

CS 302 Lecture: Chapter 1

-If you read the text, you will see different examples than I give. This is intentional as having multiple examples of concepts was always helpful to me when I learned this material.

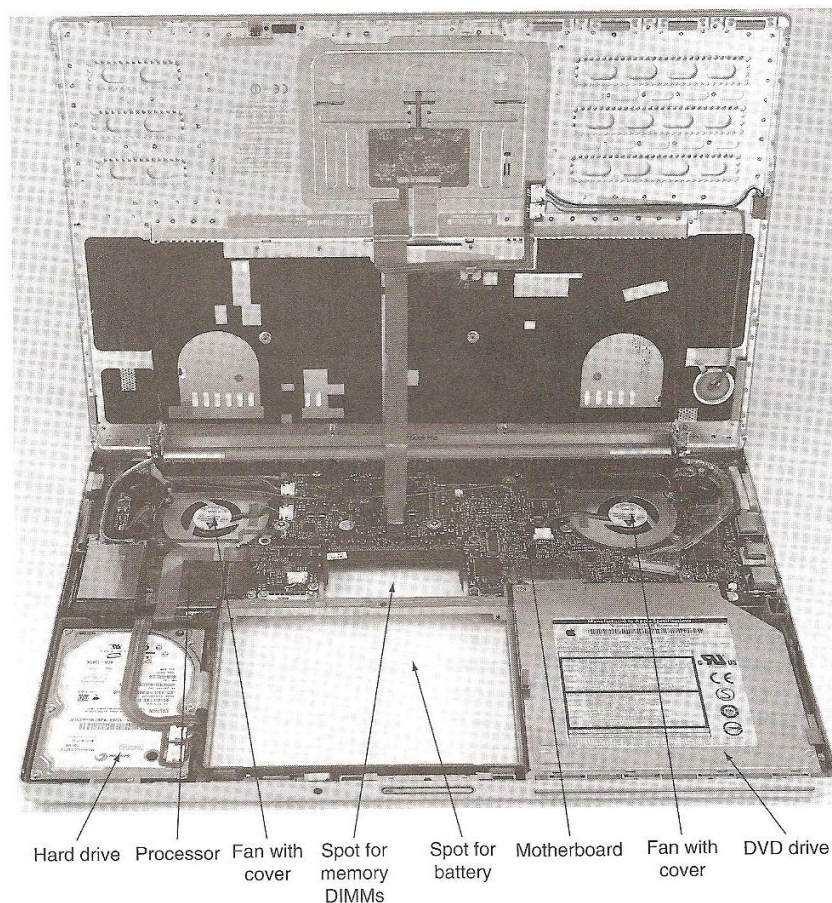
-Some things we will only touch on. We will either explain them in detail later in this class, or it is info that we want you to have an idea of now but is covered in another class.

-When we get to more advanced topics in the book, I will not have time to cover everything that is in the book, so you should read the text as well as attending lecture.

I. Parts of a Computer

a. Hardware

i. Handout: image of parts inside a laptop



ii. Definition: The physical computer and peripheral devices

iii. It makes it easier to program when you know something about the hardware of your computer.

1. Especially in more complex languages you will learn later. Ex: in Assembly, you specify in your program which memory location to store each part of your program in. You do not want to overwrite the location

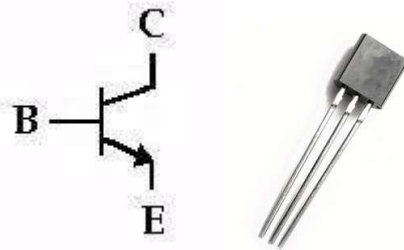
where the operating system is stored or your computer will stop working.

