Лекция 2

Игра: Ребус

Игра: Ребус

Корова мычит + Утка

Игра: Ребус

Ш + (шах, мат, ?) +

Игра: Ребус

Типы данных

1. Данные или любую информацию можно представлять в разном виде. Например: текстовые данные, целые числа, дробные числа и т.д.. Назовем это типом данных.
2. Программирование связано с данными.
3. В Java чтобы работать с информацией, вам нужно указывать ее тип.
4. У каждого типа есть свое предназначение, преимущества, недостатки, применяемые операции и способы хранения.
5. Поэтому очень важно выбирать правильный тип данных.

Типы данных: int

1. Целый тип – int – для хранения целочисленных значений: -1, 3, 234, 0, -123123
2. int number = 23;
3. int distance; // по умолчанию 0
4. distance = 100;
5. int newDistance = distance + 130;

6. int hex = 0xff01; // шестнадцатеричная система
7. int oct = 012356; //восьмеричная система
8. int bin = 0b0101001; //двоичная система Java 7

9. int parsed = Integer.parseInt(“1990”);

Типы данных: byte, short, long

1. byte, short, long – такие же целочисленные типы как int.
2. Но, размеры у них разные!

3. byte – 1 байт – от -128 до 127
4. byte w = 200; // Ошибка!

5. short – 2 байта – от -32768 до 32767;
6. short sss = 560;

7. int – 4 байта – от – 2^{31} до 2^{31} -1

8. long – 8 байтов – от -2^{63} до 2^{63}-1

Типы данных: float, double

1. float, double – числовые типы с плавающей точкой, т.е. для нецелых чисел.

2. double weight = 123.455;

3. float height = 33.6f;
4. height = 34; //можно и целые числа хранить

5. double square = height * 10.5;

6. double parsed = Double.parseDouble(“90.33”);

7. float casted = (float) parsed;
8. int casted = (int) weight;

Типы данных: Операции

1. У каждой математической операции есть результат, который имеет свой тип.
2. Например: Сумма целых чисел всегда целый тип.
3. Разность целых чисел всегда тип с плавающей точкой.
4. 3+5 – результат будет целого типа, поэтому тип результата int:
5. int result = 3+5;

6. double size = 2000/66; // результат не целое число

7. int bush = 90;
8. float sush = 3.66f;

Типы данных: boolean

1. Булевский тип – boolean – имеет всего два возможных значения: True, False
2. В нем можно хранить состояния, активность какого либо флага и тп.

3. boolean isCrazy = True;
4. isCrazy = False;

5. int a = 123;
6. int b, c;
7. …
8. boolean isABigger = a > b; // можно хранить результат булевских операторов
9. boolean isCSmallest = a > c && b > c;

Типы данных: char

1. Тип для представления одного unicode-символа.
2. Любой текстовый символ в Java пишется внутри одинарных кавычек: ‘a’, ‘b’, ‘7’, ‘-’
3. char a = ‘a’;
4. char minus = ‘-’;

5. Размер типа char – 16 битов = 2 байта
6. Поэтому в нем можно хранить любой из 65535 символов юникода
7. Так как на клавиатуре все эти символе не помещаются, используется след выражение для получения символов по номеру:
8. char someChar = ‘\u0f0f’;
9. char anotherChar = (char) 1050;

Типы данных: String

1. String- Строка -Тип для представления кучи символов юникода – т.е. Любого текста.
2. Пишется внутри двойных кавычек: “Это строка”
3. String text = “Hail, Ceaser!”;
4. text = “This is ” + “Sparta! ”; //их можно соединять

5. int x = 123123;
6. String xStr = Integer.toString(x); //конвертация
7. String auto = “x is “+x; //автоконвертация

