Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и интеллектуальных систем

02. JAVA ИЛИ ИНФОРМАТИКА?

ЛЕКЦИИ ПО ИНФОРМАТИКЕ ДЛЯ СТУДЕНТОВ ВТОРОГО КУРСА
ВЫСШЕЙ ШКОЛЫ ИТИС КФУ
2019

М.М. АБРАМСКИЙ

СТАРШИЙ ПРЕПОДАВАТЕЛЬ

КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ

Что должен включать в себя язык программирования

- Синтаксис языка (его *грамматика*) – как писать на нем правильные программы

- Программа для компьютера, которая умеет превращать правильный код на языке программирования (текстовый файл, вообще говоря) в машинный код
  - *Еще раз: .pas, .cpp, .cs, .java – это все текстовые файлы*

Java

- “Oak”, James Gosling
- 1996 Java 1.0, сейчас Java 11 (2018), но самая используемая – Java 8 (2014)

- Объектно-ориентированный, императивный, кросс-платформенный, C-образный синтаксис

- Применение:
  - Корпоративные приложения
  - Клиент-серверные приложения
  - Мобильные устройства (застали телефоны с Java ME)?

JVM

- Java Virtual Machine – виртуальная машина Java, реализующая кросс-платформенность для любого приложения на Java
  - “Write once, run everywhere”

- Java-приложение транслируется в байт код, который потом выполняется JVM (Just-In-Time)

Где Java?

- В продуктах Oracle (владеет Java)

- Amazon, eBay, LinkedIn, Yahoo
- Практически все Android-приложения
- Системы e-Commerce
    - Банки часто держат отделы квалифицированных java-разработчиков
- *и т.д.*

Чтобы все работало

- Минимум – Java Development Kit (JDK)
- Для выполнения байт кода – Java Runtime Environment (JRE)

Утилиты JDK:

- **javac** – компилятор, компилирует .java файлы в .class-файлы (байт код)
- **java** – запуск JVM, которая выполняет .class файлы.

Компилятор / интерпретатор

- **Компилятор** – выполняет целиком трансляцию программы из языка высокого уровня в машинный код
  - Сразу всю программу
  - Дальше вы можете запустить этот машинный код (например, .exe)
- **Интерпретатор** – выполняет построчную (покомандную) трансляцию и выполнение
  - Например, Python, PHP – интерпретаторы
- **Java** – компилируемый язык, но превращается в т.н. байт-код, который запускается виртуальной машиной Java (.class-файлы).
  - В литературе называется **интерпретатором компилирующего типа**

Запуск java-файла

- Компиляция
  - **Способ старой доброй командной строки:**
  - javac ИМЯ.java
    - Ничего не вывелось – повезло, нет ошибок
    - Вывелось – исправляем код, снова вызываем javac
      - » поиск по истории выполненных команд – клавиши вверх/вниз
  - По итогам появился .class файл – это байт-код, его может запускать JVM.
- Запуск
  - java ИМЯ

Литература по курсу *(первый семестр)*

- Ильдар Хабибуллин «Технология Java»
  - *и другие его книги по Java.*
- Брюс Эккель «Философия Java»
- Герберт Шилдт «Полный справочник по Java»
- Джошуа Блох «Java – эффективное программирование»
- Гради Буч «Объектно-ориентированный анализ и проектирование»
  - *Примеры на C++, но одна из лучших книг по ОО-парадигме.*
- Дональд Кнут «Искусство программирования на ЭВМ», т.1
  - *Классика жанра по алгоритмизации.*

…Previously on…

- Информация
- Информационные процессы и системы
- Управление
- Алгоритмы
    - От абстрактных машин до высокоуровневых языков

Информация к размышлению #1

Не все задачи, которые можно сформулировать, имеют алгоритм!

Пример: ***проблема остановки***

- *Помните машину Тьюринга? А то, что ее программу можно закодировать? А помните, что закодированная информация – это число? Получается – у каждой МТ есть уникальный номер.*
- **f(x) = 1**, если машина Тьюринга с номером (кодом) **x** останавливается (не зацикливается) на входе **x**, а иначе **f(x) = 0.**
- ***Нет алгоритма, вычисляющего f(x)***
  - ***Доказанная теорема***

ПРАВДА О ПАРОЛЯХ

- Их знаете только вы!

