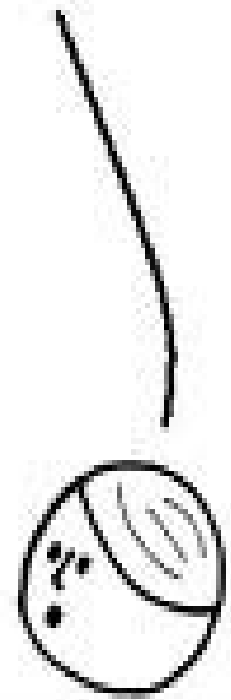




HOW DOES COMPUTER PROGRAMMING WORK?

MAGIC.



陈鹤良



Last Class

- Introduction/Administrative details
- How to go about taking a big problem and making it smaller
- Kinds of things that can happen when we have an error in our program.
- Introduction to how computers work
- HelloWorld.java ---how to set up and run a program



Assignment 1

- I'll be posting assignment 1 sometime in the next couple of days. Stay tuned through either webct or the course webpage
-
- <http://www.cs.mcgill.ca/~dpomer/comp202/summer2011>
-
- -Some written questions
- -Some short programming questions



Today

- How computers store things
- Binary numbers
- Review of HelloWorld.java
- Performing math in Java
- Storing complicated results in Java using variables.



Storing whether it is afternoon or morning

- Pick one electrical switch in memory.
- Whenever it is “on” it is AM
- Whenever it is “off” it is PM
-
- A computer stores billions or trillions of these “switches” By combining many of these, we can control many things.
-
- Note: This is just an example. Your computer probably stores this in an entirely different way.



Encoding the day of the week

- How could we encode the day of the week?
-
- If we just use 1 switch, there will not be enough information.
-
- How many “switches” will we need?




Storage is Exponential

- In general, if there are n possible values to store, we can encode it using
-
- $\log_2(n)$ “switches”
-
- Of course, there is no such thing as a fraction of a switch, so we will always have to round up.
-
- Put another way, if we have n switches, we can store 2^n values



Bits = Switch

- 1 “bit” is the same thing as a “switch”
- It has one or two values “on” or “off”
- For simplicity of notation, we will often just refer to these as 1 (on) and 0 (off)
-
- If you like, you could call them “true/false,” “yes/no,” “oui/non,” or “cats/dogs”




Byte = 8 bits

- A byte is simply 8 bits
-
- Question: How many possible values can we store in a byte?




Other Memory Units

- A kilobyte is 2^{10} bytes (1024 bytes)
- A megabyte is 2^{10} kilobytes (1024 bytes)
- Strangely, a gigabyte is just 1,000,000,000 bytes (this was a marketing decision)
-




Storing a number in a computer

- We are used to storing numbers in what is known as “base 10”
-
- What this means, is that every single digit has 10 possible values
-
- 0,1,2,3,4,5,6,7,8,9



Storing a number in a computer

- When we look at a base 10 number, we think of each digit as representing different amounts
-
- 8415
-
- really is:
- 5 ones
- 1 ten
- 4 hundreds
- 8 thousands
- 0 ten-thousands
- 0 hundred-thousands
-



Storing a number in a computer

- When we look at a base 10 number, we think of each digit as representing different amounts

- 8415

- really is:

- 5 ones = $5 * 10^0$

- 1 ten = $5 * 10^1$


- 4 hundreds = $5 * 10^2$

- 8 thousands = $5 * 10^3$

- 0 ten-thousands = $5 * 10^4$

- 0 hundred-thousands = $5 * 10^5$

-




Storing a number in a computer

- But what is special about 10?




Storing a number in a computer

- But what is special about 10?
-
- A reasonable conclusion to make is that we chose to use 10 digits since we have 10 fingers. This makes it a more natural counting system.



Storing a number in a computer

- Considering that computers think in a sequence of on/off switches, what would be a natural way to count in a computer?



