Primitive Data Types, Expressions & Operations, Casting, Variables

CSCI 161 – Introduction to Programming I

Professor Thomas Rogers

Overview

- Reading: Chapter 2 - Primitive Data and Definite Loops

- Topics:
  - Primitive Data Types
  - Expressions & Operations
  - Casting
  - Variables

Primitive Data Types

- Primitive Data Types (aka Data Types) - A name for a category of data values that are all related, as in type *int* in Java which is used to represent integer values.
- There are many data types in Java.
- A programmer uses different data types for different *kinds* of data.

Primitive Data Types (continued)

- Following are the data types in Java:

Category	Types	Size (bits)	Minimum Value	Maximum Value	Precision	Example
Integer	`byte`	8	-128	127	From +127 to -128	`byte b = 65;`
	`char`	16	0	2^{16}-1	All Unicode characters	`char c = 'A'; char c = 65;`
	`short`	16	-2^{15}	2^{15}-1	From +32,767 to -32,768	`short s = 65;`
	`int`	32	-2^{31}	2^{31}-1	From +2,147,483,647 to -2,147,483,648	`int i = 65;`
	`long`	64	-2^{63}	2^{63}-1	From +9,223,372,036,854,775,807 to -9,223,372,036,854,775,808	`long l = 65L;`
Floating-point	`float`	32	2^{-149}	$(2-2^{-23})*2^{127}$	From 3.402,823,5 E+38 to 1.4 E-45	`float f = 65f;`
	`double`	64	2^{-1074}	$(2-2^{-52})*2^{1023}$	From 1.797,693,134,862,315,7 E+308 to 4.9 E-324	`double d = 65.55;`
Other	`boolean`	1	--	--	false, true	`boolean b = true;`
	`void`	--	--	--	--	--

Expressions & Operations

- **Expression** - A simple value or set of operations that produces a value.
  - **Evaluation** - The process of obtaining the value of an expression.

```
System.out.println(42);    // A single literal value is an expression
System.out.println(2 + 2); // So is an operator acting on two literals
```

  - **Operator** - A special symbol (like + or *) that is used to indicate an operation to be performed on one or more values.
    - Arithmetic Operators:
    - **+** Addition
    - **-** Subtraction
    - * Multiplication
    - **/** Division
    - **%** Remainder or Mod

Expressions & Operations
(continued)

- Precedence
  - **Precedence** - The binding power of an operator, which determines how to group parts of an expression
  - **PEMDAS** - Acronym that gives order of precedence of operations. Easy to remember as "Please Excuse My Dear Aunt Sally". It stands for "Parentheses, Exponents, Multiplication and Division, and Addition and Subtraction" which list order of precedence.

Casting

- Mixing data types is allowed (for example: ints and doubles).
- If dividing two integers the result will be an integer.
- If dividing and either operand is of type double then the result will be a double:

```
23 / 4;       // 5
23.0 / 4;     // 5.75
23. / 4;      // 5.75
23 / 4.0;     // 5.75
23 / 4.;      // 5.75
23. / 4.;     // 5.75
23.0 / 4.0;   // 5.75
```

Casting (continued)

Cast - Sometimes you want to cast a double as an int, for example:

- You indicate a cast in your program by putting the name of the type you want to cast to in parentheses in front of the value (or variable) you want to cast.

```
23 / 4;           // 5
(int) 23.0 / 4;         // 5
23.0 / 4;         // 5.75
```

Variables

- **Variable** - A memory location with a name and a type that stores value.
- **Declaration** - A request to set aside a new variable with a given name and type.
- **Assignment** - Equals (=) operator. Used to give a value to a variable. Can be done when declaring or later.

```
<type> <name>; // Declaration without initialization/assignment

<type> <name> = <expression>; // Declaration with assignment
```

- Can only declare a variable once.
- Can assign (re-assign) a variable a value many times.

Variables (continued)

- Can declare multiple variables in single declaration, just separate variable names with commas.
- Can include multiple assignments in a single statement:

```
int x, y, z;
x = y = z = 2 * 5 + 4; // What values for x, y, z?
```

- **Increment/Decrement Operators** - These provide assignment and arithmetic operation all in one (a shorthand).

```
w += 1;  // same as: w = w + 1;
x -= 2;  // same as: x = x - 2;
y *= 3;  // same as: y = y * 3;
z /= 4;  // same as: z = z / 4;
```

- Java has an even more concise syntax for incrementing and decrementing by 1:

```
x++;   // Increment x by 1
y--;   // Decrement y by 1
```