Introduction and Java Basics

Expectations:

• You have programmed in some language (does not have to be Java).

• You know:
  • How to write arithmetic expressions.
  • Use selection: if statements.
  • Use loops:
    • for loops and while loops.
  • Use arrays: one-dimensional and two-dimensional.
  • Use methods, functions, or procedures.

Getting started:

• We will spend about four weeks covering the above topics in Java.

• Assignments once per week.
  • First assignment will be posted by Friday this week.
  • It will be due on Tuesday, January 22nd.

• Each will consist of one or more Java programs.

• Exam #1 will cover these basic Java topics.
  • (Tentatively) scheduled for Wednesday, February 13th.
  • Date will be set at least one week in advance.
What then:

- The remainder of the course will be an accelerated version of CSc 127B.
- Topics to be covered will include:
  - Linked structures: stacks, queues, lists.
  - Binary trees.
  - Recursion.
  - Hashing.
  - Sorting.
- Program design will be a recurring theme throughout the course.
- Assignments: weekly (exception: the two weeks of the exams):
  - Exam #2: (tentatively) Wednesday, April 3rd.
  - Final Exam: Friday, May 3rd (definite — set by UA final exam schedule).

Discussion Section:

- Meets once a week on either Wednesday or Thursday.
- Small size, about 25 in each.
- Led by a Section Leader (SL).
- Will provide a review, additional examples, and group exercises.
- Attendance is mandatory(?!).
  - There will be a grade for each week. Section grades make up 10% of the final course grade.
  - Sometimes it is sufficient to show up to get the grade.
  - Other times there will be an exercise in section that will be worth part of the grade that week.
  - There will be no make-ups for missed labs.
  - Three lab grades will be dropped.
Discussion Section (continued):

- First week (i.e., today or tomorrow):
  - If you are not able to make section, you **must** get in touch with your SL or with Patrick.
  - Students who do not attend the first lab meeting will be administratively dropped from the course!

<table>
<thead>
<tr>
<th>Section</th>
<th>Section Time</th>
<th>Room</th>
<th>Section Leader and email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wed., 12-12:50 pm</td>
<td>GS-813</td>
<td>Craig Barber, craigb</td>
</tr>
<tr>
<td>2</td>
<td>Wed., 1-1:50 pm</td>
<td>GS-813</td>
<td>Jesse Bright, jbright</td>
</tr>
<tr>
<td>3</td>
<td>Wed., 4-4:50 pm</td>
<td>GS-813</td>
<td>Tyson Hoekstra, tyejae</td>
</tr>
<tr>
<td>5</td>
<td>Thurs., 10-10:50 am</td>
<td>GS-701</td>
<td>Kristle Schulz, kschulz</td>
</tr>
</tbody>
</table>

- Again, if you are not able to make section, you **must** get in touch with your SL or with Patrick.

**Java Basics:**

Reading: Appendix A

- Structure of a Java Application.
- Data Types, Variables, and Constants.
- Arithmetic: Operators and Expressions.
Structure of a Java Application:

• Every Java program contains at least one class.
  • The file name for a Java application is the name of the class with .java as the extension.
• Every Java program needs one main method.
• For now, we will write programs:
  • That contain exactly one class.
  • The main method will always be inside the one class.
• Example: ShellApplication class contained in a file named ShellApplication.java:
  ```java
  /* An application shell: 
   * Minimum needed to compile and run. 
   * Does not produce any output. 
   */
  public class ShellApplication 
  { 
    public static void main( String[] args ) 
    { 
    } // end of main method 
  } // end of class ShellApplication
  ```

Structure of a Java Application (continued):

Identifiers:

• Symbolic names.
  • Used to name classes, variables, methods.
  • From the example on the previous slide: ShellApplication and main are identifiers.
• Identifier Rules:
  • Must start with a “Java letter”
    • A-Z a-z _ $
    • Unicode letters
  • Followed by more Java letters and/or numbers. baseball Football Stalag17
    • No spaces, no hyphens, no other punctuation.
  • Can be arbitrarily long. sillyLongTakesForeverToTypeIdentifierForThisExample
  • Case sensitive!
    • The identifiers theRailRoad and therailroad are different.
  • Can not be keywords or reserved words.
    • See Appendix A for a list of these.
Structure of a Java Application (continued):

Statement:
- Performs one action. Examples:
  - Prints a message:
    ```java
    System.out.println("Hello out there!");
    ```
  - Computes a sum:
    ```java
    result = x + y;
    ```
  - Terminates with a semicolon.
  - Can span multiple lines.
    ```java
    result = xray + yoke + 17 +
    zebra - 42 +
    63 + alpha;
    ```
  - Style point: Use indentation to indicate the lines go together.

