THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

COMP 110
Introduction to Programming

Tuesday September 2, 2014

Jay Aikat
Spring 2014
TR 11:00 – 12:15, GS-G100

Previous Class

- What did we discuss?

Announcements

- Assignment1 grades before Thu class
- The full grade is listed on Sakai, and the details of why you received points off will be in **feedback.txt**.
- Due to the program we used to grade and the way we uploaded grades, there might be duplicates of this file, the project you submitted, and another file created by our grader program (results.json). If you do have duplicate files, they should all show the same information.

Announcements

- Lab on Thursday 5 – 8 PM
- Readings: 1.3, 2.1, 2.2, 2.3, 2.4
- Textbook???
- Quizzes: Drop two lowest
- **Department Survey – extra credit**

Documentation and Style

- Meaningful Names
- Comments
- Indentation
- Named Constants

Documentation and Style

- Most programs are modified over time to respond to new requirements.
- Programs which are easy to read and understand are easy to modify.
- Even if it will be used only once, you have to read it in order to debug it .

Meaningful Variable Names

- A variable's name should suggest its use.
- Observe conventions in choosing names for variables.
  - Use only letters and digits.
  - "Punctuate" using uppercase letters at word boundaries (e.g. `taxRate`).
  - Start variables with lowercase letters.
  - Start class names with uppercase letters.

Comments

- The best programs are self-documenting.
  - Clean style
  - Well-chosen names
- Comments are written into a program as needed explain the program.
  - They are useful to the programmer, but they are ignored by the compiler.

Comments

- A comment can begin with //.
- Everything after these symbols and to the end of the line is treated as a comment and is ignored by the compiler.

```
double radius; //in centimeters
```


Comments

- A comment can begin with `/*` and end with `*/`
- Everything between these symbols is treated as a comment and is ignored by the compiler.

```
/**
This program should only
be used on alternate Thursdays,
except during leap years, when it should
only be used on alternate Tuesdays.
*/
```

Comments

- A *javadoc* comment, begins with `/**` and ends with `*/`.
- It can be extracted automatically from Java software.

```
/** method change requires the
  number of coins to be nonnegative
  */
```


When to Use Comments

- Begin each program file with an explanatory comment
  - What the program does
  - The name of the author
  - Contact information for the author
  - Date of the last modification.
- Provide only those comments which the expected reader of the program file will need in order to understand it.

Indentation

- Indentation should communicate nesting clearly.
- A good choice is four spaces for each level of indentation.
- Indentation should be consistent.
- Indentation should be used for second and subsequent lines of statements which do not fit on a single line.

Indentation

- Indentation does not change the behavior of the program.
- Proper indentation helps communicate to the human reader the nested structures of the program

Some Terminology follows…

Class Loader

- A Java program typically consists of several pieces called *classes.*
- Each class may have a separate author and each is compiled (translated into byte-code) separately.
- A *class loader* (called a *linker* in other programming languages) automatically connects the classes together.

Programmer, User, Package…

- The person who writes a program is called the *programmer.*
- The person who interacts with the program is called the *user.*
- A *package* is a library of classes that have been defined already.
  - `import java.util.Scanner;`

Arguments, Variables…

- The item(s) inside parentheses are called *argument(s)* and provide the information needed by methods.
- A *variable* is something that can store data.
- An instruction to the computer is called a *statement*; it ends with a semicolon.
- The grammar rules for a programming language are called the *syntax* of the language.

Algorithms

- By designing methods, programmers provide actions for objects to perform.
- An *algorithm* describes a means of performing an action.
- Once an algorithm is defined, expressing it in Java (or in another programming language) usually is easy.

Algorithms

- An algorithm is a set of instructions for solving a problem.
- An algorithm must be expressed completely and precisely.
- Algorithms usually are expressed in English or in *pseudocode.*

Object-Oriented Programming

- Our world consists of *objects* (people, trees, cars, cities, airline reservations, etc.).
- Objects can perform *actions* which affect themselves and other objects in the world.
- Object-oriented programming (*OOP*) treats a program as a collection of objects that interact by means of actions.

