate.next = duplicate.next.next;
                if (duplicate.next == null)
                    last = duplicate;
                continue;
            }
            duplicate = duplicate.next;
        }
        current = current.next;
    }
}
```

БИЛЕТ № 11

МИНОБРНАУКИ РОССИИ Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА – Российский технологический университет» Институт информационных технологий Кафедра инструментального и прикладного программного обеспечения	ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 11 Дисциплина: Программирование на языке Джава Форма обучения: очная Курс 2 Семестр 3	Утверждено на заседании кафедры (Протокол № 1 от «24» августа 2022 г.) Заведующий кафедрой  Р. Г. Болбаков 2022/2023 учебный год
---	--	---

1. Конструкторы, назначение и использование. Вызов конструктора родительского класса, неявный вызов конструктора родительского класса, порядок инициализации экземпляра Java класса.

2. Использование языка UML для проектирования и документирования объектно-ориентированных программ. Основные UML диаграммы для отображения отношений между классами в ООП программах

3. Напишите метод firstLast, который можно добавить в класс LinkedList, который перемещает первый элемент списка в конец списка. Предположим, что переменная LinkedList с именем list хранит следующие элементы спереди (слева) и сзади (справа):
[18, 4, 27, 9, 54, 5, 63]
Если вы сделали вызов list.firstLast (), список будет хранить элементы в следующем порядке:
[4, 27, 9, 54, 5, 63, 18]
Если список пуст или содержит только один элемент, его содержимое не должно изменяться.
Соблюдайте следующие ограничения в вашем решении:

- Не вызывайте никакие другие методы объекта LinkedList, такие как add, remove или size.
- Не создавайте новые объекты ListNode (хотя у вас может быть столько переменных ListNode, сколько вам нужно). Не используйте другие структуры данных, такие как массивы, списки, очереди и т. д.
- Не изменяйте данные любого существующего узла; изменять список только путем изменения ссылок между узлами.

Ваше решение должно выполняться за время O (N), где N - количество элементов в связанном списке.

Предположим, что вы добавляете этот метод в класс LinkedList (который использует класс ListNode) как показано ниже:

```
public class LinkedList {  
    ...  
}  
public class ListNode {  
    public int data;  
    public ListNode next;  
}  
...
```

1. Конструкторы, назначение и использование. Вызов конструктора родительского класса, неявный вызов конструктора родительского класса, порядок инициализации экземпляра Java класса.

Конструктор - это метод, имя которого совпадает с именем класса, инициализирующая переменные-члены, распределяющая память для их хранения. Конструктор создаётся при создании нового объекта

Конструктор определяет действия, выполняемые при создании объекта класса, и является важной частью класса. Если явного конструктора нет, то Java автоматически создаст его для использования по умолчанию.

```

class Box {
    int width; // ширина коробки
    int height; // высота коробки
    int depth; // глубина коробки

    // Конструктор
    Box() {
        width = 10;
        height = 10;
        depth = 10;
    }

    // вычисляем объём коробки
    int getVolume() {
        return width * height * depth;
    }
}

```

При вызове конструктора подкласса будет отработан конструктор и родительского класса, и самого подкласса. При создании объекта в первую очередь вызывается конструктор его базового класса, а только потом — конструктор самого класса, объект которого мы создаем.

Здесь поговорим о том, что происходит внутри компьютера и Java-машины, когда мы пишем, например:

```
Cat cat = new Cat();
```

Мы ранее уже говорили об этом, но на всякий случай вспомним:

1. Сначала для хранения объекта выделяется память.
2. Далее Java-машина создает ссылку на этот объект (в нашем случае ссылка — это `Cat cat`).
3. В завершение происходит инициализация переменных и вызов конструктора (этот процесс мы рассмотрим подробнее).

*Конструктор родительского класса вызывается с помощью метода `super()`

2. Использование языка UML для проектирования и документирования объектно-ориентированных программ. Основные UML диаграммы для отображения отношений между классами в ООП программах

UML - это язык для визуализации, специфирования, конструирования и документирования артефактов программных систем.

Словарь языка UML включает три вида строительных блоков:

- сущности;
- отношения;
- диаграммы.

Сущности - это абстракции, являющиеся основными элементами модели.

Отношения связывают различные сущности.

Диаграммы группируют представляющие интерес совокупности сущностей.

В UML имеется четыре типа сущностей:

- структурные;
- поведенческие;
- группирующие;
- аннотационные.

В языке UML определены четыре типа отношений:

- зависимость;
- ассоциация;
- обобщение;
- реализация.

В UML выделяют девять типов диаграмм:

- диаграммы классов;
- диаграммы объектов;
- диаграммы прецедентов;
- диаграммы последовательностей;
- диаграммы кооперации;
- диаграммы состояний;
- диаграммы действий;
- диаграммы компонентов;
- диаграммы развертывания.

*<https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2>

3. Напишите метод `firstLast`, который можно добавить в класс `LinkedList`, который перемешает первый элемент списка в конец списка. Предположим, что переменная `LinkedList` с именем `list` хранит следующие элементы спереди (слева) и сзади (справа):

[18, 4, 27, 9, 54, 5, 63]

Если вы сделали вызов `list.firstLast()`; список будет хранить элементы в следующем порядке:

[4, 27, 9, 54, 5, 63, 18]

Если список пуст или содержит только один элемент, его содержимое не должно изменяться.

Соблюдайте следующие ограничения в вашем решении:

- Не вызывайте никакие другие методы объекта `LinkedList`, такие как `add`, `remove` или `size`.

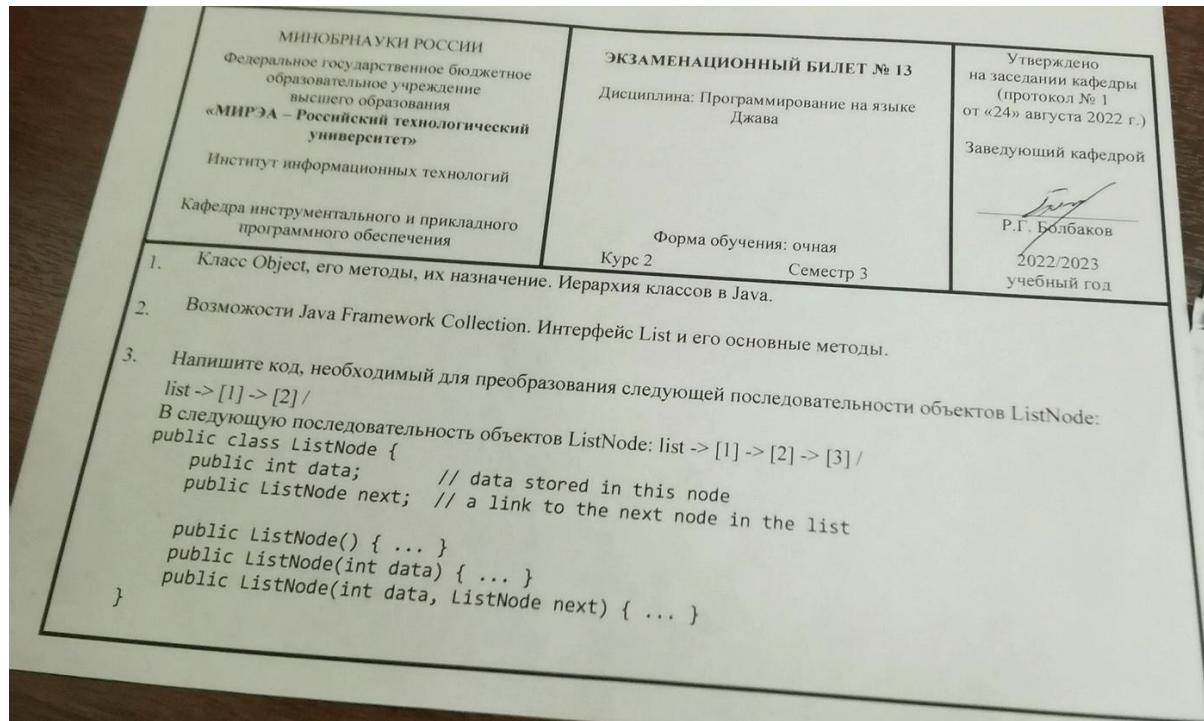
- Не создавайте новые объекты `ListNode` (хотя у вас может быть столько переменных `ListNode`, сколько вам нужно). Не используйте другие структуры данных, такие как массивы, списки, очереди и т. д. Не изменяйте данные любого существующего узла; изменять список только путем изменения ссылок между узлами.

Ваше решение должно выполняться за время $O(N)$, где N - количество элементов в связанном списке.

Предположим, что вы добавляете этот метод в класс `LinkedList` (который использует класс `ListNode`) как показано ниже:

```
public class LinkedList {  
    public class ListNode {  
        public int data;  
        public ListNode next;  
  
        public void firstLast() {  
            if (front == null || front.next == null) return;  
  
            ListNode iter = front.next;  
            while (iter.next != null) {  
                iter = iter.next;  
            }  
            iter.next = front;  
            front = front.next;  
            iter.next.next = null;  
        }  
    }  
}
```

БИЛЕТ № 13



1. Класс Object, его методы, их назначение. Иерархия классов в Java.

Класс *Object* является корнем иерархии классов. Каждый класс имеет объект в качестве суперкласса. Все объекты, включая массивы, реализуют методы этого класса.

В Java нет концепции не типизированных переменных. Все переменные в Java должны иметь определенный тип, который определяет значения, которые может содержать переменная, и операции, которые можно с ней выполнять.

Однако в Java есть тип *Object*, который можно использовать для хранения ссылки на любой объект. Тип Object является корнем иерархии классов в Java, и все классы являются подклассами Object.

Пример объявления:

```
Object obj;
```

Можно присвоить любой объект этой переменной, например:

```
obj = "Hello, world!"; // assign a String object to obj  
obj = 123; // assign an Integer object to obj
```

Необходимо обратить внимание, что когда присваивается примитивное значение (например, int или double) переменной Object, это значение автоматически помещается в объект соответствующего класса-оболочки (например, Integer или Double). Это известно как Автобоксинг.

Чтобы получить доступ к значению, хранящемуся в переменной типа Object, необходимо привести его к соответствующему типу, а затем использовать соответствующие методы для извлечения значения. Например:

```
String str = (String) obj; // cast obj to a String and assign it to str
int x = ((Integer) obj).intValue(); // cast obj to an Integer and extract
the int value
```

Важно быть осторожным при работе с нетипизированными переменными, так как они могут привести к ошибкам во время выполнения, если попытаться использовать их способом, несовместимым с их фактическим типом.

Как правило, рекомендуется использовать типизированные переменные, когда это возможно, так как они могут помочь предотвратить ошибки и сделать ваш код более понятным. Однако бывают ситуации, когда нетипизированные переменные могут быть полезны, например, когда вам нужно записать значение типа, неизвестного во время компиляции.

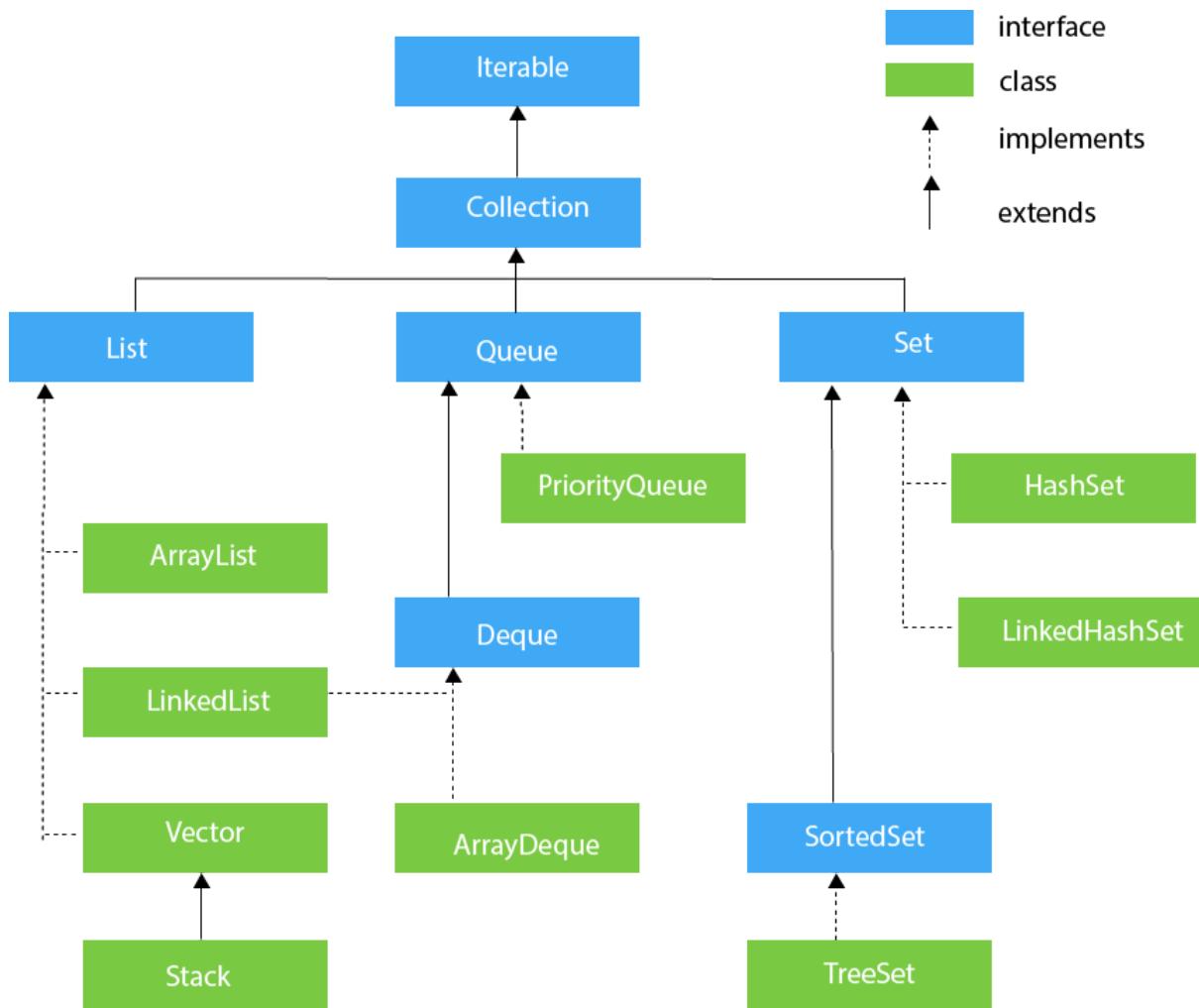
- Иерархическое наследование. При этом класс может наследовать суперкласс, который в свою очередь может наследовать другой суперкласс и т.д. Синтаксис:

```
class SubClass extends SuperClass {
    // тело класса
}
class SubSubClass extends SubClass {
    // тело класса
}
```

2. Возможности Java Framework Collection. Интерфейс List и его основные методы.

Java Framework Collection - это набор классов, которые предоставляют различные алгоритмы для работы с коллекциями. Классы Java Framework Collection расположены в пакете java.util.

Структура коллекций в Java Framework Collection представлена на рисунке ниже.



Интерфейс `List` расширяет интерфейс `Collection` и определяет методы для работы с элементами коллекции по индексу. Интерфейс `List` реализуют классы `ArrayList`, `LinkedList`, `Vector` и `Stack`. Данный интерфейс используется для хранения элементов в виде списка. `Stack` – это класс, который наследуется от класса `Vector` и реализует интерфейс `List`. `Stack` используется для хранения элементов в виде стека.

- `void add(int index, E obj)`: добавляет в список по индексу `index` объект `obj`
- `boolean addAll(int index, Collection<? extends E> col)`: добавляет в список по индексу `index` все элементы коллекции `col`. Если в результате добавления список был изменен, то возвращается `true`, иначе возвращается `false`
- `E get(int index)`: возвращает объект из списка по индексу `index`
- `int indexOf(Object obj)`: возвращает индекс первого вхождения объекта `obj` в список. Если объект не найден, то возвращается `-1`
- `int lastIndexOf(Object obj)`: возвращает индекс последнего вхождения объекта `obj` в список. Если объект не найден, то возвращается `-1`
- `ListIterator<E> listIterator ()`: возвращает объект `ListIterator` для обхода элементов списка
- `static <E> List<E> of(элементы)`: создает из набора элементов объект `List`
- `E remove(int index)`: удаляет объект из списка по индексу `index`, возвращая при этом удаленный объект

- `E set(int index, E obj)`: присваивает значение объекта `obj` элементу, который находится по индексу `index`
- `void sort(Comparator<? super E> comp)`: сортирует список с помощью компаратора `comp`
- `List<E> subList(int start, int end)`: получает набор элементов, которые находятся в списке между индексами `start` и `end`

3. Напишите код, необходимый для преобразования следующей последовательности объектов ListNode:

`list -> [1] -> [2] /`

В следующую последовательность объектов `ListNode`: `list -> [1] -> [2] -> [3] /`

```
public class Program {
    public static void main(String[] args) {
        List list = new List();
        list.add(1);
        list.add(2);
        System.out.println(list);
        list.add(3);
        System.out.println(list);
    }
}
class List {
    private ListNode head, last;
    public void add(int value) {
        ListNode newNode = new ListNode(value);
        if (head == null) {
            head = newNode;
            last = head;
        }
        last.next = newNode;
        last = newNode;
    }
    @Override
    public String toString() {
        ListNode tmp;
        StringBuilder result = new StringBuilder();
        result.append(head.data);
        tmp = head.next;
        while (tmp != null) {
            result.append(" ").append(tmp.data);
            tmp = tmp.next;
        }
    }
}
```

```
        return result.toString();
    }
}

class ListNode {
    public int data;
    public ListNode next;
    public ListNode() {
        data = 0;
        next = null;
    }
    public ListNode(int data) {
        this.data = data;
        next = null;
    }
    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}
```

БИЛЕТ № 15

1. ООП в Java. Понятие объекта. Что представляет собой Java приложение с точки зрения ООП. Основные характеристики объектов в Java.

Объект - экземпляр класса

Приложение на Java с точки зрения ООП это набор классов, методов и полей.

Любой объект может обладать двумя основными характеристиками:

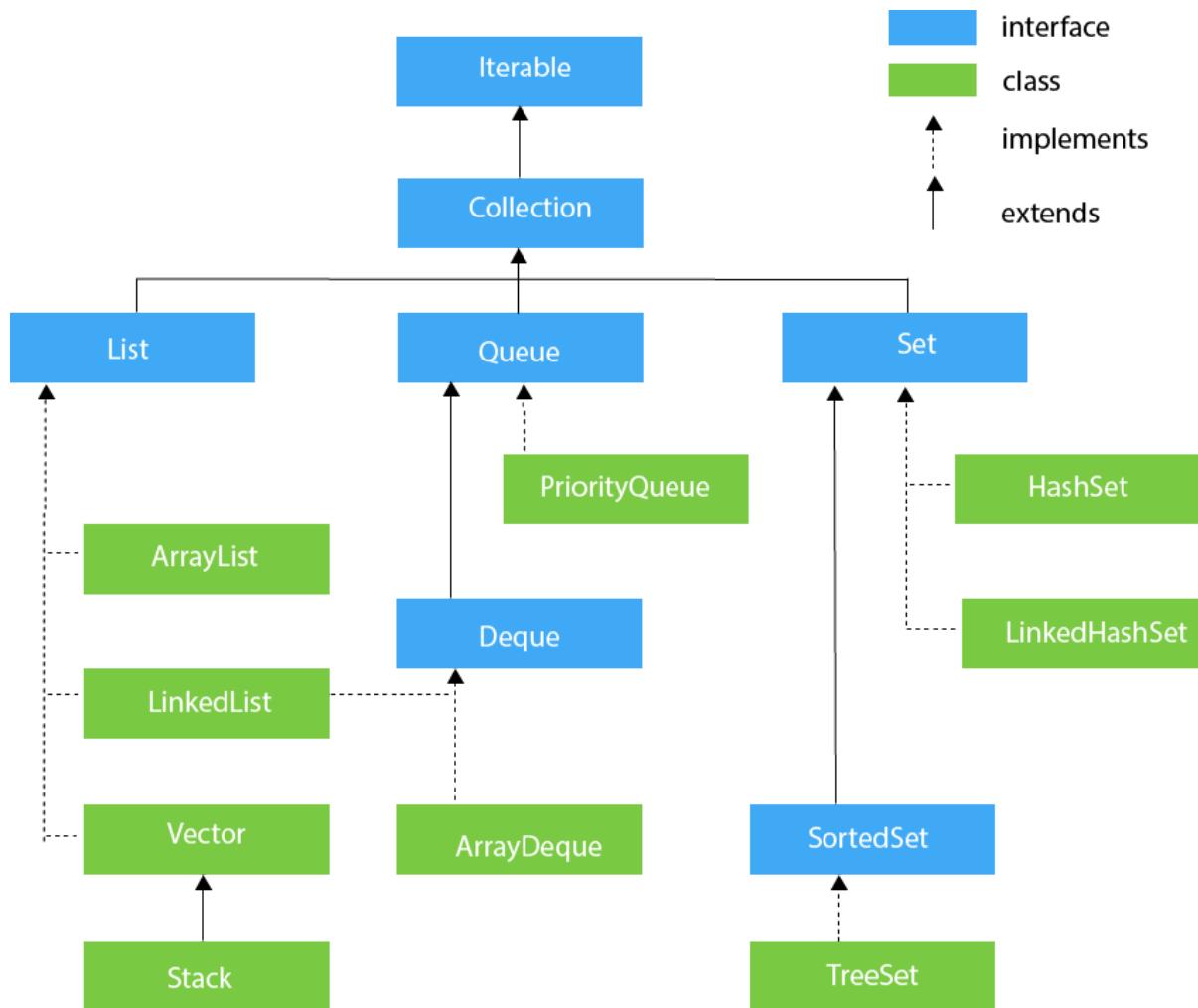
- состояние - некоторые данные, которые хранит объект
 - поведение - действия, которые может совершать объект.

Для хранения состояния объекта в классе применяются поля или переменные класса

2. Возможности Java Framework Collection. Контейнер HashMap и его основные методы.

Java Framework Collection - это набор классов, которые предоставляют различные алгоритмы для работы с коллекциями. Классы Java Framework Collection расположены в пакете `java.util`.

Структура коллекций в Java Framework Collection представлена на рисунке ниже.



Коллекция `HashMap` — это класс в пакете `java.util`, который реализует интерфейс `Map`. Коллекция `HashMap` хранит элементы в виде пар ключ-значение. Ключи должны быть уникальными, а значения могут повторяться. Коллекция `HashMap` не гарантирует порядок хранения элементов. Коллекция `HashMap` не является синхронизированной, поэтому не является потокобезопасной. Коллекция `HashMap` позволяет хранить `null` в качестве ключа и значения.

Пример создания коллекции `HashMap`:

```

import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<>();
    }
}

```

Основные методы работы с HashMap:

- `put(K key, V value)` — добавляет элемент в коллекцию. Возвращает `null`, если элемент добавлен, иначе возвращает предыдущее значение элемента с таким ключом.
- `get(Object key)` — возвращает значение элемента с указанным ключом.
- `remove(Object key)` — удаляет элемент с указанным ключом. Возвращает `null`, если элемент не найден.
- `containsKey(Object key)` — проверяет, содержит ли коллекция элемент с указанным ключом.
- `containsValue(Object value)` — проверяет, содержит ли коллекция элемент с указанным значением.
- `size()` — возвращает количество элементов в коллекции.
- `isEmpty()` — проверяет, пуста ли коллекция.
- `clear()` — удаляет все элементы из коллекции.

3. Напишите метод `splitStack`, который принимает стек целых чисел в качестве параметра и разбивает его на отрицательные и неотрицательные значения. Числа в стеке должны быть переставлены так, чтобы все отрицательные значения появлялись в нижней части стека, а все неотрицательные — в верхней части. Другими словами, если после вызова этого метода вам нужно будет вытолкнуть числа из стека, вы сначала получите все неотрицательные числа, а затем получите все отрицательные числа. Неважно, в каком порядке появляются числа, если все отрицательные находятся в стеке всегда ниже, чем все неотрицательные числа. Вы можете использовать одну очередь в качестве вспомогательного хранения.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class Main
{
    public static Stack splitStack(Stack numbers)
    {
        Stack buff=new Stack();
        int buff_num;
        while(!numbers.empty())
        {
            if ((int) numbers.peek() > 0)
            {
                if(buff.empty())
                {
                    buff.push(numbers.pop());
                }
                else if((int)buff.peek()>0)
                {
                    buff.push(numbers.pop());
                }
            }
        }
    }
}
```

```

        }
        else if((int) buff.peek()<0)
        {
            buff_num=(int)numbers.pop();
            while((int)buff.peek()<0)
            {
                numbers.push(buff.pop());
                if(buff.empty())
                {
                    break;
                }
            }
            buff.push(buff_num);
        }
    }
    else if((int) numbers.peek() < 0)
    {
        if(buff.empty())
        {
            buff.push(numbers.pop());
        }
        else if((int)buff.peek()>0)
        {
            buff.push(numbers.pop());
        }
        else if((int) buff.peek()<0)
        {
            buff.push(numbers.pop());
        }
    }
}
while(!buff.empty())
{
    numbers.push(buff.pop());
}

return numbers;
}
public static void main(String[] args)
{
    Stack test=new Stack();
    test.push(15);
    test.push(26);
    test.push(-13);
    test.push(14);
    test.push(-10);
    test.push(50);
    test.push(15);
    test.push(20);
    test.push(-3);
    System.out.println(splitStack(test));
}

```

}
 }

БИЛЕТ № 19

1. Класс Scanner. Ввод и вывод данных. Стандартные потоки ввода и вывода

Класс Scanner — это класс Java, который используется для чтения входных данных из различных источников, таких как файлы, сетевые сокеты и стандартный поток ввода (stdin). Он является частью пакета `java.util` и может использоваться для чтения входных данных различных типов, таких как целые числа, двойные числа, строки и логические значения.

Чтобы использовать класс Scanner для чтения ввода из стандартного потока ввода, вы можете создать новый объект Scanner и передать стандартный поток ввода (`System.in`) в качестве аргумента конструктору. Например:

```
Scanner scanner = new Scanner(System.in);
```

После того, как вы создали объект Scanner, вы можно использовать его различные методы для чтения ввода из стандартного потока ввода. Например, чтобы прочитать целое число из стандартного потока ввода, вы можно использовать метод nextInt():

```
int i = scanner.nextInt();
```

Чтобы прочитать двойное число, вы можно использовать метод `nextDouble()`:

```
double d = scanner.nextDouble();
```

Чтобы прочитать строку, вы можно использовать метод `nextLine()`:

```
String s = scanner.nextLine();
```

Также можно передать объект `File`, URL-адрес или строку, содержащую входные данные, конструктору сканера для чтения входных данных из этих источников. Например, чтобы прочитать ввод из файла, можно сделать что-то вроде этого:

```
File file = new File("input.txt");
Scanner scanner = new Scanner(file);
```

Или, чтобы прочитать ввод с URL-адреса, можно сделать это:

```
URL url = new URL("http://www.example.com");
Scanner scanner = new Scanner(url.openStream());
```

Также можно использовать метод `hasNext()`, чтобы проверить, доступны ли дополнительные данные для чтения, и метод `close()`, чтобы закрыть `Scanner` и освободить все используемые им ресурсы.

Класс `System` — это встроенный класс Java, предоставляющий доступ к различным функциям и переменным системного уровня. Одной из функций, предоставляемых классом `System`, является возможность работы со стандартным потоком вывода, представляющим собой поток, который используется для вывода данных на консоль или в окно терминала.

Для вывода данных в стандартный поток вывода в Java можно использовать статический метод `println()` объекта `System.out`. Этот метод принимает один аргумент, то есть данные, которые вы хотите вывести, и выводит данные на консоль, за которыми следует символ новой строки.

Например, чтобы вывести строку «Hello, world!» в консоль можно использовать следующий код:

```
System.out.println("Hello, world!");
```

Также можно использовать статический метод `print()` объекта `System.out` для вывода данных в стандартный поток вывода без символа новой строки. Например:

```
System.out.print("Hello, ");
System.out.print("world!");
```

В дополнение к методам `println()` и `print()` объект `System.out` предоставляет несколько других методов для вывода данных в стандартный поток вывода, например, `printf()` для форматированного вывода и `write()` для записи необработанных байтов.

Метод `printf()` — это метод класса `PrintStream` (который является суперклассом объекта `System.out`), который используется для вывода форматированной строки в стандартный поток вывода. Он принимает строку формата и список аргументов и заменяет заполнители в строке формата соответствующими аргументами.

Вот пример того, как можно использовать метод `printf()` для вывода форматированной строки на консоль:

Метод `write()`, с другой стороны, является методом класса `OutputStream` (который является суперклассом класса `PrintStream`), который используется для записи для записи одиночных байтов или массива байтов в стандартный поток вывода. Он принимает массив байтов в качестве аргумента и записывает байты в поток.

```
byte[] data = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x2c, 0x20, 0x77, 0x6f, 0x72, 0x6c,
0x64, 0x21};
System.out.write(data);
```

Вот пример того, как вы можете использовать метод `write()` для вывода массива байтов на консоль:

```
int x = 10;
double y = 3.14;
System.out.printf("x is %d and y is %f\n", x, y);
```

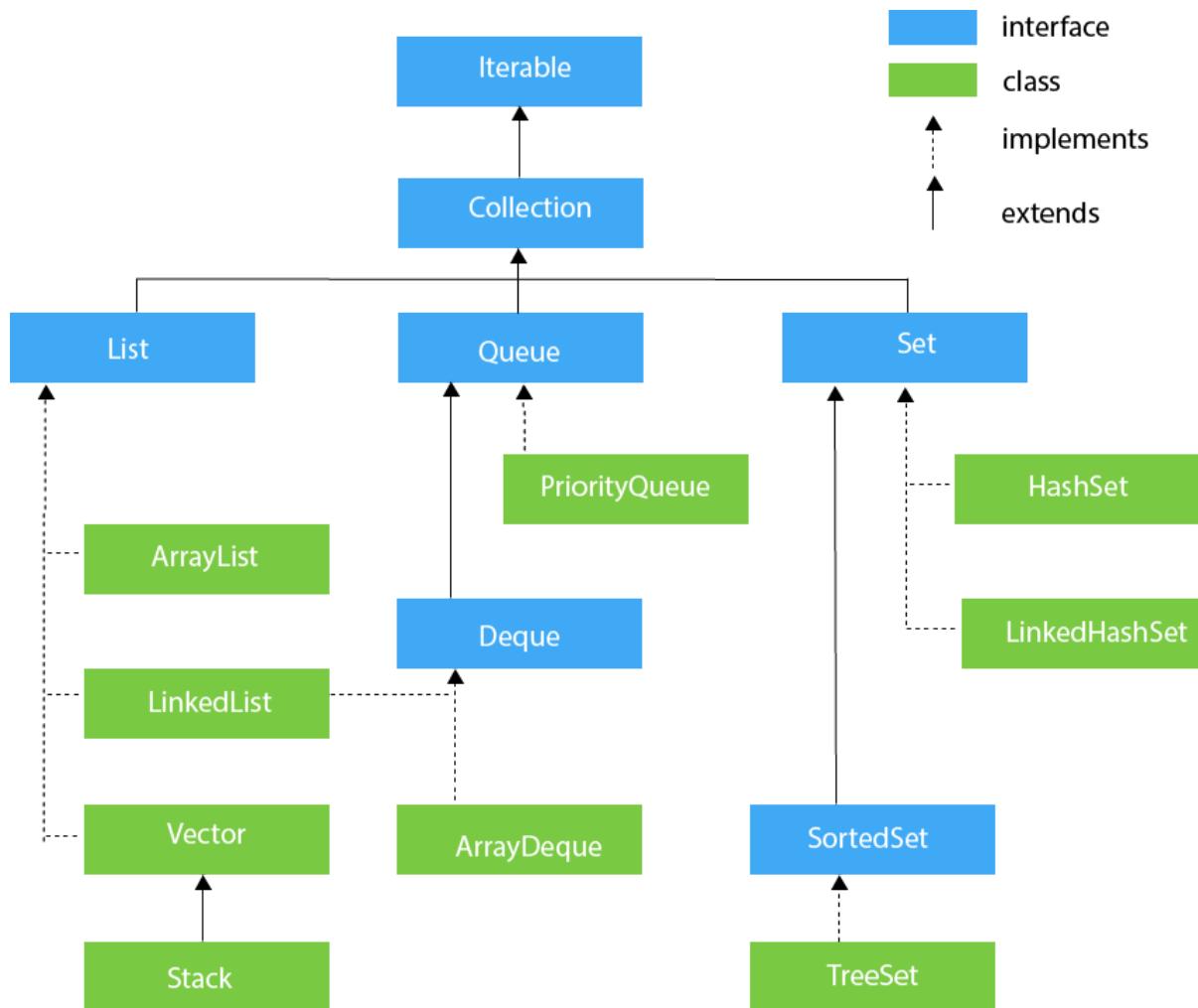
Этот код выведет строку «Hello, world!» к консоли. Массив байтов содержит коды ASCII для символов в строке, а метод `write()` записывает байты в стандартный поток вывода.

Метод `write()` не добавляет автоматически символ новой строки, поэтому вам нужно использовать отдельный вызов `println()`, если вы хотите включить новую строку в свой вывод.

2. Возможности Java Framework Collection. Интерфейс Iterator и Iterable.

Java Framework Collection - это набор классов, которые предоставляют различные алгоритмы для работы с коллекциями. Классы Java Framework Collection расположены в пакете `java.util`.

Структура коллекций в Java Framework Collection представлена на рисунке ниже.



