

# COMP 110-003

## Introduction to Programming

### *First Program*

January 15, 2013








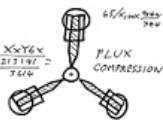



Haohan Li  
TR 11:00 – 12:15, SN 011  
Spring 2013



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL

# Learning Programming in 21 Days

<p><b>Days 1 - 10</b> Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...</p> 	<p><b>Days 11 - 21</b> Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ....</p> 	<p><b>Days 22 - 697</b> Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.</p> 
<p><b>Days 698 - 3648</b> Interact with other programmers. Work on programming projects together. Learn from them.</p> 	<p><b>Days 3649 - 7781</b> Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.</p> 	<p><b>Days 7782 - 14611</b> Teach yourself biochemistry, molecular biology, genetics,...</p> 
<p><b>Day 14611</b> Use knowledge of biology to make an age-reversing potion.</p> 	<p><b>Day 14611</b> Use knowledge of physics to build flux capacitor and go back in time to day 21.</p> 	<p><b>Day 21</b> Replace younger self.</p> 

As far as I know, this  
is the easiest way to  
"Teach Yourself C++ in 21 Days".



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL



# Today

---

- Overview of your first program
- Programming basics
- Review of your first program



THE UNIVERSITY  
*of* NORTH CAROLINA  
at CHAPEL HILL



# Miscellaneous

---

- Mid-term: Thursday, March 7<sup>th</sup>
  - Last lecture before Spring Break
- Final: **Saturday May 4, 12:00 PM**
- Typo in last lecture
  - TB = **Tera** Byte = 1,000,000,000,000 bytes
- Please install **Dropbox**. You can come to me or ask your friends about how to use it.



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL



# Miscellaneous

---

- Questions in the homework
  - What score will you get if
    - You finished all the assignments on time
    - But failed both exams
  - $60 + 59 \times 10\% + 59 \times 25\% + 5 = 85.65$  (B)
  - What score will you get if
    - You used all extensions but still missed all deadlines
    - But got full mark at both exams
  - $60 \times 50\% - 3 + 100 \times 10\% + 100 \times 25\% + 5 = 67$  (D+)





# Miscellaneous

---

- Questions in the homework
  - $60 + 59 \times 10\% + 59 \times 25\% + 5 = 85.65$  (B)
  - $60 \times 50\% - 3 + 100 \times 10\% + 100 \times 25\% + 5 = 67$  (D+)
- What if you haven't used the three extensions?
- You can use them to get the deducted half of credits from three assignments
  - So maybe 10-15 points higher, about 80 points, around C+ and B-
- The best way? **Submit every assignment on time!**



# Office Hour

---

- Official time:
  - Wednesday
  - 10:30 AM – 11:30 AM
  - 2:30 PM – 3:30 PM
- Unofficial walk-in time:
  - Tuesday, 12:30 PM – 2:30 PM
- By appointment:
  - Send me emails



THE UNIVERSITY  
*of* NORTH CAROLINA  
at CHAPEL HILL



# Self-Test Question

---

- Assume that your first program is in the memory. Now you compile it, then run it. It reads two numbers from the keyboard and output the result to the screen.
  - **What happens inside your computer during the procedure?**
  - Write down the answer, or type in on your computer.
  - Give the big picture. Several lines would be fine.
  - I will call someone to show your answer.





# Answer

---

- The compiler translates your program to instructions
  - CPU reads your Java program from memory
  - CPU does some calculation (or transformation)
  - CPU writes the results (instructions) back to memory
- Your program is now in memory as instructions
  - CPU reads these instructions from memory
  - The keyboard input is stored in a specific memory address
  - CPU reads the input and computes the result
  - CPU writes the result to another memory address (screen)