- Сайты не должны знать пароли своих пользователей:
  - Злоумышленники, украв базу, узнают пароли
  - Админы могут получать доступ туда, куда не надо.

ЧТО ПРОИСХОДИТ НА САМОМ ДЕЛЕ?

РЕГИСТРАЦИЯ:

- HASH(ваш пароль) -> хэш
  - строка вида **d6aabbdd62a11ef721d15**
  - легко подсчитать, сложно узнать исходный пароль

ВХОД НА САЙТ:

- HASH(введенный пароль) сравнивается с хэшем пароля, который вы ввели при регистрации:
  - Если хэши равны – значит и пароль совпадает с тем, что лежит в базе – вас пускают.

Пример хэша (sha256)

password
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8

Password
e7cf3ef4f17c3999a94f2c6f612e8a888e5b1026878e4e19398b23bd38ec221a

Password1
19513fdc9da4fb72a4a05eb66917548d3c90ff94d5419e1f2363eea89dfee1dd

Сасса бен Дахир

ТАК СКОЛЬКО ЗЕРЕН?

$$\mathbf{N} = 1 + 2 + 4 + \ldots + 2^{63}$$

$$= 18\,446\,744\,073\,709\,551\,615$$

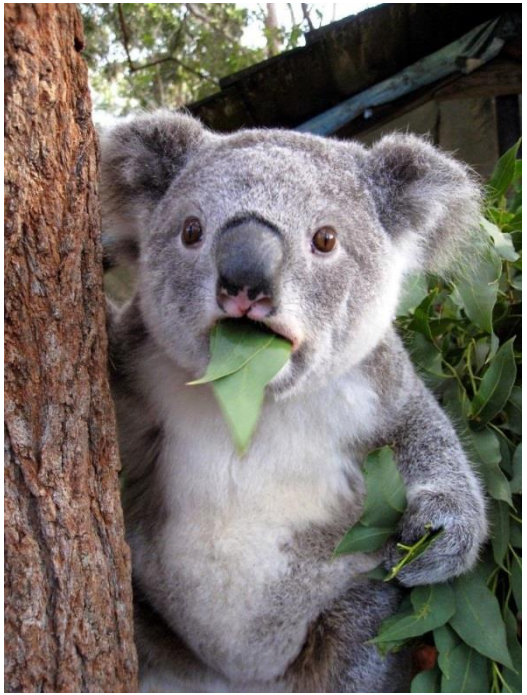
ЕСЛИ 1 ЗЕРНО = 1 СЕКУНДА:

307 445 734 561 825 860 **МИН** = 5 124 095 576 030 431 **ЧАСОВ** =
213 503 982 334 601 **ДНЕЙ** = 584 942 417 355 **ЛЕТ**

ВОЗРАСТ ЗЕМЛИ: 14 000 000 000 **ЛЕТ**

А СКОЛЬКО РАЗНЫХ ЗНАЧЕНИЙ ХЭША?

Примерно N · N · N · N
(77 цифр)



Даже считать не будем пытаться…

АТАКА НА ХЭШ?

Решение должно не просто работать, а работать эффективно

Игра, которая тормозит, тоже работает.

Другой пример: катастрофическое рекурсивное вычисление n-го числа Фибоначчи:

```
int fib(int n) {
    если n == 1, то вернуть 1
    иначе если n == 2, то вернуть 1
    иначе вернуть fib(n-1) + fib(n-2)
}
```

– что плохого в этом примере?

Программа и данные

- Обрабатывает входные данные, генерирует выходные данные.
- Должна иметь доступ к данным, которые где-то хранятся.
- Должна иметь разные инструменты работы с разными данными
  - Текст и числа обрабатываются разными способами

Хранение данных, пока программа работает

Данные должны храниться [в программе], пока программа работает.

Можно конечно писать программы, которые не хранят данные, но далеко на таком подходе не уехать.

- *in future*: потоковая обработка/обработка «на лету»

Пример без промежуточного хранения:

- работает, только если ввод – функция (а не процедура, как в Pascal)
- print(int(input()) + int(input())) – сумма двух целых на Python.

