CSCI 136
Data Structures &
Advanced Programming

Lecture 3

Fall 2018

Instructors: Bill & Bill

Administrative Details

- Lab today in TCL 217a (and 216)
  - Lab is due by 11pm Sunday
- Lab 1 design doc is "due" at beginning of lab
  - Written design docs will be required at all labs
  - Several implementation options
    - Some may be better than others....

Last Time

- Arrays, Operators, Expressions
- Some Simple Examples (Sum0-5)
  - Entering, editing, compiling, running programs
- Control structures
  - Looping: while, do – while, for, for – each

Today's Outline

- Control structures
  - Branching: if – else, switch, break, continue

- Object oriented programming Basics (OOP)

- Strings and String methods

- More on Class Types

  - Interface specification for behavior abstraction
  - Inheritance (class extension) for code reuse
  - Abstract Classes

But First: Importing Classes

In Sum2.java we used the Scanner class for input

The Java distribution has a variety of useful classes

To use such a class, you must `import` it

    Unless it is in the directory of your program

To do this, use import with the package name

Examples

```
import java.util.Scanner;
import java.util.Random;
import structure5.*; // entire package
```

Control Structures

Select next statement to execute based on value of a boolean expression. Two flavors:

- Looping structures: `while`, `do/while`, `for`
  - Repeatedly execute same statement (block)

- Branching structures: `if`, `if/else`, `switch`
  - Select one of several possible statements (blocks)
  - Special: break/continue: exit a looping structure
    - break: exits loop completely
    - continue: proceeds to next iteration of loop

If/else

```
if (x > 0)            // Exactly 1 "if" clause
      y = 1 / x;
else if (x<0) {     // 0 or more "else if" clauses
      x = - x;
      y = 1 / x;
}
else                    // at most 1 "else" clause
      System.out.println("Can't divide by 0!");
```

As with for/while/do-while, the single statement can be replaced by a *block:* any sequence of statements enclosed in {}

switch

Example: Encode clubs, diamonds, hearts, spades as 0, 1, 2, 3

```
int x = myCard.getSuit(); // a fictional method
switch (x) {
    case 1: case 2:
        System.out.println("Your card is red");
        break;
    case 0: case 3:
        System.out.println("Your card is black");
        break;
    default:
        System.out.println("Illegal suit code!");
        break;
}
```

Break & Continue

Suppose we have a method `isPrime` to test primality

Find first prime > 100

```
for( int i = 101; ; i++ )   // What's with ; ; ?
    if ( isPrime(i) ) {
        System.out.println( i );
        break;
    }
```

Print primes < 100

```
for( int i = 1; i < 100 ; i++ ) {
    if ( !isPrime(i) )
        continue;
    System.out.println( i );
}
```

Summary

Basic Java elements so far

- Primitive and array types

- Variable declaration and assignment

- Operators & operator precedence

- Expressions

- Control structures

  - Branching: if – else, switch, break, continue

  - Looping: while, do – while, for, for – each

- Edit (emacs), compile (javac), run (java) cycle

Object-Oriented Programming

- Objects are building blocks of Java software

- Programs are collections of objects
  - Cooperate to complete tasks
  - Represent "state" of the program
  - Communicate by sending messages to each other
    - Through *method invocation*

Object-Oriented Programming

- Objects can model:
  - Physical items - Dice, board, dictionary
  - Concepts - Date, time, words, relationships
  - Processing - Sort, search, simulate
- Objects contain:
  - State (instance variables)
    - Attributes, relationships to other objects, components
      - Letter value, grid of letters, number of words
  - Functionality (methods)
    - Accessor and mutator methods
      - addWord, lookupWord, removeWord

Object Support in Java

- Java supports the creation of programmer-defined types called *class types*

- A *class declaration* defines data components and functionality of a type of object

  - Data components: *instance variable (field) declarations*

  - Functionality: *method declarations*

  - *Constructor(s)*: special method(s) describing the steps needed to create an object (*instance*) of this class type

A Simple Class

Premise: Define a type that stores information about a student: name, age, and a single grade.

Declare a Java class called Student with data components (*fields/instance variables*)

```
String name;
int age;
char grade;
```

And methods for accessing/modifying fields
- getName, getAge, getGrade
- setAge, setGrade

Declare a constructor, also called Student

```java
public class Student {
    // instance variables
    private int age;
    private String name;
    private char grade;

    // A constructor
    public Student(int theAge, String theName,
                   char theGrade) {
        age = theAge;
        name = theName;
        grade = theGrade;
    }

    // Methods for accessing/modifying objects
    // ...see next slide...
```

```java
    public int getAge() {return age;}

    public String getName() {return name;}

    public char getGrade() {return grade;}

    public void setAge(int newAge) {age = newAge;}

    public void setGrade(char grade) {
        this.grade = grade;
    }
} // end of class declaration
```

Testing the Student Class

```java
public class TestStudent {

    public static void main(String[] args) {
        Student a = new Student(18, "Patti Smith", 'A');
        Student b = new Student(20, "Joan Jett", 'B');
        // Nice printing
        System.out.println(a.getName() + ", " +
            a.getAge() + ", " + a.getGrade());
        System.out.println(b.getName() + ", " +
            b.getAge() + ", " + b.getGrade());
        // Tacky printing
        System.out.println(a);
        System.out.println(b);
    }
}
```

Worth Noting

- We can create as many student objects as we need, including arrays of Students

```
Student[] class = new Student[3];
class[0] = new Student(18, "Patti Smith", 'A');
class[1] = new Student(20, "Joan Jett", 'B');
class[2] = new Student(20, "David Bowie", 'A');
```

- Fields are *private*: only accessible in Student class

- Methods are *public*: accessible to other classes

- Some methods return values, others do not

  - `public String getName();`
  - `public void setAge(int theAge);`

A Programming Principle

Use constructors to initialize the state of an object, nothing more.

- State: instance variables

- Frequently they are short, simple methods

- More complex constructors will typically use helper methods or other constructors

- See Student2 example

Access Modifiers

- `public` and `private` are called *access modifiers*

  - They control access of other classes to instance variables and methods of a given class

  - `public` : Accessible to all other classes

  - `private` : Accessible only to the class declaring it

- There are two other levels of access that we'll see later

- Data-Hiding (encapsulation) Principle

  - Make instance variables `private`

  - Use `public` methods to access/modify object data

  - Use `private` methods otherwise

More Gotchas

```java
public class Student {
    // instance variables
    private int age;
    private String name;
    private char grade;

    // A constructor
    public Student(int age, String name,
                   char grade) {
        // What would age, name, grade
        // refer to here...?
    }
```

Use This

```
public class Student {
    // instance variables
    private int age;
    private String name;
    private char grade;

    // A constructor
    public Student(int age, String name,
                   char grade) {
        this.age = age;
        this.name = name;
        this.grade = grade;
    }
```

String in Java Is a Class Type

- Java provides special support for String objects
  - String literals: "Bob was here!", "-11.3", "A", ""

- If a class provides a method with *signature*
  ```
  public String toString()
  ```
  Java will automatically use that method to produce a String representation of an object of that class type.

- For example
  ```
  System.out.println(aStudent);
  ```
  would use the toString method of Student to produce a String to pass to the println method

Pro Tip: *Always provide a toString method!*

String methods in Java

- Useful methods (also check String javadoc page)
  - `indexOf(string) : int`
  - `indexOf(string, startIndex) : int`
  - `substring(fromPos, toPos) : String`
  - `substring(fromPos) : String`
  - `charAt(int index) : char`
  - `equals(other) : bool` ← *Always use this!*
  - `toLowerCase() : String`
  - `toUpperCase() : String`
  - `compareTo(string) : bool`
  - `length() : int`
  - `startsWith(string) : bool`
- Understand special cases!