# Our First Program

---

```
import java.util.Scanner;

public class FirstProgram {
    public static void main(String[] args) {
        System.out.println("Hello out there.");
        System.out.println("I will add two number for you.");
        System.out.println("Enter two whole numbers on a line:");

        int n1, n2;

        Scanner keyboard = new Scanner(System.in);
        n1 = keyboard.nextInt();
        n2 = keyboard.nextInt();

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



# Import Packages

---

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

- **Import = borrow something from somewhere else**
- **Package = Library of tools**
  - *java.util* is a package that contains useful standard tools
    - *java.math* contains mathematical tools
    - *java.net* contains network connection tools
  - Scanner is one standard tool about keyboard inputting in this package



# Begin the Program

---

```
public class FirstProgram {  
    public static void main(String[] args) {
```

- Begin a program named ***FirstProgram***
  - Program names should make sense
  - Another name for this program could be ***AddTwoNumbers***
- You should always capitalize the first letter or each word in your program name



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL



# Output to Screen

---

```
System.out.println("Hello out there.");  
System.out.println("I will add two number for you.");  
System.out.println("Enter two whole numbers on a line:");
```

- This three lines write what in the quote to screen
  - Remember that `System.out.println("")` can write to screen.
  - Currently it's like a spell. It is magic to you now, but soon it will make sense.





# Read from Keyboard

---

```
int n1, n2;  
Scanner keyboard = new Scanner(System.in);  
n1 = keyboard.nextInt();  
n2 = keyboard.nextInt();
```

- With the help of ***Scanner***, we read two numbers from the keyboard
  - These two numbers are saved in certain memory locations, represented by ***n1*** and ***n2***



# Output Again, with Result

---

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

- Again, **System.out.println(“”)** can write to screen
- Pay attention that the second line write **the result** of **(*n1 + n2*)** to screen
  - There is **no** quotation mark, because you want the computer to output the result of ***n1+n2***, not the text “*n1+n2*”
  - **Computer programs are delicate and fragile**



# Let's See it Again

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

Get necessary tools

```
public class FirstProgram {  
    public static void main(String[] args) {  
        System.out.println("Hello out there.");  
        System.out.println("I will add two numbers");  
        System.out.println("Enter two whole numbers on a line:");
```

Name your program

Output guide information

```
int n1, n2;
```

```
Scanner keyboard = new Scanner(System.in);  
n1 = keyboard.nextInt();  
n2 = keyboard.nextInt();
```

Read two numbers  
from keyboard, and  
save them to memory

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

Calculate and output the result



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL



# Concepts Under the Hood

---

- Class: a piece of code we can use in a program
  - It is an abstract specification of a category
- Object: a piece of data/instructions in memory
  - It is a member of a class
  - It is something that actually exists (but still an abstraction)
- Example – if we need a program to record vehicle information in Chapel Hill
  - ***Car***: it can be a class
  - ***Alice's Car, Bob's Car***: they are objects, in the class of *Car*



# Concepts Cont'd

---

- Attributes
  - The characteristics of a class, and its objects
  - ***Car: Year, Make, Color, PlateNumber, Owner***, etc.
    - Remember these are specifications in a class
  - ***Alice's Car: 2003, Honda, Black, AAA-1234, Alice***, etc.
    - These are the actual **values** of the attributes.





# Concepts Cont'd

---

- Methods
  - Actions that can be performed by a class, and its objects
  - It is in the form of ***method()***, or ***method(some value)***
  - Possible methods in class *car*:
    - ***Total()***
      - Delete this car from the information system
    - ***Repaint()***
      - Change ***Color***. What will happen if we do ***Repaint("White")***?
    - ***Sell()***
      - Change ***Owner*** of the car
    - Etc.



# Meaningful Abstractions

---

- When you write code, make sure that the abstractions are useful for you to solve problems
  - If you are defining the class ***Car*** in a race game
    - What attributes shall you use?
    - You may need: ***Make, Color, Speed, Power, MaxSpeed, Weight***, etc.
    - What about methods?
    - You may need: ***Accelerate(), Brake(), Turn(), Turbo()***, etc.
  - Different abstractions due to different requirements
  - **It's impossible to include every piece of realistic details in a computer program. Select the meaningful ones.**



# Create an Object

---

- The format is
  - *ClassName* *ObjectName* = new *ClassName*();
- If you have defined a class *Car* and you want to create an object in this class
  - *Car* *MyCar* = new *Car*();
  - It means: Create an object (*MyCar*) of *Car* class
- Important tip:
  - In Java, “**x=y**” means “let x be equal to y”



# Call Attributes/Methods

---

- The format is
  - *ObjectName.Attribute*;
  - *ObjectName.Method(arguments)*;
- The **dot** invoke an attribute, or a method
- Still using the *Car* example
  - *MyCar.Owner* = “Haohan Li”;
  - *MyCar.Total()*; ☹
  - *MyCar.Repaint*(“White”);



# Primitive Types and Variables

---

- A primitive type is a special type of class
- A variable is a special type of object
- They are defined by Java Language
  - *int n1 = 10;*
  - *int* is a primitive type, and *n1* is a variable.
  - You don't need ~~*new int()*~~ to create a variable.
- They are designed to store basic data, and have assigned memory address
- They are the foundations of a program





# First Program Revisited

---

```
import java.util.Scanner;

public class FirstProgram {
    public static void main(String[] args) {
        System.out.println("Hello out there.");
        System.out.println("I will add two number for you.");
        System.out.println("Enter two whole numbers on a line:");

        int n1, n2;

        Scanner keyboard = new Scanner(System.in);
        n1 = keyboard.nextInt();
        n2 = keyboard.nextInt();

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



# Framework of a Program

---

```
import java.util.Scanner;  
  
public class FirstProgram {  
    }
```

- The *import* line get the necessary tool classes
  - *java* is a class, *util* is its attribute and is also a class, *Scanner* is *util*'s attribute and is also a class.
- *public class FirstProgram {}* defines a class. Our class is named *FirstProgram*.
  - We will learn the keyword “*public*” in the future



# Entrance of a Program

---

```
public class FirstProgram {  
    public static void main(String[] args) {  
    }  
}
```

- *main()* is called **the main method**. It is the entrance of a Java program
  - Every time you execute a Java program, the program starts as *YourClass.main()*;
  - Still, we will introduce *static*, *void* and *String[] args* in the future.



# Output to Screen

---

```
System.out.println("Hello out there.");  
System.out.println("I will add two number for you.");  
System.out.println("Enter two whole numbers on a line:");
```

- Not magical anymore?
  - **System** is an object representing the computer system, **out** is its attribute object, **println()** is a method of out
  - **println()** prints the texts to screen
  - Why don't we import **System**? Why don't we create **System**?
    - Because it is in every Java program **by default**



# Package

---

- In fact, *java.util* and *system* are both packages
  - A **package** is a collection of classes that have already been defined for you
  - Therefore, you can not actually create an object in class `java` or `system` like this:
    - ~~`java MyJava = new java();`~~
    - ~~`system newSystem = new System();`~~
  - They are only used to name a category of related classes
    - You can see packages as big classes. It doesn't hurt.





# Create Variables

---

```
int n1, n2;
```

- You create two integer type variables ***n1*** and ***n2***
  - Because you need them to save the input numbers



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL



# Create Objects

---

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

- This line creates an object *keyboard* in class `Scanner`
  - Why do we have to create the object this time?
  - Because *Scanner* can read information from somewhere other than a keyboard
    - It can read from a file
    - *Scanner file = new Scanner(new File("MyFile.txt"));*
      - This line will create a reader that reads texts from "MyFile.txt"
    - You can create multiple *Scanner* objects in a program



# Use Method to Read Keyboard Input

---

```
n1 = keyboard.nextInt();  
n2 = keyboard.nextInt();
```

- ***nextInt()*** is a method of class Scanner
- We use ***keyboard.nextInt()*** to invoke this method
  - This method returns an integer number from keyboard
- We use ***n1=*** to assign this number to n1
  - Recall that what “=” means



# Output the Result

---

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

- ***System.out.println()*** is used again
  - ***System.out.println("Some text")*** output the text
  - ***System.out.println(variable)*** output the value of that variable
    - ***System.out.println(n1+n2)*** outputs the value of ***n1+n2***



