CSCI 136
Data Structures & Advanced Programming

Lecture 3

Fall 2019

Instructors: Bill & Sam

Administrative Details

- Lab today in TCL 217a (and 216)
  - Lab is due by 10 pm 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

- Further examples : The Game of Nim
- Operators & operator precedence
- Expressions
- Control structures
  - Branching: if – else, switch, break, continue
  - Looping: while, do – while, for, for – each

Today's Outline

- Discussion: Lab 1
- Object oriented programming Basics (OOP)
- Strings and String methods

Lab 1

- Purpose
  - Exercise your Java skills by programming a game
  - Learn some new tools
    - Terminal command-line interface to Unix
    - Atom program editor
    - GitHub version control system
  - Learn some code development habits
    - Design documents
    - Pseudo-code

Lab 1

- GitHub
  - Cloud support for file storage with *version control*
  - Basic commands
    - git clone – make a local copy of an existing repository
    - git add – add files to local copy of repository
    - git rm – remove a file from local copy
    - git commit – commit staged changes
    - git push – update master repository with committed changes in local repository
    - git pull – update local repository from master

Lab 1

- CoinStrip Game
  - Two-player coin-moving game (let's play!)
  - Essentials
    - Decide on game representation
    - Build the board
      - Random coin locations
    - Allow players to take turns
      - Enter, check, process a move
    - Congratulate the winner!

Reminder : Importing Classes

In Sum4.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
```

Object-Oriented Programming

- Objects are building blocks of Java software

- Programs involve 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
  - Processes - 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

```
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...
```

```
    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

```
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
- Usually constructors 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

```
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;
      }
```

‘Objectifying’ Nim

Goal: Allow multiple ‘Nim’ instances (objects)

- Supports playing simultaneous games
- Allow each game to have its own state

How?

- Delete ‘`static`’ from data declarations
  - Except for constants `minPileSize, maxPileSize`
    - They have same (class-wide) value for all Nim objects
    - This is a subjective choice to illustrate a point
- Delete ‘`static`’ from methods that act on single Nim instance
  - Every method except `main`
- Add a *constructor* method to initialize new Nim instance
  - In fact, for convenience, add 2 constructors

Data Declaration : Object Nim

```
private static int minPileSize = 3;

private static int maxPileSize = 8;

private static Scanner in = new Scanner(System.in);

private int[] board

private int piles

private int pilesLeft;

private Random rng = new Random();
```

Constructors : Object Nim

```
public Nim2(int size) {
   piles = size;                    // Create the board
   board = new int[piles];

   // Fill the board with randomly sized piles
   for(int i=0; i< board.length; i++)
      board[i] = MIN_PILE_SIZE + rng.nextInt(
         MAX_PILE_SIZE - MIN_PILE_SIZE + 1);

   pilesLeft = piles;               // No pile is empty
}

public Nim2() { this(5); }     // Constructor chaining
```

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!*

Nim3 : Nim with toString()

```
public String toString() {
  String result = "";          // Set to empty string

  for(int i = 0; i < board.length; i++) {
    result += i + ":";

    // Display a pile
    for(int j=0; j < board[i]; j++)
      result += " O";

    result += "\n";            // Add new-line
  }
  return result;
}
```

Replace `games[i].displayBoard()` with `System.out.println(games[i])`

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) : int`
  - `length() : int`
  - `startsWith(string) : bool`
- Understand special cases!

Using Strings

- Application: Parsing an XML file of a CD collection
  - XML = eXtensible Markup Language
  - XML is used for many things
  - Music track info:

```
<TRACK>
   <NAME>Big Willie style</NAME>
   <ARTIST>Will Smith</ARTIST>
   <ALBUM>Big Willie style</ALBUM>
   <GENRE>Pop Rap</GENRE>
   <YEAR>1997</YEAR>
 </TRACK>
```

- How can we find and print just the track names?
  - See TrackTitles.java
  - java TrackTitles < trackList.xml

Summary

Java

- More on conditional control flow
  - Switch, break, continue
- Using classes from external packages
  - Random, Scanner
  - Import statement
- Use of static for non-object-based data and methods
- Introduction to classes and objects

Lecture Ends Here