2. Don't worry. Java, as we will discuss later in this lecture, has safeguards to prevent you from harming your computer.
 3. We focus on the basics. You will get into more detail in a hardware focused class.
- iv. Physical computer (* means in text book)
1. Motherboard
 - a. The plastic board containing packages of integrated circuits (what runs the computer, made of transistors), CPU, memory, and I/O connection points.
 2. Also fan and battery or electrical plug connection are in a computer
 3. CPU*
 - a. Central Processing Unit, sometimes shortened to processor
 - b. It is the active part of the computer which locates and executes program instructions to the letter
 - i. processes data (addition, subtraction, etc)
 - ii. fetches data from external memory and stores it back
 - iii. Signals I/O devices to activate
 - c. Also contains its own small cache memory
 - i. (a small portion of the main memory. Think your desk is a cache for books from the library, it only has some of them-specifically the ones you need a lot right now). Cache memory is built of SRAM.
 - d. Parts specifically (software/hardware interface class)
 - i. Datapath (part of the CPU)
 1. Perform arithmetic operations
 - ii. Control (Part of the CPU)
 1. Commands datapath, memory, and I/O devices according to the program instructions
 4. Storage*
 - a. Primary storage (memory chips), RAM, memory
 - i. Handout: shows the memory(a bunch of DRAM chips on the board)



- ii. Made of transistors usually. (CPU also has transistors often)

- 1. The electronics drawing and an image of a physical transistor



- 2. Each has a base, collector, emitter
 - 3. Collector is more positive than emitter (In pnp transistors)
 - 4. Base-emitter and base-collector circuits act like diodes. $I_c = 100$ (100 is the typical number but could be 50 to 250). $I_e = I_c + I_b$. $V_b - V_e = .6$ Volts.
 - 5. Can amplify power: the output signal has more power than the input signal.
 - 6. Enable electrical signals to control other electrical signals, making automatic computing possible.
 - 7. I can provide more info if you ask me after class, or you can take an electric engineering class or a physics electronics class to learn more.

- iii. Fast to access

- iv. Expensive storage

- v. Your Java programs will be loaded into the memory when you click on the run button.
- vi. Can only store data when they are powered.
- vii. Two types (the RAM part means it takes the same amount of time no matter what portion is read, unlike magnetic tapes which are sequentially accessed)

- 1. DRAM

- a. Dynamic Random Access Memory
 - i. Built as an integrated circuit (device made of millions of transistors)
 - ii. Can access any memory location
- b. Several DRAMs are used together to contain the instructions and data of a program

- 2. SRAM

- a. Static Random Access Memory
 - i. Faster than DRAM, less dense, more expensive
 - ii. Used in CPUs cache memory.

- b. Secondary storage (hard disk)*

- i. slower to access
- ii. continues to store the data even when not powered
- iii. show mine (looks like a plain metal CD). This rotating part “platter” is coated in a magnetic material. The read and write heads can detect and change the magnetic flux on it to store information and read what is stored.
- iv. Page 4 had a picture of the whole hard disk:



- c. Removable storage like flash drive or CD
- v. Peripheral Devices*
 - a. Often called Input/Output devices (I/O)

b. Output

- i. Printer
- ii. Monitor (we will use this output soon)
- iii. Speakers

c. Input

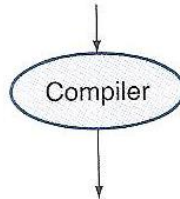
- i. Keyboard (we will learn to write programs that take keyboard input soon)
- ii. Mouse
- iii. Touch Screens allow touch as an input
- iv. Microphone

b. Software

- i. Computer Program definition: a sequence of instructions and decisions (can be as simple as if/else decisions)
 - 1. Stored on hard disk or a DVD or elsewhere
 - 2. Moved into memory when run so CPU can read it one line at a time. Then reads, modifies, and writes data as instructed. Also might output to screen or speakers or printer etc.
- ii. Programming definition: the act of designing and implementing computer programs. ie, writing code, in our case in Java.
- iii. High level language
 - 1. Definition (not in book): A portable language like C, C++, Java, Visual Basic that is composed of words and algebraic notation which can be translated by a compiler into a lower level language like assembly.
 - 2. Allows you to type in words you are used to like (in Java) print, if, new rather than 0s and 1s which the computer understands. This makes you much more productive and you can ignore many of the gritty details.
 - 3. Handout: example of translating high level language code into binary machine language

