Acknowledgements: These slides were created by Dr. Travis Doom with information, graphics, materials, or kindly aid provided by Gaddis’s “Starting Out with Java”, McConnell’s “Code Complete”, Sierra’s “Head First Java”, Guzdal’s “Introduction to computing and programming with Java”, and Barnes’s “Objects First with Java”.

CS 241 – Computer Programming II
Programming basics
(review)

Computer programming
Semantics, syntax, and style
Variables and data type
Methods and decomposition
Methods for I/O in Java
Am I in the right place?

- What is computer programming?
- What is computer science?
- Why study computer science/programming?
  - Drives innovation in sciences (makes a positive difference!)
  - Foundational knowledge in problem solving and logical thinking
  - The “renaissance man” knowledge-base of the 21st century
  - Computing jobs rated among highest paying and most satisfying
    - Lots of teamwork! Good communication skills mandatory!
  - Greatest job growth area (contrary to popular belief)
  - Its impossible to predict what will happen next in computing, there are constantly new opportunities to make new, significant contributions
CS 241: The big picture

Prior to this class you should have learned how to:
- create an executable program (semantics, syntax, and style)
- create and use typed variables to store primitive and reference values
- use standard operators to manipulate variable and literal values
- decompose a design using routines (methods), iteration, and selection
- manage simple collections of data (arrays and ArrayLists)
- handle basic console and file I/O tasks

In this class you will learn how to:
- create/design your own classes and objects
- decompose a design by developing object types (when and why!)
- create, test, debug, and manage applications of moderate complexity
- to use more sophisticated library objects and interfaces
  - Including libraries to manage events, threads, and graphics
Computer programming

- Essentially, *computers* are simple devices that:
  - allow input to memory
  - allow output from memory
  - allow limited processing (instructions) on memory

- *Computer programming* is the task of *sequencing* the limited processing operations towards solving a specific design goal.

- *High level languages* allow us to specify tasks without being aware of the specific instructions need on a system.
Compilation Vs. Interpretation

- An alternative to *compiling* your program is to *interpret* your program
  - each line of your program is translated into a machine language and immediately executed
- Like translating between natural languages
  - **Compiler**: human translator translates book in its entirety and then translated book is printed (compiled to executable format). That book can THEN be stored and read (executed) without repeating the process.
  - **Interpreter**: human interpreter translates each spoken statement in sequence AS speaker is speaking. No recording of the translated version is made. It must be re-interpreted to hear it again.
Java uses both compilation and interpretation in a two-step process.

- **Compiles program into bytecode**
  - bytecode is a generic “machine language” for a “virtual machine”
  - does not correspond to any particular machine

- **Virtual Machine (VM)** interprets bytecodes into native machine language and runs it
  - different VM implementation exists for each native computer machine language

- **Same Java bytecodes can be used on different computers** without re-compiling source code
  - each VM interprets same bytecodes
  - allows you to run Java programs by getting just bytecodes from Web page

- This makes Java code run **cross-platform**
  - “Write once, run anywhere!”
Why Java?

- Java is VERY object-oriented
- Java is popular in the field
  - Early versions of Java were slow, buggy, and hard to use.
  - Platform independence still made it amazingly popular
  - Current versions of Java have most/all features that are popular in other programming languages
  - Java and C++ are close cousins
- Java run complete with thousands of library classes, nearly everywhere
  - More power! Easier to develop with! Powerful (and free) IDEs!
- Although the fundamental concepts of this course apply to ALL programming, the specifics we will use for illustration and for formative/summative experiences will be in Java
- If you’ve never used Java before, talk to your instructor ASAP
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello World");
    } // end method main
} // end class HelloWorld

- Semantics – the “meaning”
  - When programming we should focus on semantics.
- Syntax – the rules that must be followed when writing a program
  - In algebra, what is “5 + 3”? What is “5-”?
  - Programming language syntax rules are like grammar rules in natural/spoken languages. Syntax differs by language!
  - Syntax rules are required to avoid ambiguity.
  - Compilers/IDEs help identify syntax errors but, like grammar, they must largely be memorized
- Style – conventions that affect the readability of the program
Code structure in Java

```java
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello World");
    } // end method main
} // end class HelloWorld
```

- Put a class in a source file
  - A source code file (with a .java extension) holds one class definition
  - So far you may have only used once class (and thus one file)
  - In general, each class is just a piece (unit) of the application
- Put each method in a class
  - Methods are used to perform specific tasks
  - Exactly one “main” method must exist in your source file(s)
- Put statements in each method
  - Finally, we actually get to DO something!
Control flow

- Do something
- Sequential execution of statements
  ...
  statement1;
  statement2;
  ...