Storing a number in a computer

- Considering that computers think in a sequence of on/off switches, what would be a natural way to count in a computer?
-
- Base 2 (a.k.a. binary)
-
- In this case we will have 2 choices of digits :
-
- 0 and 1



Storing a number in a computer

- 111001111

-

- We can now do the same thing to this number:

-

- 1 ones = $1 * 2^0$

- 1 twos = $1 * 2^1$

- 1 fours = $1 * 2^2$

- 1 eights = $1 * 2^3$

- 0 16s = $0 * 2^4$

- 0 32s = $0 * 2^5$

- 1 64s = $0 * 2^6$


- 1 128s = $0 * 2^7$

- 1 256s = $0 * 2^8$



Storing a number in a computer

- The following number is in base 2. What would it represent in base 10?
-
- 10011



Storing a number in a computer

- The following number is in base 2. What would it represent in base 10?
-
- 10011
-
- $= 1 * 2^0 + 1 * 2^1 + 1 * 2^4 = 19$



Other counting systems

- We can do this with many other numbers, although we don't do it as often:
-
- What is
-
- 10011 in base 3?



Other counting systems

- We can do this with many other numbers, although we don't do it as often.
-
- Ex: Hexadecimal : 16 digits (0-9 plus a-f)
-
- What is
-
- 10011 in base 3?
-
- $1 * 3^0 + 1 * 3^1 + 1 * 3^4 = 85$



Converting a number to binary

- To convert a number from base 10 to binary, you can use a “greedy” algorithm:
-
- Basically, try to take as many from the bigger columns as possible (this works for going to other bases as well)



Example: Convert 523 to binary

- Start with big powers of 2:
-
- $2^{10} = 1024$ ---> this doesn't fit
- $2^9 = 512$ ---> this fits. So we write a 1 in the 2^9 column (i.e. the 10th column from the right).
Subtracting this, we are left with 9
- $2^8 = 256$ --> doesn't fit into 9
- $2^7 = 128, 2^6 = 64, 2^5 = 32, 2^4 = 16$, doesn't fit
- $2^3 = 8$ ---> fits into 9, write a 1 in the 2^3 column.
Left with 1
- $2^2, 2^1$ --> don't fit.
- 2^0 fits.



Example: Convert 523 to binary

• 1000001001



Converting from arbitrary base to another

- What if I wanted to convert a base- n number to base- m
-
- A good way to do this is to first convert base- n to base-10 and then convert the base-10 number to base- m



Arithmetic in other bases

- To do arithmetic in other bases, you use the same procedures that we learned in kindergarten.
-
- The only difference is that you have to remember the possible digits are different.
-
- This means, for example, that if you are adding numbers in binary, and you get “2” that you actually have to write “10” ---> But this means you will most likely have to carry a number!



Congratulations

- You now can understand an incredibly nerdy computer science joke:
-
- Why do computer scientists always confuse Christmas and Halloween?



Congratulations

- You now can understand an incredibly nerdy computer science joke:
-
- Why do computer scientists always confuse Christmas and Halloween?
-
- Because DEC 25 = OCT 31



Part 3: Programming Languages



Programming Languages (1)

- We need to express our ideas in a form that a computer can understand: a program
- A *programming language* specifies the words and symbols that we can use to write a program
 - e.g. “red” belongs to English; “rouge” belongs to French
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*
 - e.g. “Banana red and” is not a valid statement in English.



Programming Languages (2)

- Computers are very intolerant of incorrect programming language statements
 - Humans are much more tolerant of incorrect natural language statements
 - You understand “The kiten is cute” even though kitten is misspelled.



Syntax and Semantics

- The *syntax rules* of a language define what words and symbols are valid in this language, and how they can be combined to make a valid program
 - “The kiten is cute” is not **syntactically** correct.
- The *semantics* of a program statement define what those words, symbols, and statements mean (their purposes or roles in a program)
 - “Banana red and.” is not **semantically** correct.



Machine Language

- Each instruction that a CPU understands is represented as a different series of bits
 - The set of all instructions that a CPU understands directly forms the *machine language* for that CPU
- **Each CPU type understands a different machine language**
 - In other words, for each different model of CPU, a given series of bits could mean a different instruction
 - For example, on an x86-compatible CPU (Intel, AMD), the series of bits `10101010` could mean `ADD`, while on a PowerPC CPU (old Macs, PlayStation 3) it could mean `LOAD`



Machine Language Example

- Here are the first 20 bytes of a machine language program that:

- asks the user to enter an integer value using the keyboard

- reads this value from the keyboard

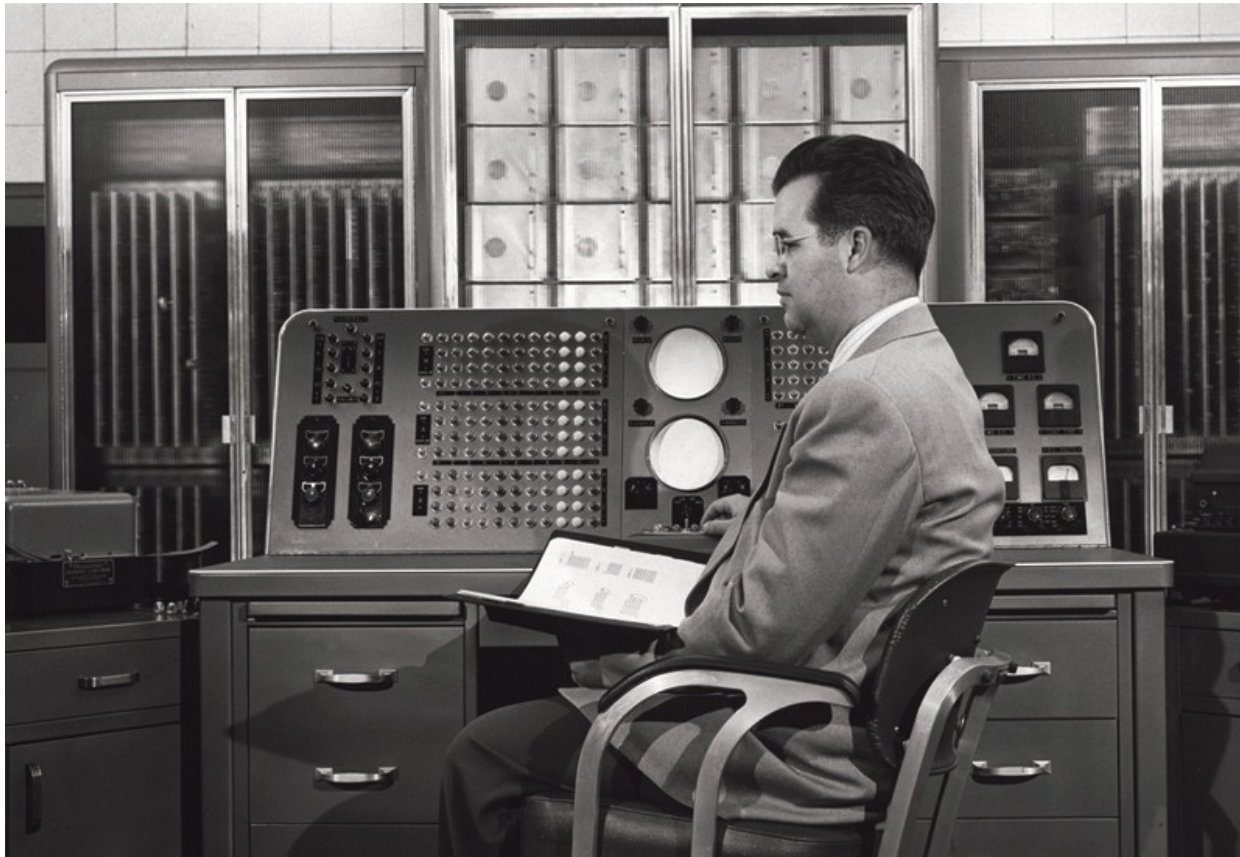
- adds one to this value, and

- displays the new value to the screen

```
01111111 01000101 01001100 01000110 00000001
00000001 00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000010 00000000 00000011 00000000
```

More the 6500 bytes in total!