Structure of a Java Application (continued):

Block:
- Contains 0, 1, 2, or more statements.
- Begins and ends with curly braces: `{   }`
- Can be used anywhere a statement is allowed.
- Example:
  ```java
  System.out.println("What's up Doc?");
  {
    System.out.println("This print statement");
    System.out.println("and this one");
    System.out.println("plus this one are in a Block.");
  }
  ```
- The `main` method contains a block of code between its `{ and }`:
  ```java
  public static void main( String args[] )
  {
    System.out.println("This is inside main's block of code");
  } // main method ends
  ```
Structure of a Java Application (continued):

White Space:
- Space, tab, newline are white space characters.
- At least one white space character is required between a keyword and an identifier.
  - Example:
    ```java
    int myNumber; // correct, need white space between int and myNumber
    int myNumber; // error. Compiler does not recognize int as a keyword here
    int myNumber, myOtherNumber, myLastNumber; // correct
    ```
- Any amount of white space characters are permitted between identifiers, keywords, operators, and literals.

  *Style point: Readability:*
  - White space makes code easier to read.
  - Put white space around operators and operands and identifiers.
    ```java
    result = lima + juliet + oscar; // this one is easier to read!
    result = lima + juliet + oscar; // harder to read (and SL's will not like it!)
    ```
  - Put blank lines between logical sections of the program.

Comments:
- Explain the program to others and to yourself.
  - Why to yourself? Don't you know what you are doing?

- Two types of comments.
  - **Block comment**: The comment can span multiple lines.
    - Begins with /*
    - Ends with */
    ```java
    System.out.println("message here"); // This is a comment that continues here and finally ends here */
    ```
  - **Line comment**: The comment is no longer than one line.
    - Begins with //
    - Ends at the end of the line.
    ```java
    System.out.println("message here"); // A line comment
    ```
Structure of a Java Application (continued):

Comments (continued):
- **Style point**: Include a block comment at the beginning of each Java source file.
  - Identify yourself as the author of the program and your section number and Section Leader’s name.
  - Briefly describe what the program does.

/* Your first and last name here
 * Your section leader’s name here
 * Section: Your section number here
 * Short description of what your program does.
 * Description can occupy more than one line, as needed.
 */

Data Types, Variables, and Constants:
Reading: Section A.2.
- Declaring Variables.
- Primitive Data Types.
- Initial Values and Literals.
- String Literals and Escape Sequences.
- Constants.
Data Types, Variables, and Constants (continued):

- For all data, assign a name (an identifier) and a data type.

```java
int xray;
```

- The data type tells the compiler:
  - How much memory to set aside.
  - How the memory will store the value.
  - Types of operations that can be performed on the data.

- Compiler monitors the use of the data.
  - Java is a “strongly typed” language — very strict about how data can be used.

- Primitive data types in Java (you will use these a lot!):
  - `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`.

---

Data Types, Variables, and Constants (continued):

**Declaring Variables:**

- Variables hold one value at a time.
- That one value can change during execution.

**Syntax:**

```java
dataType identifier;
```

- Or

```java
dataType identifier, anotherIdentifier, andAnother, ...
```

**Style point:** Naming `convention` for variable names:

- First letter is lowercase.
- Embedded words begin with uppercase letter; called “camel casing”.

**Examples:**

```java
int xray, limeJuice, oneMonth;
float myBankBalance, profitAndLoss;
```

- A `convention` is not required by the language (the Java compiler will not complain).
- `Conventions` generally exist to make code more readable and understandable.
- You are expected to follow these `conventions` in the code you develop in this (and other) courses.
Data Types, Variables, and Constants (continued):

Declaring Variables:

- Variable names should be meaningful:
  - The name should tell us how the program will use the value.
  - Makes the logic of the program clearer.
- Do not skimp on characters.
  - Abbreviations can be confusing.
  - Avoid extremely long names.
- Avoid names similar to Java keywords.

Data Types, Variables, and Constants (continued):

Integer Types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
</tbody>
</table>

- Examples:
  ```java
  int testGrade;
  int numPlayers, highScore, diceRoll;
  short xCoordinate, yCoordinate;
  byte ageInYears;
  long worldPopulation;
  ```
Data Types, Variables, and Constants (continued):

Floating-Point Data Types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
<th>Minimum Positive Nonzero Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4</td>
<td>1.4E-45</td>
<td>3.4028235E+38</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>4.9E-324</td>
<td>1.7976931348623157E+308</td>
</tr>
</tbody>
</table>

- Use `double` when you need:
  - Larger range of values than provided by `float`.
  - Greater precision than provided by `float`.

- Examples:
  ```java
  float courseAverage;
  float battingAverage, sluggingPercentage;
  double distanceToAndromeda;
  ```

---

Character Data Type:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>2</td>
<td>character encoded as 0</td>
<td>character encoded as 65535</td>
</tr>
</tbody>
</table>

- Contains standard Latin alphabet, digits, punctuation.
- Supports many character sets (i.e., Greek, Cyrillic, Hebrew).
- Known as Unicode.

- Examples:
  ```java
  char myInitial;
  char firstLetter, nextLetter, lastLetter;
  char newline, tab, quoteMark;
  ```
Boolean Data Type:
- Only two values possible:
  - true
  - false
- Used for decision making or as “flag” variables.
- Examples:
  
  ```java
  boolean done;
  boolean programStarted, programFinished;
  boolean reachedLastLevel;
  boolean hasWizardPowers;
  ```

---

What can we do now?
- So far, we can declare variables (which creates memory space for them), but cannot do anything else with them.

```java
public class NotMuchYet {
    public static void main( String [] args ) {
        int xray, yoke, zebra;
        int someValue, numSingles;
        float battingAverage;
        double lightSpeed;
        System.out.println("What now?");
    } // end of method main
}
```
Data Types, Variables, and Constants (continued):

Assigning Values to Variables:

- Assignment operator =
  - Value on the right of the operator is assigned as new value of the variable on the left.
  - Value on the right can be:
    - a literal: text representing a specific value.
    - another variable.
    - an expression (which we will cover later).
- Syntax:
  ```
  dataType variableName = initialValue;
  ```
  Example:
  ```
  int maxStudents = 175;
  ```
  Or
  ```
  dataType variable1 = initialValue1, variable2 = initialValue2, ...
  ```
  Example:
  ```
  int bases = 4, players = 9, balls = 17, bats = 12, battingHelmets = 6;
  ```

Data Types, Variables, and Constants (continued):

Assigning Values to Variables (continued):

- Literals for integer types.
  - `int, short, byte`.
    - Optional initial sign (+ or -) followed by digits 0-9 in any combination.
  - `long`.
    - Optional initial sign (+ or -) followed by digits 0-9 in any combination, terminated with L or l.
    - Style point: Use the capital L, not the lowercase l. The lowercase l (ell) can be confused with 1 (one).
  - Note: there are no commas or periods in an integer literal.
  - Examples:
    ```
    short myAge = 62;
    int numSongs = +4406, nitrogenLiquid = -210;
    long ageOfUniverse = 15000000000L;
    // This next one will generate a compile error:
    short smallValue = 103467;  // value too big to go into a short
    ```
Data Types, Variables, and Constants (continued):

Assigning Values to Variables (continued):

- Literals for floating-point types.
  
  - `float`.
    - Optional initial sign (+ or -) followed by a floating-point number in fixed or scientific format, terminated by an `F` or `f`.

  - `double`.
    - Optional initial sign (+ or -) followed by a floating-point number in fixed or scientific format.

- Note: there are no commas and no periods in an integer literal.

- Examples:
  ```java
  float pi = 3.1415926536F;         // fixed format
  float small = 0.00000000456f;     // fixed format
  double lightSpeed = 2.99792E+8;   // scientific format
  float tinyValue = 1.837E-12F;     // value close to zero
  double negTinyValue = -1.837E-12; // negative value close to zero
  ```

Data Types, Variables, and Constants (continued):

Assigning Values to Variables (continued):

- Literals for `char` type.
  
  - `char`.
    - Any printable character enclosed in single quotes: `'Q'`
    - A decimal value in the range 0 - 65535.

  - Certain escape sequences:
    - `	` represents a tab, `\n` represents a newline.

  - Examples:
    ```java
    char myInitial = 'P';
    char aTab = '\t', percent = '\%';
    ```

- Literals for `boolean` type.
  
  - `boolean`.
    - Use the value `true` or `false`.

  - Examples:
    ```java
    boolean afternoon = true;
    boolean doneYet = false;
    ```
Data Types, Variables, and Constants (continued):

Assigning Values to Variables (continued):

```java
/* Variables Class */
public class Variables {

    public static void main( String [] args )
    {
        // This example shows how to declare and initialize variables
        int    hits = 98;
        int    atBats = 324;

        long   lightYear = 5879000000000L; // miles in one light year

        byte   ageInYears = 61;
        float  mortgageRate = .035F;       // 3.5% interest rate
        float  lightSpeed = 2.99792E8F;    // speed of light in meters/second
        double solarMass = 1.988435E+30;   // mass of our Sun in kilograms

        char finalGrade = 'A';
        boolean isEmpty = true;

        System.out.println( "The grade for the course is " + finalGrade );
        System.out.println( "Mortgage rate is " + mortgageRate );
        System.out.println( "Speed of light is " + lightSpeed );
        System.out.println( "Mass of our Sun is " + solarMass );
        System.out.println( "Hits is " + hits + " in " + atBats + " at bats" );
        System.out.println( "isEmpty is " + isEmpty );
    } // main
} // class Variables
```
Assigning Values of Other Variables:

- Syntax:
  ```java
dataType variable2 = variable1;
```
- Rules:
  - `variable1` needs to be defined earlier in the code.
  - `variable1` and `variable2` need to be compatible data types.
    - **Compatible data types**: the precision of `variable1` must be lower than or equal to that of `variable2`.

### Data Types, Variables, and Constants (continued):

**Assigning Values of Other Variables (continued):**

- Any type in the right column can be assigned to the type in the left column:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Compatible Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>byte</td>
</tr>
<tr>
<td>short</td>
<td>byte, short</td>
</tr>
<tr>
<td>int</td>
<td>byte, short, int, char</td>
</tr>
<tr>
<td>long</td>
<td>byte, short, int, long, char</td>
</tr>
<tr>
<td>float</td>
<td>float, byte, short, int, long, char</td>
</tr>
<tr>
<td>double</td>
<td>float, double, byte, short, int, long, char</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>char</td>
<td>char</td>
</tr>
</tbody>
</table>

- **Hints:**
  - Small items fit into larger items.
  - Integers fit into floats or doubles.
Data Types, Variables, and Constants (continued):

Assigning Values of Other Variables (continued):

- Example assignments:
  ```java
  short smallValue = 42;
  int largerItem = smallValue;
  long biggerStill = largerItem;
  long bigExample = 2787L;
  long aBigOne = smallValue;
  float myBankBalance = 273.42f;
  double largeBalance = myBankBalance;
  float littleValue = 2738.2F;
  double anotherBalance = biggerStill;
  ```

- What is the current value stored in:
  - aBigOne
  - largerItem
  - anotherBalance

Data Types, Variables, and Constants (continued):

Assigning Values of Other Variables (continued):

- Beware! What is wrong with the following?
  ```java
  int numBooks = 173;
  short howMany = numBooks;
  short againBooks = 173;
  
  double milesToGo = 435.7F;
  float howFar = milesToGo;
  ```
Assigning Values of Other Variables (continued):

- You can also assign the contents of one variable to another variable after both have been declared.

```java
int numBooks = 173042;
short howMany = 97;
System.out.println("numBooks is " + numBooks);
numBooks = howMany;
System.out.println("numBooks is " + numBooks);

float milesToGo = 435.7F;
System.out.println("milesToGo is " + milesToGo);
double away;
away = milesToGo;
System.out.println("away is " + away);
away = howMany;
System.out.println("away is " + away);
```

String Literals (continued):

- *String* is actually a class, not a basic data type (and, not a reserved word).
  - *String* variables are objects.
  - More on classes and objects a bit later.
- *String literal*: text contained within double quotes.
- Examples of String literals:
  - "The answer is 
  - "Four-score and seven years ago"
  - "Gallia est omnis divisa in partes tres"
  - "To boldly go where no man has gone before."
  - "Please Sir, can I have some more?"
String Concatenation:

- Can combine two String literals together.
- Can combine one String literal with another data type for printing.

```java
String howdy = "Howdy";
String buddy = "Pardner!";
String greeting = howdy + ' ' + buddy;
System.out.println( greeting );
Prints:
   Howdy Pardner!

String greeting3 = howdy + buddy; // when printed will give: HowdyPardner!
System.out.println( greeting3 );
Prints:
   HowdyPardner!

String greeting2 = howdy + "         " + buddy;
System.out.println( greeting2 );
Prints:
   Howdy         Pardner!
```

Data Types, Variables, and Constants (continued):

String Concatenation (continued):

- Common error (!!!):

  ```java
  System.out.println("Four-score and seven years ago,
                      Our Fathers brought forth on this continent");
  ```

  Generates these errors:
  - unclosed string literal
  - ')' expected
  - String literals must start and end on the same line.

- Break long strings into shorter strings and use the concatenation operator:

  ```java
  System.out.println("Four-score and seven years ago," +
                      "Our Fathers brought forth on this continent");
  ```

- Or, use two separate print statements:

  ```java
  System.out.println("Four-score and seven years ago,");
  System.out.println("Our Fathers brought forth on this continent");
  ```
Data Types, Variables, and Constants (continued):

Escape Sequences in String literals:

<table>
<thead>
<tr>
<th>Character</th>
<th>Escape Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newline</td>
<td>\n</td>
</tr>
<tr>
<td>Tab</td>
<td>\t</td>
</tr>
<tr>
<td>Double quotes</td>
<td>&quot;</td>
</tr>
<tr>
<td>Single quote</td>
<td>'</td>
</tr>
<tr>
<td>Backslash</td>
<td>\</td>
</tr>
<tr>
<td>Backspace</td>
<td>\b</td>
</tr>
<tr>
<td>Carriage return</td>
<td>\r</td>
</tr>
<tr>
<td>Form feed</td>
<td>\f</td>
</tr>
</tbody>
</table>

- Use `\n` when you need a newline or a blank line.
- We will *not* use `\r` or `\f`.

```java
/* Literals Class */
public class Literals {
    public static void main( String[] args )
    {
        System.out.println( "How to get multiple\nlines from one statement\n" );

        System.out.println( "When you want to indent a line" );
        System.out.println( "\tUse a tab" );
        System.out.println( "\tFor each indented line" );

        System.out.println( "It started: \"Four-score and seven years ago\"\" );
        System.out.println( "He said \"NO\\" and meant it!\" );
    } // end of method main
} // end of class Literals
```
Data Types, Variables, and Constants (continued):

Declare a Variable exactly Once:

```java
double myTwoCents;
double myTwoCents = 0.02;
```

- Generates this compile error:
  ```java
  myTwoCents is already defined in main(java.lang.String[])
  ```

Once a variable is declared, its data type cannot be changed:

```java
double cashInMyPocket = 5.0;
int cashInMyPocket;
```

- Generates this compile error:
  ```java
  cashInMyPocket is already defined in main(java.lang.String[])
  ```

Data Types, Variables, and Constants (continued):

Constants:

- Value cannot change during program execution.
- Syntax:
  ```java
  final dataType constantIdentifier = assignedValue;
  ```
- Note: assigning a value when the constant is declared is optional.
- But, the value must be assigned before the constant is used.
- Style Point: The convention is to use CAPITAL LETTERS for constants and separate words with an underscore.
- Example:
  ```java
  final double HOURLY_PAY = 18.75;
  ```
- Declare constants at the beginning of the method so their values can easily be seen.
- Declare as a constant any data that should not change during execution.
Data Types, Variables, and Constants (continued):

 Constants (continued):

 ```java
/* Constants Class */
public class Constants {

    public static void main( String[] args ) {
        final char WILDCATS = 'W';
        final double SQRT2 = Math.sqrt(2), SQRT3 = Math.sqrt(3);
        final int DISTANCE_TO_FIRST = 90;

        System.out.println("Constant WILDCATS is " + WILDCATS);
        System.out.println("The square root of 2 is " + SQRT2);
        System.out.println("The square root of 3 is " + SQRT3);
        System.out.println("First base is " + DISTANCE_TO_FIRST + " feet from home plate");

    } // main

} // class Constants
```

Expressions and Arithmetic Operators:

- The Assignment Operator and Expressions.
- Arithmetic Operators.
- Operator Precedence.
- Integer Division and Modulus.
- Division by Zero.
- Mixed-Type Arithmetic and Type Casting.
- Shortcut Operators.
Expressions and Arithmetic Operators (continued):

Assignment Operator:
- Used to change the value of a variable.
- Syntax:
  \[
  \text{target} = \text{expression};
  \]
- The target is the variable whose value will change.
- The expression can be:
  - A variable or a value:
    ```java
    int xray;
    int sierra = 25;
    xray = sierra;
    ```
  - An arithmetic expression involving multiple variables, values, and operators:
    ```java
    int zebra = 42;
    int november = 30;
    int william;
    william = zebra + november;
    ```
  - The value of an expression must be compatible with the target's data type.

- Examples:
  ```java
  int numPlayers = 10;  // numPlayers holds 10
  numPlayers = 8;       // numPlayers now holds 8
  int legalAge = 18;
  int voterAge = legalAge;
  ```

- The following examples have errors:
  ```java
  int height = weight * 2;  // weight has not been defined yet
  int weight = 20;
  ```
  Produces the error message: “cannot find symbol”

  ```java
  int numPlayers = 10;
  numPlayers = ;
  ```
  Produces the error message: “illegal start of expression”
Expressions and Arithmetic Operators (continued):

**Arithmetic Operators:**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>%</td>
<td>modulus (remainder after division)</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
</tbody>
</table>

```java
/* SimpleOperators Class */
public class SimpleOperators {
    public static void main( String[] args ) {
        int a = 6;
        int b = 2;
        int result;

        result = a + b;
        System.out.println( a + " + " + b + " is " + result );

        result = a - b;
        System.out.println( a + " - " + b + " is " + result );

        result = a * b;
        System.out.println( a + " * " + b + " is " + result );
    } // end of main method

} // end of class SimpleOperators
```
Expressions and Arithmetic Operators (continued):