  {
  // a block of
  // statements are
  // treated as
  // one statement
  }

- Do something
- under this condition

- Selection
  ...
  if (condition) {
    statement(s);
  }
  -or-
  if (condition) {
    statement(s);
  } else {
    statement(s);
  }
  ...

- Do again and again
- Iteration
  ...
  while (condition) {
    statement(s);
  }
  -or-
  for (initializer;
    condition; update
    expression) {
    statement(s);
  }
  -or-
  do {
    statement(s);
  } while (condition)
  ...

Wright State University, College of Engineering
Dr. T. Doom, Computer Science & Engineering
CS 241
Computer Programming II
Variables in review

- Semantics – abstract place to hold information
  - Be aware of the realities/limitations of the implementation:
    - type, scope, and precision.
- Syntax – variables need a type and an identifier
  - Identifier characters (a-z, A-Z, 0-9, _, $)
  - Leading character must not be a digit
  - Case sensitive
- Style
  - Naming conventions
    - numOfStudents Vs. num, n, students, loopCheckValue, tempInt
    - NUM_OF_STUDENTS (for unchanging constant/“final” values)
  - Declaration conventions
    - Declaration block
    - Just in Time
  - Initialization conventions
**Variables**

<table>
<thead>
<tr>
<th>Variable declaration</th>
<th>Assignment statement</th>
<th>Initialized declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>int value;</td>
<td>value = 5;</td>
<td>int value = 5;</td>
</tr>
</tbody>
</table>

This is a String *literal*. It will be printed as is.

```java
System.out.print("The value is ");
System.out.println(value);
```

The integer 5 will be printed out here.

Notice no quote marks.
Data types in Review

- Select data types with care!
  - int, double, boolean, and char most frequent
- Be wary of compiler assumptions! In Java:
  - integer literals are cast as int
  - real number literals are cast as double
  - results of mixed types are promoted
  - Example: 2/3 is cast int 0
  - Example: 2.0/3 is cast double 0.666
- Java automatically promotes lower precision types to higher precision types but not visa-versa!
  - byte -> short -> int -> long -> float -> double
- The rules can be overridden by explicit typecasting

float x = (float) (5/3.0);
Semantics of Methods

- Smaller, simpler, subcomponent of program
- Provides abstraction
  - hide low-level details
  - give high-level structure to program, easier to understand overall program flow
  - enables separable, independent development of modules as single components that can be later used to build more complex components
- Method have a *signature*
  - methods have a name (by convention, only one method named “main”)
  - zero or multiple *arguments* passed in as *typed parameters*
  - single result returned
    - void
    - Primitive data type or Derived data type (class/object)
- In other languages, called functions, procedures, subroutines, ...
Calling a value-returning method

```java
public static int sum(int num1, int num2) {
    int result;
    result = num1 + num2;
    return result;
} // end method sum
```

- At this point, you should be familiar with `public static` methods that return primitive data types.
  - Later: other accesses modifiers (public, protected, private)
  - Later: static Vs. regular (non-static) methods
Methods and your data

- A local variable is declared inside a method and is not accessible to statements outside the method.
  - **Scope**
  - Different methods can have local variables with the same names because the methods cannot see each other’s local variables.
  - A method’s local variables exist only while the method is executing.
- The values/arguments passed to the parameters of the method are copied and become local variables to that method.
  - **Call by value**
  - Changes to the local copies of variables do not change the original
  - Use returned value to make changes to primitive data types
- If a reference to an object is passed to a method, then the method makes a copy of the reference and can use that reference to make changes to the actual object.
  - **Call by reference**
printf: formatting console output

```java
System.out.printf("UID=%d\n", uid);
System.out.printf("UID=%d\tlogin=%s\n", uid, loginName);
printf("%+-5d  %+5d", 10, -20)
+1 0 _____ - 2 0
printf("%8.3f, %7.2f", 1.234567, 1.234567)
___1.234___1.23
```

