Java introduction

Ionut
Spalatelu

Outline

- About me
- Goals
- Variables
- Data types
- Operators
- Control flow statements
- Arrays

About me

~ 7 years of programming experience (mostly Java)

~ 6 years since my first training

- Oracle Certified Associate, Java SE 8
- Oracle Certified Professional, Java SE 8
- Oracle Certified Professional, Java SE 11
- Oracle Certified Expert for Oracle DB SQL
- Oracle Certified Expert, JEE - JPA
- Oracle Certified Expert, JEE - Web Components
- Oracle Certified Expert, JEE - Web Services
- Spring 5 Certified Professional
- acredited trainer by National Certification Authority

ORACLE Certified Associate

ORACLE Certified Professional

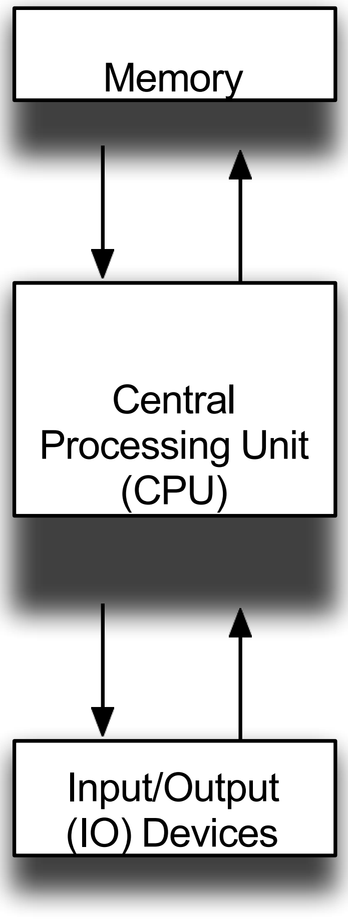
ORACLE Certified Expert Oracle Database SQL

Pivotal CERTIFIED SPRING PROFESSIONAL

Goals

- Learn enough Java to do something useful

The Computer



CPU Instructions

$z = x + y$

Read location x

Read location y

Add

Write to location z

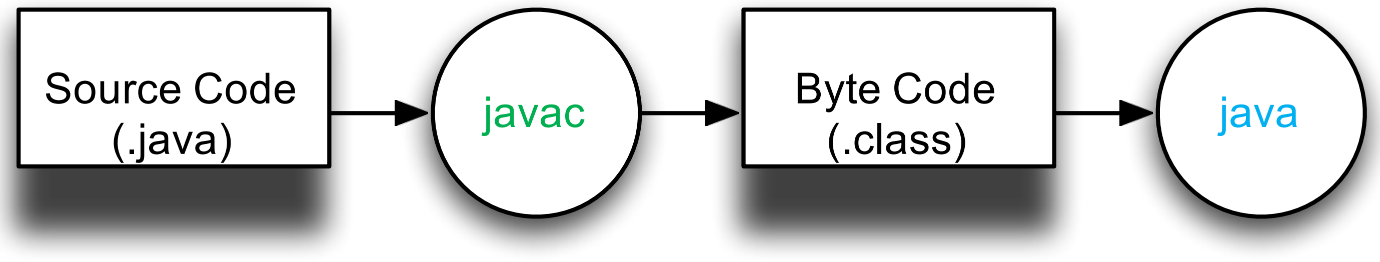
Programming Languages

- Easier to understand than CPU instructions
- Needs to be translated for the CPU to understand it

Java

- One of the “most popular” languages
- Runs on a “virtual machine” (JVM)
- More complex than some (eg. Python)
- Simpler than others (eg. C++)

Compiling Java



First program

```
public class Hello {
    public static void main(String[] args) {
        // Program execution begins here
        System.out.println("Hello world.");
    }
}
```

Program structure

```
class CLASSNAME {
    public static void main(String[] arguments) {
        STATEMENTS
    }

    //OR

    @Test
    public void myTestMethod() {
         STATEMENTS
     }
```

Output

System.out.println(*something*) outputs to the console

Example:

```
System.out.println("output");
```

Variables

- A way of storing information inside the computer

Variables

- A way of storing information inside the computer
- As its name suggests, it's content can be changed