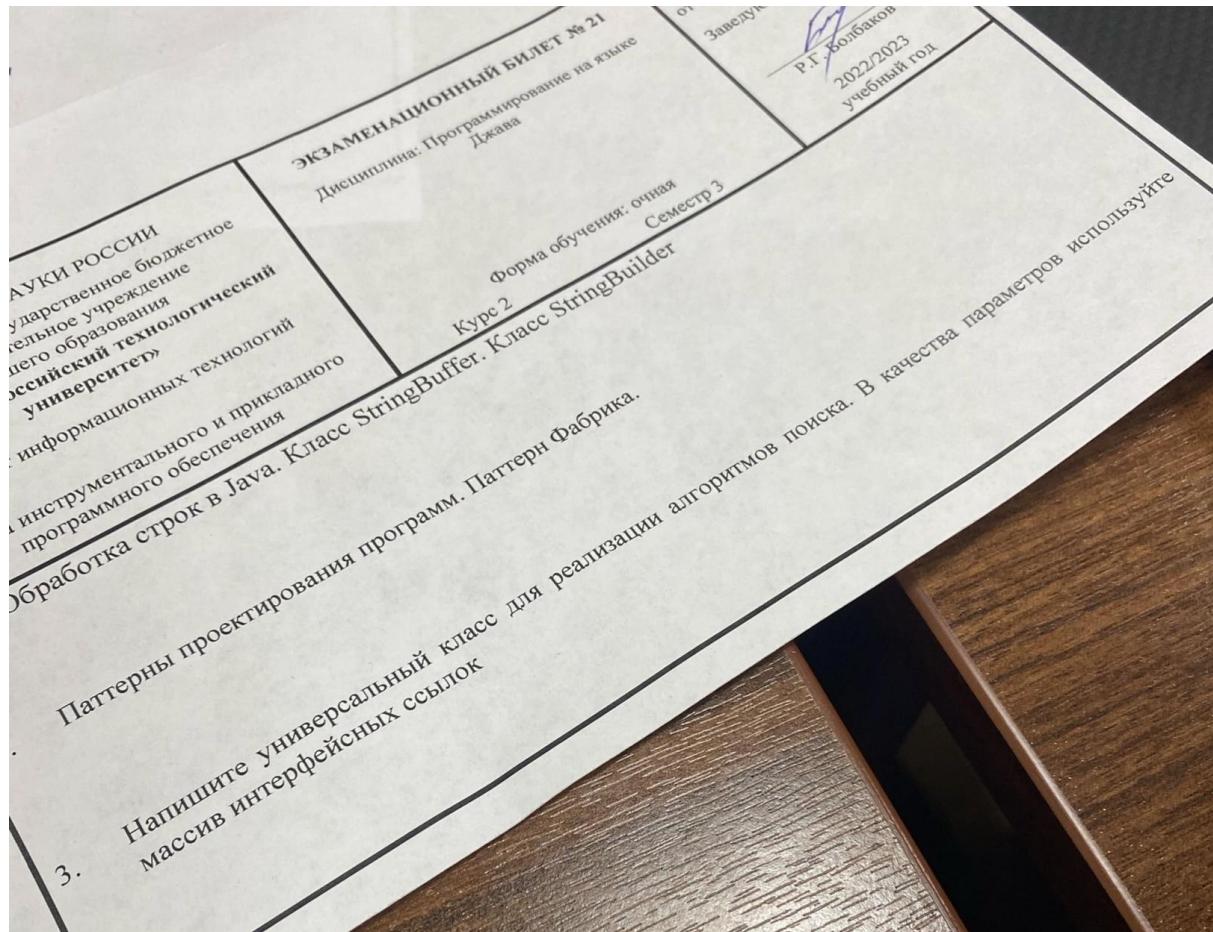
3. Обработка строк в Java. Класс StringBuffer. Класс StringBuilder

```

import java.util.*;

public class Main {
    static public void main(String[] args) {
        ArrayDeque<Integer> priorityQueue = new ArrayDeque<>();
        priorityQueue.addAll(Arrays.asList(1, 8, 7, 2, 9, 18, 12, 0));
        Stack<Integer> stack = new Stack<>();
        int c = priorityQueue.size();
        for (int i = 0; i < c; i++) {
            if (i % 2 == 0) priorityQueue.add(priorityQueue.poll());
            else stack.add(priorityQueue.poll());
        }
        for (int i = 0; i < c; i++) {
            if (i % 2 == 0) priorityQueue.add(priorityQueue.poll());
            else priorityQueue.add(stack.pop());
        }
        System.out.println(priorityQueue);
    }
}
  
```

БИЛЕТ № 21



1. Обработка строк в Java. Класс StringBuffer. Класс StringBuilder

При почти любой работе со строками в джаве создается новый строковый объект, в который копируются содержимое старых строк. Чтобы процесс конкатенации строк был более эффективным можно использовать классы `StringBuilder` или `StringBuffer`.

`StringBuilder` и `StringBuffer` похожи на `String`, но они изменяемы, что означает, что вы можете изменять их содержимое, не создавая новый объект. Это делает их более эффективными для объединения строк, поскольку они позволяют избежать расходов на создание новых объектов для каждого объединения.

В Java при создании строкового объекта с помощью класса `StringBuffer` вы можете использовать операцию конкатенации строк, а также методы класса `StringBuffer` для изменения строки.

Класс `StringBuffer` — это изменяемая (модифицируемая) версия класса `String`, которая позволяет изменять строку без создания нового объекта для каждой операции.

Чтобы создать строковый объект с помощью класса StringBuffer, используйте следующий синтаксис:

```
StringBuffer имя_объекта = new StringBuffer(строка);
```

Чтобы объединить две строки с помощью класса StringBuffer, используйте метод append().

```
StringBuffer strBuffer = new StringBuffer("Hello");
strBuffer.append(" World");
System.out.println(strBuffer); // Hello World
```

2. Паттерны проектирования программ. Паттерн Фабрика

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

В чем суть?

Фабричный метод (Factory method) также известный как Виртуальный конструктор (Virtual Constructor) - пораждающий шаблон проектирования, определяющий общий интерфейс создания объектов в родительском классе и позволяющий изменять создаваемые объекты в дочерних классах.

Шаблон позволяет классу делегировать создание объектов подклассам. Используется, когда:

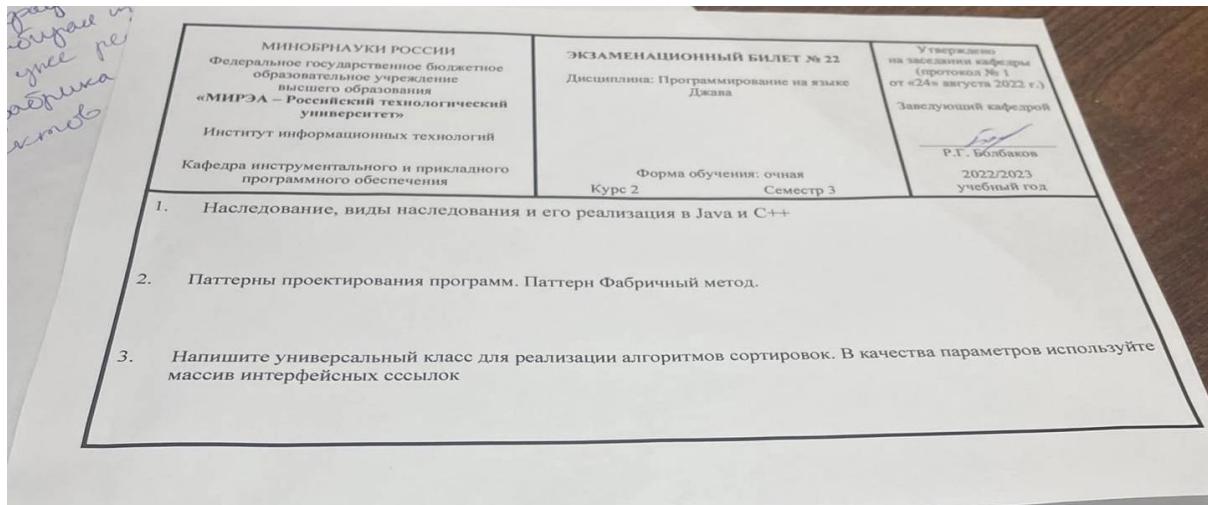
1. Классу заранее неизвестно, объекты каких подклассов ему нужно создать.
2. Обязанности делегируются подклассу, а знания о том, какой подкласс принимает эти обязанности, локализованы.
3. Создаваемые объекты родительского класса специализируются подклассами.

3. Напишите универсальный класс для реализации алгоритмов поиска. В качестве параметров используйте массив интерфейсных ссылок

```
public class Test {  
    static public void main(String[] args) {  
        Searcher<Animal> Find = new Searcher<>();  
        Animal d1 = new Dog(false, "d1");  
        Animal d2 = new Dog(false, "d2");  
        Animal d3 = new Dog(false, "d3");  
        Animal c1 = new Cat(false, "c1");  
        Animal c2 = new Cat(true, "c2");  
        Animal c3 = new Cat(false, "c3");  
        Animal[] Animals = {d1, d2, d3, c1, c2, c3};  
        Animal cutie = Find.Cuties(Animals);  
        System.out.println(cutie.toString());  
    }  
}  
interface Animal {  
    boolean isCutie();  
}  
class Cat implements Animal {  
    public boolean isCutie;
```

```
public String name;
public Cat(boolean status, String name) {
    isCutie = status;
    this.name = name;
}
public boolean isCutie() {
    return isCutie;
}
@Override
public String toString() {
    return "Cat{" +
        "isCutie=" + isCutie +
        ", name='" + name + '\'' +
        '}';
}
}
class Dog implements Animal {
    public boolean isCutie;
    public String name;
    public Dog(boolean status, String name) {
        isCutie = status;
        this.name = name;
    }
    public boolean isCutie() {
        return isCutie;
    }
    @Override
    public String toString() {
        return "Dog{" +
            "isCutie=" + isCutie +
            ", name='" + name + '\'' +
            '}';
    }
}
class Searcher<T extends Animal> {
    public T Cuties(T[] IArray) {
        for (T obj : IArray) {
            if (obj.isCutie())
                return obj;
        }
        return null;
    }
}
```

БИЛЕТ № 22



1. Наследование, виды наследования и его реализация в Java и C++

45. Наследование в Джава. Вид наследования и синтаксис Ключевое слово extends

Наследование в Джава - это механизм, который позволяет создавать новые классы на основе уже существующих. Новый класс называется подклассом (дочерним классом, производным классом), а существующий класс называется суперклассом (родительским классом, базовым классом).

Ключевое слово `extends` используется для наследования класса. Оно указывает, что класс, который объявляется после ключевого слова `extends`, является суперклассом для класса, который объявляется до ключевого слова `extends`.

Существует несколько видов наследования:

- Одиночное наследование (single inheritance). При этом класс может наследовать только один суперкласс. Синтаксис:

```
class SubClass extends SuperClass {  
    // тело класса  
}
```

- Множественное наследование. В Java множественное наследование не поддерживается. Однако стоит упомянуть, что класс может реализовывать несколько интерфейсов.
- Иерархическое наследование. При этом класс может наследовать суперкласс, который в свою очередь может наследовать другой суперкласс и т.д. Синтаксис:

```
class SubClass extends SuperClass {  
    // тело класса  
}  
class SubSubClass extends SubClass {  
    // тело класса  
}
```

Синтаксис наследования в C++:

class имя_произв_класса : модификатор_доступа имя_род_класса
дочернее private поле: public и protected становится private
дочернее protected поле: public становится protected
дочернее public поле: public становится public

2. Паттерны проектирования программ. Паттерн Фабричный метод.

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция,

представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

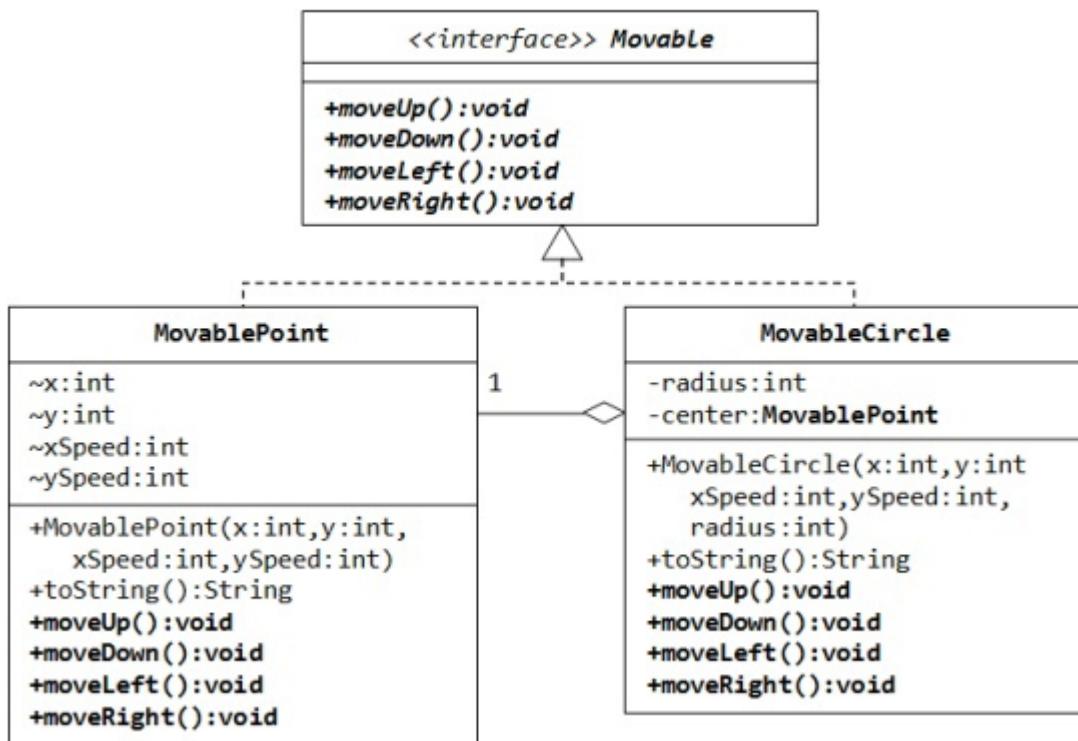
В чем суть?

Фабричный метод (Factory method) также известный как Виртуальный конструктор (Virtual Constructor) - пораждающий шаблон проектирования, определяющий общий интерфейс создания объектов в родительском классе и позволяющий изменять создаваемые объекты в дочерних классах.

Шаблон позволяет классу делегировать создание объектов подклассам. Используется, когда:

1. Классу заранее неизвестно, объекты каких подклассов ему нужно создать.
2. Обязанности делегируются подклассу, а знания о том, какой подкласс принимает эти обязанности, локализованы.
3. Создаваемые объекты родительского класса специализируются подклассами.

Пример на UML.



3. Напишите универсальный класс для реализации алгоритмов сортировок. В качестве параметров используйте массив интерфейсных ссылок

БИЛЕТ 24

МИНОБРНАУКИ РОССИИ Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА – Российский технологический университет» Институт информационных технологий Кафедра инструментального и прикладного программного обеспечения	ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 24 Дисциплина: Программирование на языке Джава Форма обучения: очная Курс 2 Семестр 3	Утверждено на заседании кафедры (протокол № 1 от «30» августа 2022 г.) Заведующий кафедрой Р.Г. Болбаков 2022/2023 учебный год
1. Расширение классов. Переопределение методов. Скрытие полей данных. 2. Паттерны проектирования программ. Паттерн Observer и модель MVC 3. Разработайте класс иерархии классов Комплексное число, Рациональное число. Используйте паттерн Фабрика		

1 Расширение классов. Переопределение методов. Скрытие полей данных.

45. Наследование в Джава. Вид наследования и синтаксис Ключевое слово extends

Наследование в Джава - это механизм, который позволяет создавать новые классы на основе уже существующих. Новый класс называется подклассом (дочерним классом, производным классом), а существующий класс называется суперклассом (родительским классом, базовым классом).

Ключевое слово `extends` используется для наследования класса. Оно указывает, что класс, который объявляется после ключевого слова `extends`, является суперклассом для класса, который объявляется до ключевого слова `extends`.

Существует несколько видов наследования:

- Одиночное наследование (single inheritance). При этом класс может наследовать только один суперкласс. Синтаксис:

```
class SubClass extends SuperClass {  
    // тело класса  
}
```

- Множественное наследование. В Java множественное наследование не поддерживается. Однако стоит упомянуть, что класс может реализовывать несколько интерфейсов.
- Иерархическое наследование. При этом класс может наследовать суперкласс, который в свою очередь может наследовать другой суперкласс и т.д. Синтаксис:

```
class SubClass extends SuperClass {  
    // тело класса  
}  
class SubSubClass extends SubClass {  
    // тело класса  
}
```

Переопределение метода - это возможность создавать методы с одинаковым именем и параметрами, но с разным телом. При этом методы должны быть объявлены в одном классе, но в разных подклассах.

Пример:

```
public class calc {  
    public int add(int a, int b) {  
        return a + b;
```

```
    }
}

public class calc2 extends calc {
    @override
    public int add(int a, int b) {
        return a + b + 1;
    }
}
```

Строка `@override` не является обязательной, но ее использование позволяет избежать ошибок при переопределении методов.

Стоит отметить, что переопределение методов не является полной заменой перегрузки методов. Перегрузка методов используется для создания различных вариантов одного и того же метода, а переопределение методов используется для изменения поведения метода в подклассе.

2 Паттерны проектирования программ. Паттерн Observer и модель MVC

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

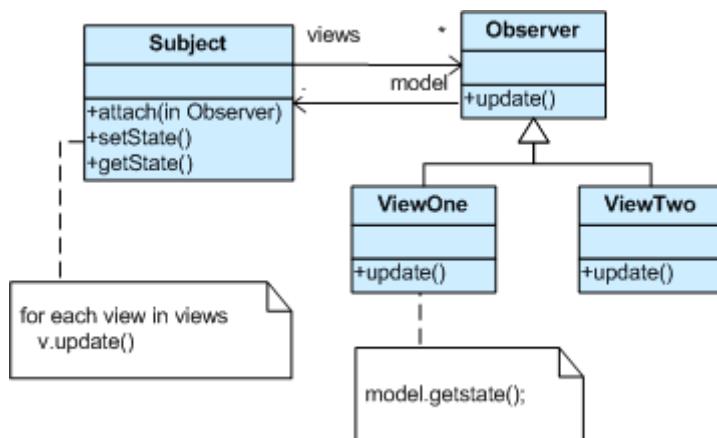
Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

- Паттерн Observer определяет зависимость "один-ко-многим" между объектами так, что при изменении состояния одного объекта все зависящие от него объекты уведомляются и обновляются автоматически.

- Паттерн Observer инкапсулирует главный (независимый) компонент в абстракцию Subject и изменяемые (зависимые) компоненты в иерархию Observer.
- Паттерн Observer определяет часть "View" в модели Model-View-Controller (MVC).

Паттерн Observer определяет объект Subject, хранящий данные (модель), а всю функциональность "представлений" делегирует слабосвязанным отдельным объектам Observer. При создании наблюдатели Observer регистрируются у объекта Subject. Когда объект Subject изменяется, он извещает об этом всех зарегистрированных наблюдателей. После этого каждый обозреватель запрашивает у объекта Subject ту часть состояния, которая необходима для отображения данных.

Такая схема позволяет динамически настраивать количество и "типы" представлений объектов.



З Разработайте класс иерархию классов Комплексное число, Рациональное число. Используйте паттерн Фабрика

БИЛЕТ № 23

1. Расширение классов. Порядок создания экземпляра дочернего класса.

2. Понятие сериализации и ее использование в ООП программах.

Сериализация - это процесс преобразования объекта в последовательность байтов, которая может быть записана в файл или передана по сети.

В Java сериализация объектов реализуется с помощью интерфейса `Serializable`. Этот интерфейс не содержит никаких методов, он просто помечает классы, которые могут быть сериализованы.

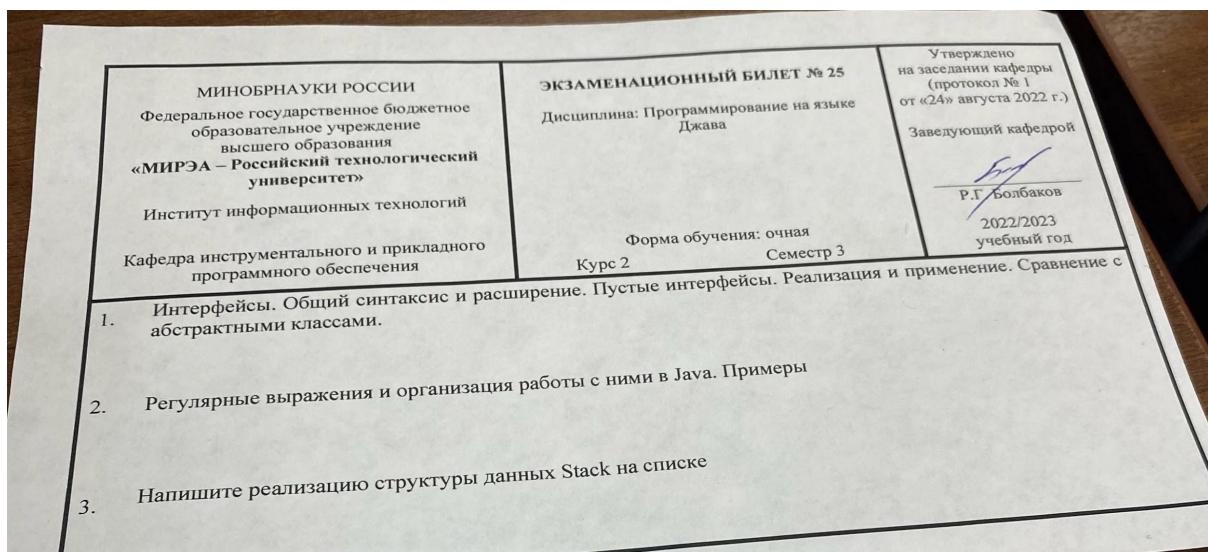
В Java можно сериализовать только объекты, которые реализуют интерфейс `Serializable`. Если класс не реализует этот интерфейс, то при попытке

сериализовать объект этого класса будет выброшено исключение NotSerializableException.

Все поля класса должны быть сериализуемыми. Если поле не сериализуемое, то при сериализации объекта будет выброшено исключение NotSerializableException.

3. Разработайте класс иерархию классов Геометрическая фигура, Прямоугольник, Круг. Используйте паттерн Фабрика

БИЛЕТ № 25



1. Интерфейсы. Общий синтаксис и расширение. Пустые интерфейсы. Реализация и применение. Сравнение с абстрактными классами.

Интерфейс - интерфейс Java представляет собой набор абстрактных методов и констант (необязательно). Абстрактный метод представляет собой заголовок метода без тела. Абстрактный метод объявляется с помощью модификатора `abstract`, потому что все методы в интерфейсе являются как правило, абстрактными, то у них нет реализации. Интерфейс используется для создания набора методов, что класс будет реализовать.

```
public class Main {  
    static public void main(String[] args) {  
        sddd sddd = new sddd();  
        sddd.print();  
        System.out.println(gggg.dssd);  
    }  
}  
interface as {  
    int a =333;  
}  
interface fff {  
    int a = 22;  
    void print();  
}  
abstract class gggg {  
    public int dssd = 2;  
    abstract void pgpg();  
}  
class sddd extends gggg implements as, fff {  
    @Override
```

```
public void print() {  
    dssd = 3;  
    fff.a = 2;  
}  
@Override  
void pgpg() {  
}  
}
```

2. Регулярные выражения и организация работы с ними в Java. Примеры

Регулярные выражения – это система обработки текста, основанная на специальной системе записи образцов для поиска. Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Язык программирования Java также поддерживает регулярные выражения для работы со строками. Основными классами для работы с регулярные выражения являются класс `java.util.regex.Pattern` и класс `java.util.regex.Matcher`.

Класс Pattern — это класс стандартной библиотеки Java. Он представляет собой компилированный шаблон регулярного выражения. Этот класс используется для создания объекта `Matcher`, который выполняет сопоставление с шаблоном регулярного выражения.