**Operator Precedence:**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Order of evaluation</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>left to right</td>
<td>parenthesis for explicit grouping</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
<td>multiplication, division, modulus</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
<td>addition, subtraction</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
<td>assignment</td>
</tr>
</tbody>
</table>

- Operator precedence follows normal arithmetic rules. Makes the following work correctly:
  ```java
  int singles = 43, doubles = 17, triples = 2, homeRuns = 28;
  int hits, totalBases;
  hits = singles + doubles + triples + homeRuns;
  totalBases = singles + 2 * doubles + 3 * triples + 4 * homeRuns;
  ```

- Batting average in baseball is computed as \( \frac{\text{number of hits}}{\text{plate appearances} - \text{walks}} \)

- The following does not work (why?):
  ```java
  float hits = 153f, appearances = 482f, walks = 27f;
  float average = (hits / (appearances - walks));
  ```

- Where are parentheses needed?
Expressions and Arithmetic Operators (continued):

**Integer Division & Modulus:**

- When dividing two **integers**:
  - The quotient is an integer.
  - The answer is **truncated**; that is, any remainder is discarded.
  - Example:
    ```java
    int i = 413, j = 499;
    int resultOne = i / 100;
    int resultTwo = j / 100;
    ```
  - What is the value of `resultOne`? of `resultTwo`?

- This applies to all the integer types: **byte, short, int, and long**.

- What happens with **negative** numbers?
  ```java
  int i = 78, j = 12, w = -42;
  int resultOne = i / j;
  System.out.println( i + " / " + j + " = " + resultOne);
  int resultTwo = -i / j;
  System.out.println( "-" + i + " / " + j + " = " + resultTwo);
  ```
  - What is the value of `resultOne`? of `resultTwo`?

  ```java
  int resultTwo = j / 100;
  int another = j % 100;
  int again = i % 100;
  i = 2793;
  j = 93847509187;
  again = i % j;
  double zap;
  zap = i / 100f;
  ```

- ```java
  int resultTwo = j / 100;
  int another = j % 100;
  int again = i % 100;
  i = 2793;
  j = 93847509187;
  again = i % j;
  double zap;
  zap = i / 100f;
  ```
Expressions and Arithmetic Operators (continued):

Integer Division & Modulus (continued):

• Modulus example: (Furlong.java)

```java
public class Furlong {
    public static void main(String[] args) {
        int numFeet = 23487;                // assume this is measured in feet
        System.out.println("length in feet = " + numFeet);

        // Calculate number of miles and feet left over
        final int FEET_PER_MILE = 5280;
        int miles = numFeet / FEET_PER_MILE;
        int leftOver = numFeet % FEET_PER_MILE;

        System.out.print("miles = " + miles + 	");
        System.out.println("leftOver = " + leftOver);

        // Calculate number of furlongs and feet left over
        final int FEET_PER_FURLONG = 660;
        int furlongs = numFeet / FEET_PER_FURLONG;
        leftOver = numFeet % FEET_PER_FURLONG;  // gets number of feet after last furlong

        System.out.print("furlongs = " + furlongs + \\	");
        System.out.println("leftOver = " + leftOver);
    }
}
```

• Modulus example: (Furlong.java)

```java
// Calculate number of miles and feet left over
final int FEET_PER_MILE = 5280;
int miles = numFeet / FEET_PER_MILE;
int leftOver = numFeet % FEET_PER_MILE;
System.out.print("miles = " + miles + \\	");
System.out.println("leftOver = " + leftOver);
```

• Modulus example: (Furlong.java)

```java
// Calculate number of furlongs and feet left over
final int FEET_PER_FURLONG = 660;
int furlongs = numFeet / FEET_PER_FURLONG;
leftOver = numFeet % FEET_PER_FURLONG;  // gets number of feet after last furlong
System.out.print("furlongs = " + furlongs + \\	");
System.out.println("leftOver = " + leftOver);
```
Expressions and Arithmetic Operators (continued):

Division by Zero:

- Integer division by 0:
  - Example:
    ```java
    int answer = 173 / 0;
    ```
  - No error from the compiler.
  - At runtime, generates an `ArithmeticException` and program stops executing.

- Floating-point division by 0:
  - Two possibilities:
    - Dividend is not zero; Divisor is zero.
      ```java
double resultAlpha = 173.628 / 0;
System.out.println("resultAlpha = " + resultAlpha);
```
    - Both dividend and divisor are zero.
      ```java
double resultBeta = 0.0 / 0.0;
System.out.println("resultBeta = " + resultBeta);
```

---

```java
public class DivideZero
{
    public static void main(String[] args)
    {
        int answer = 173 / 0;    // generates an ArithmeticException when executed

double resultAlpha = 173.628 / 0;
System.out.println("resultAlpha = " + resultAlpha);

double resultBeta = 0.0 / 0.0;
System.out.println("resultBeta = " + resultBeta);

double tinyNumber = 1.0E-310;
System.out.println("tinyNumber = " + tinyNumber);

resultAlpha = 173.628 / tinyNumber;
System.out.println("resultAlpha = " + resultAlpha);
    }
}
```
Expressions and Arithmetic Operators (continued):

Mixed-Type Arithmetic:
- Calculations can be performed with operands of different data types.
  ```java
  int sample = 17;
  float price = 37.95F;
  double cost = sample * price;
  ```
- There are rules:
  - Lower-precision operands are *promoted* to higher-precision data types before the operation is performed.
  - *Promotion* is only done while evaluating the expression. It is not a permanent change.
  - Called “*Implicit* type casting”.

Expressions and Arithmetic Operators (continued):

Mixed-Type Arithmetic (continued):
- Rules of Promotion, apply the first rule that fits:
  1. If either operand is a *double*, the other operand is converted to a *double*.
  2. If either operand is a *float*, the other operand is converted to a *float*.
  3. If either operand is a *long*, the other operand is converted to a *long*.
  4. If either operand is an *int*, the other operand is promoted to an *int*.
  5. If neither operand is a *double, float, long*, or an *int*, both operands are promoted to an *int*.

```java
  double cost = 730 * 4.95;    // 730 converted to double, rule 1
  float height = 16.75F;
  float stories = height * 7;  // 7 converted to float, rule 2
  byte months = 7;
  short days = 31;
  long hours = days * 24L;     // days converted to long, rule 3
  int seconds = days * 3600 * 24;  // days converted to int, rule 4
```
Expressions and Arithmetic Operators (continued):

Explicit Type Casting:
- Want to change the type of an expression.
- Syntax:
  \[(\text{datatype}) \text{ expression}\]
- Example:
  ```java
  public class Weeks {
      public static void main(String[] args) {
          final int DAYS_IN_WEEK = 7;
          int days = 2873;
          float weeks;
          weeks = days / DAYS_IN_WEEK;
          System.out.println("weeks = \" + weeks);
          weeks = days / (float) DAYS_IN_WEEK;
          System.out.println("weeks = \" + weeks);
          weeks = (float) days / (float) DAYS_IN_WEEK;
          System.out.println("weeks = \" + weeks);
      }
  }
  ```

Expressions and Arithmetic Operators (continued):

Explicit Type Casting (continued):
  ```java
  weeks = (float) days / (float) DAYS_IN_WEEK;
  System.out.println("weeks = \" + weeks);
  weeks = (float) days / DAYS_IN_WEEK;
  System.out.println("weeks = \" + weeks);
  weeks = days / (float) DAYS_IN_WEEK;
  System.out.println("weeks = \" + weeks);
  }
  // end of main method
  } // end of class Weeks
  ```
Expressions and Arithmetic Operators (continued):

Shortcut Operators:
- Often want to add 1 or subtract 1.
- ++ lets us add 1  -- lets us subtract 1.
- Examples:
  - Can be used in a postfix form, where the operator appears after the variable:
    ```java
    int xray = 17, sample = 24;
    xray++;     // xray now contains 18, same as typing: xray = xray + 1;
    sample--;  // sample now contains 23
    sample--;  // sample now contains 22
    ```
  - Can be used in a prefix form, where the operator appears before the variable:
    ```java
    System.out.println("xray becomes " + --xray);  // xray now contains 17
    ```

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>alpha += 3;</td>
<td>alpha = alpha + 3;</td>
</tr>
<tr>
<td>-=</td>
<td>bravo -= 17;</td>
<td>bravo = bravo - 17;</td>
</tr>
<tr>
<td>*=</td>
<td>charlie *= 8;</td>
<td>charlie = charlie * 8;</td>
</tr>
<tr>
<td>/=</td>
<td>delta /= 9;</td>
<td>delta = delta / 9;</td>
</tr>
<tr>
<td>%=</td>
<td>echo %= 60;</td>
<td>echo = echo % 60;</td>
</tr>
</tbody>
</table>
Expressions and Arithmetic Operators (continued):

Shortcut Operators (continued):

- **No spaces** are allowed between the arithmetic operator and the equals sign.
  
  \[ x + = 12; \]
  
  Causes a compiler error.

- The order of the arithmetic operator and the equals sign is important:
  
  \[ \text{int foxtrot = 30;} \]
  \[ \text{foxtrot += 42;} \]

  The compiler does not find fault with this.
  
  The statement will also execute correctly.

- What will be the value of \texttt{foxtrot} after the statement executes?

Expressions and Arithmetic Operators (continued):

Operator Precedence (updated):

<table>
<thead>
<tr>
<th>Operator</th>
<th>Order of evaluation</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>left to right</td>
<td>parenthesis for explicit grouping</td>
</tr>
<tr>
<td>++ --</td>
<td>right to left</td>
<td>preincrement, predecrement</td>
</tr>
<tr>
<td>++ --</td>
<td>right to left</td>
<td>postincrement, postdecrement</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
<td>multiplication, division, modulus</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
<td>addition or \textit{String} concatenation, subtraction</td>
</tr>
<tr>
<td>= += -= *= /= %=</td>
<td>right to left</td>
<td>assignment</td>
</tr>
</tbody>
</table>
Formatted Printing:

* When you print numbers:
  
  ```java
  double foxtrot = 30.7 / 7.3;
  System.out.println( "The answer is " + foxtrot );
  The answer is 4.205479452054795
  `}

  We don't usually need so many decimal places!!

  * Use the `printf` method instead:
    
    ```java
    System.out.printf( "The answer is %4.2f\n", foxtrot);
    The answer is 4.21
    `}

  * `printf` uses C-style formatting. The `%` marks the start of a formatting instruction.

  * In the example above, `f` means “floating-point”, and is used with float or double types.

  * The `4.2` part means:
    - Use 2 decimal places; value will be rounded.
    - Use a total of 4 characters.
      - 2 for the decimal places
      - 1 for the decimal point
      - 1 for the value in front

Formatted Printing:

  ```java
  foxtrot = 1020.29876;
  System.out.printf( "The answer is 6.3f\n", foxtrot);
  The answer is 6.3f
  `}

  String first = "Patrick";
  String last = "Homer";
  System.out.printf( "The name is %10s", first);
  System.out.println();
  System.out.printf( "The name is %5s, %7s\n", last, first);
  `}
Formated Printing (continued):

- Most commonly used format codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Formats</th>
<th>Example Use</th>
<th>Corresponding Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Integers (Base 10)</td>
<td>&quot;%5d&quot;, 29</td>
<td>29</td>
</tr>
<tr>
<td>x</td>
<td>Integers (Base 16)</td>
<td>&quot;%x %x&quot;, 29, 32</td>
<td>1D 20</td>
</tr>
<tr>
<td>o</td>
<td>Integers (Base 8)</td>
<td>&quot;%o&quot;, 29</td>
<td>35</td>
</tr>
<tr>
<td>f</td>
<td>Floating-point</td>
<td>&quot;%8.2f&quot;, 874.9163</td>
<td>874.92</td>
</tr>
<tr>
<td>e</td>
<td>Exponential Floating-point</td>
<td>&quot;%1.2e&quot;, 874.9163</td>
<td>8.749163E+02</td>
</tr>
<tr>
<td>c</td>
<td>Character</td>
<td>&quot;%c&quot;, 'Y'</td>
<td>Y</td>
</tr>
<tr>
<td>s</td>
<td>Strings</td>
<td>&quot;%10s&quot;, &quot;Hi&quot;</td>
<td>Hi</td>
</tr>
</tbody>
</table>

- There are more. See the Java API:
  - `System` is part of `java.lang`
  - `System.out` returns a `PrintStream`.
  - `PrintStream` is part of `java.io`
  - look at the `printf` method.

- See also (on lectura and the lab machines):
  - `man printf`
  - `man -s 3 printf`
  - under the heading "Format of the format string"

```
public class Printing {
    public static void main( String [] args )
    {
        double foxtrot = 30.7 / 7.3;
        System.out.printf( "The answer is \%4.2f\n", foxtrot);
        System.out.println( "The answer is " + foxtrot );

        int value = 1782;
        System.out.printf( "value =\%2d\n", value);
        System.out.printf( "value =\%4d\n", value);
        System.out.printf( "value =\%7d\n", value);
        System.out.printf( "value =\%d\n", value);

        float small = 0.000000029872F;
        System.out.printf( "small =\%f\n", small);
        System.out.printf( "small =\%4.2f\n", small);
        System.out.printf( "small =\%1.5f\n", small);
        System.out.printf( "small =\%10.8f\n", small);
        System.out.printf( "small =\%11.8f\n", small);
        System.out.printf( "small =\%15.8f\n", small);
    } // main
} // Printing
```