Do you think it would be fun or easy to write a program in





Machine Language

Disadvantages

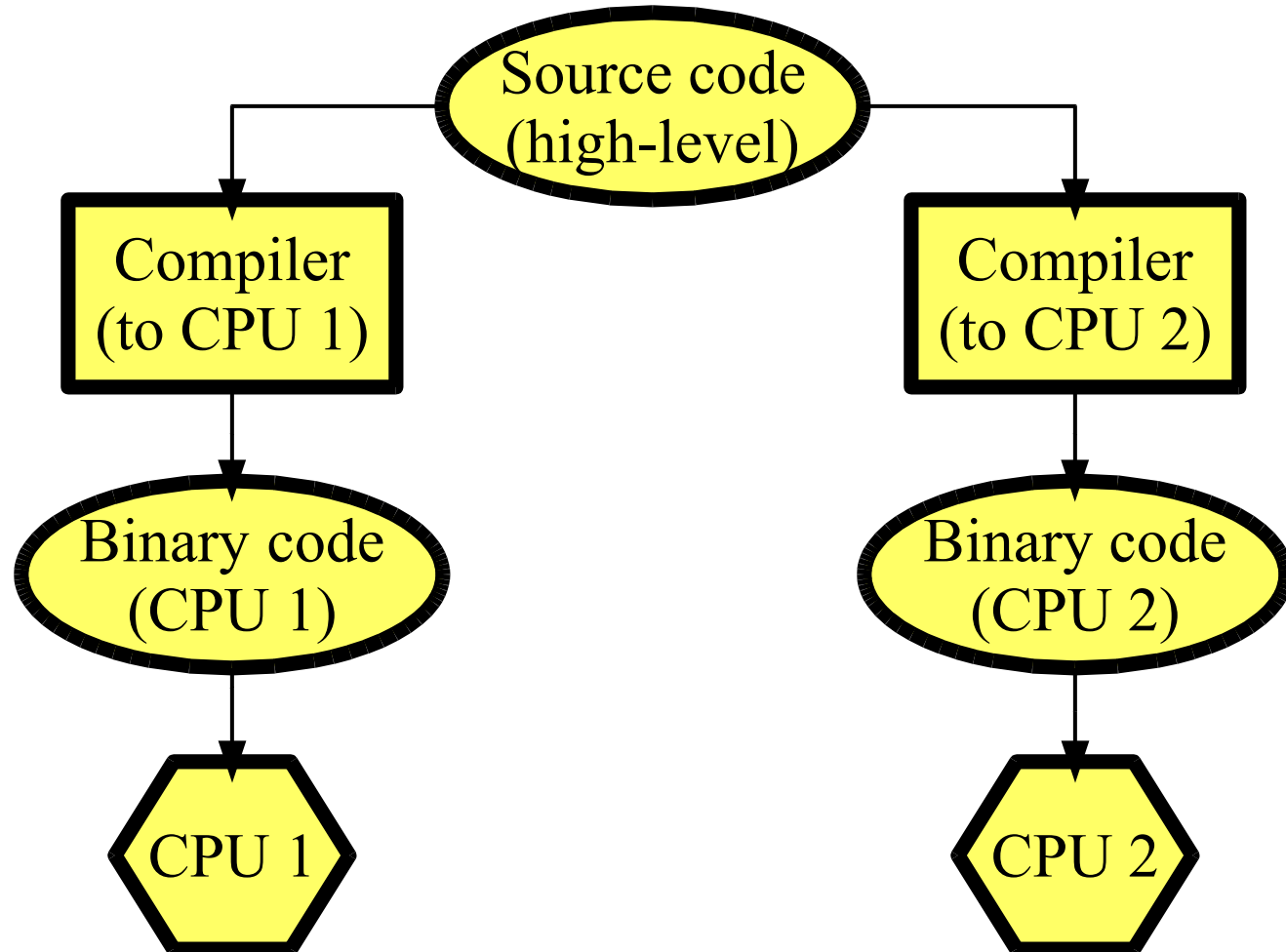
- Very tedious and confusing: machine language is extremely difficult for humans to read
- Error-prone
 - If you change one bit from 1 to 0 (or vice-versa), or forget a bit, your program's behavior will likely be not even close to what you expected
 - Moreover, errors are hard to find and correct
- Programs are not *portable*
 - Running the program on a different processor or CPU requires a complete rewrite of the program



High-Level Languages (1)

- To make programming more convenient for humans, *high-level languages* were developed
- No CPU understands high-level languages directly
 - Programs written in these languages must all be translated in machine language before a computer can run them (that's what a **compiler** is for)
- Basic idea:
 - Develop a language that looks like a mix of English and mathematical notation to make it easier for humans to read, understand, and write it
 - For each CPU type, develop a program that translates a program in high-level language to the corresponding machine language instructions (**a compiler**)

