PRIMITIVE VARIABLES

CS302 – Introduction to Programming
University of Wisconsin – Madison
Lecture 3

By Matthew Bernstein – matthewb@cs.wisc.edu

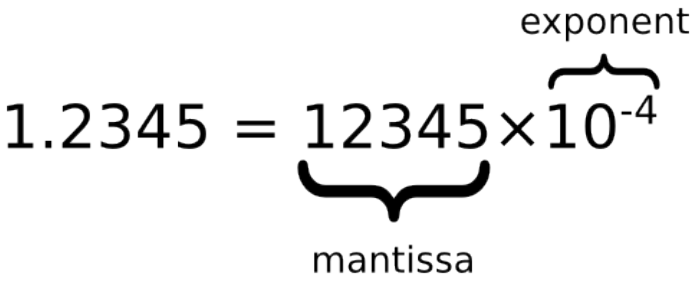
Variables

- A **variable** is a storage location in your computer
- Each variable has a **type**, **name**, **value**
  - Example:

    x = 4
- There are many different types of variables for storing different types of values.
- For example, a computer stores an integer differently than it stores any real number. Different types of variables reflect the underlying way the computer stores the value

Primitive Numeric Variables – Two Basic Types

- **Integer** – Whole numbers without a fractional part (-1,0,1,2…)
  - In the java programming language, an integer variable is called an **int**
- **Floating-Point Numbers** – Numbers that include a fractional part (6.2434)
  - Java has multiple variable types for holding floating-point numbers. The one we will use most commonly is called a **double**
  - Floating point numbers are stored in the computer as an integer and a location for the decimal place



Java Primitive Data Types

Type Name	Kind of Value	Memory Used	Range of Values
byte	Integer	1 byte	–128 to 127
short	Integer	2 bytes	–32,768 to 32,767
int	Integer	4 bytes	–2,147,483,648 to 2,147,483,647
long	Integer	8 bytes	–9,223,372,036,8547,75,808 to 9,223,372,036,854,775,807
float	Floating-point	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$
double	Floating-point	8 bytes	$\pm 1.79769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$
char	Single character (Unicode)	2 bytes	All Unicode values from 0 to 65,535
boolean		1 bit	True or false

For more information on Java Primitive Data types, visit:
http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Declaring Variables

- You must declare a variable in a declaration statement before using it!
- Declaration requires specifying the type of the variable and the name of the variable:

```
int x;          // Declare an int called “x”
double y;       // Declare a double called “y”
```

Assigning Values to Variables

- Assignment is done using the **assignment operator (=)**
- Example :

```
int x;     // Initialize an int called “x”
x = 4;    // Assign x a value of 4
```

- You can initialize a variable and assign it a value in the same statement. This is called **initializing** a variable:

```
int x = 4;
```

Initialization and Assignment Rules

- You can only declare a variable once
- After declaring the variable, you can reassign it as often as you like
- Example:

```
int x = 4;      // Declare once
x = 5;          // Reassign x to another int
x = 6;          // And again…
```

- A variable of a one type cannot be set to the value of a different type (for the most part)

Variable Naming Conventions

- Variable names should describe their function
- Names should be short, yet descriptive
- Bad practice to use single letters for variables

- CamelCase:

  playerAge
  numApples

- Underscore:

  player_age
  num_apples

- Pick one naming convention and stay consistent!

Reserved Words

- There are certain words that have special meaning in the Java programming language. These are called **reserved words**
- You cannot name a variable with a reserved word. The compiler will see the reserved word and treat it accordingly rather than understand that you meant for that word to be a variable name.
- Reserved words are highlighted purple in Eclipse editor
- Example reserved words:

class

public

static

Common Issues With Numeric Variables

- A computer has limited memory. Thus, it cannot possibly store every decimal of an irrational number. Floating point numbers only have a certain decimal precision. Thus, you may experience **rounding errors** when dealing with floating point numbers.

- Numeric variables have a limited range of the numbers they can store. For example, an int variable can only store numbers between -2,147,483,648 and 2,147,483,647

- How do we get around these limitations?
  - Use an object that allows us to deal with big numbers (i.e. java.math.BigInteger)

Constant Variables

- When a variable is defined with the reserved word final, its value can never change.
- Variables defined this way are called **constants**
- Example:

```
final double BOTTLE_VOLUME = 2.0;
```

- You will get a compile time error if you write any statement that assigns a new value to BOTTLE_VOLUME
- Constants should be named with all capital letters to distinguish them from non-constant variables

Arithmetic

- Four arithmetic operators:
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
- The combination of variables, literals, operators, and methods is called an **expression**
- Follows standard order of operations. Exceptions are made explicit using parenthesis.
- Example:
  - a + b / 2
  - (a + b) / 2
  - a + (b / 2)

Increment & Decrement Operators

- In programming you will commonly have to increment or decrement a numeric variable by 1
- Increment and decrement operators provide an easy way to do this:

- Both of these statements do the same thing:

```
counter = counter + 1;
counter++;
```

- Similarly:

```
counter = counter – 1;
counter--;
```

Dividing with Floating-point Numbers

- Division works as we would expect provided that at least one of the numbers is a floating-point number
- For example all of the following statements return the number 1.75:

```
7.0 / 4.0;
7 / 4.0;
7.0 / 4;
```

Dividing ints

- If both numbers are integers than the result of the division is always an integer, with the remainder discarded.
- This is often a common source of error. Always know what variables you are dealing with.
- The following example results in 1 (NOT 1.75):

7 / 4

Modulus Operator (%)

- Also called “Modulo” or “Mod”
- This operator returns the remainder when dividing two integers.
- The following example results in 3:

```
7 % 4
```

7 divided by 4 is 1 with a remainder of 3

Find Dollars and Cents

- Let's say we have a variable pennies. We want to know how many dollars and extra cents we have. How do we do this?

```
public class Main
{
        public static void main(String[] args)
        {
                int pennies = 2347;  // Number of pennies

                // …WRITE SOLUTION HERE…
        }
}
```

Solution

```
public class Main
{
        public static void main(String[] args)
        {

                // Initialize variables
                final int PENNIES_PER_DOLLAR = 100;
                int pennies = 2347;  // Number of pennies
                int dollars;
                int cents;

                // Compute and output solution
                dollars = pennies / PENNIES_PER_DOLLAR;
                cents = pennies % PENNIES_PER_DOLLAR;

                System.out.println(dollars + " dollars and " + cents + " cents");
        }
}
```

Powers and Roots

- There are no operators for powers or roots. You have to use the operators available.
- Thankfully performing these operations has already been implemented for us. We just need to use the code in a **library** of code called java.lang.Math
- How do we code this?

$$b \times \left(1 + \frac{r}{100}\right)^{n}$$

- Solution:

```
b * Math.pow(1 + r / 100, n);
```

Other Mathematical Methods

- Examples:
  - Math.sqr(x)
  - Math.pow(x,y)
  - Math.sin(x)

- Consult:

http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

Converting Integers to Floating-Point Numbers

- This is easy and allowed. Think about it…a floating point number is simply an integer (the mantissa) and a decimal location
- Example:

```
int x = 9;
double y = x;
```

- Java just takes the integer and assigns the correct decimal location

Converting Floating-Point Numbers to Integers

- This is not allowed because it is dangerous.
- This is NOT allowed:

```
double x = 4.5;
int y = x;
```

- Why?
  - The fractional component is lost
  - The magnitude of the floating-point number might be too large to fit into an integer type variable

So how do we make this conversion?

- Answer: The **cast** operator
- Example:

```
double x = 4.5;
int y = (int) x;      // First we cast x to an int type
```

- You are essentially “overriding” the compiler and demanding it to treat the x as an integer. If x is too large to store as an int, then y will be assigned to the largest possible int (2,147,483,647).
- BE CAREFUL WHEN CASTING

Using other code in your code

- The **import** statement
- An import statement allows you to “import” code from other locations and run it in your program
- The import statement goes at the top of your file before your class declaration
- Example:

```
import java.util.Scanner;
```

- Now we can use the code in Scanner

Using Scanner to get input from user

- We use the code in Scanner to prompt data input by the user.
- Example:

```
import java.lang.Scanner

public class Main  {
        public static void main(String[] args)  {

                // Create a Scanner object
                Scanner in = new Scanner(System.in);

                // Prompt user for for an integer
                System.out.println("Please input an integer:");
                int userInteger = in.nextInt();

                // Output the integer
                System.out .println( userInteger );
        }
}
```

Using the Scanner in our Pennies To Dollar Example

-- See Demo --

Programming Exercise

- For next class create a program that will ask the user to input a number corresponding to a temperature in degrees Fahrenheit. Convert this temperature to Celsius and output this value to the user.
- Example:

Please input a temperature in degrees F: **32**

0.0

Output

Input

Cool CS Link of the Day

- TIME Magazine article, “2045 The Year Man Becomes Immortal”:
- http://content.time.com/time/magazine/article/0,9171,2048299,00.html

PHOTO-ILLUSTRATION BY PHILLIP TOLEDANO FOR TIME, PROP