IT-Academy

# Концепция ООП в Java

# Рассматриваемые вопросы

IT-Academy

- Типы данных
- Концепция ООП
- Понятия класс, объект
- Переменные и объекты
- Методы
- Сборка мусора

# Типы данных в языке Java

IT-Academy

Поля, переменные, аргументы и возвращаемое значение методов кроме идентификатора имеют **тип данных.**

Типы данных в языке Java делятся на:

- **примитивные**
- **ссылочные**

В Java есть 8 примитивных типов, которые делятся на 3 группы:

- Целые числа – byte, short, int, long
- Числа с плавающей точкой(вечественные) – float, double
- Логические – boolean
- Символьные - char

Все остальные – ссылочные.

# Примитивные типы данных

IT-Academy

| Тип | Размер (байт) | По умолчанию | Мин | Макс |
|---|---|---|---|---|
| boolean | 1 или 4 | false | - | - |
| char | 2 | '\u0000' | Unicode 0 | Unicode $2^{16}$-1 |
| byte | 1 | 0 | -128 | 127 |
| short | 2 | 0 | $-2^{15}$ | $2^{15}$ -1 |
| int | 4 | 0 | $-2^{31}$ | $2^{31}$ -1 |
| long | 8 | 0L | $-2^{63}$ | $2^{63}$ -1 |
| float | 4 | 0.0f | $1.4*10^{-45}$ | $1.4*10^{38}$ |
| double | 8 | 0.0d | $4.9*10^{-324}$ | $1.8*10^{308}$ |

# Переменные

**Переменная** – это именованная ячейка памяти, содержимое которой может изменяться. При объявлении переменной сначала указывается тип переменной, а затем идентификатор задаваемой переменной.

```
int age;
age = 0;
```

```
int age = 0;
```

Значение null соответствует ссылочной переменной, которой не назначен адрес ячейки с данными.

# Типы byte, short, int, long

IT-Academy

- **int** — основной целочисленный тип, используемый в Java по умолчанию. Любое целое число будет рассматриваться компилятором как число типа **int**. Используется в качестве счётчика циклов, индексов массивов и индексов символов в строках.
- **long** — целочисленный тип содержащий практически бесконечное количество значений. Используется в случаях, где числа превосходят 2 миллиарда и стандартного int уже не хватает. Используется в повседневной жизни для создания уникальных значений.
- **byte** — используется для передачи данных по сети, записи и чтения из файла. В математических операциях, как правило, не используется.
- **short** — самый редко используемый тип в Java, может использоваться только в целях экономии памяти.

# Числовые переменные

IT-Academy

Можно использовать литералы в разных системах исчисления:

int age = 29; (десятичная система)

age = 011101; (двоичная система)

age = 035; (восьмеричная система)

age = 0x1D; (шестнадцатиричная система)

# Нумерические литералы

IT-Academy

Сколько нулей в значении переменной?

```
int age = 100000000;
```

Для удобства чтения значения переменны можно использовать следующее написание:

```
int age = 100_000_000;
```

# Числа с плавающей точкой

IT-Academy

Числа с плавающей точкой (или действительные числа) представлены типами **float** и **double**. Используются для хранения значений с точностью до определенного знака после десятичной точки.

- **double** — это числа с двойной точностью, максимально приближённые к заданным или полученным в результате вычислений значениям. Используется в Java для любых математических вычислений (квадратный корень, синус, косинус,..).
- **float** — менее точный тип с плавающей точкой. Используется очень редко с целью экономии памяти.

# Специальные числа с плавающей точкой

IT-Academy

В языке Java есть три специальных числа плавающей точкой, которые используются для обозначения переполнения и ошибок:

- **положительная бесконечность** - результат деления положительного числа на 0. Представлены константами Double.POSITIVE_INFINITY и Float.POSITIVE_INFINITY.
- **отрицательная бесконечность** - результат деления отрицательного числа на 0. Представлены константами Double.NEGATIVE_INFINITY и Float.NEGATIVE_INFINITY.
- **NaN (не число)** - вычисление 0/0 или извлечение квадратного корня из отрицательного числа. Представлены константами Double.NaN и Float.NAN.

Числа с плавающей точкой нельзя использовать в финансовых расчетах, где ошибки округления недопустимы.

Если же требуется исключить ошибки округления, то следует воспользоваться классом **BigDecimal**.

# Тип char

IT-Academy

Символы описываются в языке Java **char** типом. Символы преобразуются по таблице кодировки UTF-16. По большому счёту это все буквы, числа и специальные символы существующие на нашей планете.

*Размер в байтах* - 2 байта

*Возможные значения (от..до)* - 0..65,535

*Значение по умолчанию* - '\u0000'

```
char c1 = 'a';
```

Пример объявление символьных типов:

Тип **char** является псевдоцелочисленным типом, поэтому значения этого типа можно задавать в виде числа - кода символа из таблицы кодировки UTF-16. Каждому символу соответствует определённое число из таблицы и Java при виде этого числа в рамках типа **char** выводит его

```
char c2 = 97;
System.out.println(c2);
```

```
a
```

# Тип boolean

IT-Academy

Примитивный тип boolean предназначен для хранения логических значений. Логические переменные этого типа могут принимать только два значения: true – истина и false – ложь. Памяти на переменную такого типа требуется 1 бит.

Пример объявление логических типов:

```
boolean b1 = false;
boolean b2 = true;
```

# Преобразование и приведение примитивных типов

IT-Academy

Иногда возникают ситуации, когда необходимо переменной одного типа присвоить значение переменной другого типа.

Например:

```
int i = 11;
byte b = 22;
i = b;
```

# Преобразование и приведение примитивных типов

В Java существует два типа преобразований - автоматическое преобразование (неявное) и приведение типов (явное преобразование).

# Автоматическое преобразование типов

IT-Academy

Если оба типа совместимы, их преобразование будет выполнено в Java автоматически. Например, значение типа byte всегда можно присвоить переменной типа int.

Для автоматического преобразования типа должно выполняться два условия:

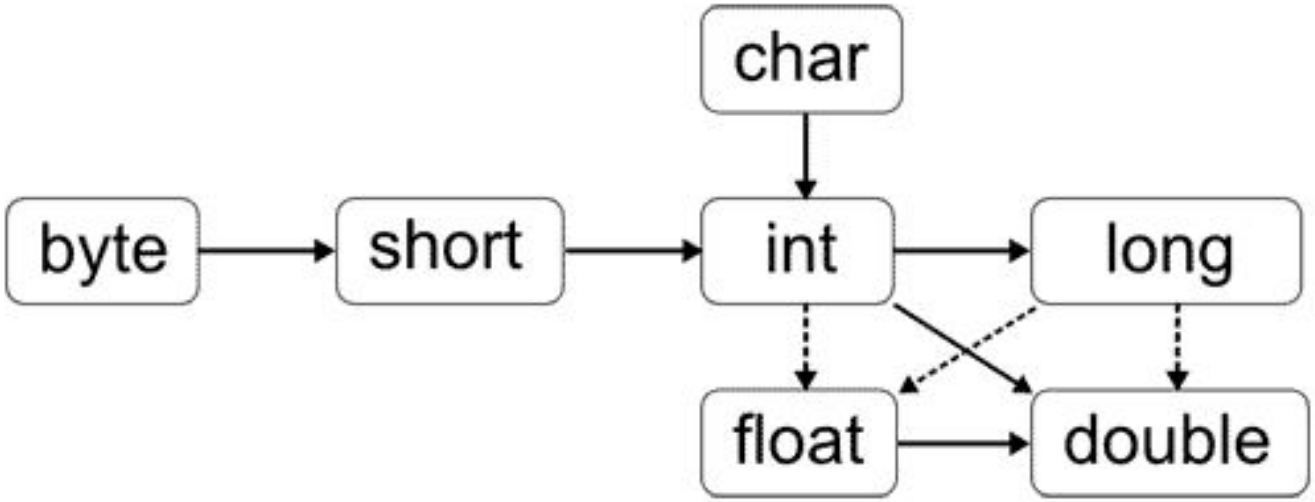
- оба типа должны быть совместимы
- длина целевого типа должна быть больше длины исходного типа

В этом случае происходит преобразование с расширением.

# Автоматическое преобразование типов

**Сплошные** линии обозначают преобразования, выполняемые без потери данных.

**Штриховые** линии говорят о том, что при преобразовании может произойти потеря точности.



# Приведение типов

IT-Academy

Что делать, если значение типа *int* нужно присвоить переменной типа *byte*?

Такое преобразование не будет выполняться автоматически, поскольку длина типа *byte* меньше, чем у типа *int*.

Иногда этот вид преобразования называется **сужающим преобразованием**, поскольку значение явно сужается, чтобы уместиться в целевом типе данных.

**Приведение** - это всего лишь явное преобразование типов.

```
int i = 11;
byte b = 22;
b = (byte) i;
```

# Автоматическое продвижение типов в выражениях

IT-Academy

В языке Java действуют следующие правила:

1. Если один операнд имеет тип double, другой тоже преобразуется к типу double.
2. Иначе, если один операнд имеет тип float, другой тоже преобразуется к типу float.
3. Иначе, если один операнд имеет тип long, другой тоже преобразуется к типу long.
4. Иначе оба операнда преобразуются к типу int.
5. В выражениях совмещенного присваивания (+=,-=,*=,/=) нет необходимости делать приведение.

# Немного кода

IT-Academy

```
public class Main{
  public static void main(String... args) {
      int myAge = 24;
      long passportNumber = 1234567890L;
      char yourMark = 'A';
      double averageMark = 4.12;
      byte test = 512; // ???
    System.out.println("your exam results: ");
    System.out.println("your age is: " + myAge + ", passport number: " +
    passportNumber);
    System.out.println("you got " + yourMark + ", average mark is " +
    averageMark);
  }

}
```

# ООП - парадигма

IT-Academy

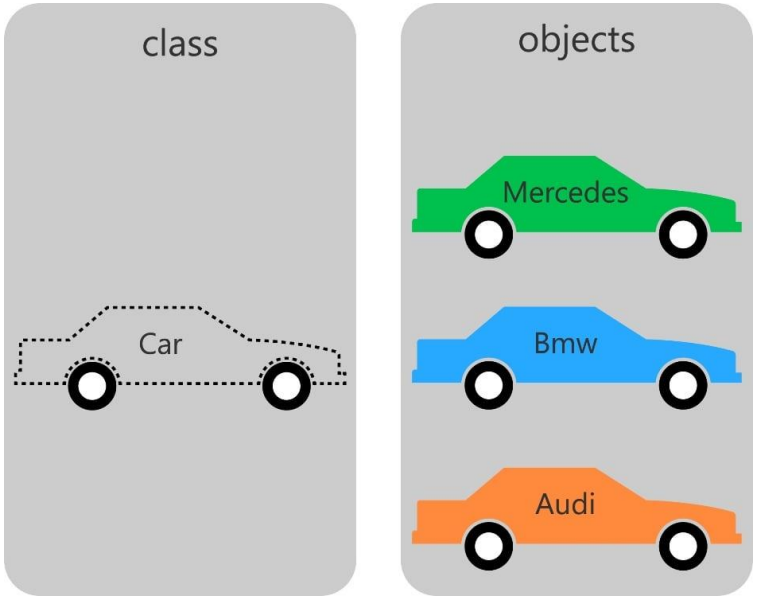
**Окъектно-ориентированное программирование или ООП –** парадигма программирования, где основными концепциями является понятия объектов и классов.

**Объект –** это сущность или экземпляр класса, который может посылать сообщения и может на них реагировать, используя свои данные.

**ООП** возникло в результате развития идей процедурного программирования, где данные и функции (методы) их обработки формально не связаны.

# Понятие Класс и Объект

- **Класс –** прототип, чертеж, определяет структуру и поведение создаваемых объектов
- **Объект –** конкретный, реальный экземпляр класса



# Поле и метод класса

IT-Academy

**Класс в Java имеет два основных элемента:**

**1. Поле (field) –** имеет модификатор доступа, тип данных, идентификатор и значение (переменное или постоянное)

**2. Метод (method) –** имеет модификатор доступа, идентификатор аргументы (входные параметры), возвращаемое значение или void, тело

# Класс в Java

```
public class Man {

   // Comments one line
   int hairLength;      //поле класса

     /*
       * Comments several lines
       */
   public void grow() {
    hairLength = hairLength + 1;
   }
}
```

# Идентификаторы

**Идентификаторы –** это имена пакетов, классов, интерфейсов, объектов, полей, методов, переменных, параметров методов и т.д.

```
package by.pvt; //идентификатор пакета

public class Car {

    int price; //поле

    public int getPrice() {  //метод, который позвращает
значение
        return price;
    }

    public void setPrice(int price) {  //метод с парамметрами
        this.price = price;
    }
}
```

# Идентификаторы

IT-Academy

Названия идентификаторов выбираются по следующим правилам:

- они должны начинаться с буквы или символа подчеркивания “_” и “$”;
- они могут содержать латинские буквы, символы подчеркивания или цифры без пробелов;
- названия идентификаторов не должны совпадать с ключевыми словами языка Java;

Длина идентификатора в Java - любая.

А лучше почитать про coding conventions java.

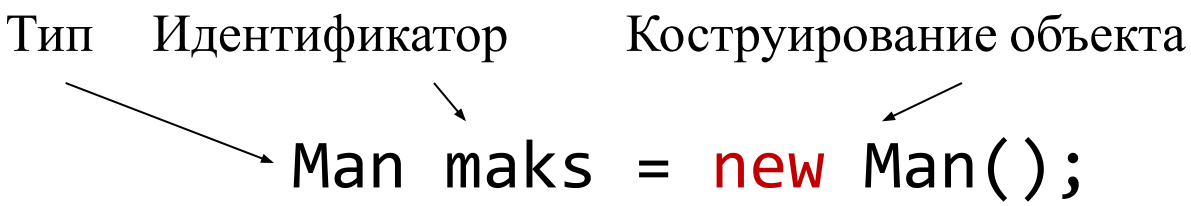
https://en.wikibooks.org/wiki/Java_Programming/Coding_conventions

# Объекты

IT-Academy

Все объекты класса имеют одинаковые наборы полей данных (атрибуты объекта), но с независимыми значениями этих данных для каждого объекта.

Значения полей данных объекта задают его состояние, а методы – его поведение. Сами объекты безымянны, и доступ к ним осуществляется только через ссылочные переменные:



# Конструкторы

**Конструктор** — это метод класса, который инициализирует новый объект после его создания. Имя конструктора всегда **совпадает** с именем класса, в котором он расположен. У конструкторов нет типа возвращаемого результата - никакого, даже void.

# Конструкторы по умолчанию

IT-Academy

```
public class Man {
   int armsNumber;

   public Man() {
      this.armsNumber = 2;
   }
}
```

Вызов конструктора по умолчанию:

```
Man man = new Man();
```

# Конструкторы с параметрами

IT-Academy

```
public class Man {
   int armsNumber;
   int hairLength;

   public Man(int hairLength) {
      this.armsNumber = 2;
      this.hairLength = hairLength;
   }
}
```

Вызов конструктора с параметрами:

```
Man man = new Man(10);
```

# Сборка мусора

IT-Academy

- В языках без сборки мусора (например Си), при работе с памятью (при записи в память), нужно не забывать вручную очищать память.
- В Java реализован сборщик мусора, переодически останавливающий на очень короткое время программу (обычно менее секунды), уничтожающий неиспользуемые данные.
- Сборщиком мусора - специальный процесс периодически освобождает память, удаляя объекты, которые уже не будут востребованы приложениями — то есть производит «сбор мусора».

# Класс Runtime

IT-Academy

Свободная память у виртуальной машины:
Runtime.getRuntime().freeMemory();

Вызов сборщика мусора из приложения:
Runtime.getRuntime().gc();

# Создаем мусор

IT-Academy

Создаем объекты класса Man и помещаем их в динамическое хранилище List в памяти JVM.

```
ArrayList list = new ArrayList();

for (int i = 0; i < 15_000_000; i++) {
        list.add(new Man(i));
}
```

# Наблюдаем память JVM

IT-Academy

```
public class Main {

public static void printMemory() {
        System.out.println("Max mem: " + Runtime.getRuntime().maxMemory());
        System.out.println("Total mem: " + Runtime.getRuntime().totalMemory());
        System.out.println("Free mem: " + Runtime.getRuntime().freeMemory());
 }

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

}
```

# Зачем собирать мусор?

IT-Academy

Физическая память компьютера ограничена.
Компьютер выделяет конечное количество памяти для JVM.

Выделением памяти JVM можно управлять:

>java –X

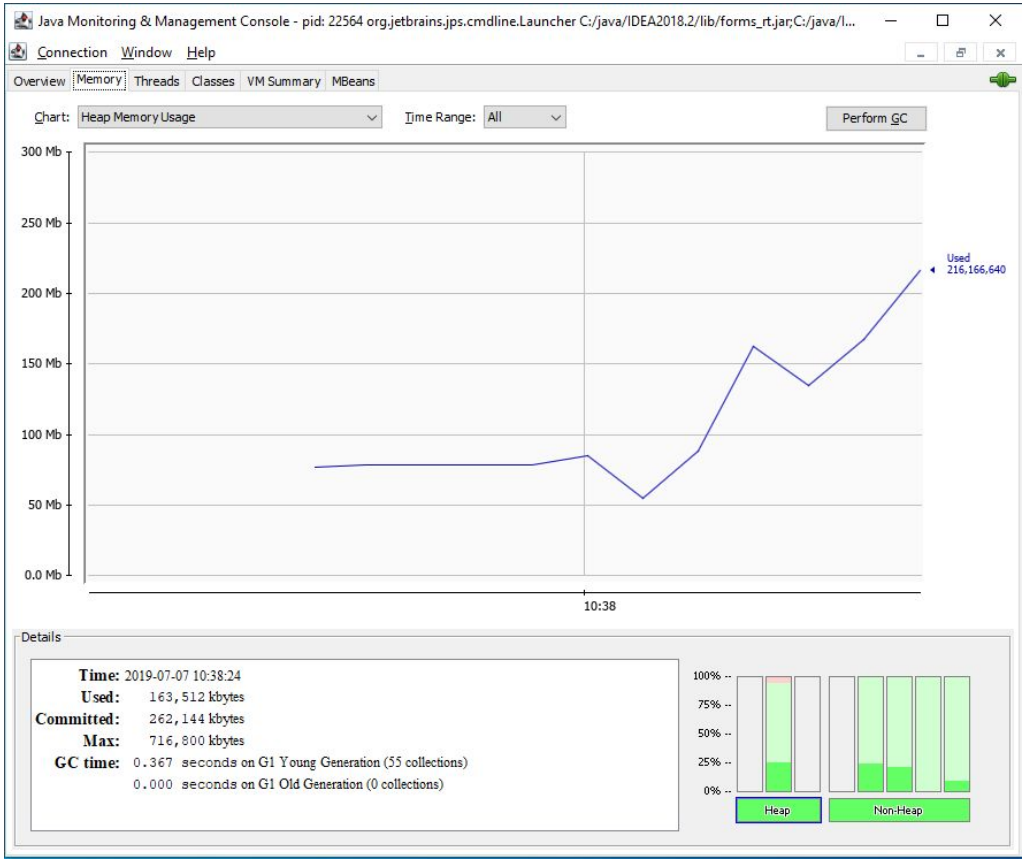
>java -Xmx1024M -Xms1024M Main

# Инструменты

IT-Academy

Мониторим использование памяти:

> %JAVA_HOME%**bin\jconsole.exe**



# Вопрос

IT-Academy

IT-Academy

# Дополнительные материалы

1. **Object-Oriented Programming Concepts**
https://docs.oracle.com/javase/tutorial/java/concepts/index.html

2. **Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide**

https://docs.oracle.com/en/java/javase/11/gctuning/introduction-garbage-collection-tuning.html

IT-Academy

# Спасибо за внимание