Java Basics: Variables

CPSC 233: Introduction to Computer Science for Computer Science Majors II
Winter 2022

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Thursday, 4 November 2021

UNIVERSITY OF CALGARY

Java Variables (primitive/Object)

- Java has 2 types of simple variables
- Primitive
  - int, byte, short, long
  - float, double
  - boolean
  - char
- Object
  - Integer, Byte, Short, Long, BigInteger
  - Float, Double, BigDecimal
  - Boolean
  - Character
  - String

Java Variables (Python mapping)

- Java has 2 types of simple variables
- Primitive
  - int, byte, short, long
  - float, double
  - boolean
  - char
- Object
  - Integer, Byte, Short, Long, **BigInteger (int -flexes in size as integer grows)**
  - **Float(float -32 bit vm)**, **Double (float -64 bit vm)**, BigDecimal
  - **Boolean (bool)**
  - Character
  - **String (str)**

Java Variables (primitive/Object)

- Java has 2 types of simple variables
- Primitive – like c++ data types (just store data, primitives are mutable)
  - int, byte, short, long
  - float, double
  - boolean
  - char
- Object – are objects (have methods and primitive wrapper objects are immutable)
  - Integer, Byte, Short, Long, BigInteger
  - Float, Double, BigDecimal
  - Boolean
  - Character
  - String

Typing

Variables (typing)

- Unlike python all variables are **typed explicitly**
  - This means you have to tell Java what type of data the variable can store (and this is permanent)
  - If typing doesn't match during compile time (or possibly while running) the program will fail to compile (or crash if running)
- Usual simple mappings

Python type		Java type	
int	x = 1	int	int x = 1;
float	x = 1.0	double	double y = 1.0;
bool	x = True y=False	boolean	boolean x = true; boolean y = false;
str	x = "Hello"	String	String x = "Hello";
str	x = "H"	char	char x = 'H';

Variables (typing)

- More exact mappings (not as common but software engineering reasons to prefer)

Python type		Java type	
int	x = 1	BigInteger	BigInteger x = 1;
float	x = 1.0	Double	Double y = 1.0;
bool	x = True y=False	Boolean	Boolean x = true; Boolean y = false;
str	x = “Hello”	String	String x = “Hello”; //Double quotes
str	x = “H”	Character	Character x = ‘H’; //Single quotes

- There are some performance consequences from using wrapper objects over primitives, but in most cases not an issue
- Wrapper objects lead to clearer software engineering design

What do the most common names mean

- int/Integer – 4 byte integer (signed) -2^31 to 2^31-1
- double/Double – 8 byte fraction 64 bit (signed)
- boolean/Boolean – 1 bit
- String – array of char (2 byte per character/letter stored)

What do the other names mean

- byte/Byte – 1 byte integer (signed) -2^7 to 2^7-1
- short/Short – 2 byte integer (signed) -2^15 to 2^15-1
- long/Long – 8 byte integer (signed) -2^63 to 2^63-1
- float/Float – 4 byte fraction 32 bit (signed)
- char/Character – 2 bytes (single character/letter – example one ASCII)
- *BigInteger -> scaling integer*
- *BigDecimal -> scaling fractional number*

Creation/Assignment

Variables - Creation

- Variable creation (can be done without assignment):
  - **<type>** <*name*>**;**
- Variable assignment
  - <name> = <new data>;
- Variable initialization (both at once):
  - **<type>** <*name*> = <*data*>**;**
- **We must end with a semi-colon**
  - **this means a line of Java code is done**
  - **syntax errors if missing!!!**

UNIVERSITY OF CALGARY

Variables – explicit typing

- In Java**, the type is explicit**

- When assigning new data if it doesn't match and Java can tell that on compile time it will crash (ex.)