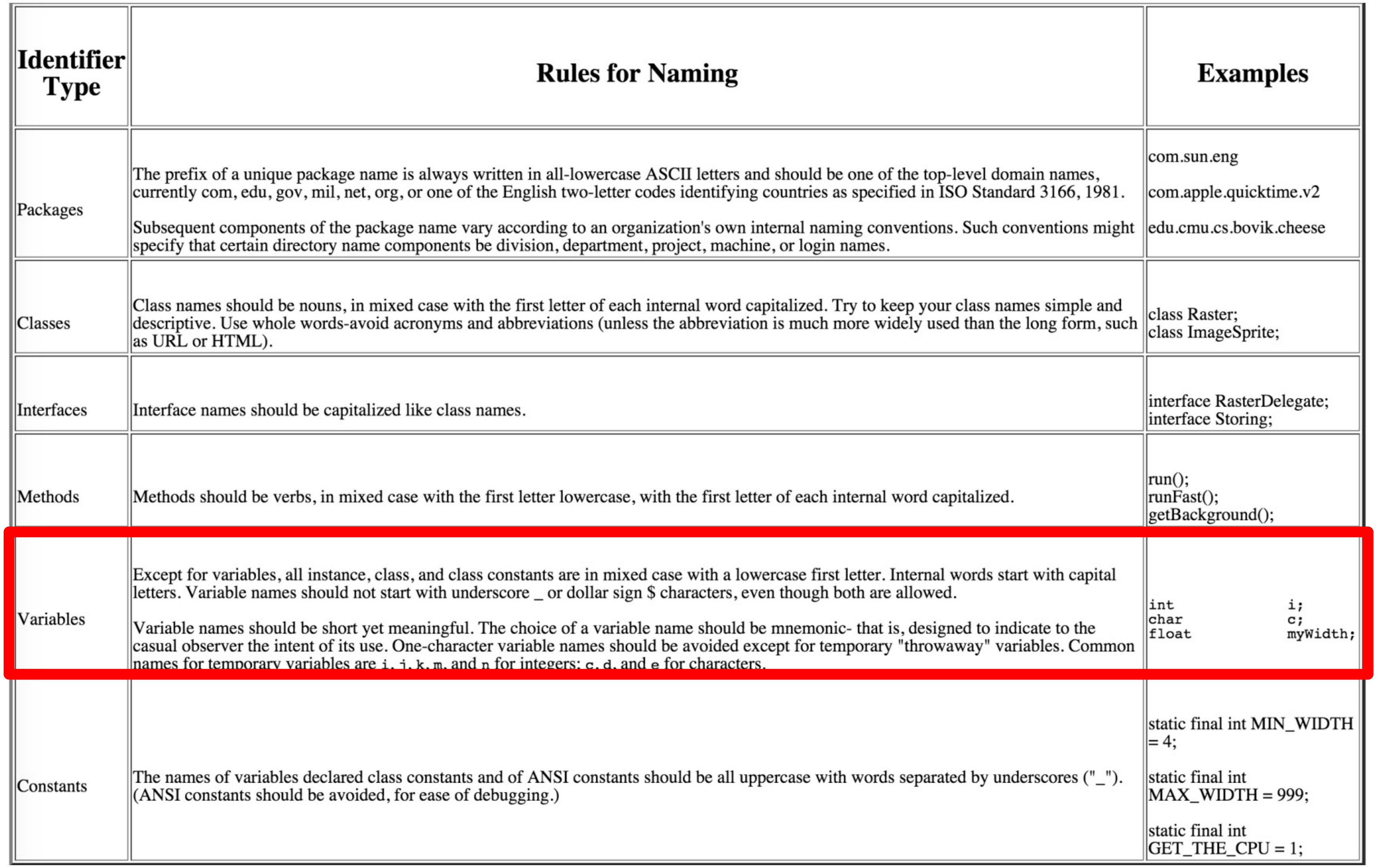
Variables

- A way of storing information inside the computer
- As its name suggests, it's content can be changed
- So, to define a variable we need to tell computer what type of information we need to store in it, and give it a name

Variables

- A way of storing information inside the computer
- As its name suggests, it's content can be changed
- So, to define a variable we need to tell computer what type of information we need to store in it, and give it a name
- There are lots of different types of data that can be used to define our variables, also known as **data types**

Variables



Identifier Type	Rules for Naming	Examples
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese

Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	class Raster;
		class ImageSprite;

Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate;
		interface Storing;

Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	run();
		runFast();
		getBackground();

Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.
	
	Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.
		`int i;`
		`char c;`
		`float myWidth;`

Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4;
		
		static final int MAX_WIDTH = 999;
		
		static final int GET_THE_CPU = 1;

Data types

Terminology

- **Data type** = a set of values (definition domain) and a set of operations defined on them

Data types

Terminology

- **Data type** = a set of values (definition domain) and a set of operations defined on them

- 8 **primitive** (**built-in**) data types in Java, mostly different types of **numbers**

Data types

Terminology

- **Data type** = a set of values (definition domain) and a set of operations defined on them
- 8 **primitive** (**built-in**) data types in Java, mostly different types of **numbers**
- OOP is centered around the idea of creating **our own data types** out of existing ones (we'll see later)

Data types

Terminology

```
int a, b, c;
a = 5;
b = 6;
c = a + b;
int d = 0;
```

The first statement declares 3 variables with the identifiers **a, b,** and **c** to be of type **int**.

Data types

Terminology

```
int a, b, c;
a = 5;
b = 6;
c = a + b;
int d = 0;
```

The next 2 assignment statements change the values of the variables using the **literals** 5 and 6.

Data types

Terminology

```
int a, b, c;
a = 5;
b = 6;
c = a + b;
int d = 0;
String foo = " Something important";
System.out.println(foo);
```

The last 2 statements assigns c the value of the expression a + b, and define and initialize in the same time variable d

Primitives

Integer numbers

- **byte:** range -2^7 and 2^7-1 (8 bits = 1 byte)
- **short:** range -2^{15} and $2^{15}-1$ (16 bits = 2 bytes)
- **char:** range 0 to 65535 (16 bits = 2 bytes)
  char myChar = ‘a’;
- **int:** range -2^{31} and $2^{31}-1$ (32 bits = 4 bytes)
- **long:** range -2^{63} and $2^{63}-1$ (64 bits = 8 bytes)

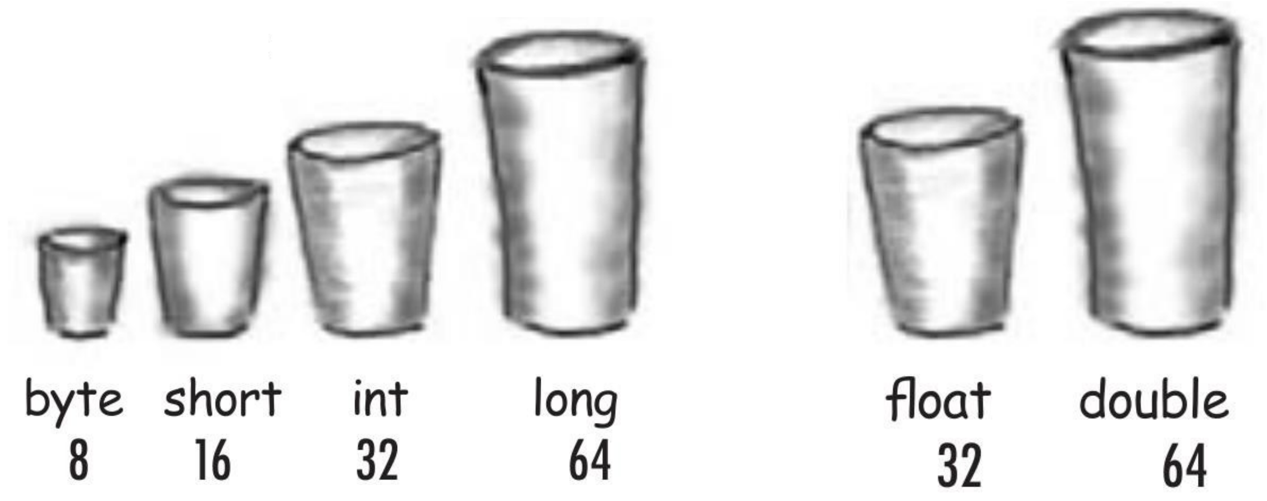
Primitives

Real numbers

- **float:** range (32 bits = 4 bytes)

- **double**: range (64 bits = 8 bytes)

Primitives



Primitives

Booleans

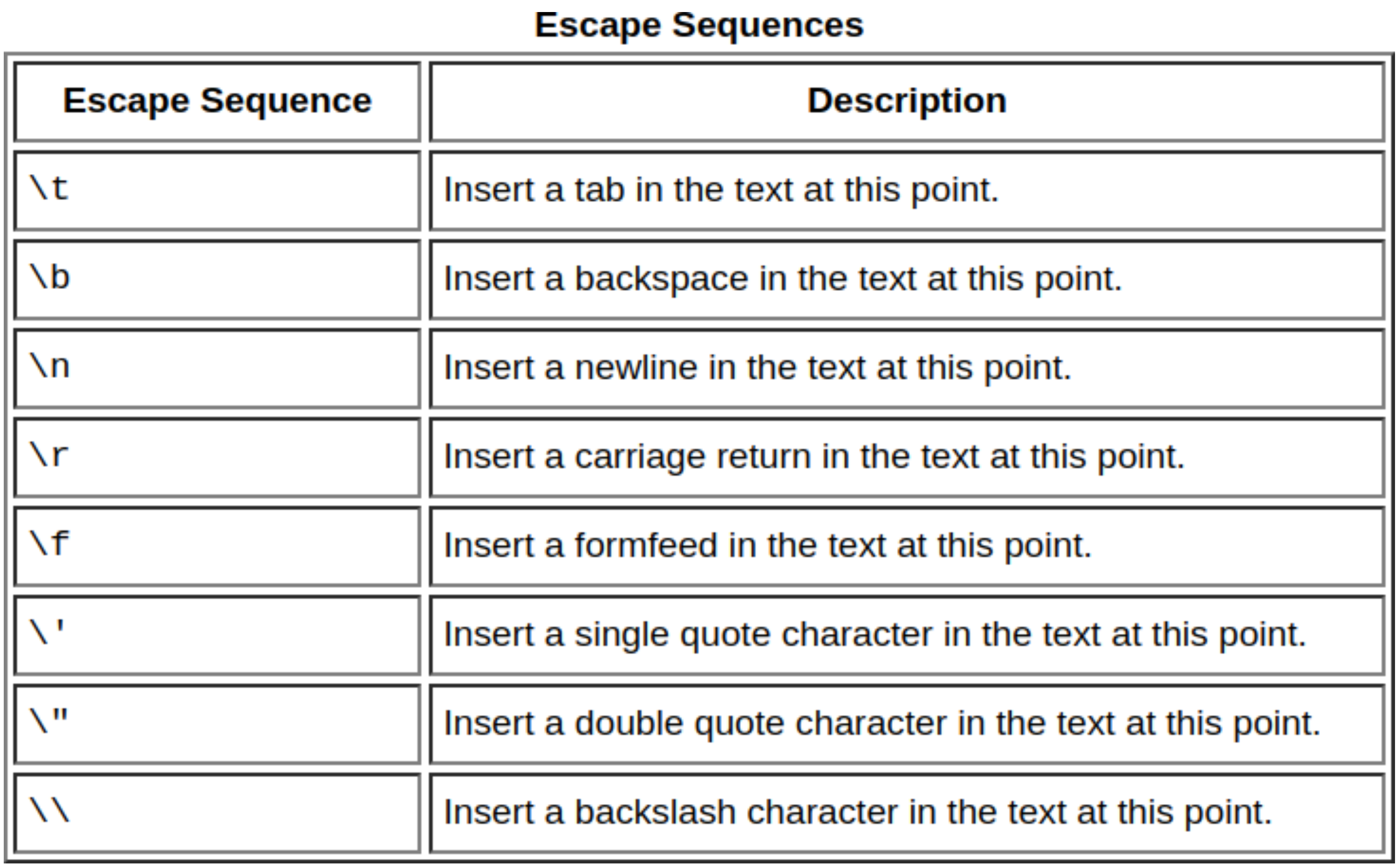
- **boolean:** only values **true** and **false**

Unicode

- Character encoding: the process of assigning numbers to graphical characters by which each letter, digit, or symbol is assigned a **unique** numeric value that applies across different platforms and programms

- Is an international encoding standard, maintained by the Unicode Consortium, used for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems

Escape sequences

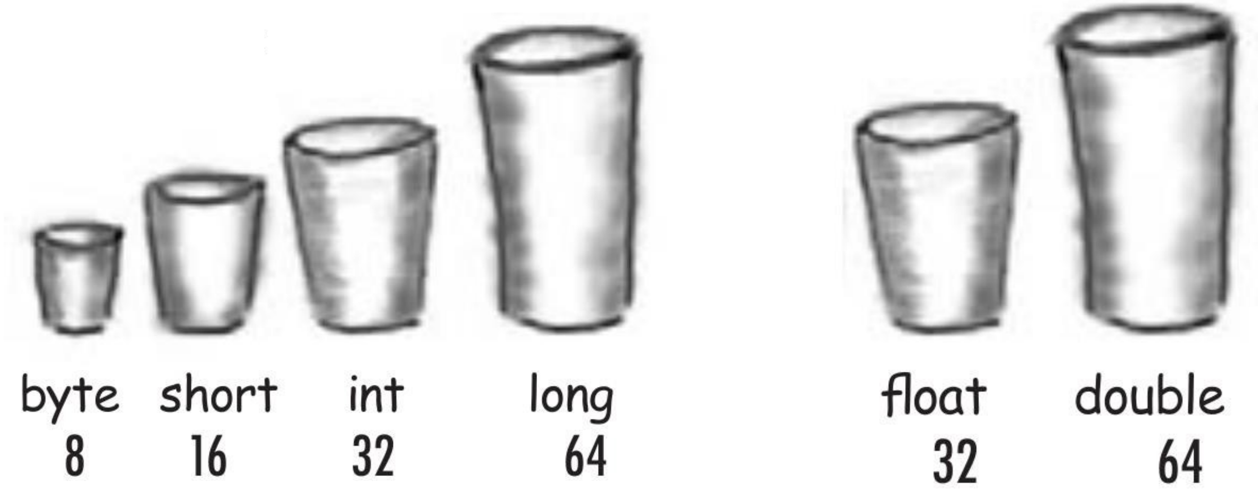


Escape Sequences

Escape Sequence	Description
`\t`	Insert a tab in the text at this point.
`\b`	Insert a backspace in the text at this point.
`\n`	Insert a newline in the text at this point.
`\r`	Insert a carriage return in the text at this point.
`\f`	Insert a formfeed in the text at this point.
`\'`	Insert a single quote character in the text at this point.
`\"`	Insert a double quote character in the text at this point.
`\\`	Insert a backslash character in the text at this point.

Casting in Java

- Converting a number from a type to another type



- NOTE: parsing is not casting

Conversion by casting

```
int a = 2;        // a = 2
double a = 2;     // a = 2.0 (Implicit)

int a = 18.7;          // ERROR
int a = (int)18.7;          // a = 18

double a = 2/3;        // a = 0.0
double a = (double)2/3;   // a = 0.6666…
```

String

- A String is a sequence of characters. (actually is backed by a char array)
- The max size of a String is Integer.MAX_VALUE
- Is not a primitive, it’s a Java class, contined in the JDK library, so a string will be an object

Operators

Type	
Arithmetic	+, -, /, *, % --, ++
Relational	<, >, >=, <=, ==
Bitwise	&, |, ^, ~, << , >>, >>
Logical	&& , ||, !
Assignment	=, +=, -=, *=, /=, %=
Misc	ternary (? :) instanceof

Order of Operations

Follows standard math rules + few more

Level	Operator	Description	Associativity
16	() [] .	parentheses array access member access	left-to-right
15	++ --	unary post-increment unary post-decrement	left-to-right
14	+ - ! ~ ++ --	unary plus unary minus unary logical NOT unary bitwise NOT unary pre-increment unary pre-decrement	right-to-left
13	() new	cast object creation	right-to-left
12	* / %	multiplicative	left-to-right
11	+ - +	additive string concatenation	left-to-right
10	<< >> >>>	shift	left-to-right
9	< <= > >= instanceof	relational	left-to-right
8	== !=	equality	left-to-right
7	&	bitwise AND	left-to-right
6	^	bitwise XOR	left-to-right
5	|	bitwise OR	left-to-right
4	&&	logical AND	left-to-right
3	||	logical OR	left-to-right
2	?:	ternary	right-to-left
1	= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	right-to-left
0	->	lambda expression arrow	right-to-left

Level	Operator	Description	Associativity
16	()	parentheses	left-to-right
	[]	array access	
	.	member access	
15	++	unary post-increment	left-to-right
	--	unary post-decrement	
14	+	unary plus	right-to-left
	-	unary minus	
	!	unary logical NOT	
	~	unary bitwise NOT	
	++	unary pre-increment	
	--	unary pre-decrement	
13	()	cast	right-to-left
	new	object creation	
12	* / %	multiplicative	left-to-right
11	+ -	additive	left-to-right
	+	string concatenation	
10	<< >>	shift	left-to-right
	>>>		
9	< <=	relational	left-to-right
	> >=		
	instanceof		
8	==	equality	left-to-right
	!=		
7	&	bitwise AND	left-to-right
6	^	bitwise XOR	left-to-right
5	|	bitwise OR	left-to-right
4	&&	logical AND	left-to-right
3	||	logical OR	left-to-right
2	?:	ternary	right-to-left
1	= += -=	assignment	right-to-left
	*= /= %=		
	&= ^= |=		
	<<= >>= >>>=		
0	->	lambda expression arrow	right-to-left

Operators

```
class DoMath {
    public static void main(String[] args) {
        double score = 1.0 + 2.0 * 3.0;
        System.out.println(score);
        score = score / 2.0;  →
        score \= 2.0;
        System.out.println(score);
        score = (1.0 + 2.0) * 3.0;
        System.out.println(score);
    }
}
```

Operators

```
class DoMath {
    public static void main(String[] args) {
        double a = 5.0/2.0; // a = 2.5
        int b = 4/2; // b = 2
        int c = 5/2; // c = 2
        double d = 5/2.0; // d = 2.0/2.5
    }
}
```

String Concatenation (+)

```
String text = "hello" + " world";
text = text + " number " + (5 + 5);
// text = "hello world number 10"
```

Control flow statements

if-else statement

```
if (<boolean expression>) {
    // statements;
} else {
    // statements;
}
```

```
if (<boolean expression>) {
    // statements;
} else if (<boolean expression>){
    // statements;
} else {
   // statements;
}
```

Control flow statements

```
public static void main(String[] args) {
        int x = 6;
        if (x > 5) {
            System.out.println(x + " is > 5");
        }
}
```

Control flow statements

```
public static void main(String[] args) {
        int x = 6;
        if (x > 5) {
            System.out.println(x + " is > 5");
        }
        if (x > 5 && x < 10) {
            System.out.println(x + " is between 5 and 10");
        }


}
```

Control flow statements

while statement

```
while(<condition>)                {
    // statements;
}
```

```
int i = 0;
while (i < 3) {
  System.out.println("Rule #" + i);
  i = i + 1; // i++;/ i+=1;
}
```

Control flow statements

for statement

simple for

```
for (<initialization>; < condition >; < update >;) {
    // statements;
}
```

foreach

```
for (DataType varName: array | iterable collection){
    // statements;
}
```

```
for (int i = 0; i < 3; i = i + 1) {
  System.out.println("Rule #" + i);
}  // Note:   i = i+1 may be replaced by ??
```

Control flow statements

switch statement

```
switch (expression) {
    case value1:
      // do something
      break; // optional

    case value2:
      // do something else
      break; // optional
    ...
   default: // optional
      // do something if value is none of the cases above
}
```

Limitations:

- expression can be one of types: boolean, integer type, string, enum.
- Duplicate case values are not allowed.
- Case values must be of the same type as the variable in the switch.
- The value for a case must be a constant or a literal
- break and default may be omitted

break and continue

```
int[ ] numbers = { 10, 20, 30, 40, 50 };
 int sum = 0;
for (int x : numbers) {

if (x % 15 == 0) {
        continue;
    }

    sum += x;

    if (sum > 100) {
        break;
    }
    System.out.println(x);
}

System.out.print("sum = " + sum);
```

Arrays

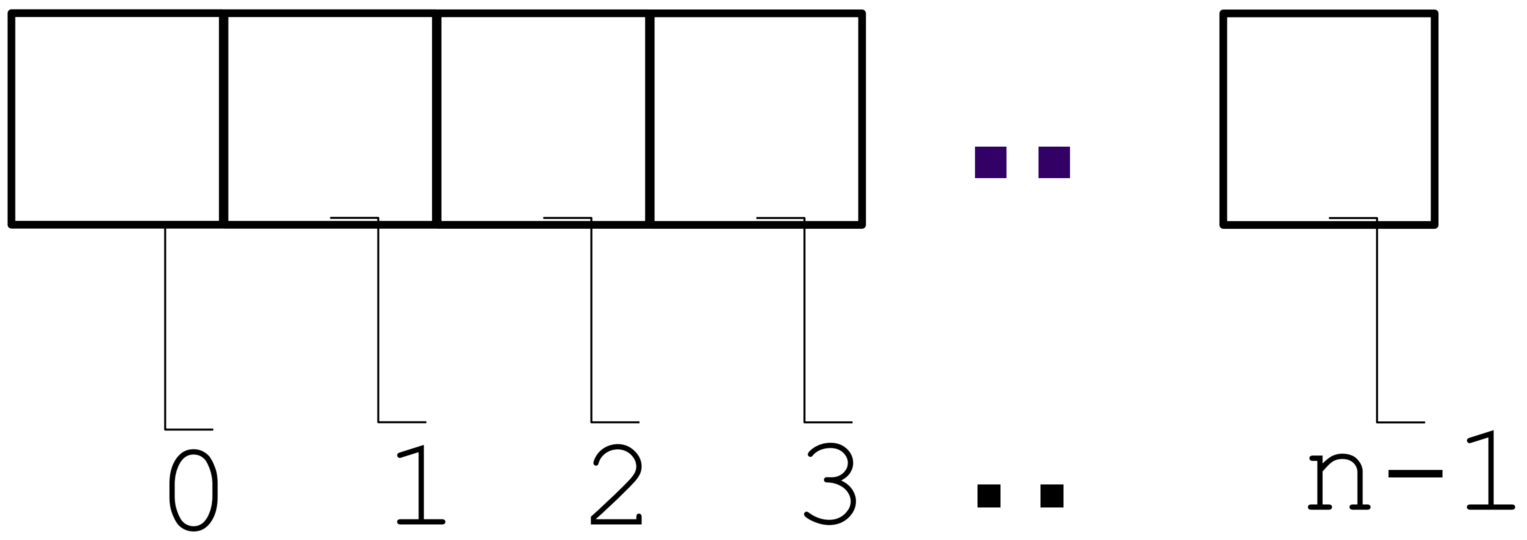
An array is an indexed list of values.

You can make an array of any type

- int, double, String, etc..

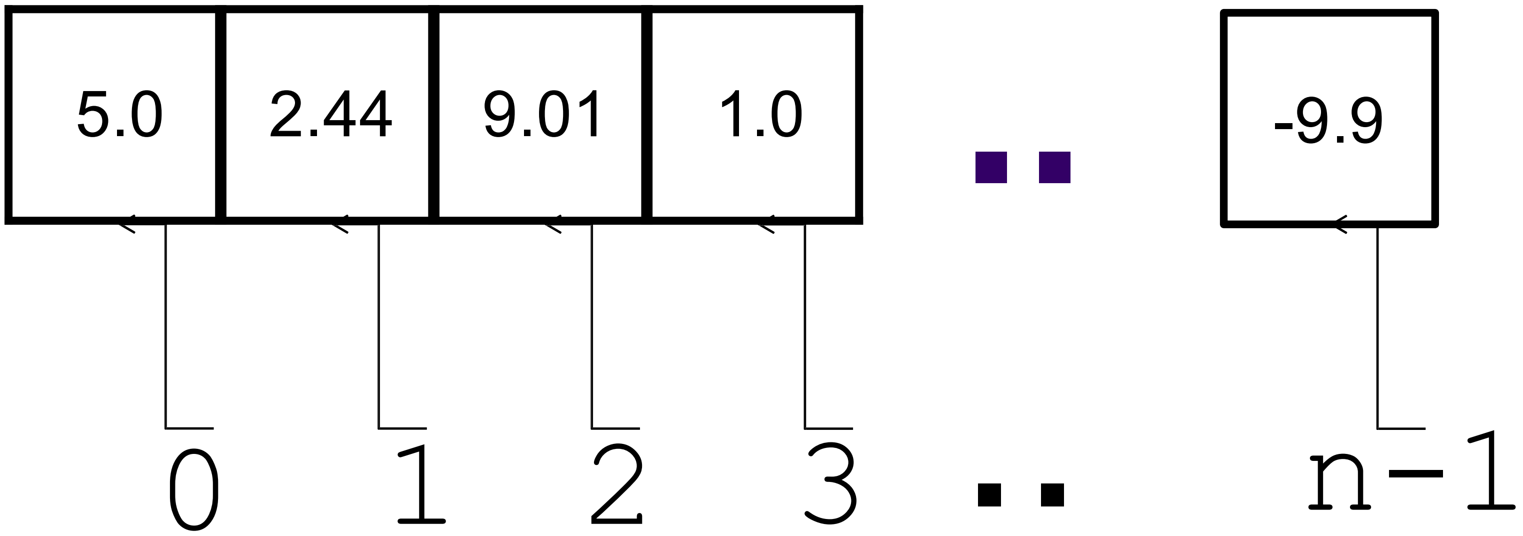
All elements of an array must have the same type.

Arrays



Arrays

Example: double [ ]



Arrays

The index starts at zero and ends at length-1.

Example:

```
int[] values = new int[5];
values[0] = 12; // CORRECT
values[4] = 12; // CORRECT
values[5] = 12; // WRONG!! compiles but
                // throws an Exception
                // at run-time
```

Arrays

An array is defined using TYPE`[]`.

Arrays are just another type.

```
int[]   values;    // array of int

int[][] values;    // int[] is a type
```

Arrays

To create an array of a given size, use the operator `new` :

```
    int[] values = new int[5];
```

or you may use a variable to specify the size:

```
    int size = 12;
    int[] values = new int[size];
```

Arrays

Curly braces can be used to initialize an array.

It can ONLY be used when you declare the variable.

```
int[] values  = { 12, 24, -23, 47 };
int[] values  = new int[]{ 12, 24,
                -23, 47 };
int[] values  = new int[30];
```

Summary

- Variables
- Data types, primitives, bounds, Unicode, casting
- Operators: pre/postincrement, short-circuiting, shothand operators
- Control flow statements, branching keywords
- Arrays

Summary

1. Consider the following code snippet.

```
arrayOfInts[j] > arrayOfInts[j+1]
```

2. Which operators does the code contain?

3. Consider the following code snippet.

```
int i = 10;
int n = i++%5;
```

   1. What are the values of i and n after the code is executed?
   2. What are the final values of i and n if instead of using the postfix increment operator (i++), you use the prefix version (++i))?

4. To invert the value of a boolean, which operator would you use?

5. Which operator is used to compare two values, = or == ?

Questions

Bibliography

- https://docs.oracle.com/javase/tutorial/java/concepts/
- **Thinking in Java 4th Edition**, by Bruce Eckel
- http://beginnersbook.com/2013/04/oops-concepts/
- https://introcs.cs.princeton.edu/java/home/
- https://ocw.mit.edu/