# Let's See it Again (and Again)

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

Get necessary tools

```
public class FirstProgram {  
    public static void main(String[] args) {  
        System.out.println("Hello out there.");  
        System.out.println("I will add two numbers");  
        System.out.println("Enter two whole numbers on a line:");
```

Name your program

Output guide information

```
int n1, n2;
```

```
Scanner keyboard = new Scanner(System.in);  
n1 = keyboard.nextInt();  
n2 = keyboard.nextInt();
```

Read two numbers  
from keyboard, and  
save them to memory

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

Calculate and output the result



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL



# Final Tip: Semicolon

---

- Is there anything missing?
  - The **semicolon**
    - *System.out.println("Hello out there.");*
  - Why do we need the semicolon?
    - Let the computer know it's the end of a line
    - When computer reads a program, it reads this:
      - *import java.util.Scanner; public class FirstProgram {public static void main(String[] args) {System.out.println("Hello out there."); System.out.println("I will add two number for you.");}}*
    - We write the programs in proper format so people can read it more easily





# Object-Oriented Programming (OOP)

---

- When designing a program, start with abstracting the objects that you will deal with
  - Objects have details (attributes)
  - Objects can perform actions which affect themselves and other objects in the world (methods)
  - Object-oriented programming (OOP) treats a program as a collection of objects that interact by means of actions



THE UNIVERSITY  
*of* NORTH CAROLINA  
at CHAPEL HILL



# Object-Oriented Programming (OOP)

---

- Compared to **Procedural Programming**
  - Different ways to abstract the world
  - Procedural programming is **temporal** abstraction;
  - Object-oriented programming is **spatial** abstraction.



THE UNIVERSITY  
*of* NORTH CAROLINA  
at CHAPEL HILL



# Car Information System

---

- Object-oriented programming thinks in objects:
  - The world **has** cars
    - Cars **have** *Make, Color, PlateNumber, Owner*, etc.
    - Cars **can** *Total(), Repaint(), ChangePlate()*, etc.
  - The world **has** drivers
    - Drivers **have** *Name, Age, Sex, LicenseNumber*, etc.
    - Drivers **can** *ChangeName(), GetOld(), RevokeLicense()*, etc.
  - Drivers and cars **interact**
    - A driver **can have** a car (or many cars)
    - A driver **can** buy/sell a car



# Car Information System

---

- Procedural programming thinks in procedures:
  - **First**, we input all the car and driver records into system
    - We need to save them as variables
  - **Second**, we update the records in case
    - If a car is repainted, we change the color variable of the car
    - If a car is totaled, we delete the record of the car
    - If a driver changes his name, we change the name variable of this driver
    - .....
  - **Finally**, in the end of every day/week/month, we backup/print/report all information



# Object-Oriented Programming

---

- OOP is easy to understand
  - We will learn some principles in OOP that are more complicated
- But we can not abandon procedural programming
  - Because we need procedural programming to write all the methods
  - Such procedures are called **algorithms**



THE UNIVERSITY  
*of* NORTH CAROLINA  
at CHAPEL HILL



# Algorithm

---

- A set of instructions for solving a problem
- 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) is usually easy





# PBJS Algorithm

---

- Get a slice of bread from loaf and put it on plate
- Repeat following two steps until you get enough peanut butter
  - Put knife into peanut butter jar and get peanut butter
  - Transfer peanut butter from knife to slice of bread
- Transfer other slice of bread from loaf to plate
- Repeat following two steps until you get enough jelly
  - Put knife into jelly jar and get jelly
  - Transfer jelly from knife to other slice of bread
- Put one slice of bread (pb side down) on other slice of bread
- Enjoy!