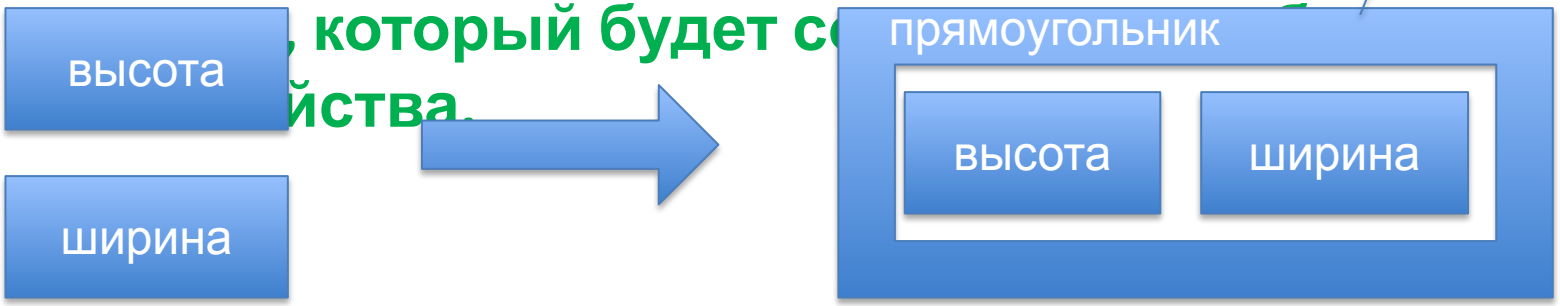
8. String destiny = 123123; // ошибка!

9. String s = “198998998”
10. char selected = s.charAt(4); //получить символ

Классы

1. Класс – это составной тип данных.
2. Их используют чтобы представлять такие сущности, для которых обычного числа или строки будет недостаточно.
3. Класс объединяет некоторое количество данных разного типа в одно целое.
4. Например мы хотим программно представить прямоугольник. У прямоугольника всегда есть два свойства: высота и ширина.
5. Для него мы можем создать новый тип , который будет с йства.

Класс это-контейнер



Классы

1. Также классы могут иметь какие либо функции, которые они могут выполнять, операции – они называются методами класса.
2. Например класс Калькулятор у нас мог вычислять сумма, разность, произведение и частное двух чисел. А класс Принтер мог печатать на экран.
3. Метода могут возвращать результат и менять состояние класса.

Класс Car

```
class Car{
    public  String number;
    public double speed = 0;

    public void go(){
        speed = 20.5;
    }

    public void stop(){
        speed = 0;
    }

    public double getSpeed(){
        return speed;
    }
}
```

Поля представляют какие-то свойства Машины. В данном случае скорость и номер.

Номер машины типа String, потому что номер это текст.

Скорость машины – число с плавающей точкой

Методы – это то что может делать наша Машина

Создание объектов класса

1. Можно создать сколько угодно экземпляров класса


```
Car car1 = new Car();
```

Тип объекта car1. Так как классы тоже типы данных, указываем название класса.

Переменная в котором будет храниться ссылка на наш создаваемый объект.