Где хранить данные

В специальных ячейках

- У них должно быть имя, чтобы можно было обращаться к данным в них (**имя**)
- У них должно быть достаточно места для хранения (**тип**)
- Эти данные могут изменяться в процессе работы
  - изменяться по-английски? (не change)

Переменная

- **Имя**
  - идентификатор
    - Регулярное выражение [A-Za-z_][A-Za-z0-9_]*
    - Начинается на букву или _, а дальше могут быть буквы, цифры или _.

- **Тип**
  - не просто характеризует переменную
    - С точки зрения теории программирования определяется все возможные значения, которые принимает переменная
    - С точки зрения архитектуры определяет размер данных, которые хранятся в переменной.

Какие могут быть данные (в зависимости от того, что с ними можем сделать)?

- Текст
- Число
- В общем случае – бинарные – закодированные в двоичном виде данные, которые что-то означают (опять же чаще – число или текст)

- Данные программ (переменные) хранятся в оперативной памяти.

- Но есть специфика в зависимости от типов данных
  - Примитивные типы
  - Ссылочные типы

Примитивные типы в Java (8 типов)

- Также известны как «скалярные»
- Память под переменную примитивного типа выделяется в момент объявления
  - Объявляется как «ТИП ИМЯ», например «int x»

- В выражениях переменная означает свое значение
  - x = y + 2; (y – только в качестве значения, x – переменная в которую значение записывается)

Примитивные

- Числа
  - Целые
    - **byte** – 1 байт
    - **short** – 2 байта
    - **int** – 4 байта
    - **long** – 8 байт
  - Вещественные
    - **float** – 4 байта
    - **double** – 8 байтов
- Символы
  - **char** - 2 байта
    - Часто считается тоже целочисленным
- Логический тип
    - **boolean**

Объявил переменную = выделил память

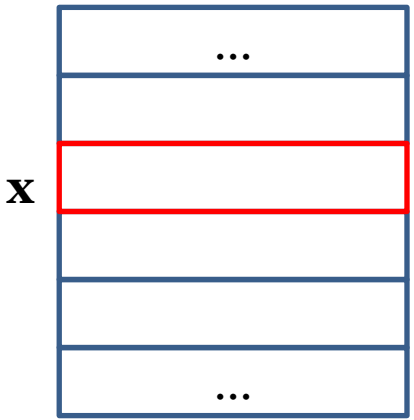
Ссылочные типы в Java (все остальные)

- Переменная хранит не содержимое, а ссылку на область памяти, где содержимое лежит.
  - К данным нужно обращаться по другому
    - user.username

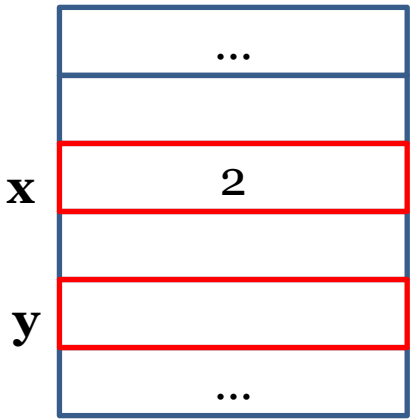
- Объявление переменной и выделение памяти – 2 разных действия
  - StringBuilder s = new StringBuilder(“ITIS”);
    - » StringBuilder – изменяемая строка в Java

Память и примитивные переменные

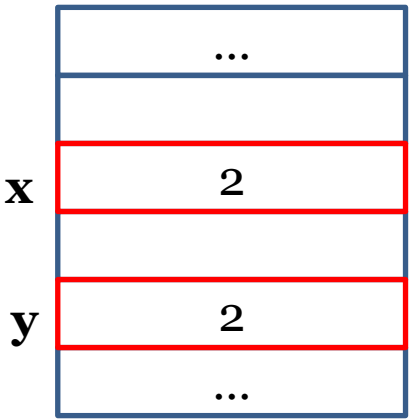
int x;



x = 2;
int y;



y = x;