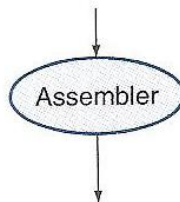
High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```



Assembly
language
program
(for MIPS)

```
swap:
  muli $2, $5, 4
  add $2, $4, $2
  lw $15, 0($2)
  lw $16, 4($2)
  sw $16, 0($2)
  sw $15, 4($2)
  jr $31
```



Binary machine
language
program
(for MIPS)

```
000000001010000100000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

- a. This example is in C, so we don't care what this program is doing really. And you don't need to learn the Assembly in this class.
 - b. Look how much longer it is in machine code. When you take more classes, you will know how to write your code in 0's and 1's but it is tedious and harder to find mistakes.
 - c. With even longer programs (like Java ones that are 100s of lines long), it becomes even more helpful that we have high level languages
4. Assembly language is not a high level language.
- a. You have to specify what memory location to store things in.

- b. You cannot just write 12^5 , you would have to actually do each of the multiplications.
 - c. The lw and sw in the example, mean load and store, so you can see you have to specify extra things in Assembly.
 - d. I found it interesting to learn about, but it makes you write very long programs even for simple tasks. This means it is harder to do complex tasks.
5. In Java we do not have to do the tedious steps because a compiler translates your Java code into what the computer needs.

We are now going to talk about a couple general programming ideas, before we start learning about Java.

6. Pseudo Code definition: informal description of a sequence of steps for solving a problem.
- a. You do not have to write in Java, just write what your steps will be in normal language.
 - b. Very helpful in the early stages of a program to make it clear to yourself how you will solve the problem. You might start with one idea and then change to another. It's nice to do in pseudo code so you don't have to rewrite all the Java.
7. Algorithm definition: a sequence of steps that is unambiguous, executable, and terminating.
- a. Unambiguous-precise instructions for each step and what to do next. (If this is true then jump to this part of the program, otherwise jump to here. Either way the program knows where it goes next).
 - b. Executable-all the needed information is there. (if there is a formula and you don't have one of the important values, then it cannot be calculated so it is not executable)
 - c. Terminating-does not run forever
 - d. An example of an algorithm.
 - i. Suppose you and a partner have just graded a pile of papers and now you have to sort them alphabetically. We can write an algorithm to do it.
 - ii. Each of you first sort your own students papers. We now have an A list $a_1, a_2, a_3, \dots, a_n$ and a B list $b_1, b_2, b_3, \dots, b_n$.
 - iii. Maintain a current pointer in each list, starting at the first elements in the list.
 - iv. Drawing an example helps with lots of program. We are pointing at the highlighted element. We copy the

smaller one into our new list and then point to the next item following the one we copied.

A= 1,5,7,8,9 B=2,4,6,7
C= 1 A=1,5,7,8,9 B=2,4,6,7
C= 1,2 A=1,5,7,8,9 B=2,4,6,7
C= 1,2,4 A=1,5,7,8,9 B=2,4,6,7
C= 1,2,4,5 A=1,5,7,8,9 B=2,4,6,7
C= 1,2,4,5,6 A=1,5,7,8,9 B=2,4,6,7
C=1,2,4,5,6,7 A=1,5,7,8,9 B=2,4,6,7
C=1,2,4,5,6,7,7 A=1,5,7,8,9 B=2,4,6,7..

Now that we have run out of items in list B, we copy the rest of A into C. and we are done.

C=1,2,4,5,6,7,7,8,9

v. Algorithm:

1. While both lists are non empty
 - a. Let a_i and b_i be the elements pointed to by their own pointers
 - b. Append the smaller one to the output list
 - c. Advance the current pointer in the list from which the smaller element was selected
2. Once one list is empty, append the remainder of the other list to the output.

e. A simpler algorithm example

- i. Suppose you want to find the maximum number in a list of n numbers. (this is desired in many situations)
- ii. Pseudo-code:
 1. Set Max to the first one in the list
 2. Look at each of the other elements in sequence. If the one you are looking at is $>$ max, change the value of Max. Continue until done with the list.

II. Java basics

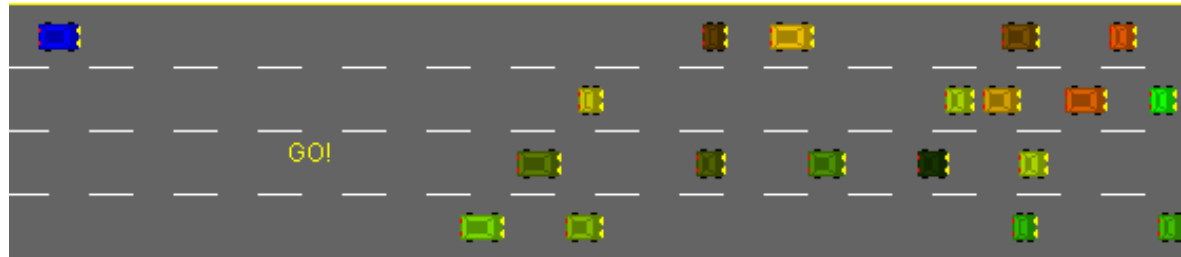
a. Safe to use and portable

- i. Rather than translate your Java into CPU instructions, it is translated into instructions for the Java Virtual Machine (a program that simulates a computer)
- ii. You cannot overwrite operating system information, but get error reports from the program if you do something wrong.

- iii. Also, this JVM means that you don't have to write code differently for mac and windows machines.
 - b. Libraries-code already written for many things you need a lot. (Math functions like sqrt are already defined and pi and e, Random class use for games or statistics, graphics packages for displaying pictures on the monitor). We will use some of these later on in the class.
 - c. Integrated Development Environment (IDE)
 - i. Definition-software program that allows you to write and test your code
 - ii. We will use Eclipse in this class. It has a window that shows your code you have written, the output that your program has generated including error messages, and a window with a list of all the files in your program.
 - d. Java is case sensitive so if you are doing math and make a variable x, it is not the same as X and Java will complain if you type the wrong one.
 - e. Strings must be enclosed in quotation marks (i below), otherwise the program will look for some other thing of the name you have (ii below). Book example is printing Hello World. If not in quotes computer looks for an item named Hello and can't find it.
 - i. `System.out.println("Hello");` //this one works
 - ii. `System.out.println(Hello)` //this one fails
 - f. Always end each statement with a semicolon just as English sentences end in a period. Example: `String name = "Alicia";`
 - g. Process of writing a program in Java
 - i. Write your code and edit it to fix any problems you noticed
 - ii. Compile. The compiler translates your code into code the JVM understands. Using a command line, you must perform this step but in Eclipse hitting run causes this to occur.
 - iii. Run the program
 - iv. Also always back up your code on a flash drive or save in multiple files and save frequently so it is not lost.
- III. Java classes and methods
- a. Files
 - i. Whenever you write a program, the code you actually type is stored in a filename.java file. This is called the "source code."
 - ii. There is also a filename.class file that the compiler creates when it translates your code into code the JVM can read.
 - b. Class
 - i. Definition: the fundamental building block of a Java program.
 - ii. Declaration:


```
public class NameOfClass
{
    //body of class
}
```

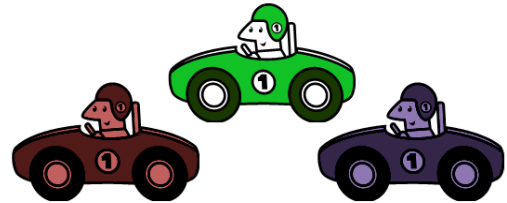
iii. Instantiable class example: a race-car driving game



1. What do we need for our game?
 - a. Cars
 - b. A race track
 - c. Possibly coins that can be collected for points
 - d. Possibly hazards that must be avoided
2. Each of these will become a class (blueprint for a specific object)
3. Car class



Blueprint of car



Instances of the car class

iv. Instantiable Class example 2.

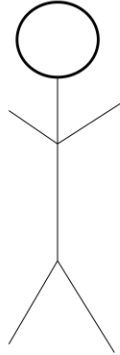
1. Suppose I am a CEO of a company and now I must pay all my workers, but I want the computer to do it.
2. I make a class file called Employee. It will contain things like a name, and ID, perhaps a wage, and other info. But it does not as yet represent a specific employee, just what an employee is in general.

```

public class Employee
{
    private String name;
    private double wage;
    private double hoursWorked;

    //possibly more data and methods
}

```



- Now to pay my employees, I make another class file, this one called Paychecks. In this file, I will calculate how much each Employee has earned and perhaps have the program write the check as well.
- I need to create specific Employees now, these are objects. Say I want to pay Mary and Joe first. (lower case names, 2 different objects b/c although both are employees, they are clearly not the same employee, and each should receive their own check. We sometimes must give information to new objects-here we must give the names)

```

Employee mary = new Employee("Mary");
Employee joe = new Employee("Joe");

```

- Now actual Employee objects exist.

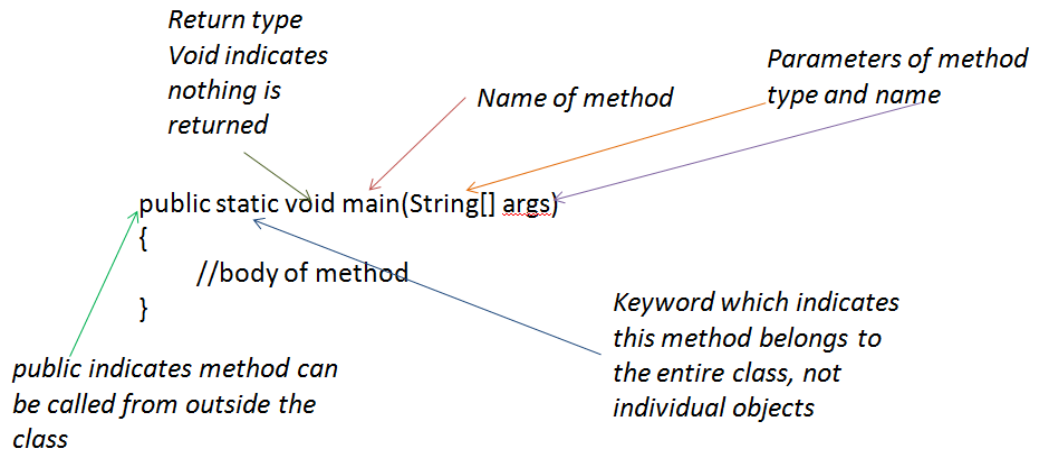


- From here on in my Paychecks.java file, I can refer to the objects mary and joe and the computer will know all the information in the Employee class that pertains to this object.
- Now, the first programs you write will have only one class file and you will not need to make any objects. For me this made the idea of objects seem unimportant and confusing at first. I wanted to present an

example where you can see how they are useful now. We will study objects in more detail later.

c. Methods

- i. The code that controls the program's actions
- ii. main method.
 - 1. This method is automatically run when the program is started. It must start with the line `public static void main(String[] args) {}` with your instructions inside the brackets.
 - 2. Required in non-instantiable classes
- iii. There may be other methods as well in your program.
 - 1. Example: writing an online banking program you might have one method that tells computer what to do when adding money and another when taking out money.
- iv. Whenever you create a method, you must specify the name (by convention, it starts with a lower case letter) and the information to be passed to it when it is used.
 - 1. The book uses the `System.out.println()` method as the example. You must tell it what you want to print, by putting that inside the parentheses.
 - 2. Often, you want to print some words to the screen as in `System.out.println("Hello, World");`
- v. Declaration:



IV. Analyzing a Java program

```
public class RightTriangle
{
    public static void main(String[] args)
    {
        double hypot = getHypot(1.1, 2.1);
        System.out.println(hypot);
    }
} //end main
```

```

public static double getHypot(double a, double b)
{
    double aSq = a*a;
    double bSq = b*b;
    double cSq = aSq+bSq;
    return Math.sqrt(cSq);
} //end getHypot
} //end class

```

- a. Suppose you have a geometry program you are working on and often need to find the hypotenuse of a right triangle with given side lengths. In a real geometry application, there would probably be other things you want to find as well, but we will not address those in this short example.
- b. We know the formula $a^2+b^2=c^2$ and this program does that
- c. We declare a class for this program
- d. Contains a main method with the required header.
- e. Calls method getHypot() and stores the result in a double variable named hypot. This method needs the a and b values to calculate c. Math works. Returns value. Prints value to screen. Result is 2.37065...
- f. We will learn more about writing methods in a later chapter

V. Errors

- a. Compile Time Errors
 - i. Definition: A violation of the programming language rules that is detected by the compiler.
 - ii. The compiler complains that it has no idea what you mean by something that you wrote in your code.
 - iii. Also called a syntax error (the way you wrote your Java 'sentence' is wrong)
 - iv. Example problem we could have in the program we just wrote:

```
System.out.println(hypot);
```

Problem: println is not a java command, it is println so the method will not work correctly
 - v. Example two:

```
System.out.println(hypo);
```

Problem: hypo is not defined anywhere in our code
 - vi. In Eclipse, red lines will underline whatever is causing the error and a little red x will appear along the left side of the code. You can mouse over the x and it will tell you what the problem is. In our example here it will say "the method println is undefined" and "hypo cannot be resolved to a variable."
 - vii. The compiler will not compile your program, that is it will not translate your code into code that the JVM requires, so you cannot run your program.
 - viii. These are common errors, especially if they are just typing errors like our second example. Even experienced programmers make these mistakes and

have to fix them. That is the nice thing about using Eclipse, not just typing code in a text editor, you are informed of your syntax errors along the way.

b. Run Time Errors

- i. Definition: a mistake that causes a program to take an action that the programmer did not intend.
 - ii. The program is syntactically correct, that is you wrote the Java language correctly, and it will compile. But you made a mistake somewhere in what you told the computer to do so the result will still be incorrect.
 - iii. Also called a logic error (because there is a logical flaw in the program).
 - iv. Example:
Suppose we forgot and returned cSq rather than the square root. This is not a problem in Java, but then we are saying $c = b^2 + a^2$ which is not the correct formula. Our results will come out wrong because we coded the formula wrong.
 - v. Exceptions are sometimes generated if there is a severe enough problem. An exception is a special kind of error message. The program will stop running at the point the exception was generated and a message will print onto the screen. It will tell you what line of your program the exception came from. You can also click on the part that tells you the line and it will highlight that line in your code. We will learn how to 'catch' exceptions and prevent them from crashing your program later.
 - vi. Example with an exception:
Between the two lines we have in the main method, add
`System.out.println(1/0);`
The problem here is that mathematically we are not allowed to divide by 0. The program will stop without telling you what hypot was and print "Exception in thread "main" java.lang.ArithmeticException: / by zero at RightTriangle.main(RightTriangle.java:8) " Arithmetic type of Exception, with an error message of '/ by 0.'
 - vii. These will not be found by the compiler b/c compiler is looking for 'can I translate this into computer's language?'. This is one reason you should always test your programs before handing them in. By test I mean, with our RightTriangle class, compute some c values yourself and test if your program gives the same values. If the results are incorrect, look at your code again and make sure it is doing what you meant it to when you wrote it and that you didn't make a mistake.
- c. We will learn about using debuggers to help us fix mistakes and we will also learn other tools to help see where the problem is in a more complex program. Often I will use `System.out.println()` of values at different points in the program to see if they are what they should be for a given example I have worked out by hand to test.

