Quick review: parameter passing

- Names of variables are not important, only the number and types
- Ditto for return types: only the type is important

Java has 2 kinds of data types

- **Primitive data types** already have memory allocated for them, so we don't have to create them
  - e.g., int, double
  - String acts like one most of the time
- **Reference data types** do not have memory allocated automatically
  - when we create a new object of a class, we are allocating memory to store it
  - e.g., Canvas, Rectangle

Numerical data types in Java

- **byte** integer 8 bits [-128, 127]
- **short** integer 16 bits [-32768, -32767]
- **int** integer 32 bits [-2.1x10^9, 2.1x10^9]
- **long** integer 64 bits [-9.2x10^18, 9.2x10^18]
- **float** real 32 bits [-3.4x10^38, 3.4x10^38]
- **double** real 64 bits [-1.7x10^308, 1.7x10^308]
What happens when we initialize a primitive data type variable?

```java
int a = 265;
```

- Set aside 4 bytes of memory in the next available memory slot in the program
- Copy the value 256 into this spot
- Label this spot “a”

What happens when we create a new object?

```java
Canvas c1 = new Canvas(“My drawing”);
```

- Set aside the next slot of memory in the program
- Create a new Canvas object somewhere else in memory. Get this spot’s address.
- Copy the address into the slot
- Label this slot “c1”

Why the difference?

- Primitive data types, like `int`, always take up the same amount of memory
  - each `int` is 4 bytes
- Reference data types, like `Canvas`, take up varying amounts of memory
  - the size of a `Canvas` object depends on many factors: the values of the data fields, the size of the frame, the number of objects inside the frame, ...
- The Java compiler likes consistency
  - so, it sets aside the same amount of memory for all reference data types, and uses these memory slots to store the addresses of the actual objects
  - it then creates the objects wherever there’s room, and just “points” to them (stores their addresses)

Naming conventions in Java

- Names can only contain letters, digits, and the `_` character
  - must start with a letter
  - case-sensitive
  - examples:
    - good: max_grade, isDigit, ChangeMaker
    - bad: *myvar, max grade, seti@home, 2much_work
- Good practice: names should be descriptive
  - good: Canvas, drawRectangle, numSalesReps;
  - bad: JavaIsCool, monkeypants, int1
Naming conventions (cont.)

- Variables and methods:
  - start with a lowercase letter
  - “words” should be separated by an ‘_’ or denoted by capitalizing the second (preferred)
- Classes
  - start with a capital letter
  - same rules for separating “words” as variables and methods

Variable and class names should be “nouns”
- example: grade1, acctNum, CheckingAccount

Method names should be “verbs”
- example: calculateInterest(), drawOval(), init()

Arithmetic expressions in Java

- Binary operators: + - * / %
  - * is multiplication
  - / is division
    - dividing 2 integers also yields an integer:
      \[
      3/2 = 1 \quad \text{but} \quad 3/2.0 = 1.5 \quad \text{and} \quad 3.0/2 = 1.5
      \]
  - % is modulo division (the remainder of dividing two numbers)
    - 10 modulo 5 is 0
    - 10 modulo 7 is 3
- Unary operators: + -

Arithmetic precedence rules

- Determines the order of operations in an arithmetic expression
- Rules:
  - parentheses first
  - then unary (+/-)
  - then multiplication, division, modulo
  - then addition, subtraction
Examples

- $15 + 4 \times 6 / 3 - 9$
- $3 + 5.0/3$
- $(18 - 4) / 6$
- $(6 \times 3 + 2) \% (5 + 3)$

Examples

- $15 + 4 \times 6 / 3 - 9 = 14$
- $3 + 5.0 / 3 = 4.6666667$
- $(18 - 4) / 6 = 2$
- $(6 \times 3 + 2) \% (5 + 3) = 2$

More advanced arithmetic operations

- The Math class contains common math functions:
  - Trig functions
    - $\sin(a)$, $\cos(b)$, $\tan(a+b)$, $\acos(a)$,...
  - Logarithms/exponential
    - $\exp(-a) = e^{-a}$
    - $\log(b) = \ln(b)$
  - Powers
    - $\text{pow}(a,b) = a^b$
  - Rounding
    - $\text{ceil}(a)$ rounds up: $\text{ceil}(10.6) = 11.0$
    - $\text{floor}(a)$ rounds down: $\text{floor}(10.6) = 10.0$
- See chart on page 114-115 of Wu

Constants (review)

- A data value that does not change during the program's execution
- Use the keyword `final` to indicate a constant
- Declare constants with the rest of the variables, at the top of the program (after the class declaration)
- Typically, names are all capital letters
- Examples
  
  final int MAX_GRADE = 100;
  final double MIN_BALANCE = 100.00;
Predefined constants

- Java defines some constants for us already
  - \( \pi, e \)
- Can access these through the `Math` class (class variables):
  
  ```java
  double circleArea = Math.PI * r * r;
  double number = 2 * Math.E;
  ```