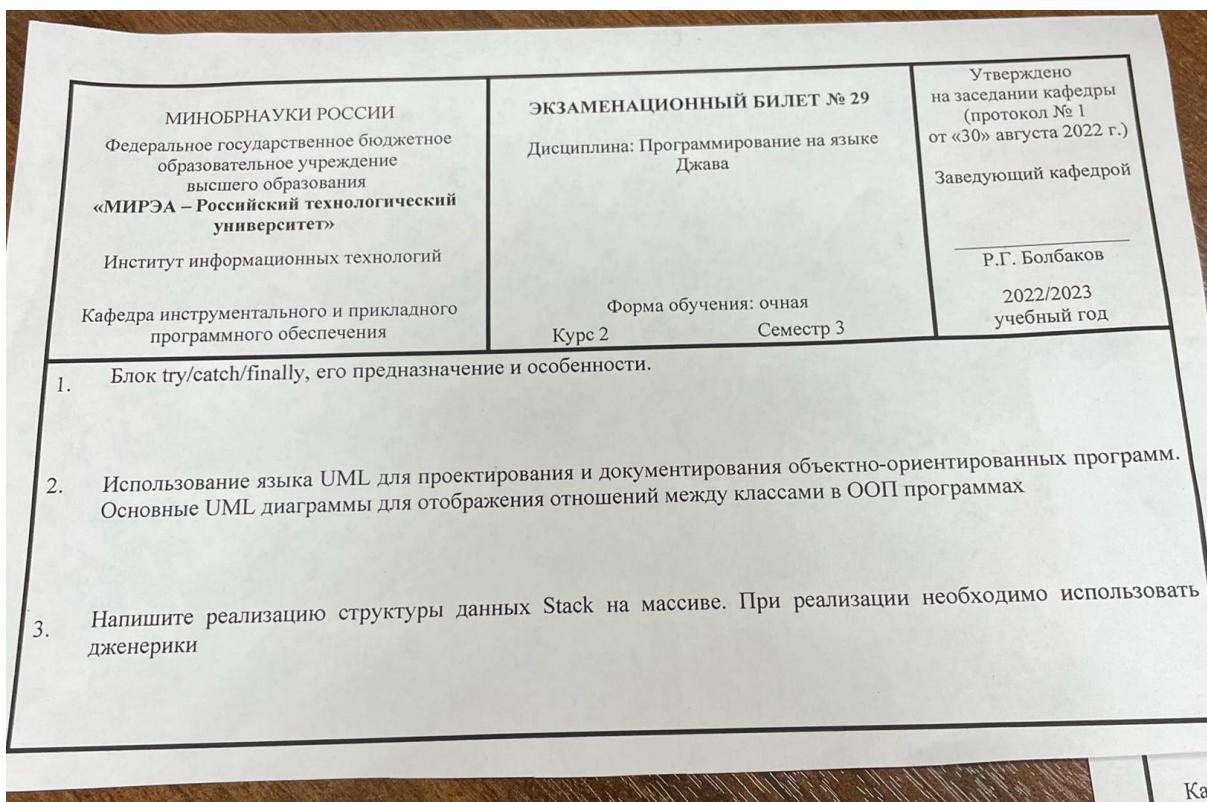
Пример № 1:

https://github.com/CyberGeo335/Java/blob/main/Task_14/Main.java

Пример № 2:

3. Напишите реализацию структуры данных Stack на списке

БИЛЕТ № 29



1. Блок try/catch/finally, его предназначение и особенности.

В Java исключительная ситуация представляет собой событие, которое происходит во время выполнения программы и прерывает ее нормальное выполнение. Исключительная ситуация может быть вызвана ошибкой в программе, неправильным вводом данных или другими причинами.

В Java все исключения делятся на два основных типа: Error и Exception. Error используется для ошибок, которые не могут быть обработаны программой. Exception используется для ошибок, которые могут быть обработаны программой.

Чтобы обработать исключительную ситуацию, вы можете использовать оператор try-catch. Оператор try-catch позволяет вам определить блок кода, который может вызвать исключительную ситуацию, и блок кода, который будет выполнен, если исключительная ситуация произойдет.

Оператор try-catch имеет следующий синтаксис:

```
try {
    // code that may throw an exception
} catch (ExceptionType1 ex1) {
    // code to handle ExceptionType1
} catch (ExceptionType2 ex2) {
    // code to handle ExceptionType2
} catch (ExceptionType3 ex3) {
    // code to handle ExceptionType3
} finally {
    // code to be executed regardless of whether an exception occurs
}
```

2. Использование языка UML для проектирования и документирования объектно-ориентированных программ. Основные UML диаграммы для отображения отношений между классами в ООП программах

Unified Modeling Language (UML) — унифицированный язык моделирования. UML описывает объект в едином заданном синтаксисе

Графически можно представить класс в виде **UML¹** диаграммы как прямоугольник в виде как трех секций, в котором присутствует секция наименования класса, секция инкапсуляции данных и методов (функций или операций) класса. Пример общего представления диаграммы класса представлен на рисунке 1.1.

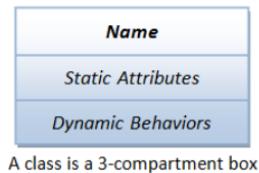


Рисунок 1.1 - Диаграмма класса. Общее представление.

Рассмотрим подробнее диаграмму класса. Имя (или сущность): определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области).

Методы (или поведение, функции, работа с данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“контейнере” или “боксе”), представленном на рисунке в виде прямоугольника.

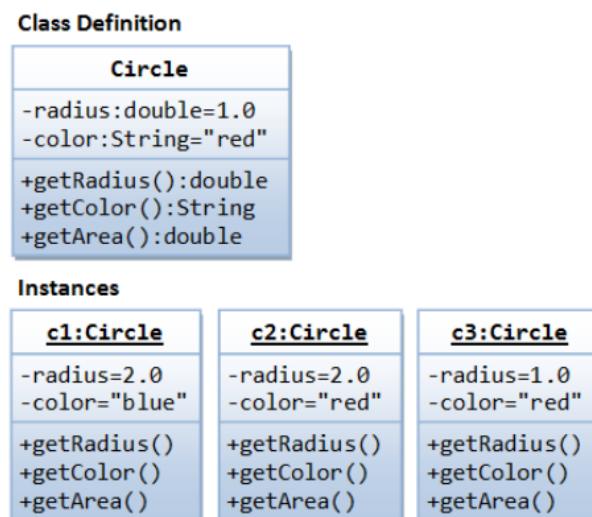
На рисунке 1.2 показано несколько примеров классов. У каждого из них есть имя, переменные класса и методы.

Name (Identifier)	Student	Circle
Variables (Static attributes)	name grade	radius color
Methods (Dynamic behaviors)	getName() printGrade()	getRadius() getArea()
SoccerPlayer	Car	
name number xLocation yLocation	plateNumber xLocation yLocation speed	
run() jump() kickBall()	move() park() accelerate()	

Examples of classes

7. Теперь соберем все вместе: Пример ООП

На рисунке ниже представлена диаграмма класса и его трех экземпляров.



- - private
- + public

3. Напишите реализацию структуры данных Stack на массиве. При реализации необходимо использовать

БИЛЕТ № ??

Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА – Российский технологический университет» Институт информационных технологий Кафедра инструментального и прикладного программного обеспечения	Дисциплина: Программирование на языке Джава Форма обучения: очная Курс 2 Семестр 3	(протокол № 1 от «24» августа 2022 г.) Заведующий кафедрой Р.Г. Болбаков 2022/2023 учебный год
<p>1. Инициализация полей класса и локальных переменных (отличие), инициализатор и статический инициализатор (когда вызывается).</p> <p>2. Возможности Java Framework Collection. Интерфейс Map и его основные методы.</p> <p>3. Напишите метод isPalindrome, который принимает в качестве параметра очередь целых чисел и возвращает true, если числа в очереди представляют палиндром (и false в противном случае). Последовательность чисел считается палиндромом, если она совпадает в обратном порядке. Например, предположим, что очередь с именем q хранит эти значения:</p> <pre>front [3, 8, 17, 9, 17, 8, 3] back</pre> <p>Тогда вызов isPalindrome(q); должен вернуть true, потому что эта последовательность одинакова в обратном порядке. Если в очереди хранятся эти значения:</p> <pre>front [3, 8, 17, 9, 4, 17, 8, 3] back</pre> <p>Вызов isPalindrome вместо этого вернул бы false, потому что эта последовательность не совпадает в обратном порядке (9 и 4 в середине не совпадают). Пустую очередь следует считать палиндромом. Вы не можете делать какие-либо предположения относительно количества элементов в очереди, и ваш метод должен восстановить очередь, после использования очереди, чтобы после вызова он сохранял ту же последовательность значений, что и раньше. Вы можете использовать стек в качестве вспомогательного хранения.</p>		

№3

```
import java.util.ArrayDeque;
import java.util.Objects;
import java.util.Queue;

public class Palindrome {
    public static void main(String[] args) {
        Queue<Integer> q = new ArrayDeque<>();
        q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);
        q.add(3);
        q.add(2);
        q.add(1);
        while (!q.isEmpty())
            System.out.println(q.poll());
        System.out.println(isPalindrome(q));
    }

    private static boolean isPalindrome(Queue<Integer> q) {
        int left = 0;
        int right = q.size() - 1;
        while (left < right) {
            if (!Objects.equals(q.peek(), q.peekLast()))
                return false;
            q.poll();
            q.pollLast();
            left++;
            right--;
        }
        return true;
    }
}
```

```
}

public static boolean isPalindrome(Queue<Integer> q) {
    boolean flag = false;
    if (q.isEmpty())
        return true;
    Queue<Integer> buffer = new ArrayDeque<>();
    Stack<Integer> stack = new Stack<>(100);
    while (!q.isEmpty()) {
        int a = q.poll();
        buffer.add(a);
        stack.push(a);
    }
    while (!buffer.isEmpty()) {
        if (!(Objects.equals(buffer.poll(), stack.pop())))
            return false;
    }
}
return true;
}
```