Compilers

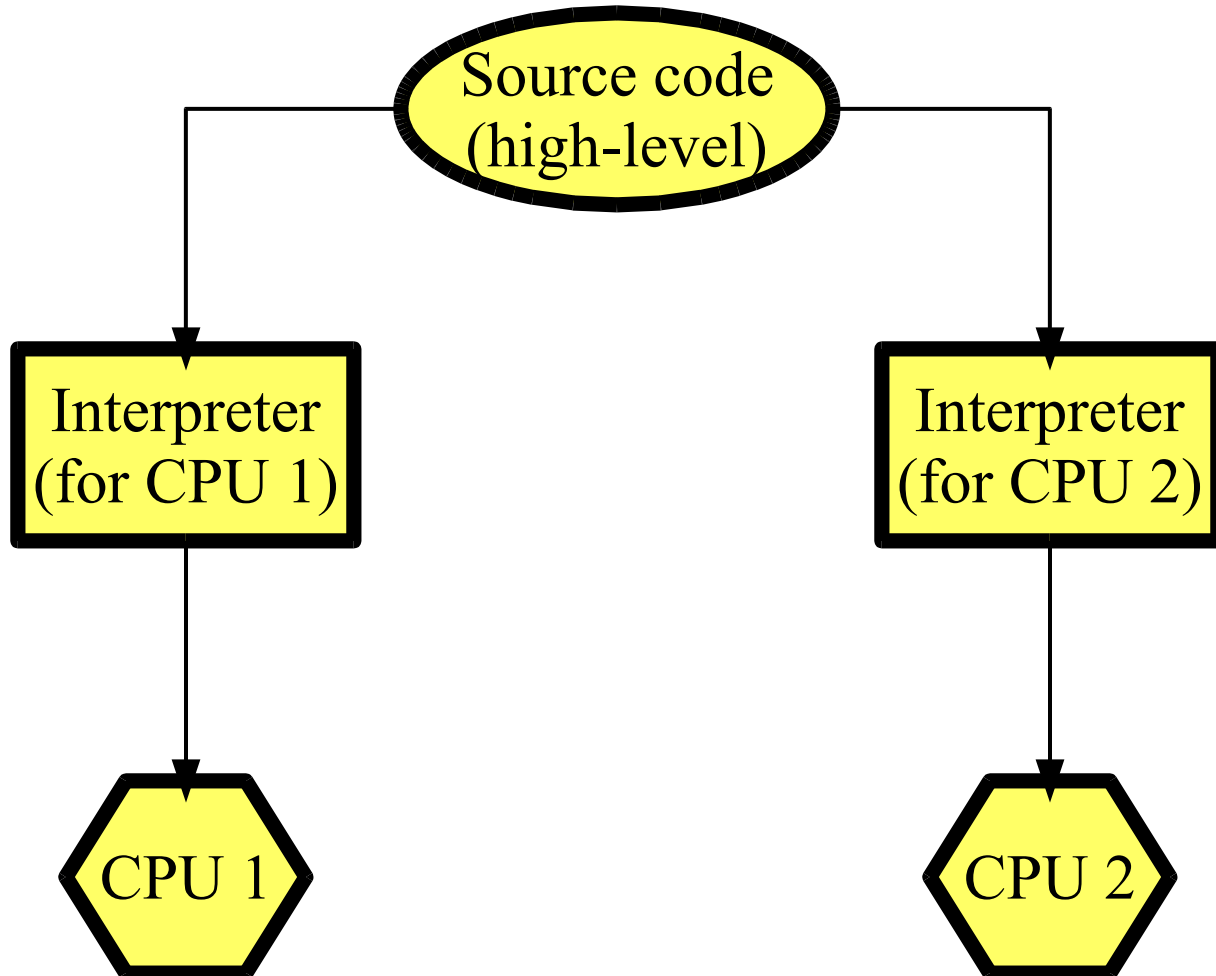




Interpreters (1)

- An *interpreter* is another kind of program. It takes source code and translates it into a target language
 - However, the target language instructions it produces are executed **immediately**
 - No executable file is created**

Interpreters (2)



Java combines a compiler with an interpreter

- Java **compiler** (javac, included in JDK 6) takes source and translates it into **bytecode**

foo.java $\xrightarrow{\text{javac}}$ foo.class
(Java) (bytecode)

foo.class can then be executed using an **interpreter**, the Java Virtual Machine (JVM)

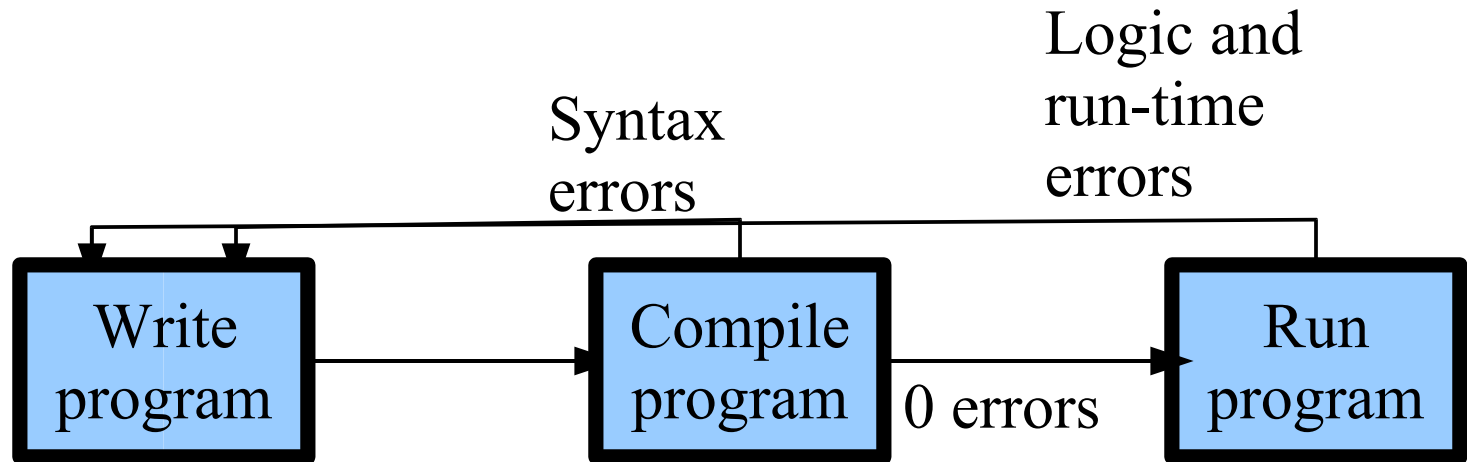


Programming Errors

- A program can have three types of errors
- **Compile-time errors:** the compiler finds problems with syntax and other basic issues
- **Run-time errors:** a problem occurs during program execution, and causes the program to terminate abnormally (or *crash*)
 - Division by 0
- **Logical errors:** the program runs, but produces incorrect results


- ```
celcius = (5.0 / 9.0) * fahrenheit - 32;
// Incorrect equation; should be
// (5.0 / 9.0) * (fahrenheit - 32)
```

# Development Life Cycle



- Errors may take a long time to debug!
  - Important Note:** When you compile for the first time and see 150 errors, do not despair. Only the first 1 or 2 errors are relevant. Fix those and compile again. There should be fewer errors (like 50). Repeat until there are no more errors

# Exercises to practice this at home

- 
- 
- A) Practice breaking the following tasks into smaller pieces. Make sure the pieces are small enough that you can manage them.
    - 1) Choosing what channel to watch on TV
    - 2) Writing a 5 paragraph essay
    - 3) Studying for an exam
    - 4) Arguing a speeding ticket in court
    -
  - B) Look at the resources on the course website and try to compile the HelloWorld program
    - Note: The file MUST be called HelloWorld.java (case-sensitive) or else the program will not compile
  - C) Take any number in base 10 and convert it to several other bases.



# COMP-202

---

## Unit 1: Introduction to Java

### CONTENTS:

Java Basics  
Variables



# Java Program: Hello World

---

```
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello World!");
 }
}
```

To print something to the screen, you would write a program as above.

This should be inside a file called HelloWorld.java

You then compile it by typing `javac HelloWorld.java`

You can run it by typing `java HelloWorld`



# Java Program: Hello World

---

```
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello World!");
 }
}
```

The first thing is that every program in java MUST be inside of a **class**. We'll go into more detail on what this means, but you can think of a class as grouping things together

"public class NAMEOFFILE { " in it.

(Note: we'll see later that this isn't always on the first line though!)





# Java Program: Hello World

---

```
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello World!");
 }
}
```

Inside a class, there are (usually) 1 or more methods. A method is simply a group of instructions to Java that can have both an input and an output. Conceptually, it is like a function in math.

In this case, the method is main()



# Java Program: Hello World

---

```
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello World!");
 }
}
```

Every Java program you ever write has to have a main method.

Not every class that you ever write in Java will have a main method.

If you don't have a main method, you can compile your class, but you can't run it.



# Java Program: Hello World

---

```
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello World!");
 }
}
```

Experiment: See what happens if you change

`public static void main(String[] args)` to

`public static void Main(String[] args)`

Try

1) `javac HelloWorld.java`

2) `java HelloWorld`



# Java Program: Hello World

---

```
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello World!");
 }
}
```

This method is called main because main is written before (String[] args).

We often will call this “the main method”



# Java Program: Hello World

---

The main method can consist of many statements or instructions

```
public class KnockKnock {
 public static void main(String[] args) {
 System.out.println("Knock Knock!");
 System.out.println("Who's there?");
 System.out.println("Java");
 }
}
```



# Java Program: Hello World

---

A class can consist of many methods. Here is a class with many methods. They are called elmer, bugs, Bugs, daffy, and yosemitesam

```
public class LoonieToons {
 public static void bugs() { }
 public void Bugs() { } //no link to bugs
 public int elmer() { }

 private static double daffy() {
} public void yosemitesam(int wile, double coyote){}

}
```



# Java Program: Hello World

---

Because the class LoonieToons does not have a main method you can not run this class

```
public class LoonieToons {
 public static void bugs() { }
 public void Bugs() { } //no link to bugs
 public int elmer() { }

 private static double daffy() {
 } public void yosemitesam(int wile, double coyote){}

}
```



# Java Program: Hello World

---

Any program you ever write in Java will always start the the beginning of the main method.

Remember that the beginning of a method is always marked by the `{` that immediately follows the name of the method.





# About that line:

---

```
public void yosemitesam(int wile, double coyote) {
```

There are a lot of different pieces to this.

Remember that in Java we can space things however we want. Anywhere that 1 space is allowed, multiple spaces can be allowed too (or new lines or tabs)

```
public
 void
 yosemitesam(int
wile
, double coyote)
{
```



# About that line:

---

```
public void yosemiteSam(int wile, double coyote) {
```

This line here is known as a method header. It gives you information about the method

yosemiteSam is the name of the method



# About that line:

---

```
public void yosemitesam(int wile, double coyote) {
```

This line here is known as a method header. It gives you information about the method

void is what is known as the return type of the method. We will talk more about this next class, but effectively this means “the thing that the method produces”



# About that line:

---

```
public void yosemitesam(int wile, double coyote) {
```

This line here is known as a method header. It gives you information about the method

public is a descriptor for the method. For the time being, we will almost always write the 2 descriptive words “public” and “static” before methods. But we will see in a few weeks what it means to have other things



# About that line:

---

```
public void yosemitesam(int wile, double coyote) {
```

This line here is known as a method header. It gives you information about the method

Between the parenthesis is the input to a method. Here is a list of things that the method will always be given when it is run. We will see that this is actually a list of values provided to the method.



# About that line:

---

```
public void yosemitesam(int wile, double coyote) {
```

This line here is known as a method header. It gives you information about the method

Finally we have an opening `{` which denotes the start of the method. Technically the `{` is not considered part of the method header.

# Java Program: Mathematical operations

```
public class DoMath {
 public static void main(String[] args) {
 System.out.println("My Comp 202 grade");
 System.out.print("is based on");
 System.out.println("my" + "assignments" +
 ",midterm");
 System.out.print("and final");
 System.out.println("If I get 60,70,80 on each");
 System.out.println("I'll get a ");
 System.out.println(60 * .3 + 70*.0 + 80*.7);
 }
}
```



# Java Program: Mathematical operations

---

Java will evaluate all of the above expressions.

Essentially, Java will always print whatever is between the ( ) after println (or print)

Sometimes, the stuff between the ( ) will be simple like "HelloWorld"

Java will always *evaluate the expression* between the ( ) before printing it.





# Java Program: Mathematical operations

---

There are rules for determining how it will evaluate the expression. For example, if the value between the ( ) consists of two things between " " that are separated by a +, then Java will just combine the 2 things.

```
System.out.println("hello" + "world");
```

is the same as

```
System.out.println("helloworld");
```



# Java Program: Mathematical operations

---

If there are 2 numbers separated by a mathematical symbol, then Java will evaluate them according to the math rules.

If it sees something complicated such as

```
System.out.println(1 + (2+3))
```

then it will have to do the parenthesis part first.



# What if I have a very complicated expression?

---

What if I have a very complicated expression to evaluate and I want to break it up into multiple steps.

Or what if I want to use some value many times throughout my program. For example, maybe I have 5 numbers and I want to print the difference of each number from the average. For example:

1,2,3,4,5 has average 3

1 is -2 from the average, 2 is -1 from average, 3 is 0 from average, etc.



# What if I have a very complicated expression?

---

```
public class RepeatConfusing {
 public static void main(String[] args) {
 System.out.println(1 - (1+2+3+4+5)/5);
 System.out.println(2 - (1+2+3+4+5)/5);
 System.out.println(3 - (1+2+3+4+5)/5);
 System.out.println(4 - (1+2+3+4+5)/5);
 System.out.println(5 - (1+2+3+4+5)/5);
 }
}
```



# What if I have a very complicated expression?

---

This is a little silly though because you'll notice we do the same operation many times.

Any time you can reduce the number of calculations the computer has to do, your program will run a bit more efficiently and things will be easier to read. (In this case the difference is so small you won't notice the difference.)



# Solution: Store the results into a variable

---

If we have a case like this, we can use a variable to store the results of a computation.

To make a variable, you have to do 2 things:

- 1) Decide what kind of thing or type you want to store.
  - If you want to store an integer, this is called int in Java
  - If you want to store a number with a fractional part, this is normally called double in Java (also could be float)
  - If you want to store letters, this is usually done with something called a String in Java.



# Solution: Store the results into a variable

---

2)Decide on a name for your variable.

Your variable can be named anything you like with a few exceptions:

- 1)It can only contain letters, numbers, and \_ (no ; for example)
- 2)It must start with a letter
- 3)There must not be another variable with the same name in scope
- 4)Variable names are case sensitive so Foo is different than foo
- 5)There are a few words in Java that are reserved. You can't call your variables these (for example "public")



# Solution: Store the results into a variable

---

Once you decide on a name and type, you can do what is known as declaring a variable by writing first the type and then the name and then a ;

For example:

```
int mean;
```

would declare a variable which will store an integer. It will be called mean in further computations.

At the beginning the variable mean has no value and is called uninitialized





# Solution: Store the results into a variable

---

To store a value into a variable, you write:

`variablename = expression`

What this means is "assign the value of the variable called `variablename` to be the value of *expression*"

The equals in Java is very different from the = in math.

- 1) It is not symmetric. `a = b` is not the same as `b = a`
- 2) It is a one time assignment. All that Java does is evaluate the expression and assign its value.
- 3) The types on the left and right side of the equal have to be the same. For example, you can't store letters into a number.



# Solution: Store the results into a variable

---

Once you have a variable initialized, you can use it in any other computation:

```
int mean;
```

```
mean = (1 + 2 + 3 + 4 + 5) / 5;
```

```
System.out.println(1 - mean);
```

```
System.out.println(2 - mean);
```

```
.....
```



# Solution: Store the results into a variable

---

You can also initialize the variable mean at the same time as you declare it:

```
int mean = (1 + 2 + 3 + 4 + 5) / 5;
```

```
System.out.println(1 - mean);
```

```
System.out.println(2- mean);
```

```
.....
```



# Question:

---

Suppose I make 2 variables:

```
int x = 0;
int y;
y = 1;
```

```
x = y + 1;
y = x + 1;
```

What would be the value of x and y at the end?



# Question:

---

Suppose I make 2 variables:

```
int x =0;
int y;
```

```
x = y + 1;
y = x + 1;
```

What would be the value of x and y at the end?



# Question:

---

Suppose I make 2 variables:

```
int x =0;
int y;
```

```
x = x + 1;
y = x + 1;
```

What would be the value of x and y at the end?



# Question:

---

What if I have 2 variables  $x$  and  $y$  and I want to swap the contents of them? In other words, I want to write something so that afterwards  $x$  has the old value of  $y$  and  $y$  has the old value of  $x$ .



# What if you don't declare x

---

If you write

```
x = 5;
```

without declaring `x`, you will get a compiler error.

The error will complain that it does not recognize `x`.





# Some basic types

---

int : stores an integer number

String : stores letters. For example "Hello World"

double : stores real numbers (fractions)

long : stores a long integer (up to 9 quintillion!)

float : like double, can store numbers

boolean : stores either true or false

char : stores one character



# Mismatching types

---

If you try to store something of type X in something of type Y, the Java compiler will complain.

For example,

```
int x;
```

```
x = "Hello"
```

What are the types of "Hello" and x



# Why does Java care anyway?

---

If the computer is as stupid as you say it is, why does the Java compiler care that you are trying to store a string in an int?



# Why does Java care anyway?

---

Answer: The problem is when you write

```
int x;
```

Java is setting aside enough memory to store 1 integer value.

If you try to store a String in it, it doesn't know whether it will fit or not!



# Next Class

---

- How to perform more complex operations
- 
- Differences between different types
- 
- How to group complex operations together using methods