ЛЕКЦИЯ ПО ОСНОВАМ ЯЗЫКА ПРОГРАММИРОВАНИЯ JAVA SE

Составил : Шляпкин Андрей Владимирович, к.т.н., доцент
Кафедра «Прикладная математика и информатика»
teachmi_tgys@mail.ru

Тематический план

1. Введение в язык программирования Java
2. Основные элементы языка Java
3. Базовые конструкции языка Java
4. Объектно-ориентированное программирование
5. Абстрактные классы и интерфейсы
6. Работа со строками
7. Коллекции

Характеристика языка программирования Java

Java - это одновременно язык программирования и платформа.

Java представляет собой высокоуровневый объектно-ориентированный язык программирования. При компиляции, которая выполняется один раз во время сборки приложения, код на Java преобразуется в код на промежуточном языке (байт-код). В свою очередь, байт-код анализируется и выполняется (интерпретируется) виртуальной машиной Java (JVM), которая играет роль транслятора между языком Java и аппаратным обеспечением с операционной системой. Все реализации Java должны эмулировать JVM, чтобы создаваее приложения могли выполняться на любой системе, включающей виртуальную машину Java.

Платформа включает в себя JVM и интерфейс прикладного программирования на Java (API), представляющий собой обширный набор готовых программных компонентов (классов), облегчающих разработку и развертывание апплетов и приложений

Характеристика языка программирования Java

Существуют три редакции платфор Java.

- Java SE (Java Platform, Standard Edition). Используя Java SE, вы можете создавать и развертывать Java-приложения для настольных компьютеров и серверов, а также разрабатывать встроенное программное обеспечение и програм для систем реального времени.
- Java EE (Java Platform, Enterprise Edition). Эта корпоративная версия платфор помогает разработчикам создавать и развертывать переносить, надежные, масштабируемые и безопасные серверные приложения на Java.
- Java ME (Java Platform, Micro Edition). Java ME предоставляет среду для выполнения приложений, созданных для широкого круга мобильных и встроенных систем, например мобильных телефонов, карманных компьютеров, телевизионных приставок и принтеров. Приложения, базирующиеся на спецификациях Java ME, могут быть запущены на множестве устройств и при этом способны эффективно задействовать их системные возможности.

Типы данных (примитивы) в Java

byte (число, 1 байт)
short (число, 2 байта)
int (число, 4 байта)
long (число, 8 байтов)
float (число с плавающей точкой, 4 байта)
double (число с плавающей точкой, 8 байтов)
char (символ, 2 байта)
boolean (true (истина) или false (ложь), 1 байт)

Пример объявления целого числа

1 способ:

```
int myNumber;
myNumber = 5;
```

2 способ:

```
int myNumber = 5;
```

Базовые арифметические операции

Символ	Характеристика
+	Оператор сложения. Бинарный оператор. Результатом выполнения a+b будет сумма значений переменных.
-	Оператор вычитания. Бинарный оператор. Результатом выполнения a-b будет разность значений переменных.
*	Оператор умножения. Бинарный оператор. Результатом выполнения команды a*b будет произведение значений переменных.
/	Деление. Бинарный оператор. Результатом команды a/b является частное от деления значений переменных a и b. Для int выполняется деление нацело.
%	Остаток. Бинарный оператор. Результатом команды a%b является остаток от целочисленного деления значений переменных a и b.
++	Инкремент. Унарный оператор. Команда a++ (или ++a) является тем же самым, что и команда a=a+1
--	Декремент. Унарный оператор. Команда a-- (или --a) является эквивалентом команды a=a-1

Логические операции

&&	Сокращенное логическое «И». Бинарный оператор. Особенность оператора, по сравнению с оператором &, состоит в том, что если значение первого операнда равно false, то значение второго операнда не проверяется
||	Сокращенное логическое «ИЛИ». Бинарный оператор. Особенность оператора, по сравнению с оператором |, состоит в том, что если значение первого операнда равно true, то значение второго операнда не проверяется
!	Логическое отрицание. Унарный оператор. Результатом команды !A является true, если значение операнда A равно false. Если значение операнда A равно true, результатом команды !A является значение false