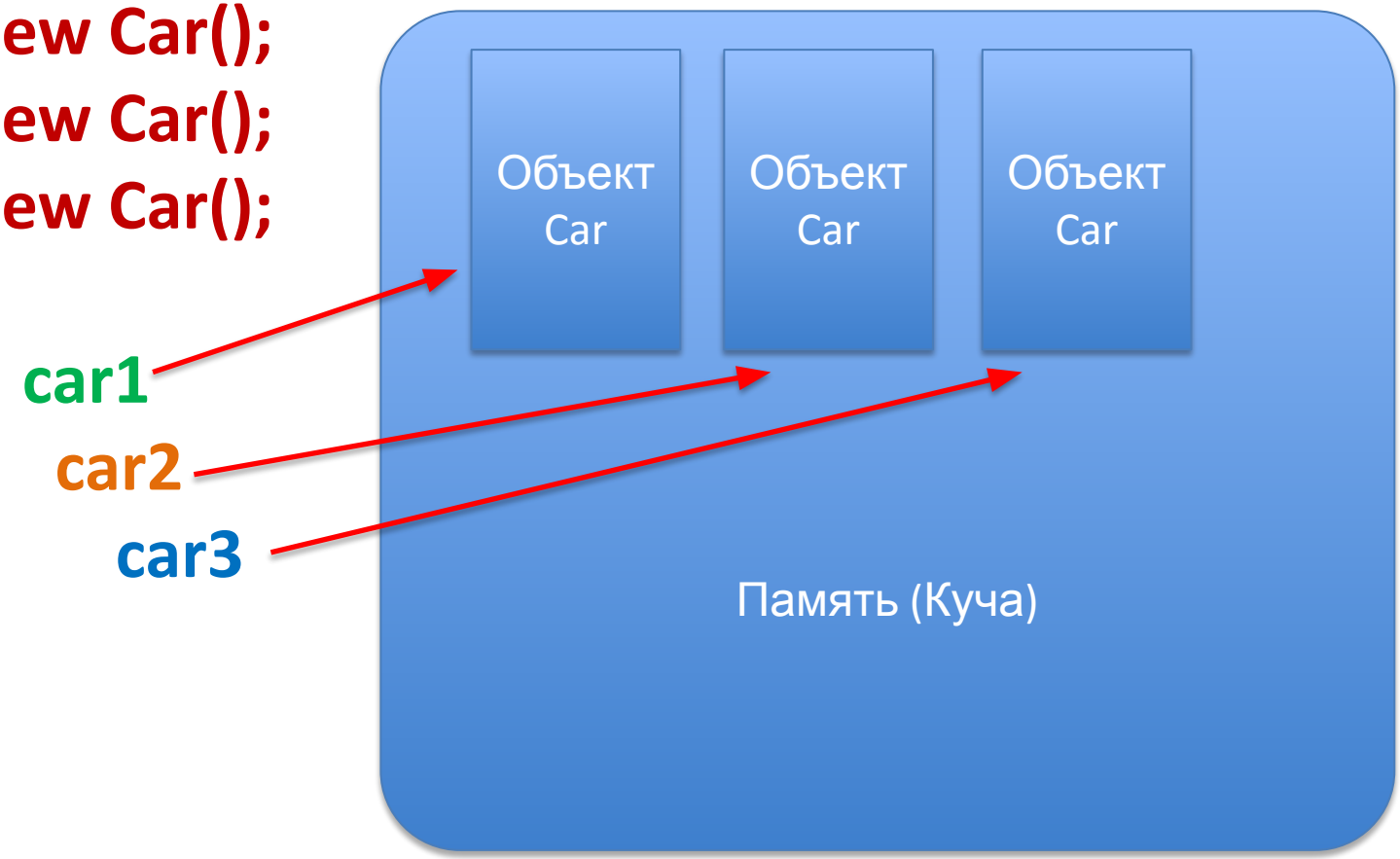
Ключевое слов для создания новых объектов.

Метод-Конструктор создаваемого класса

Объекты в памяти

1. Можно создать сколько угодно экземпляров класса


```
Car car1 = new Car();
Car car2 = new Car();
Car car3 = new Car();
```



Объекты в памяти

1. **Каждый объект имеет свою собственные поля.**

Доступ к свойствам и методам класса

1. После создания объекта, мы можем получить доступ к его public свойствам и методам.
2. Car car1 = new Car();
3. car1.number = “A8909U”; // Изменение значения свойства объекта
4. car1.go(); // вызов метода объекта
5. Вызов метода всегда имеет круглые скобки в конце. При этом внутри скобок могут быть параметры. car1.setNewSpeed(120);
6. Метод может возвращать какое-либо значение определенного типа. Возврат значения производится с помощью ключевого слова return. После возвращения значения метод всегда завершается.

Методы

1. Вся реализация программы происходит в методах.
2. В методе можно выполнять любые действия.
3. Менять значения свойств класса:
4. public void go(){
   this.speed = 20; //this – ссылка на текущий объект
5. Вызывать другие методы // у которого вызывается данный метод
6. public void stop(){
   this.stopEngine();
   this.speed = calcNewSpeed();
   System.out.println("Car deactivated");
7. }

Параметры методов(аргументы)

1. Методы могут принимать разные параметры и выполняться по разному в зависимости от их значения.
2. Принимаемые параметры перечисляются внутри скобок при определении метода. Для каждого параметра указывается тип.

```
public int sum(int num1, int num2){
    int result = num1+num2;
    return result;
}
...

int number3 = sum(100, 300);
```