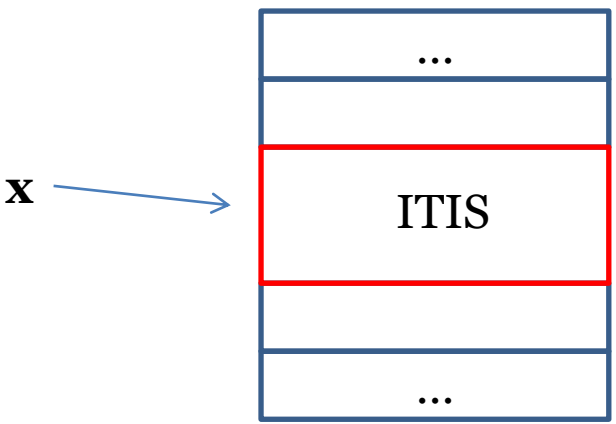
Значение скопировалось

Примитивная переменная «хранит» в себе значение (можно это себе так представлять)
Пишу переменную, подразумеваю значение

Память и ссылочные переменные

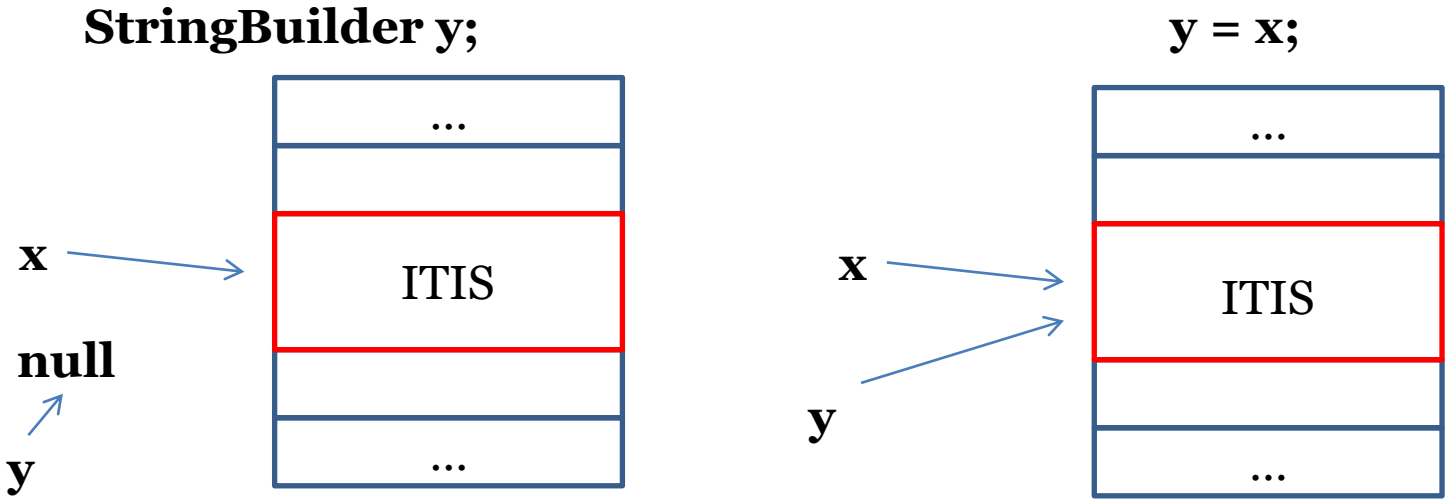
StringBuilder x;

x = new StringBuilder(...);



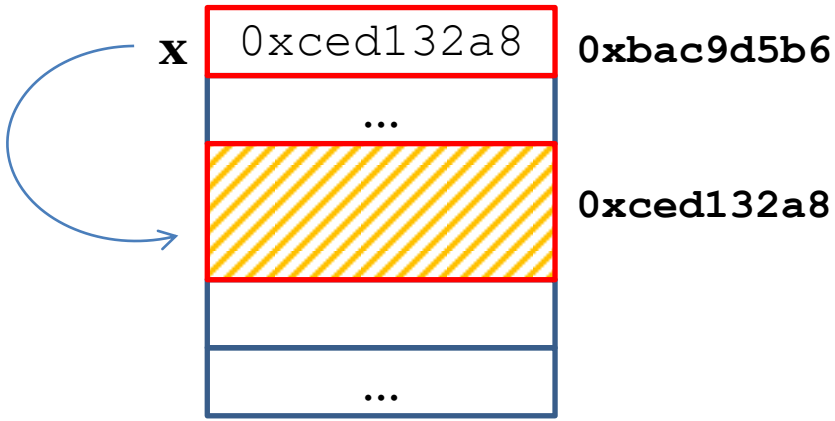
Ссылочная переменная «хранит» в себе ссылку на то место в памяти, где данные лежат.
Т.е. сама по себе хранит просто число (номер ячейки, адрес).
null – пустое значение такой ссылки.