Побитовые логические операции

& (and)	Побитовое «И». Бинарный оператор. Логическая операция И применяется к каждой паре битов операндов. Результатом является 1, если каждый из двух сравниваемых битов равен 1. В противном случае результат равен 0
| (or)	Побитовое «ИЛИ». Бинарный оператор. Логическая операция ИЛИ применяется к каждой паре битов операндов. Результатом является 1, если хотя бы один из двух сравниваемых битов равен 1. В противном случае результат равен 0
^ (бинарный XOR)	Побитовое «ИСКЛЮЧАЮЩЕЕ ИЛИ». Бинарный оператор. Логическая операция ИСКЛЮЧАЮЩЕЕ ИЛИ применяется к каждой паре битов операндов. Результатом является 1, если один и только один из двух сравниваемых битов равен 1. В противном случае результат равен 0
~	Побитовое отрицание. Унарный оператор. Выполняется инверсия двоичного кода: 0 меняется на 1, а 1 меняется на 0
<<	Сдвиг влево. Бинарный оператор. Результатом является число, получаемое сдвигом влево в позиционном представлении первого операнда (слева от оператора) на количество битов, определяемых вторым операндом (справа от оператора). Исходное значение первого операнда при этом не меняется. Младшие биты заполняются нулями, а старшие теряются
>>	Сдвиг вправо. Бинарный оператор. Результатом является число, получаемое сдвигом вправо в позиционном представлении первого операнда (слева от оператора) на количество битов, определяемых вторым операндом (справа от оператора). Исходное значение первого операнда при этом не меняется. Младшие биты теряются, а старшие заполняются дублированием знакового бита
>>>	Беззнаковый сдвиг вправо. Бинарный оператор. Результатом является число, получаемое сдвигом вправо в позиционном представлении первого операнда (слева от оператора) на количество битов, определяемых вторым операндом (справа от оператора). Исходное значение первого операнда при этом не меняется. Младшие биты теряются, а старшие заполняются нулями

Общая структура программы языка Java

```
имя пакета;

Import   ……..;

public class Main {                  // комментарии

// комментарии
 public static void main(String[] args) {        //точка входа программы

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

Public - метод, класс общедоступен
Static- означает, что выполнение этого метода без создания экземпляра класса main
void -означает, что метод не возвращает никакого значения
main -точка входа Java программы

Модификаторы доступа

В Java существуют следующие модификаторы доступа:

public -члены класс доступны всем

Private- члены класса доступны только внутри класса

default (package-private) (модификатор, по-умолчанию):-члены класса видны внутри пакета (если класс будет так объявлен он будет доступен только внутри пакета);

Protected -члены класса доступны внутри пакета и в наследниках

ВВОД/ВЫВОД ДАННЫХ НА ЯЗЫКЕ JAVA

```
package massiv.summa;
import java.util.Scanner;
import java.util.Random;

public class Summ {

public static void main(String[] args) {
                // TODO Auto-generated method stub
                Scanner in = new Scanner (System.in);
                Random random = new Random();
                System.out.println("Enter array length: ");
         int size = in.nextInt();// Читаем с клавиатуры размер массива и записываем в size
          System.out.println("Enter array length1: ");
        int size1 = in.nextInt();
        int array[] [] = new int[size][size1];// Создаём массив int размером в size

 System.out.println("Insert array elements:");
           /*Пройдёмся по всему массиву, заполняя его*/
                  for (int i = 0; i < size; i++) {
                for (int j = 0; j < size1; j++) {
                array[i][j] = random.nextInt(7);
                System.out.println ("Inserted array elements:"+ array[i][j]);

                                }
```

Листинг программы «побитовые логические операции»

```
public static void main(String args[]) {
    int a = 60;
    int b = 13;
    int c = 0;

    c = a & b;
    System.out.println("a & b = " + c );

    c = a | b;
    System.out.println("a | b = " + c );

    c = a ^ b;
    System.out.println("a ^ b = " + c );

    c = ~a;
    System.out.println("~a = " + c );

    c = a << 2;
    System.out.println("a << 2 = " + c );

    c = a >> 2;
    System.out.println("a >> 2  = " + c );

    c = a >>> 2;
    System.out.println("a >>> 2 = " + c );
```

Условный оператор IF… ELSE

Синтаксис условного оператора:

IF (условие) {

……

}

else {

……

}

Пример программы

```
public class NewClass{
    public static void main(String args[]){
        int x = 30;

        if( x == 10 ){
            System.out.print("Значение X = 10");
        }else if( x == 20 ){
                System.out.print("Значение X = 20");
            }else if( x == 30 ){
                    System.out.print("Значение X = 30");
                }else{
                    System.out.print("Это оператор else");
                }
    }
}
```

Команда выбора SWITCH

Синтаксис команды выбора SWITCH:

```
switch(ВыражениеДляСравнения) {
   case Совпадение1:
          команда;
                break;
        case Совпадение2:
          команда;
                break;
        case Совпадение3:
          команда;
                break;
        default:
          оператор;
          break;
}
```

Пример программы

```
String monthString;
switch (month) {
   case 1:  monthString = "Январь";
         break;
   case 2:  monthString = "Февраль";
         break;
default: monthString = "Не знаем такого";
         break;
}
System.out.println(monthString);
```

Оператор цикла WHILE

Синтаксис оператора цикла WHILE:

WHILE (условие) {

……

}

Пример программы

```
public class NewClass{
    public static void main(String args[]) {
        int x = 10;

        while( x < 15 ) {
            System.out.print("Значение x: " + x );
            x++;
            System.out.print("\n");
        }
    }
}
```

Оператор цикла FOR

Синтаксис оператора цикла FOR:

For (переменная = начальное значение; переменная<конечного значения; приращение) {

…….....

}

Пример программы

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

        for(int x = 10; x < 15; x = x+1) {
            System.out.print("Значение x: " + x )
            System.out.print("\n");
        }
    }
}
```

Объявление массивов на языке Java

1 способ:

```
int[] cats;  //Объявляем переменную одномерного массива
cats = new int[10];   //Определение массива
```

2 способ:

```
int[] cats = new int[10]; //объявление переменной и определение массива
```

Пример программы

```
int[] mice = {4, 8, 10, 12, 16};
int result = 0;

for(int i = 0; i < 5; i++){
        result = result + mice[i];
}
result = result / 5;
System.out.println ("Среднее арифметическое: " + result);
```

Объявление массивов на языке Java

1 способ:

```
int[] [] twoD;  //Объявляем переменную двумерного массива
twoD = new int[10] [5];   //Определение массива
```

2 способ:

```
int[] [] twoD = new int[10] [5]; //объявление переменной и определение массива
```

Пример программы

```
int[][] twoD = new int[3][4]; // объявили двухмерный массив
int i, j, k = 0;

for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++) {
                twoD[i][j] = k;
                k++;
        }
```

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Основными концепциями здесь являются понятия объект и класс. Программа на языке Java состоит из классов, а каждый класс, в свою очередь, представляет объект реального света. Например, в качестве объекта возьмем робота, в этом случае, в классе Robot, представляющем этот объект, будут описаны его характеристики и поведение, то есть атрибуты и методы.

4 главных принципа, на которых строится объектно-ориентированное программирование:

- **Инкапсуляция** -это механизм, объединяющий атрибуты и методы (которые составляют объект) и охраняющий их от внешнего вмешательства. Инкапсуляция — защитная оболочка, позволяющая обращаться к атрибутам и методам класса только внутри этого класса или при помощи специально спроектированного интерфейса.
- **Наследование** -помогает избежать дублирования кода в случае, если нам нужно создать объект на основе уже существующего. В этом случае говорится, что новый объект (дочерний) унаследовал свойства уже существующего (родительского). Если атрибуты или поведение существующего объекта нужно частично изменить, то их можно просто переопределить.
- **Полиморфизм** -это способность предоставлять один и тот же интерфейс для различных базовых форм (типов данных). Это означает, что классы, имеющие различную функциональность, совместно используют один и тот же интерфейс и могут быть динамически вызваны передачей параметров по ссылке.
- **Абстракция** означает разработку классов исходя из их интерфейсов и функциональности, не принимая во внимание реализацию деталей. Абстрактный класс представляет собой интерфейсы без включения фактической реализации. Он отличает реализацию объекта от его поведения. Абстракция упрощает код, скрывая несущественные детали.

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Описание класса начинается с ключевого слова class, после которого указывается идентификатор — имя класса. Затем в фигурных скобках перечисляются атрибуты и методы класса. Атрибуты в языке Java называются полями (в дальнейшем мы будем использовать это наименование). Поля и методы называются членами класса.
Поля описываются как обычные переменные.

Конструктор — это особенный метод класса, который вызывается автоматически в момент создания объектов этого класса. Имя конструктора совпадает с именем класса.
Например, в классе Cat может быть конструктор с двумя параметрами, который при создании новой кошки позволяет сразу задать ее кличку и возраст.

```
public Cat (String n, int a) {
name = n;
age = a;
}
```

Конструктор вызывается после ключевого слова new в момент создания объекта. Теперь, когда у нас есть такой конструктор, мы можем им воспользоваться:

```
Cat cat1 = new Cat(«Мурка", 2);
```

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

```
package cat1;

public class Cat {
private int weight1;
private String name1;
private String color1;
public void eat() {
System.out.print ("Eating...\n");
}
public void sleep() {
System.out.print ("Sleeping zz-zz-z-z.....\n");
        }
public String speak(String world) {
String phrase = world + "... mauu.....\n";
return phrase;
}
}
```

```
package cat1;

public class HelloWorld {

        public static void main(String[] args) {
                // TODO Auto-generated
method stub
Cat ourcat = new Cat();
ourcat.eat();
ourcat.sleep();
String say = ourcat.speak("Play with me");
System.out.println(say);
        }

}
```

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Для того, чтобы один класс был потомком другого, необходимо при его объявлении после имени класса указать ключевое слово extends и название суперкласса.
Например:

```
class Britan extends Cat {
// дополнительные поля и методы
...
}
```

Перегрузка

В одном классе можно создать несколько методов с одним и тем же именем, различающихся по своим параметрам. Этот прием называется перегрузкой методов. Когда один из этих методов будет вызван, произойдет сопоставление переданных ему параметров (их количества и типов) с параметрами всех методов класса с таким именем. Если подходящий метод будет найден, выполнится именно он.

ИНТЕРФЕЙСЫ И АБСТРАКТНЫЕ КЛАССЫ

Ключевое слово interface используется для создания полностью абстрактных классов. Создатель интерфейса определяет имена методов, списки аргументов и типы возвращаемых значений, но не тела методов.
Чтобы создать интерфейс, используйте ключевое слово interface вместо class.
Класс, который собирается использовать определённый интерфейс, использует ключевое слово **implements.**
Интерфейсов у класса может быть несколько, тогда они перечисляются за ключевым словом implements и разделяются запятыми.
Интерфейсы могут вкладываться в классы и в другие интерфейсы.
Если класс содержит интерфейс, но не полностью реализует определённые им методы, он должен быть объявлен как abstract.
объявление интерфейсной переменной:
List<String> catNames;
интерфейсная переменная должна ссылаться на объект класса, реализующего данный интерфейс.
List<String> catNames = new ArrayList<>();
Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы помечаются ключевым словом **abstract.**
Фигура - это абстрактное понятие и мы не можем создать универсальный метод для вычисления площади. Поэтому мы создаём другой класс Треугольник и пишем код, вычисляющий площадь треугольника

Модификатор STATIC в Java

Модификатор static в Java напрямую связан с классом, если поле статично, значит оно принадлежит классу, если метод статичный, аналогично — он принадлежит классу. Исходя из этого, можно обращаться к статическому методу или полю используя имя класса.

```
class Vehicle{
    public static void kmToMiles(int km){
        System.out.println("Внутри родительского класса/статического метода");
    }}

class Car extends Vehicle{
    public static void kmToMiles(int km){
        System.out.println("Внутри дочернего класса/статического метода ");
    }}

public class Solution{
    public static void main(String args[]){
        new Car().kmToMiles(10);
        new Vehicle().kmToMiles(10);
        Car.kmToMiles(10);
        Vehicle.kmToMiles(10);
    }
}
```

Определение final в Java

Определение final применяется в нескольких случаях. Во-первых, оно позволяет запретить изменение значения переменной. Т.е. если вы создадите переменную с определением final, то значение этой переменной уже нельзя будет поменять.

```
public class TestClass
{
   public static void main(String[] arg) {
      final Integer f = new Integer(100);
      f = 200;
   }
}
```

Строка f = 200; не скомпилируется, т.к. переменная f уже проинициализирована. Когда определение final применяется к свойству класса, то его можно инициализировать либо в месте описания, либо в конструкторе

Геттеры и сеттеры в Java

Геттер - это метод возвращающий значение некоего свойства класса, а сеттер соответственно то что устанавливает свойство класса

```
public class MyClass {
  private String name; //свойство

  public String getName() { //геттер
    return this.name;
  }

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

Коллекции в Java

1. List - Представляет собой неупорядоченную коллекцию, в которой допустимы дублирующие значения. Иногда их называют последовательностями (sequence). Элементы такой коллекции пронумерованы, начиная от нуля, к ним можно обратиться по индексу.

2. Set - описывает неупорядоченную коллекцию, не содержащую повторяющихся элементов. Это соответствует математическому понятию множества (set).

3. Queue - очередь

Графический интерфейс на Java Swing

В Java есть 2 основных пакета для создания графических интерфейсов (Graphics User Interface). Это Abstract Windows Toolkit (AWT) и Swing.

Для группировки компонент интерфейса используются контейнеры (Container). Для создания основного контейнера для приложения чаще всего используется контейнер JFrame (есть еще JWindows и JApplet). Проще всего унаследоваться от JFrame тем самым получить доступ ко множеству методов, например:
setBounds(x, y, w, h) - указывает координаты верхней левой вершины окна, а также его ширину и высоту.
setResizable(bool) - указывает, можно ли изменять размер окна.
setTitle(str) - устанавливает название окна.
setVisible(bool) - собственно отображает окно.
setDefaultCloseOperation(operation) - указывает операцию, которая будет произведена при закрытии окна.
Основные элементы управления:
JLabel - элемент для отображения фиксированного текста;
JTextField - простой edit-box;
JButton - обычная кнопка (button);
JCheckBox - элемент выбора (аналог checkbox);
JRadioButton - радио кнопка

Графический интерфейс на Java Swing

При отображении элементов управления используются специальные менеджеры - LayoutManager. У всех LayoutManager'ов есть методы для добавления у удаления элементов.
FlowLayout - используется для последовательного отображения элементов. Если элемент не помещается в конкретную строку, он отображается в следующей.
GridLayout - отображения элементов в виде таблицы с одинаковыми размерами ячеек.
BorderLayout - используется при отображении не более 5 элементов. Эти элементы располагаются по краям фрейма и в ценрте: North, South, East, West, Center.
BoxLayout - отображает элементы в виде рядка или колонки.
GridBagLayout - позволяет назначать месторасположение и размер каждого виджета. Это самый сложный, но и самый эффективный вид отображения.

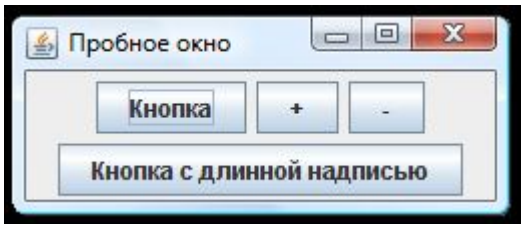
Стоит еще обратить внимание на обработку событий. Для этого используются так называемые Event Listeners.

Графический интерфейс на Java Swing

```
public class SimpleWindow extends JFrame {

SimpleWindow(){
super("Пробное окно");
setDefaultCloseOperation(EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());
panel.add(new JButton("Кнопка"));
panel.add(new JButton("+"));
panel.add(new JButton("-"));
panel.add(new JButton("Кнопка с длинной надписью"));
setContentPane(panel);
setSize(250, 100);
}
```

```
public class Program {
public static void main (String [] args) {
JFrame myWindow = new SimpleWindow();
myWindow.setVisible(true);
}
}
```



Спасибо за внимание!