OOP Terminology

- Objects, appropriately, are called *objects.*
- Actions are called *methods.*
- Objects of the same kind have the same *type* and belong to the same *class.*
  - Objects within a class have a common set of methods and the same kinds of data
  - but each object can have it's own data values.

Printing to the Screen

```
System.out.println ("Whatever you want to print");
```

- **`System`** is a class
- **`System.out`** is an object for sending output to the screen.
- **`println`** is a method to print whatever is in parentheses to the screen.

Printing to the Screen

- The object performs an action when you *invoke* or *call* one of its methods

```
objectName.methodName(argumentsTheMethodNeeds);
```

Simple Screen Output

```
System.out.println("The count is " + count);
```

- Outputs the sting literal `"the count is "`
- Followed by the current value of the variable `count`.

Keyboard and Screen I/O

- Keyboard input
- Screen output

Keyboard Input

- Java has reasonable facilities for handling keyboard input.
- These facilities are provided by the `Scanner` class in the `java.util` package.
  - A *package* is a library of classes.

Simple Input

- Data can be entered from the keyboard using

```
Scanner keyboard =
    new Scanner(System.in);
```

followed, for example, by

```
eggsPerBasket = keyboard.nextInt();
```

which reads one `int` value from the keyboard and assigns it to `eggsPerBasket`.

Using the Scanner Class

- Near the beginning of your program, insert

```
import java.util.Scanner;
```

- Create an object of the `Scanner` class

```
Scanner keyboard =
   new Scanner (System.in)
```

- Read data (an `int` or a `double`, for example)

```
int n1 = keyboard.nextInt();
double d1 = keyboard,nextDouble();
```


Some `Scanner` Class Methods

Scannner_Object_Name`.next()`
Returns the `String` value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are whitespace characters.

Scannner_Object_Name`.nextLine()`
Reads the rest of the current keyboard input line and returns the characters read as a value of type `String`. Note that the line terminator `'\n'` is read and discarded; it is not included in the string returned.

Scannner_Object_Name`.nextInt()`
Returns the next keyboard input as a value of type `int`.

Scannner_Object_Name`.nextDouble()`
Returns the next keyboard input as a value of type `double`.

Scannner_Object_Name`.nextFloat()`
Returns the next keyboard input as a value of type `float`.

Some `Scanner` Class Methods

Scannner_Object_Name`.nextLong()`
Returns the next keyboard input as a value of type `long`.

Scannner_Object_Name`.nextByte()`
Returns the next keyboard input as a value of type `byte`.

Scannner_Object_Name`.nextShort()`
Returns the next keyboard input as a value of type `short`.

Scannner_Object_Name`.nextBoolean()`
Returns the next keyboard input as a value of type `boolean`. The values of `true` and `false` are entered as the words *true* and *false*. Any combination of uppercase and lowercase letters is allowed in spelling *true* and *false*.

Scannner_Object_Name`.useDelimiter(`*Delimiter_Word*`);`
Makes the string *Delimiter_Word* the only delimiter used to separate input. Only the exact word will be a delimiter. In particular, blanks, line breaks, and other whitespace will no longer be delimiters unless they are a part of *Delimiter_Word*.

This is a simple case of the use of the `useDelimiter` method. There are many ways to set the delimiters to various combinations of characters and words, but we will not go into them in this book.

`nextLine()` Method Caution

- The `nextLine()` method reads
  - The remainder of the current line,
  - Even if it is empty.

nextLine() Method Caution

- Example – given following declaration.

```
int n;
String s1, s2;
n = keyboard.nextInt();
s1 = keyboard.nextLine();
s2 = keyboard.nextLine();
```

- Assume input shown

```
42
and don't you
forget it.
```

n is set to 42

but s1 is set to the empty string.

In-class Exercise 2

```
import java.util.Scanner;

public class addTwoNumbers {
    public static void main(String[] args) {
    System.out.println("Hello!");
    System.out.println("I will add two number for you.");
    System.out.println("Enter two whole numbers, one on each
line when prompted");

    int n1, n2;

    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter first number:");
    n1 = keyboard.nextInt();
    System.out.println("Enter second number:");
    n2 = keyboard.nextInt();

    System.out.println("The sum of those two numbers is");
    System.out.println(n1 + n2);
    }
}
```

Next class (Thu, Sep 4)

- Quiz 2