Память и ссылочные переменные



При присваивании не происходит копирования данных. Копируется только ссылка. (*логично – у вас у всех есть ссылка на универ, в котором вы учитесь, но универ при этом не дублируется*)

Память – подход С.



x (ссылочная переменная) хранит в себе явно адрес (сам хранит число как примитивная переменная) и в то же время через этот адрес указывает на данные.

В си вообще говоря все не так просто, тут просто продемонстрирована модель.

В Java прямого доступа к адресам нет, эту работу JVM прячет от греха подальше.
Но модель **такая же** (для понимания)

Где хранятся переменные

- **Стек (Stack)** – место хранения переменных примитивного типа
  - Небольшой размер (Stack Overflow)
  - Быстрые операции добавления и удаления
  - *Подробнее разбирается в стеке вызовов в методах*

- **Куча (Heap)** – место хранения данных ссылочного типа
  - Динамически выделяемая память (большого размера)
  - *Подробнее разбирается в теме про ООП*

Где хранятся переменные

~~В оперативке!~~

Оперативная память не то, чем кажется

Память

- Физическая
  - Те самые платки оперативной памяти
- Виртуальная
  - Место на жестком диске, которое «прикидывается» оперативной памятью
    - » Гораздо больше физической памяти!

Какие числа влезают в целые типы?

- На примере byte.

- 1 байт – 8 бит, каждый бит – 0 или 1.
  – сколько возможных различных вариантов?

256

- Комбинаторика: 2^8 = 256
- Или в лоб:

```
00000000 = 0
00000001 = 1
00000010 = 2
...
11111110 = 254
11111111 = 255
```

Но как быть с отрицательными числами?

Как реализован знак?

- Представьте, что был бы еще 9й бит

```
1 0000 0000 -
          1
  1111 1111
```

-1

```
1111 1111 - 1 = 1111 1110
```

-2

и т.д.

Числа со знаком

данные в byte, short, int, long – не всегда соответствуют своему двоичному коду

```
00000000 = 0
00000001 = 1
00000010 = 2
…
01111111 = 127
10000000 = -128 (хотя должно было быть 128)
…
11111110 = -2
11111111 = -1 (хотя должно было быть 255)
```

Число со знаком

- Первый бит – знак (0 – плюс, 1 – минус).
- Получается на само число остается на 1 бит меньше.
- byte – целые числа из [-128; 128)
  - 128 = 2^7, хотя в байте 8 бит
  - По аналогии с int, short, long

Вещественные типы

- double – 8 байтов
- float – 4 байта

- Запись **2.8e-8** означает **2.8*10^-8**

Прикол с 2.0 – 1.1

- System.out.println(2.0 – 1.1);
- Выведет: 0,8999999999999
- Почему?

Объявление и инициализация

```
int x;
int y, s;
int u = 2, z;
```

Неинициализированные переменные использовать нельзя!

- %Username%, сколько будет x + 5?
- А что такое x?

Арифметика

- +, -, *, /, % (остаток от деления)
- любая целочисленная константа – int
- любая вещественная константа– double
- результат арифметического выражения – всегда int (если нет long)
- даже такой:

```
byte b1 = 50;
byte b2 = 51;
byte b3 = b1 + b2;// не сработает!
```

Приведение типов

- (тип)
- `byte b3 = (byte)(b1 + b2);`
- если не влезает
  - нет явного приведения типов – ошибка
  - есть – сужение
    - `byte b1 = 100;`
    - `byte b2 = 100;`
    - `byte b3 = (byte)(b1 + b2);` ***(у двоичного представления 200 убирают все биты, кроме последних 8ми)***

Еще операторы

***a*++** (инкремент)

- Увеличение на единицу
- Разница в “x = ++a”, “x = a++”
  - » Постфиксный и префиксный инкременты

a = 2; b = 3;
c = a+++++b

Аналогично ***a--*** (декремент)

Еще операторы

- +=, -=, *=, /= и т.д.

x += y

- Это не “x = x + y”, а “x = (тип x)(x + y);”

Приоритет операций

- Приведение типов
- Скобки
- *, /
- +, -