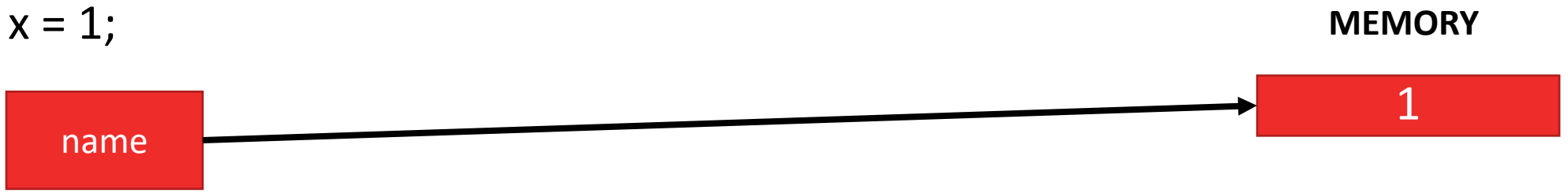
```
int x = 1;
String y = "a";
x = y;
```

incompatible types: java.lang.String cannot be converted to int

UNIVERSITY OF CALGARY

Variable Memory - primitive

```
int x = 1;
```

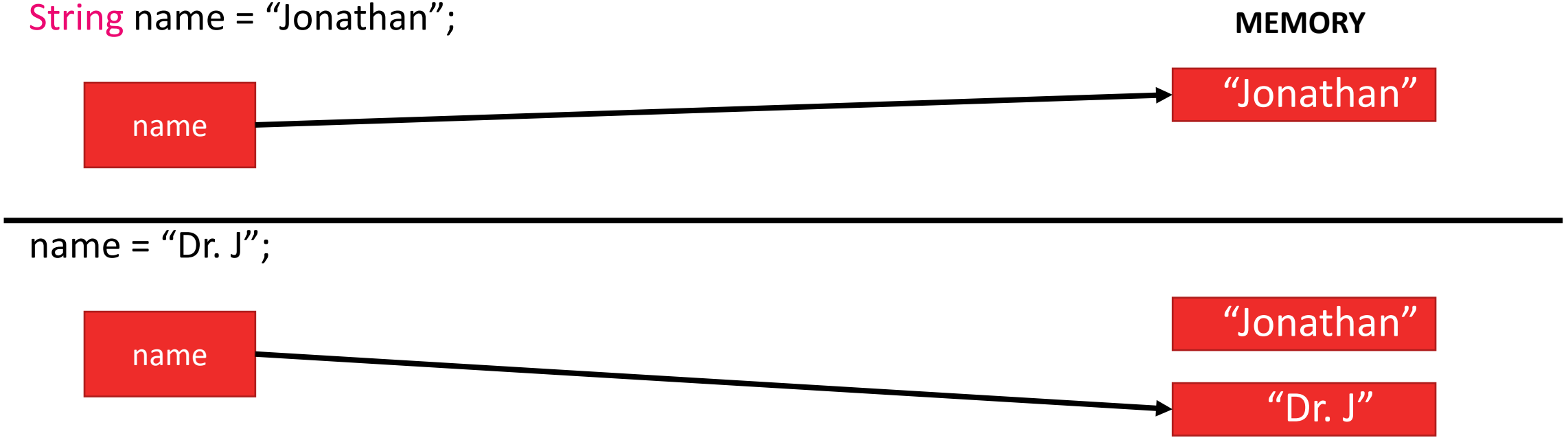



```
x = 2;
```

- Changing a **primitive** changes memory.

Variable Memory - Object

String name = “Jonathan”;



name = “Dr. J”;

- Changing an Object variable doesn’t change memory. It actually just points the name to a new memory spot with the new information. (Java eventually will get around to throwing away “Jonathan” via something called *garbage collection*)

Naming

Variable Naming – Requirements

- **Java naming convention**:
  - Names **must start with a letter** (e.g., a, ..., z, A, ..., Z) (can also start with _,$)
  - A name **may contain any letter, any number (0, ..., 9), and the special character "_" (underscore).**
  - **White spaces and signs** with special meanings (e.g., "+", "-", "*", "/") **are not allowed.**
- **Case sensitive** "sum" is different than "Sum"
- **Cannot use reserved keywords**.
- Legal variable names: fooBar, X15Y, _variable, $var
- Illegal variable names: **4**2Bars, How Much, **-**Bar

Variable Naming – Constants

- **ALL CAPITALS is used to indicate a constant where value won't change**
  - **(not enforced unless we use syntax-> final)**

- **Local constant (ex. in a function)**
  - `final int LOCAL_CONSTANT = 1;`
  - `int local_variable = 2;`

- **Global constant (ex. in a class)**
  - `static final int GLOBAL_CONSTANT = 3;`
  - `static int global_variable = 4;`