Условный оператор

1. Условный оператор if позволяет выполнять тот или иной код в зависимости от указанного условия.
2. Условие указывается внутри круглых скобок ()

```
3. if (a>b){
        //Блок кода который выполнится если условие выполняется
4. } else {
        //Блок кода который выполнится если условие ложное
5. }


6. if (x == 100){
        System.out.print(“x равна 100”);
7. } else {
        System.out.print(“x не равна 100”);
```

Условный оператор

1. **Условием для оператора if может быть любое логические выражение:**
   - **True или False**
   - **переменная типа boolean**
   - **результат операций сравнения >, <, ==, >=, <=, !=**
   - **результат логических операций: ||, &&, !**

2. **if(True)** // всегда выполнится
3. **if(False)** //никогда не выполнится
4. **if(aBigger && cSmaller)** // оператор логичекое И
5. **if(aBigger || cSmaller)** // оператор логичекое ИЛИ
6. **if(!active)** // оператор логичекое НЕ

7. При этом все переменные: aBigger, cSmaller,

Видимость переменных

1. Свойства класса тоже переменные, к ним можно получить доступ из любого метода данного класса:

```
class Computer{
    int memory;

    public void method1(){
        memory = 100;
    }

    private int getMethod(){
        memory = 100 + 3;
        return memory;
    }
}
```

Видимость переменных

1. **В теле каждого метода можно создать переменные: Они называются локальными**

```
class Computer{
    int memory;

    public void method1(){
        int count = 20;
        memory = count * 45;
    }

    public void method2(){
        memory = count * 34;  //Ошибка
    }
}
```

count – локальная переменная

Переменная count не определена в данном методе

Видимость переменных

1. **Если имя локальной переменной совпадает с именем свойства класса, используется ключевое слово this для обращения к свойству класса**

```
class Computer{
    int memory;

    public void method1(){
        int memory= 20;
        this.memory = memory * 8;
    }
}
```

локальная переменная memory

Свойство класса

Массивы

1. **Массивы предназначены для хранения множества значений одного типа.**
2. String[] students; // объявление массива
3. students = new String[]; //инициализация
4. students[0] = “Ernazar”; // иниц. первого элемента
5. students[1] = “Malik”; // иниц. второго элемента
6. students[2] = “Syrym”; // иниц. третьего элемента

7. **Нумерация элементов начинается с 0**

8. **Также можно инициализовать массив перечислив все элементы сразу:**
9. String[] students = {“Ernazar”, “Malik”, “Syrym”};

Массивы

1. Длину любого массива можно получить через свойство length.
2. int[] days = {31,29,31,30};
3. System.out.println(days.length); // 4

4. Длину массива нельзя менять
5. Удалять и добавлять новые элементы тоже нельзя

6. При обращении к элементу которого нет, Java выдаст ошибку IndexOutOfBoundsException
7. int tenthDay = days[10]; // Exception

Циклы

1. **Чтобы повторить один и тот же участок кода несколько раз, используются циклы.**
2. **Например: нам нужно напечатать все имена студентов**
3. **Студенты перечислены в массиве**
4. `String[] students = new String[13];`
5. **Без использования циклов получится:**
6. `System.out.println(students[0]);`
7. `System.out.println(students[1]);`
8. `System.out.println(students[2]);`
9. `System.out.println(students[3]);`
10. `System.out.println(students[4]);`
11. `System.out.println(students[5]);`
12. `System.out.println(students[6]);`
13. `….// и т.д.`

Циклы

1. Все элементы массива можно обойти другим способом:

2. for(String student: students){
   System.out.println(student);
3. }

4. Внутри круглых скобок указывается переменная которая будет хранить значение текущего элемента массива для текущей итерации
5. После каждой итерации, переменная будет переходит на след элемент.

Циклы

1. Оператор цикла for
2. for состоит из четырех блоков
3. for(int i=0; i<13; i++){
   System.out.println(students[i]);
4. }
5. int i =0 – начальная инициализация, не обязательно указывать
6. i<13 – условие окончания цикла, проверяется после каждой итерации цикла
7. i++ - пост операция – код который выполняется после каждой итерации цикла
8. Внутри фигурных скобок {} – тело цикла, которое выполняется при каждой итерации.

Спасибо!