# Pseudocode

---

- Combination of code and English used to express an algorithm **before** writing algorithm into code
  - As long as it's not actual code, it can be called pseudocode
  - Programmers communicate through pseudocode
    - Some of them insist that you should communicate through codes – you can call them geeks



# Additional Vocabulary

---

- Statements – instructions to the computer
  - Specifically, it means a line of code ended by a semicolon
- Syntax – grammar rules for a language
  - As you've seen
    - Start with ***import***
    - Define the class with ***public class ClassName{}***
    - Assign a value with **=**
  - We will learn them progressively by examples
    - You learn a language by reading and writing, not by a grammar handbook



# Helpful Tips for Lab 1

---

- The primitive type ***int*** can only handle integer numbers. You need ***double*** to handle numbers having a decimal point and a fractional part after the decimal point.
- The symbol “\*” represents multiplying.
- Comment
  - “/\* ***something*** \*/” means texts between “/\*” and “\*/” will not be ignored by computer
  - “// ***something***” means this line will be ignored



# Assignments For This Week

- Sign Honor Pledge
- Install Java and Eclipse
- Prepare for Lab 1

**“If the authors  
of computer  
programming  
books wrote  
arithmetic  
textbooks”**

Suppose you have one rabbit.



Now suppose someone gives you one more rabbit.



Now, if you count your rabbits, you have two rabbits. So one rabbit plus one rabbit equals two rabbits. So one plus one equals two.

$$1 + 1 = 2$$

And that is how arithmetic is done.

Now that you understand the basic idea behind arithmetic, let's take a look at a simple easy-to-understand example that puts into practice what we just learned.

Try It Out  
Example 1.7

$$\log \Pi(N) = \left(N + \frac{1}{2}\right) \log N - N + A - \int_N^{\infty} \frac{\overline{B}_1(x) dx}{x}, \quad A = 1 + \int_1^{\infty} \frac{\overline{B}_1(x) dx}{x}$$

$$\log \Pi(s) = \left(s + \frac{1}{2}\right) \log s - s + A - \int_0^{\infty} \frac{\overline{B}_1(t) dt}{t + s}$$

$$\begin{aligned} \log \Pi(s) &= \lim_{n \rightarrow \infty} \left[ s \log(N+1) + \sum_{n=1}^N \log n - \sum_{n=1}^N \log(s+n) \right] \\ &= \lim_{n \rightarrow \infty} \left[ s \log(N+1) + \int_1^N \log x dx - \frac{1}{2} \log N + \int_1^N \frac{\overline{B}_1(x) dx}{x} \right. \\ &\quad \left. - \int_1^N \log(s+x) dx - \frac{1}{2} [\log(s+1) + \log(s+N)] \right. \\ &\quad \left. - \int_1^N \frac{\overline{B}_1(x) dx}{s+x} \right] \\ &= \lim_{n \rightarrow \infty} \left[ s \log(N+1) + N \log N - N + 1 + \frac{1}{2} \log N + \int_1^N \frac{\overline{B}_1(x) dx}{x} \right. \\ &\quad \left. - (s+N) \log(s+N) + (s+N) + (s+1) \log(s+1) \right. \\ &\quad \left. - (s+1) - \frac{1}{2} \log(s+1) - \frac{1}{2} \log(s+N) - \int_1^N \frac{\overline{B}_1(x) dx}{s+x} \right] \\ &= \left(s + \frac{1}{2}\right) \log(s+1) + \int_1^{\infty} \frac{\overline{B}_1(x) dx}{x} - \int_1^{\infty} \frac{\overline{B}_1(x) dx}{s+x} \\ &\quad + \lim_{n \rightarrow \infty} \left[ s \log(N+1) + \left(N + \frac{1}{2}\right) \log N \right. \\ &\quad \left. - \left(s + N + \frac{1}{2}\right) \log(s+N) \right] \\ &= \left(s + \frac{1}{2}\right) \log(s+1) + (A-1) - \int_1^{\infty} \frac{\overline{B}_1(x) dx}{s+x} \\ &\quad + \lim_{n \rightarrow \infty} \left[ s \log(N+1) + \left(N + \frac{1}{2}\right) \log N - \left(s + N + \frac{1}{2}\right) \log(s+N) \right] \end{aligned}$$



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL

