Java Basics *(ala Goodrich & Tamassia)*

- Everything is in a class
- A minimal program:

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Computer Science");
    }
}
```

- Output?
- Where?
- Do colors mean something?
- Identify the pieces . . .
Java Basics

Objects

- Every object is an instance of a class (which defines its type)
- Objects contain data (state) and function (methods)

State

- Stored in instance variables (fields, members)
- Can be base types (e.g. integers)
  or instances of (objects) of other classes

Function

- Expressed as methods (subroutines, functions, procedures…)
- These define the behavior of objects of this class
Java Basics

- Declaring a Class:

```java
public class Counter {
    protected int count;
    Counter() {
        count = 0;
    }
    public int getCount() {
        return count;
    }
    public void incCount() {
        count = count + 1;
    }
    public void decCount() {
        count = count - 1;
    }
}
```

- Identify the methods by kind
  - Constructor
  - Accessor
  - Mutator (modifier)

- Note Syntax from this and previous examples
  - Braces
  - Semicolons
  - Parentheses
  - Identifiers
  - ...

CompSci 100E
Java Basics

- **Class Modifiers**
  - Abstract, final, public, default

- **Reserved Words**
  - May not be used as identifiers
  - Shown in red by Eclipse and many of our examples
  - See table in text (p4) or in any Java text

- **Comments**
  - For human consumption: ignored by compiler
  - Inline comments: //
    - Effective for rest (to end) of current line
  - Block comments: /* */
    - Effective between start and stop groups
Java Basics

- **Primitive Types (base types)**
  - Built-in data types; native to most hardware
  - Note: not objects (will use mostly first four)

  ```java
  boolean
  int
  double
  char
  ```

- **Constants (by example):**

  ```java
  boolean f = false;
  int i = 32769;
  double d = 0.333333;
  char c = 'x';
  byte b = 33;
  short s = 21;
  long l = 289L;
  float = 3.141592F;
  ```
Java Basics

Creating and Using Objects (Example)

```java
public class Example {
    public static void main(String[] args) {
        Counter c; // Counter defined on a previous slide
        Counter d = new Counter();
        c = new Counter();
        System.out.println("c = " + c.getCount()
            + " d = " + d.getCount());
        c.incCount();
        d.decCount();
        System.out.println("c = " + c.getCount()
            + " d = " + d.getCount());
        d = c; // what does this really mean???
        c.incCount();
        d.incCount();
        System.out.println("c = " + c.getCount()
            + " d = " + d.getCount());
    }
}
```
Java Basics

- **String Objects**
  - string is a sequence of characters (**char**)
    - Unicode (16 bit)
  - String is a built-in class
    - Constants: "this is an example"

- **String Concatenation (+)**
  
  ```java
  String s = "Happy birthday to you."
  s = s + "\n" + s;
  System.out.println(s); // what ?
  ```
Java Basics

- **Object References**
  - When creating object with `new`, get *location* or *address* of new object
  - Typically assign this to a reference variable:
    ```java
    Counter c = new Counter();
    ```
  - Every object reference variable refers to object or `null`
  - Null is an important value that indicates object not created or not available.
  - Can have multiple references to same object
  - Access members of class using *dot operator* (" . ").
    ```java
    Counter c = new Counter();
    c.incCount();
    ```
  - May have multiple methods with same name but different *signature*: e.g.:
    ```java
    c.incCount(); c.incCount(5);
    ```
Java Basics

- **Instance Variables**
  - Classes have 0 or more instance variables
    - Also called fields
    - Keep state of object
  - May be primitive type
    - E.g. int, double
  - May be reference type (object)
    - E.g., String, Counter, (an array), ...
  - If public can:
    - Access or alter reference variables using dot operator
      ```java
      Counter c = new Counter();
      System.out.println(c.count + " = " + c.getCount());
      ```
Java Basics

- **Variables Modifiers: scope**
  - **public**
    - Anyone can access
  - **protected**
    - Only subclass or same package may access
  - **private**
    - Only methods of same class may access
  - (omitted) **default**
    - Anyone in same package may access

- **Other Variable Modifiers**
  - **static**
    - Associated with whole class, shared among instances
  - **final**
    - Must be initialized, then not changed: CONSTANT
Java Basics - Methods

- **Methods**
  - Like functions, procedure, subroutines, ...
  - Has *header* and *body*
  - Syntax:
    
    ```java
    modifiers type name(parameter_declarations){
        method_body
    }
    ```
  - *Modifiers* like those of variables:
    - public, private, protected, static, final
  - *Type* is return type and give type of information being passed back
  - *Name* is any valid Java identifier name
  - *Parameters* define type of info being passed into method
Java Basics - Methods

- **Method modifiers**
  - `public`: anyone can invoke (call)
  - `protected`: only called from subclass of same package
  - `private`: only called from same class
  - (omitted) (default): only called from same package
  - `abstract`: has no code (dealt with in subclass)
  - `final`: cannot be overridden in subclass
  - `static`: associated with class, not with instance

- **Return types**
  - Use `void` is no information to be returned (*procedure*)
  - Use actual type of information to be returned (*function*)
    - requires `return` statement(s)
    - only one item returned (may be compound object, e.g., array)
Java Basics - Methods

- **Parameters**
  - Parameter list may be empty (*parentheses still required*).
  - Parameter list consists of comma separated pairs of types and parameter names.
    ```java
    public void setAge(String name, int age){...}
    ```

- **Constructors**
  - Used to initialize new objects
  - Has **same name as class and no return type**
    ```java
    public Counter() {
        count = 0;
    }
    
    public Professor(String aName, String aDept){
        name = aName;
        department = aDept;
    }
    ```
Java Basics

- Using a Constructor
  - Invoked using a new operator
    - Examples:
      ```java
      Professor compSciProf = new Professor("Jeff Chase", "Computer Science");
      Counter tally = new Counter();
      ```
  - Class may have multiple constructors as long a signatures are different
  - If class has no constructors defined, then a default constructor is used that does not initialize anything
Java Basics - Methods

- **The main Method**
  - Required for an Application
    - This is a stand-alone Java program
    - Typically invoked from a command line
    - Must include the following code:
      ```java
      public static void main(String[] args){
          // main body of the main method
      }
      ```
    - (The parameter name `args` can actually be any name you choose.)
    - Argument may be used to pass command line arguments to the program.
Java Basics - Methods

- Blocks and Local Variables
  - Body of a method is a block:
    - a sequence of statements and declarations enclosed in braces ({  });
      - Blocks may have blocks nested inside
      - Variables declared with a block are known only in that block
      - These variables are called local variables
      - (We say their scope is limited to that block.)
      - (Method parameters are also local to that method.)
      - Examples:

```java
public static int sumThree(int a, int b, int c){
    int sum;
    int partsum = a + b;
    sum = partsum + c;
    return sum;
}
```
  - a, b, sum, and partsum are all local to that method
Java Basics - Expressions

- **Literals**
  - A literal is a constant value also called a *self-defining term*
  - Possibilities:
    - **Object:** `null`, the only object literal available
    - **Boolean:** `true` or `false`
    - **Integer:** e.g., `127`, `-13`, `42`, or `0` create 32-bit integers
      - For 64-bit `long` append `L` or `l`, e.g., `17L`
    - **Floating Point:** `3.14592` or `0.0` or `2.1E16` for 64-bit doubles
      - For 32-bit `float` append `F` or `f`, e.g., `2.56F` or `0.5e-12f`
    - **Character:** e.g., `'A', 'Z', 'w', '$', '%' for 16 bit Unicode
      - control: `\n`, `\b`, `\f`, `\t`, `\r`
      - escape: `\'`, `\\`, `\"`
    - **Strings:** e.g., "How are things?" or "" (null string)
      - Use mostly same control and escape characters as `char`
Java Basics - Expressions

- Operators
  - Arithmetic
    - +, −, *, /, % (remainder or mod)
  - Increment/Decrement
    - e.g., k++, k--, ++k, --k
  - Logical (results in boolean value)
    - <, <=, ==, !=, >=, >
    - Used only for numbers except == and !=
    - For boolean only: !, &&, ||
  - String Concatenation
    - “I’m “ + 19 + “ years old and live in “ + city
  - Assignment
    - variable = expression
    - variable op= expression
    - (shorthand for: variable = variable op expression)
Java Basics - Expressions

- **Operator Precedence**
  - Determines order of operation
  - See table in text
  - For arithmetic, matches grammar school learning
    - multiplication and division before addition and subtraction
    - what is the value of \( 4.0 + 5.0 / 9.0 * 27.0 \) ?
    - (what is the value for the integer version?)
  - Parentheses override precedence rules (and don’t do harm when not needed)
  - For equal precedence (e.g., \(*\) and \(/\)) work strictly left to right
    except for assignment and prefix operations which work right to left
  - Precedence rules same as for C and C++
Java Basics - Expressions

- **Casting**
  - Allows us to change the type of the value of an expression
  - (Type change must be reasonable and supported.)
  - Simple example:
    ```java
    double x = 5.5, y = 2.9999;
    int k = (int) x;
    int m = (int) y;
    double z = (double) k; // what is in x, y, z, k, m ?
    ```

- **Implicit Casting**
  - When an `int` expression is assigned to a `double`, casting is automatic (no information is lost).
    - (double cast at end of previous example not needed)
  - When `double` is on one side of an operator and `int` at other, `int` is automatically cast to a `double` before op is used.
    ```java
    5 / 9 * (68 - 32) vs. 5.0 / 9 * (68 - 32)
    ```
Java Basics - Expressions

- **Autoboxing/Unboxing**
  - Since Java 5.0, there is automatic casting between primitive types and their related Object types (also called *wrapper classes*).
  - Simple examples:
    ```java
    Double d = 2.9;
    used to require:
    Double d = new Double(2.9);
    and
    double x = d;
    used to require
    double x = d.doubleValue();
    ```
Java Basics – Control of Flow

• If Statement

  • if (boolean_exp) {
    what_to_do_if_true
  }

  • if (boolean_exp) {
    what_to_do_if_true
  } else {
    what_to_do_if_false
  }

  • if (1st_boolean_exp) {
    what_to_do_if_1st_true
  } else if (2nd_boolean_exp){
    what_to_do_if_2nd_true
  } else {
    what_to_do_if_all_false
  }
Java Basics – Control Flow

❖ Switch Statement

```java
switch (int_type_exp) {
    case CONST1:
        action_for_CONST1;
        break;
    case CONST1:
        action_for_CONST1;
        break;
    case CONST2:
        action_for_CONST2;
        break;
    case CONST3:
        action_for_CONST3;
        break;
    ...
    default:
        action_for_no_match;
        break;
}
```
Java Basics – Control Flow

- **Switch Statement Example**
  ```java
  switch (stars) {
      case 4:
          message = "truly exceptional";
          break;
      case 3:
          message = "quite good";
          break;
      case 2:
          message = "fair";
          break;
      case 1:
      case 0:
          message = "forget it";
          break;
      default:
          message = "no info found";
          break;
  }
  ```
Java Basics – Loops

- **While Loops**
  - Syntax
    
    initialize
    
    while (boolean_exp) {
      work_to_be_done
      update
    }
  
  - Example
    
    int counter = 10;
    while (counter > 0) {
      System.out.println(counter);
      counter--;
    }
    System.out.println("Blast Off!");

  - What is the output?
  - What if we exchange order of two statements in loop?
Java Basics – Loops

❖ For Loops
  ❖ Syntax
  
  ```java
  for (initialization; boolean_exp; update) {
      work_to_be_done
  }
  ```

  ❖ Example

  ```java
  for (int counter = 10; counter > 0; counter--) {
      System.out.println(counter);
      System.out.println("Blast Off!");
  }
  ```

  ❖ What is the output?
  ❖ When is update performed?
  ❖ What is value of `counter` after loop?
Java Basics – Loops

- Do-While Loops
  - Syntax
    initialize
do{
    work_to_be_done
    update
} while (boolean_exp);
  - NOTE REQUIRED SEMICOLON!!!

- Example
  ```java
  int counter = 10;
  for (counter > 0 {  
    System.out.println(counter);
    counter--;  
  } while (counter > 0);
  System.out.println("Blast Off!");
  ```
Java Basics – Loops

Which Kind of Loop Do I Use?

- While Loop
  - Don’t know how often it’s going be
  - Update can be anywhere in the loop body
- For Loop
  - Know how often in advance
  - All information controlling loop together, in front
- Do-While Loop
  - Least popular
  - Often used with data input

What is the minimum number of times each of these loop?

- while?
- for?
- do-while?
Java Basics – Control Flow

- **Returning from a Method**
  - Executing a `return` statement means you exit from the method. Subsequent statements are ignored!
  - **void** Methods
    - Implicit `return` at end of body
    - Can make it explicit
    - Can have other `return` statements as logic dictates
  - **Functions (non-void Methods)**
    - Require `return` as last statement (with argument of correct type)
    - Can have other `return` statements as logic dictates
Java Basics – Control Flow

- **Break Statement**
  - Use to exit from loop or switch
    - One level only!
    - With nested loops, only leave loop immediately surrounding `break`

- **Continue Statement**
  - Use to go to the end of a loop, ignoring remaining statements
    - Loop continues with next iteration (if needed)
    - One level only!
    - With nested loops, only got to end of loop immediately surrounding `continue`
Java Basics – Arrays

❖ Should be a very familiar idea
  □ Problem: Deal with exam grades in a course
    o Could have variable for each student
    o Would need unique name for each variable
    o Need lots of custom code
    o Instead, assume named array; use index to get values

❖ Example: method to count number of A grades

```java
public static int getAs(int[] grades) {
    int aCount = 0;
    for (int k = 0; k < grades.length; k++){
        if (grades[k] >= 90) {
            aCount++;
        }
    }
    return aCount;
}
```

❖ Explain