<table>
<thead>
<tr>
<th>Flags</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Left justified</td>
</tr>
<tr>
<td>+</td>
<td>prefix with +/-</td>
</tr>
<tr>
<td>0</td>
<td>pad with zeros</td>
</tr>
<tr>
<td></td>
<td>separate by thousands</td>
</tr>
<tr>
<td></td>
<td>negatives in parens</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>integer (digits)</td>
</tr>
<tr>
<td>%f</td>
<td>floating point</td>
</tr>
<tr>
<td>%e</td>
<td>exponential/scientific</td>
</tr>
<tr>
<td>%b</td>
<td>boolean</td>
</tr>
<tr>
<td>%c</td>
<td>character</td>
</tr>
<tr>
<td>%s</td>
<td>string</td>
</tr>
</tbody>
</table>
import java.util.Scanner;
public class Main {

    public static int getValue (String prompt) {
        Scanner keyboard = new Scanner(System.in);
        int userValue;
        System.out.println(prompt);
        userValue = keyboard.nextInt();
        return userValue;
    } // end method getValue

    public static void main(String [] args){
        int inputValue = getValue("Enter integer value: ");
        System.out.println("Value is ", inputValue);
    } // end method main

} // end class GetValue
import java.io.File;
import java.util.Scanner;
public class Main {
    public static void main (String[] args) throws Exception {
        String filename = "Data.txt";  // uses working directory
        File fileHandle = new File(filename);
        Scanner inputFile = new Scanner (fileHandle);

        String line;
        while (inputFile.hasNextLine()) {
            line = inputFile.nextLine();
            System.out.println(line);
        }
        inputFile.close();
    } // end method main
} // end class Main
import java.io.FileWriter;
import java.io.PrintWriter;
public class Main {
    public static void main (String[] args) throws Exception {
        final boolean APPEND_EXISTING_FILE = true;
        final boolean CLOBBER_EXISTING_FILE = false;
        String filename = "Data.txt";  // uses working directory
        String filename2 = "C:\Documents and Settings" + 
            "\w001ted\Desktop\Data.txt ";
        FileWriter fileHandle =
            new FileWriter(filename,CLOBBER_EXISTING_FILE);
        PrintWriter outputFile = new PrintWriter(fileHandle);
        outputFile.println("Hello file!");
        outputFile.close();
    } // end method main
} // end class Main
public static void main(String[] args) {
    int[][] numList;
    int[] xList = {1,2,3,4},
              yList = {9, 8};
    numList = new int[3][];

    numList[0] = xList;
    numList[1] = yList;
    xList = yList;

    printList (numList[0]);
    printList (numList[1]);
    printList (xList);
    printList (yList);
    printList (numList[2]);
} // end method main

public static void printList (int[] list) {
    for (int i=0; i < list.length; i++) {
        System.out.print (list[i] + " ");
    }
    System.out.println();
} // end method printList
for / for-each loops

- Some of the brightest people in Computer Science suggest that arrays should never be accessed *randomly* but only *sequentially*
  - Random accesses in arrays tend to be undisciplined, error prone, and hard to prove correct.
  - They suggest other (more advanced) data structures for random access

- The *enhanced* for loop (a type of for-each loop) provides a means to implement this stylistic constraint.

```java
for (datatype elementVariable : collection){ statements;}
```

- Compare:

```java
double[] numberList = new double[20];
...
for(int i=0; i< numberList.length; i++) {
    System.out.println(numberList[i]);
}
double[] numberList = new double[20];
...
for( double number : numberList) {
    System.out.println(number);
}
```
The ArrayList class in the Java API is similar to an array, but it does not store primitive data types as elements.
- ArrayList data structures store any object using a generic data type
- An ArrayList can hold objects of different types!

ArrayList data structures provide the following abstractions/features:
- Add an element: The ArrayList object automatically expands as items are added to it
- Remove an element: The ArrayList object automatically reduces as items are removed from it

```java
import java.util.ArrayList;               //...
ArrayList nameList = new ArrayList();   // NOTE NO SIZE
nameList.add( new String( "Bob" ) );    // ADDING OBJECTS
nameList.add( new String( "Pat" ) );
nameList.add( new String ("April") );
String name = (String) nameList.get( 0 ); // NOTE TYPECASTING!
```
ArrayList methods

- Useful ArrayList methods:
  - .size() returns the size of the ArrayList
  - .add( object ) adds the object reference to the end
  - .add( index, object ) inserts the object before the index (updates others)
  - .set( index, object ) overwrites an existing object
  - .get( index ) returns the object reference, but doesn’t remove it
  - .remove( index) returns the object reference and removes it

- ArrayList can be used as a parameterized class
  ArrayList<String> nameList = new ArrayList<String>();

- Parameterized ArrayLists can only hold objects of the named type and returns/gets are automatically typed
  - Strongly typed data structures help reduce errors!
The Random class

- Useful methods
  - `nextInt ( int n)`: returns an integer number \([0, n]\)
  - `nextDouble()`: returns a double \([0.0, 1.0]\)
  - `nextInt()`: returns an integer \([\text{minInt}, \text{maxInt}]\)

```java
import java.util.Random

... Random randomNumbers = new Random();
int dieRoll = randomNumbers.nextInt(6)+ 1;
...```

Wright State University, College of Engineering
Dr. T. Doom, Computer Science & Engineering
### Complete Java Order of Precedence

<table>
<thead>
<tr>
<th>Order</th>
<th>Operator</th>
<th>Operation</th>
<th>Associates</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( )</td>
<td>parenthesis</td>
<td>L to R</td>
</tr>
<tr>
<td>1</td>
<td>[] . ,  ++ --</td>
<td>array subscript, member selection, comma delimiter, post increment, post decrement</td>
<td>L to R</td>
</tr>
<tr>
<td>2</td>
<td>++ -- + - !</td>
<td>prefix increment, prefix decrement, positive, negative, NOT</td>
<td>R to L</td>
</tr>
<tr>
<td>3</td>
<td>(type) new</td>
<td>type cast, object instantiation</td>
<td>R to L</td>
</tr>
<tr>
<td>4</td>
<td>* / %</td>
<td>multiplication, division, modulo</td>
<td>L to R</td>
</tr>
<tr>
<td>5</td>
<td>+ - +</td>
<td>addition, subtraction, string concatenation</td>
<td>L to R</td>
</tr>
<tr>
<td>6</td>
<td>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</td>
<td>left shift, right shift, unsigned right shift</td>
<td>L to R</td>
</tr>
<tr>
<td>7</td>
<td>&gt;= &lt;= &gt; &lt; instanceof</td>
<td>greaterThanOrEqual, lessThanOrEqual, greaterThan, lessThan, type comparison</td>
<td>L to R</td>
</tr>
<tr>
<td>8</td>
<td>== !=</td>
<td>equalTo, notEqualTo</td>
<td>L to R</td>
</tr>
<tr>
<td>9</td>
<td>&amp;</td>
<td>Bitwise AND, boolean AND (no short circuit)</td>
<td>L to R</td>
</tr>
<tr>
<td>10</td>
<td>^</td>
<td>Bitwise XOR, boolean XOR (no short circuit)</td>
<td>L to R</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>Bitwise Or, boolean OR (no short circuit)</td>
</tr>
<tr>
<td>12</td>
<td>&amp;&amp;</td>
<td>logical AND (short circuits)</td>
<td>L to R</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>?:</td>
<td>Conditional operator (boolean)?(do if true):(do if false)</td>
<td>R to L</td>
</tr>
<tr>
<td>15</td>
<td>=, +=, -=, *=, /=, %=, &lt;&lt;=, &gt;&gt;&gt;=, &gt;&gt;==, &amp;=,</td>
<td>assignment, plusAssign, minusAssign, timesAssign, dividesAssign, moduloAssign, leftShiftAssign, rightShiftAssign, rightShiftAssignUnsigned, ANDAssign, XORAssign, ORAssign</td>
<td>R to L</td>
</tr>
</tbody>
</table>
Example: Guessing game

- Guess a number game
  - Computer generates a random number between 1-16
  - You guess a number
  - Program responds with one of three responses
    - Your guess is too high!
    - Your guess is too low!
    - Your guess is correct!
  - You get four guesses.
  - If your forth guess is wrong, then you loose!

- Math note: What percentage of the time do you expect to win?

- Example 2: draw a random card from a standard 52 card deck?

Wright State University, College of Engineering
Dr. T. Doom, Computer Science & Engineering CS 241 Computer Programming II