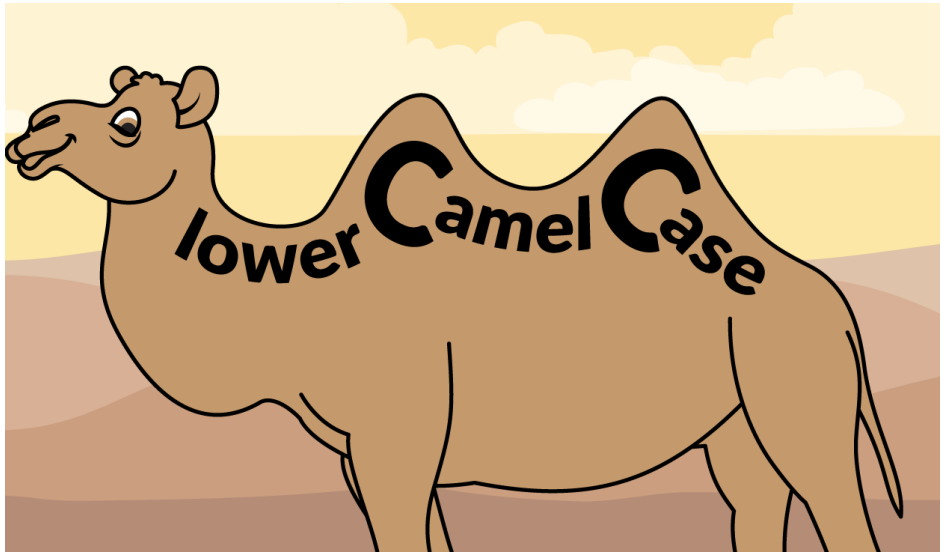
Variable Naming - Style

- **Style requirement:**
  - Use meaningful names for readability.
  - Balance between brevity and descriptiveness

IndexToDatabseOfMyCourses (too long)
IndexToCourseDB (better)
INDEXTOCOURSEDB (hard to read -> **CONSTANT**)

- Avoid using variables that are only distinguishable by upper/lower case
  - (e.g., don't use "Sum" and "sum" in the same program)

Variable Naming - Case



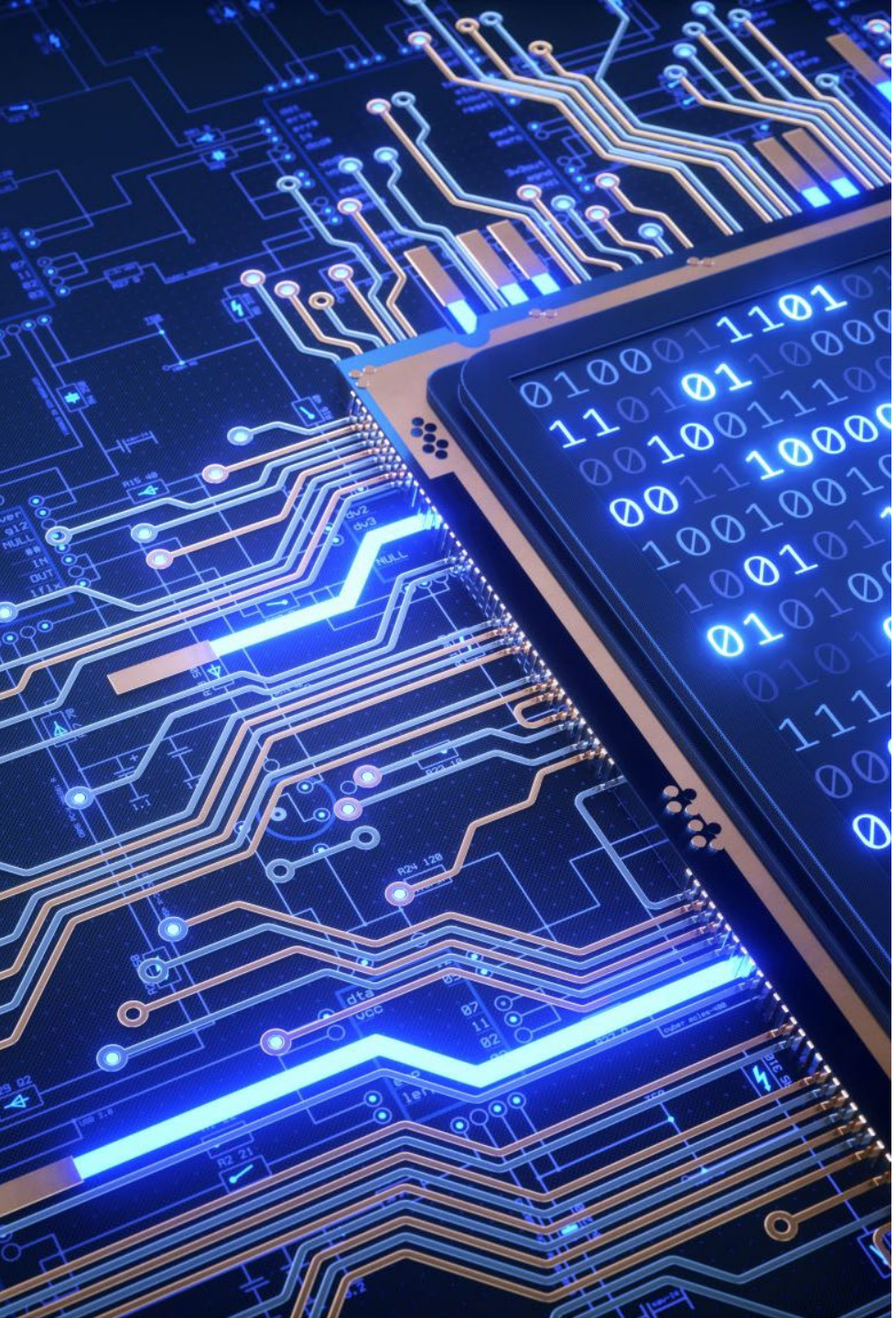
sumNetProfit

- Popular in Java, used in Python

sum_net_profit

- used more often in Python

Operators



Operators and Operands

- Operators are special tokens that represent computations like addition, multiplication and division. The values the operator works on are called operands.
- Operates:
  - **+, -, *, / ,%** → are clear
  - **Math.pow(num, exp)** → exponentiation
  - **//** → integer division (doesn't exist)
  - <,<=,>=,>,!=,== → relation operators
  - &&, ||, ! → boolean operators (and, or, not)
  - =, +=, -=, *=, /= → assignment operators
  - ++, -- → increment by 1 (+= 1, -= 1)

UNIVERSITY OF CALGARY

Division (integer division is based on types used)

```
int x_int = 1;
int y_int = 2;
double x_dbl = 1.0;
double y_dbl = 2.0;

x_int  / y_int  = 0;      -> integer
x_int  / y_dbl = 0.5;  -> double
x_dbl / y_int  = 0.5;  -> double
x_dbl / y_dbl = 0.5; -> double
```

Casting (changing types)

Widening Casting (automatically) - converting a smaller type to a larger type size

byte -> short -> char -> int -> long -> float -> double

```
int x = 9;
double y = x;                    //now has 0s after decimal point (y stores 9.0)
```

Narrowing Casting (manually) - converting a larger type to a smaller size type

double -> float -> long -> int -> char -> short -> byte

```
double x = 3.14;
int y = (int) x;                 //Loses data after decimal point [truncation] (y stores just 3)
```

Precedence

Precedence

Order	Operations	Precedence
1	()	Highest
2	++,--	Increment 1
3	-x, +x, **!x**	Unary (not)
4	x * y, x / y, x % y	Multiplicative
5	x + y, x – y	Additive
6	<, <=, >=, >	Relational
7	==, !=	Equality
8	**&&**	and
9	**||**	or
10	=, +=, -=, *=, /=	Lowest

The order of evaluation is determined by operator precedence (highest -> lowest)

Highest means the expression is collapse on execution of this operator first

Strings

Strings

- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html
- Let us store text (technical each text symbol is 2-bytes each)

- No indexing (use function charAt(index) instead, only positive indices, no [-1])
- No slicing via [], need to use substring() function
- String + String → concatenation works and makes a new String
- String * int → does not work
- String hello = “Hello, world!”; → define
- hello.length(); → character count, like python len(hello)

Comparing (Equality and Ordering)

- For Strings (and other **object** types)
- use .equals() instead of ==
  - Use if (s1.equals(s2)) ← this tells you if contents to two string are the same
  - instead of if (s1 == s2) ← this tells you if two strings are the same object

- Use .compareTo() instead of <, <=, > , >= (and possibly ==)
  - s1.compareTo(s2) < 0 instead of s1 < s2
  - s1.compareTo(s2) == 0 instead of s1 == s2
  - s1.compareTo(s2) > 0 instead of s1 > s

Onward to … IO.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY