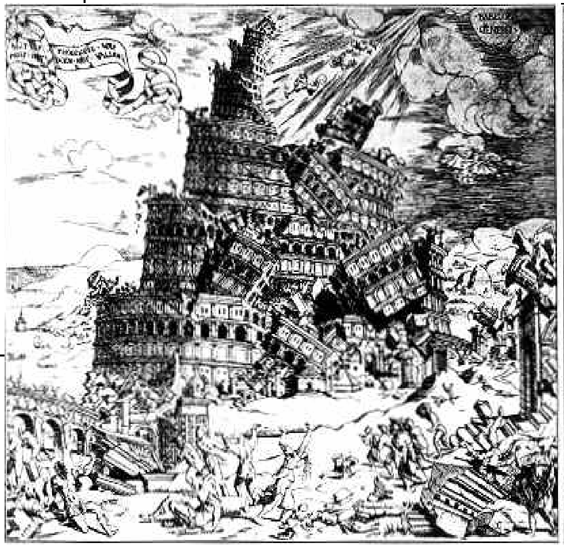
Topic 2
Introduction to Java Programming

"When a programming language is created that allows programmers to program in simple English, it will be discovered that programmers cannot speak English."

- Anonymous



Based on slides for Building Java Programs by Reges/Stepp, found at
http://faculty.washington.edu/stepp/book/

What We Will Do Today

- What are computer languages?
- Java editors
  - text editor and command line
  - BlueJ
- First programming concepts
  - output with println statements
  - syntax and errors
- structured algorithms with static methods
- identifiers, keywords, and comments

Computers and Computer Languages

- Computers are everywhere
  - how many computers do you own?
- Computers are useful because they run various programs
  - program is simply a set of instructions to complete some task
  - how many different programs do you use in a day?

Definitions

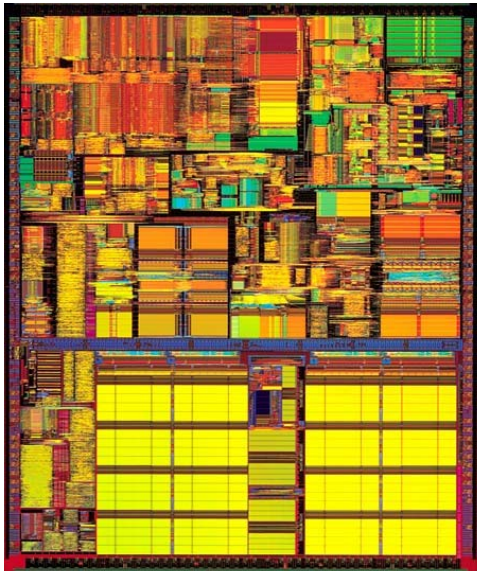
- **program**: A set of instructions that are to be carried out by a computer.

- **program execution**: The act of carrying out the instructions contained in a program.
  - this is done by feeding the instructions to the CPU

- **programming language**: A systematic set of rules used to describe computations, generally in a format that is editable by humans.
  - in this class will are using Java

High Level Languages

- Computers are fast
  - Pentium 4 chip from 2001 can perform approximately 1,700,000,000 computations per second
  - made up of 42,000,000 transistors (a switch that is on or off)
- Computers are dumb
  - They can only carry out a very limited set of instructions
    - on the order of 100 or so depending on the computer's processor
    - machine language instructions, aka instruction set architecture (ISA)
    - Add, Branch, Jump, Get Data, Get Instruction, Store



Machine Code

- John von Neumann - co-author of paper in 1946 with Arthur W. Burks and Hermann H. Goldstine,
  - "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument"
- One of the key points
  - program commands and data stored as sequences of bits in the computer's memory
- A program:

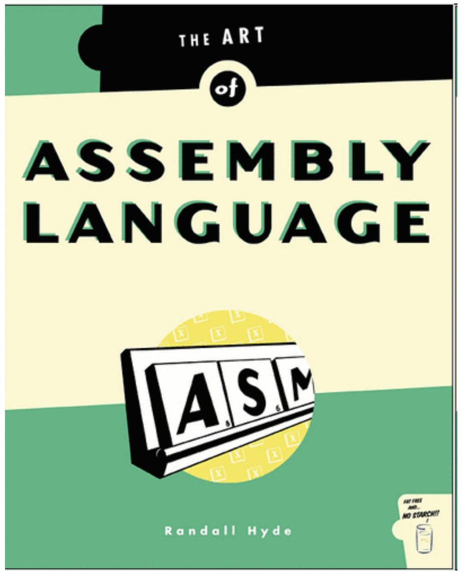
```
1110001100000000
0101011011100000
0110100001000000
0000100000001000
0001011011000100
0001001001100001
0110100001000000
```

```
'  ' 000b 0000              NOP
'PH' 000c 5048              AND $0048
'YR' 000d 5952              DSZ $0152
'EC' 000e 4543              JSM Lbl_0052
'  ' 000f 2020              ADA $0020
'RE' 0010 5245              AND $fe45
'V ' 0011 5620              AND $fe31
'B.' 0012 422e              JSM $fe2e
'  ' 0013 8000 Lbl_0001:    LDA A,I
'  ' 0014 ffff              SES *-1,S [Lbl_0
' $' 0015 1624 Lbl_0002:    CPA $fe39
'  ' 0016 10f9              CPA $00f9
```

Say What?

- Programming with Strings of bits (1s or 0s) is not the easiest thing in the world.
- Assembly language
  - mnemonics for machine language instructions

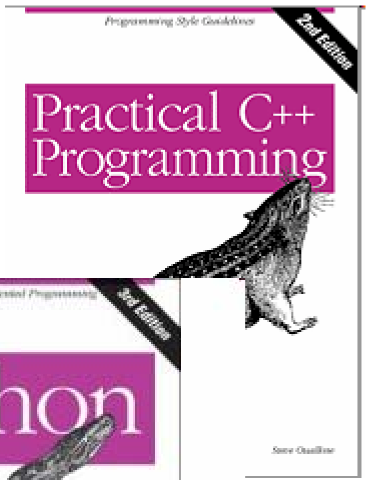
```
.ORIG      x3001
LD         R1, x3100
AND        R3, R3 #0
LD         R4, R1
BRn        x3008
ADD        R3, R3, R4
ADD        R1, R1, #1
LD         R4, R1
BRnzp      x3003
```

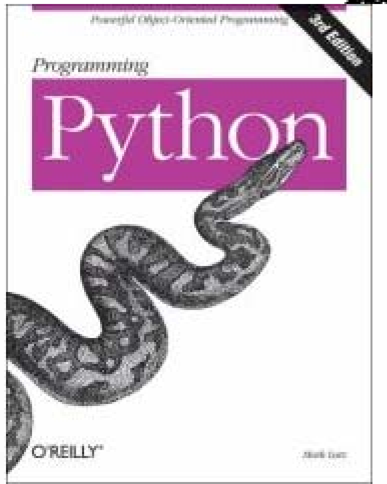


High Level Languages

- Assembly language, still not so easy, and lots of commands to accomplish things
- High Level Computer Languages provide the ability to accomplish a lot with fewer commands than machine or assembly language in a way that is hopefully easier to understand

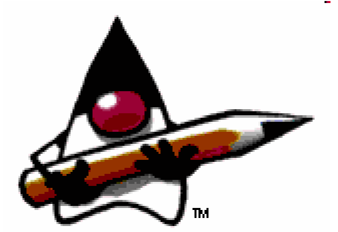
```
int sum;
int count = 0;
int done = -1;
while( list[count]!= -1 )
     sum += list[count];
```





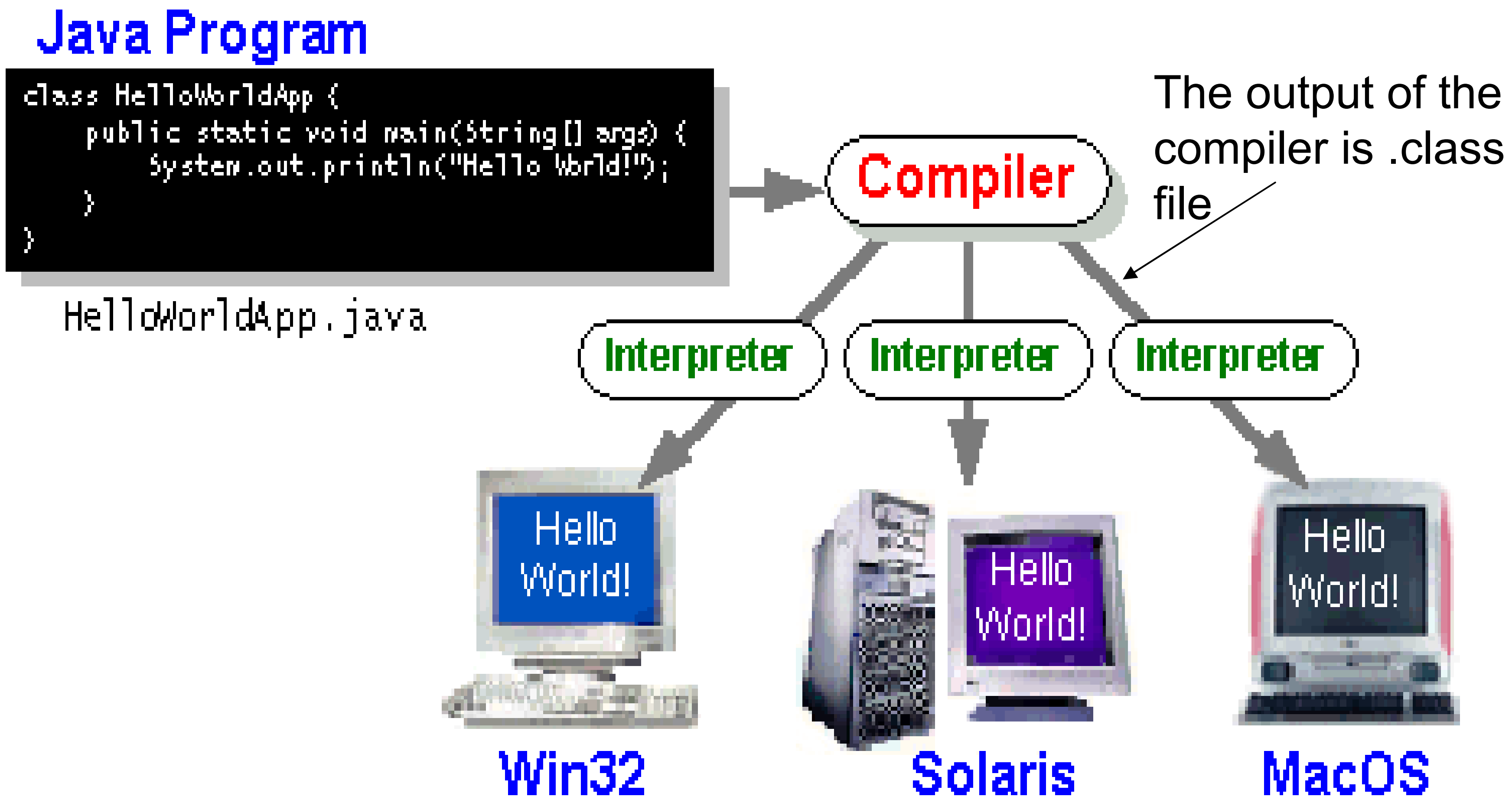
Java

- There are hundreds of high level computer languages. Java, C++, C, Basic, Fortran, Cobol, Lisp, Perl, Prolog, Eiffel, Python
- The capabilities of the languages vary widely, but they all need a way to do
  - declarative statements
  - conditional statements
  - iterative or repetitive statements



- A compiler is a program that converts commands in high level languages to machine language instructions

A Picture is Worth…



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

The Interpreter's are sometimes referred to as the Java Virtual Machines

A Simple Java Program

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

This would be in a text file named Hello.java

DEMO of writing and running a program via notepad and the command line

More Definitions

- **code or source code**: The sequence of instructions in a particular program.
  - The code in this program instructs the computer to print a message of **Hello, world!** on the screen.
- **output**: The messages printed to the computer user by a program.
- **console**: The text box or window onto which output is printed.

Compiling and Running

- **Compiler**: a program that converts a program in one language to another language
  - compile from C++ to machine code
  - compile Java to *bytecode*
- **Bytecode**: a language for an imaginary cpu
- **Interpreter**: A converts one instruction or line of code from one language to another and then executes that instruction
  - When java programs are run the bytecode produced by the compiler is fed to an interpreter that converts it to machine code for a particular CPU
  - on my machine it converts it to instructions for a Pentium cpu

The command line

To run a Java program using your Command Prompt:

- change to the directory of your program

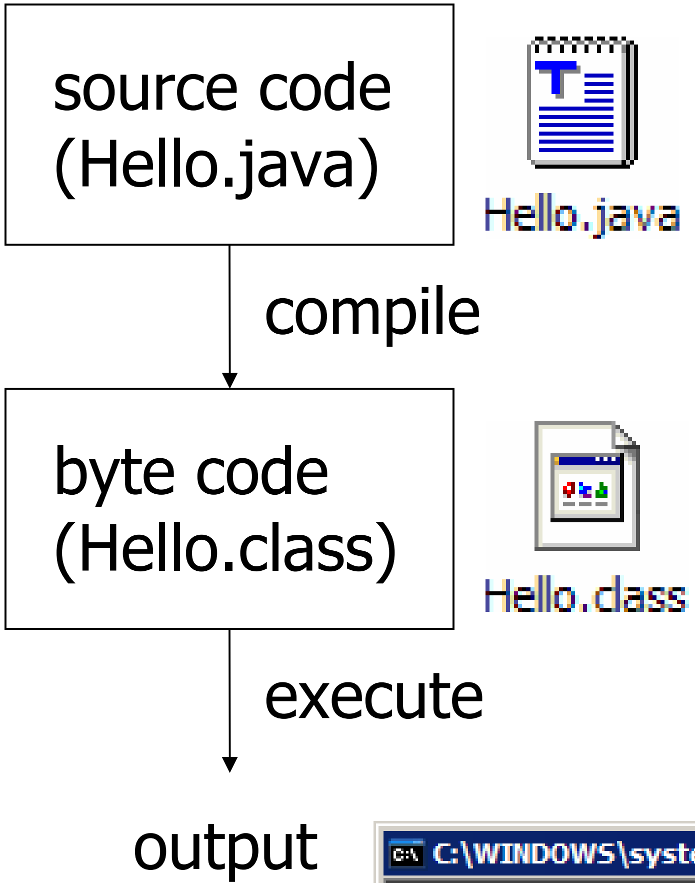
```
cd
```

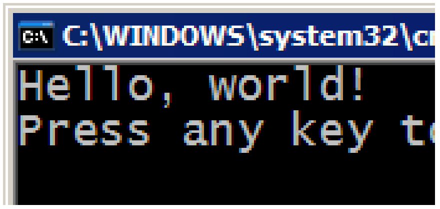
- compile the program

```
javac Hello.java
```

- execute the program

```
java Hello
```



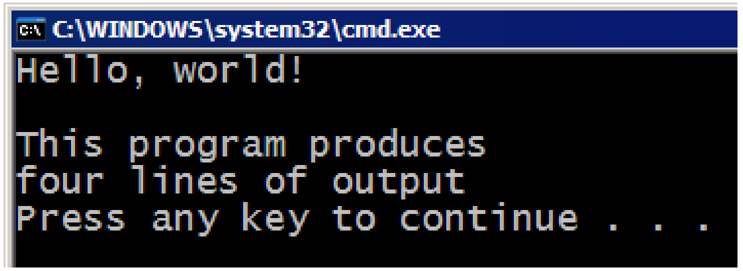


```
Hello, world!
Press any key t
```

Another Java program

```
public class Hello2 {
   public static void main(String[] args) {
      System.out.println("Hello, world!");
      System.out.println();
      System.out.println("This program produces");
      System.out.println("four lines of output");
   }
}
```

- The code in this program instructs the computer to print four messages on the screen.



C:\WINDOWS\system32\cmd.exe

```
Hello, world!

This program produces
four lines of output
Press any key to continue . . .
```

Structure of Java programs

```
public class <name> {
    public static void main(String[] args) {
        <statement(s)>;
    }
}
```

- Every executable Java program consists of a **class**...
  - that contains a **method** named `main`...
    - that contains the **statements** to be executed
- The previous program is a class named `Hello`, whose `main` method executes one statement named `System.out.println`

Java terminology

- **class**:
  (a) A module that can contain executable code.
  (b) A description of a type of objects. (seen later)
- **statement**: An executable piece of code that represents a complete command to the computer.
  - every basic Java statement ends with a semicolon `;`
- **method**: A named sequence of statements that can be executed together to perform a particular action or computation.

Syntax and syntax errors

- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **syntax error** or **compiler error**: A problem in the structure of a program that causes the compiler to fail.
  - If you type your Java program incorrectly, you may violate Java's syntax and see a syntax error.

```
public class Hello {
    pooblic static void main(String[] args) {
        System.owt.println("Hello, world!")_
    }
}
```

Compiler Output

- The program on the previous slide produces the following output when we attempt to compile it

compiler output:

```
H:\summer\Hello.java:2: <identifier> expected
    pooblic static void main(String[] args) {
           ^
H:\summer\Hello.java:5: ';' expected
}
^
2 errors
Tool completed with exit code 1
```

Fixing syntax errors

- Notice how the error messages are sort of cryptic and do not always help us understand what is wrong:

```
H:\summer\Hello.java:2: <identifier> expected
    pooblic static void main(String[] args) {
            ^
```

  - We'd have preferred a friendly message such as, "You misspelled 'public' "

- The compiler does tell us the line number on which it found the error, which helps us find the place to fix the code.
  - The line number shown is a good hint, but is not always the true source of the problem.

- Java has a fairly rigid syntax.

System.out.println

- Java programs use a statement called `System.out.println` to instruct the computer to print a line of output on the console
  - pronounced "*print-linn*"; sometimes called a *println statement* for short

- Two ways to use `System.out.println` :
  - 1. `System.out.println("`***<Message>***`");`
    - Prints the given message as a line of text on the console.
  - 2. `System.out.println();`
    - Prints a blank line on the console.

Strings and string literals

- **string**: A sequence of text characters (not just letters) that can be printed or manipulated in a program.

- **literal**: a representation of a value of a particular type
  - String literals in Java start and end with quotation mark characters

```
"This is a string"
```

Details about Strings

- A string literal may not span across multiple lines.

```
"This is not
a legal String."
```

- A string may not contain a `"` character. `'` is OK

```
"This is not a "legal" String either."
"This is 'okay' though."
```

- A string can represent certain special characters by preceding them with a backslash `\` (this is called an **escape sequence**).
  - `\t` tab character
  - `\n` new line character
  - `\"` quotation mark character
  - `\\` backslash character

Practice Program 1

- What sequence of println statements will generate the following output?

```
This program prints the first lines
of the song "slots".

"She lives in a trailer"
"On the outskirts 'a Reno"
"She plays quarter slots in the locals casino."
```

Practice Program 2

- What sequence of println statements will generate the following output?

```
A "quoted" String is
'much' better if you learn
the rules of "escape sequences."

Also, "" represents an empty String.
Don't forget to use \" instead of " !
'' is not the same as "
```

Practice Program 3

- What is the output of the following `println` statements?

```
System.out.println("\ta\tb\tc");
System.out.println("\\\\");
System.out.println("'");
System.out.println("\"\"\"");
System.out.println("C:\nin\the downward spiral");
```

Answer to Practice Program 3

Output of each println statement:

```
        a       b       c
\\
'
"""
C:
in      he downward spiral
```

Practice Program 4

- Write a `println` statement to produce this output:

```
/ \ // \\ /// \\\
```

Answer to Practice Program 4

println statement to produce the line of output:

```
System.out.println("/ \\ // \\\\ /// \\\\\\");
```

A structured example

- What sequence of println statements will generate the following output?

```
  ______
 /      \
/        \
\        /
 \______/

  ______
 /      \
/        \
|        |
|        |
|        |
\        /
 \______/
  ______
 /      \
/        \
+--------+

  ______
 /      \
/        \
```

- What observations can we make about the output that is generated?
  - It has a noticeable structure.
    (draw first figure, draw second figure, draw third figure, ...)
  - The output contains redundancy. Certain figures (or large parts of figures) are repeated in the output.

Structured algorithms

- How does one bake sugar cookies?
  - Mix the dry ingredients.
  - Cream the butter and sugar.
  - Beat in the eggs.
  - Stir in the dry ingredients.
  - Set the oven for the appropriate temperature.
  - Set the timer.
  - Place the cookies into the oven.
  - Allow the cookies to bake.
  - Mix the ingredients for the frosting.
  - Spread frosting and sprinkles onto the cookies.
  - ...

- Can we express this process in a more structured way?

A structured algorithm

- **structured algorithm**: A list of steps for solving a problem, which is broken down into cohesive tasks.
- A structured algorithm for baking sugar cookies:
  - 1. Make the cookie batter.
    - Mix the dry ingredients.
    - Cream the butter and sugar.
    - Beat in the eggs.
    - Stir in the dry ingredients.
  - 2. Bake the cookies.
    - Set the oven for the appropriate temperature.
    - Set the timer.
    - Place the cookies into the oven.
    - Allow the cookies to bake.
  - 3. Add frosting and sprinkles.
    - Mix the ingredients for the frosting.
    - Spread frosting and sprinkles onto the cookies.
  - ...

Redundancy in algorithms

- How would we express the steps to bake a double batch of sugar cookies?

Unstructured:

- Mix the dry ingredients.
- Cream the butter and sugar.
- Beat in the eggs.
- Stir in the dry ingredients.
- Set the oven ...
- Set the timer.
- Place the first batch of cookies into the oven.
- Allow the cookies to bake.
- Set the oven ...
- Set the timer.
- Place the second batch of cookies into the oven.
- Allow the cookies to bake.
- Mix the ingredients for the frosting.

Structured:

- 1. Make the cookie batter.
- 2a. Bake the first batch of cookies.
- 2b. Bake the second batch of cookies.
- 3. Add frosting and sprinkles.

- *Observation*: A structured algorithm not only presents the problem in a hierarchical way that is easier to understand, but it also provides higher-level operations which help eliminate redundancy in the algorithm.

Static methods

- **static method**: A group of statements that is given a name so that it can be executed in our program.
  - Breaking down a problem into static methods is also called "procedural decomposition."
- Using a static method requires two steps:
  - **declare** it (write down the recipe)
    - When we declare a static method, we write a group of statements and give it a name.
  - **call** it (cook using the recipe)
    - When we call a static method, we tell our main method to execute the statements in that static method.

- Static methods are useful for:
  - denoting the structure of a larger program in smaller, more understandable pieces
  - eliminating redundancy through reuse

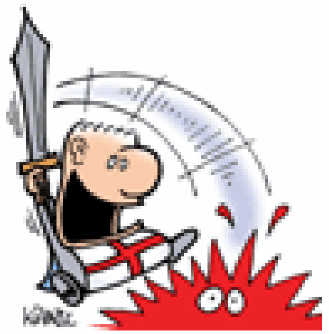
Static method syntax

- The structure of a static method:

```
public class <Class Name> {

    public static void <Method name> () {
        <statements>;
    }

}
```

- Example:

```
public static void printCheer() {
    System.out.println("Three cheers for Pirates!");
    System.out.println("Huzzah!");
    System.out.println("Huzzah!");
    System.out.println("Huzzah!");
}
```



Static methods example

```
public class TwoMessages {
    public static void main(String[] args) {
        printCheer();
        System.out.println();
        printCheer();
    }

    public static void printCheer() {
        System.out.println("Three cheers for Pirates!");
        System.out.println("Huzzah!");
        System.out.println("Huzzah!");
        System.out.println("Huzzah!");
    }
}
```

Program's output:

```
Three cheers for Pirates!
Huzzah!
Huzzah!
Huzzah!


Three cheers for Pirates!
Huzzah!
Huzzah!
Huzzah!
```

Methods calling each other

- One static method may call another:

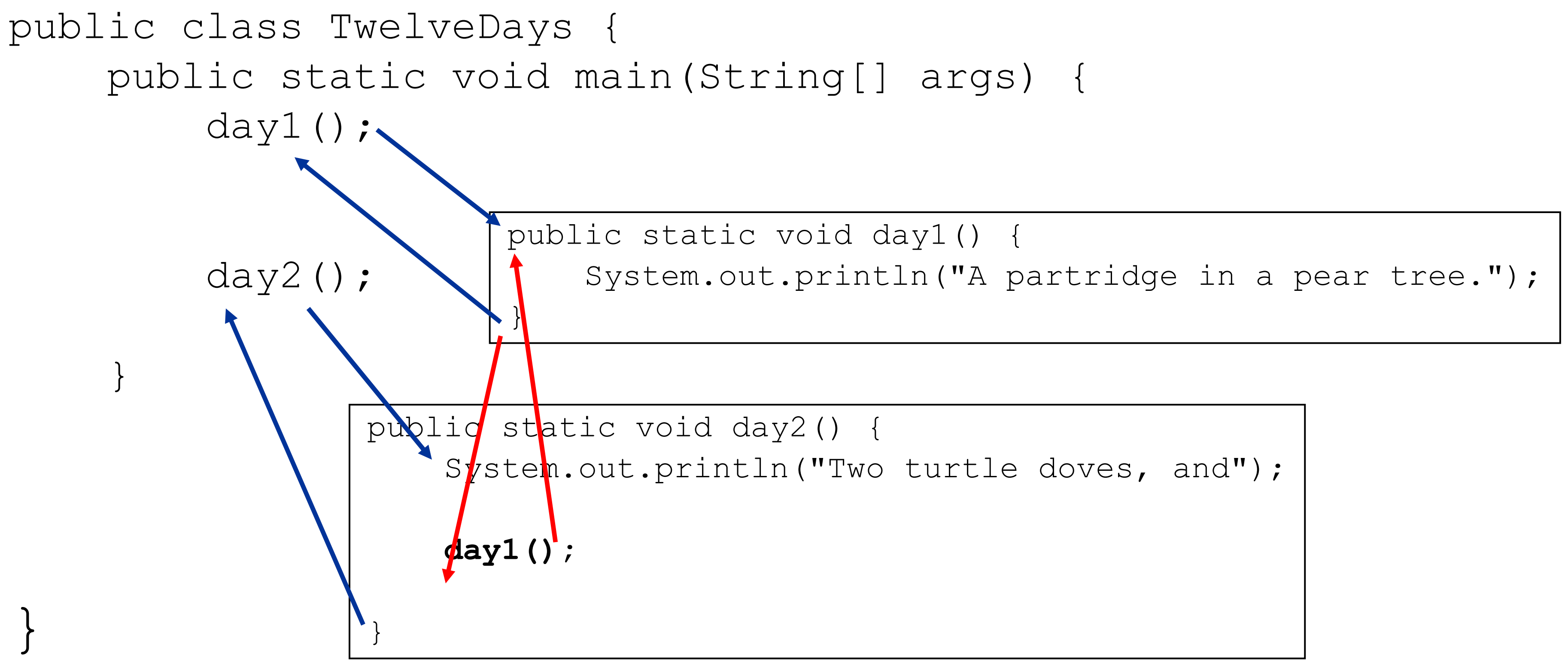
```
public class TwelveDays {
    public static void main(String[] args) {
        day1();
        day2();
    }

    public static void day1() {
         System.out.println("A partridge in a pear tree.");
    }

    public static void day2() {
         System.out.println("Two turtle doves, and");
         day1();
    }
}
```

Program's output:

```
A partridge in a pear tree.
Two turtle doves, and
A partridge in a pear tree.
```

Control flow of methods

- When a method is called, a Java program 'jumps' into that method, executes all of its statements, and then 'jumps' back to where it started.



```
public class TwelveDays {
    public static void main(String[] args) {
        day1();

        day2();

    }
}
```

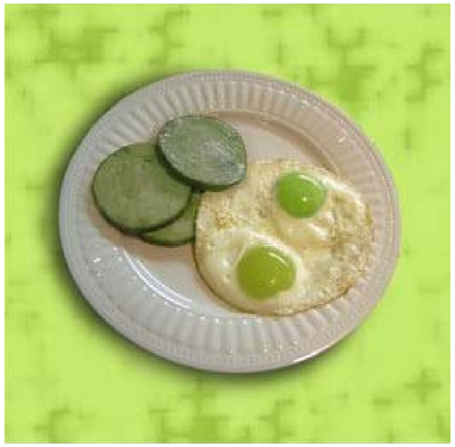
```
public static void day1() {
    System.out.println("A partridge in a pear tree.");
}
```

```
public static void day2() {
    System.out.println("Two turtle doves, and");

    day1();

}
```

Static method problems

- Write a program that prints the following output to the console. Use static methods as appropriate.

```
I do not like green eggs and ham,
I do not like them, Sam I am!
I do not like them on boat,
I do not like them with a goat.
I do not like green eggs and ham,
I do not like them, Sam I am!
```

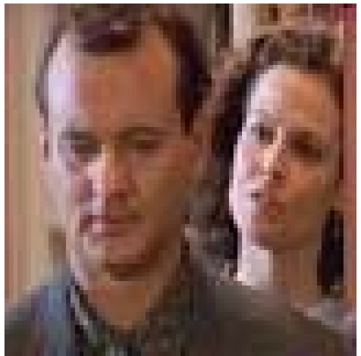


- Write a program that prints the following output to the console. Use static methods as appropriate.

```
Lollipop, lollipop
Oh, lolli lolli lolli

Lollipop, lollipop
Oh, lolli lolli lolli
Call my baby lollipop
```

When to use static methods

- You should place a group of statements into a static method if any of the following conditions is met:
  - The statements are related to each other and form a combined part of the program's structure.
  - The statements are repeated in the program.
- You need not create static methods for the following:
  - Individual statements.
    (One single println in its own static method does not improve the program, and may make it harder to read.)
  - Unrelated or weakly related statements.
    (If the statements are not closely related, consider splitting the method into two or more smaller methods.)
  - Only blank lines.
    (It's fine to have blank `System.out.println();` statements in the main method.)
- Remember, these are *guidelines!*



Identifiers

- **identifier**: A name that we give to a piece of data or part of a program.
  - Identifiers are useful because they allow us to refer to that data or code later in the program.
  - Identifiers give names to:
    - classes
    - methods
    - variables (named pieces of data; seen later)

- The name you give to a static method is an example of an identifier.
  - What are some other example identifier we've seen?

Details about identifiers

- Java identifier names:
  - first character must a letter or `_` or `$`
  - following characters can be any of those characters or a number
  - identifiers are case-sensitive; `name` is different from `Name`

- Example Java identifiers:
  - legal:

```
olivia      second_place   _myName
TheCure     ANSWER_IS_42   $variable
```

  - illegal:

```
me+u         :-)        question?
side-swipe   hi there   ph.d
belles's     2%milk     kelly@yahoo.com
```

    - explain why each of the above identifiers is not legal.

Keywords

- **keyword**: An identifier that you cannot use, because it already has a reserved meaning in the Java language.

- Complete list of Java keywords:

`abstract`	`default`	`if`	`private`	`this`
`boolean`	`do`	`implements`	`protected`	`throw`
`break`	`double`	`import`	**`public`**	`throws`
`byte`	`else`	`instanceof`	`return`	`transient`
`case`	`extends`	`int`	`short`	`try`
`catch`	`final`	`interface`	**`static`**	**`void`**
`char`	`finally`	`long`	`strictfp`	`volatile`
`class`	`float`	`native`	`super`	`while`
`const`	`for`	`new`	`switch`	
`continue`	`goto`	`package`	`synchronized`	

- You may not use `char` or `while` or `this` or any other keyword for the name of a class or method; Java reserves those words to mean other things.
  - You could use CHAR, While, or ThIs, because Java is case-sensitive. However, this could be confusing and is not recommended.

Comments

- **comment**: A note written in the source code by the programmer to make the code easier to understand.
  - Comments are not executed when your program runs.
  - Most Java editors turn your comments a special color to make it easier to identify them.

- Comment, general syntax:

  `/*` ***<comment text; may span multiple lines>*** `*/`

  or,

  `//` ***<comment text, on one line>***

- Examples:
  - `/* A comment goes here. */`
  - `/* It can even span`
    `multiple lines. */`
  - `// This is a one-line comment.`

Using comments

- Comments can be put in many standard places.
  - Most all programs have a "comment header" at the top of each file, naming the author and explaining what the program does.
  - Most programmers also place a comment at the start of every method, describing the method's behaviour.
  - Lastly, we can use comments inside methods to explain particular pieces of code.
- Comments provide important documentation.
  - At this stage in our learning, it is not very useful to write comments, because we only know println statements.
  - More complicated programs span hundreds or thousands of lines, and it becomes very difficult to remember what each method is doing. Comments provide a simple description.
  - When multiple programmers work together, comments help one programmer understand the other's code.

Comments example

```
/* Olivia Scott
   CS 305j, Fall 2006
   This program prints lyrics from a song! */
public class PartOfSong {
  /* Runs the overall program to print the song
     on the console. */
  public static void main(String[] args) {
    displayVerse();

    // Separate the two verses with a blank line
    System.out.println();

    displayVerse();
  }

  // Displays the first verse of song.
  public static void displayVerse() {
    System.out.println("The road goes on forever,");
    System.out.println("And the party never ends!");
  }
}
```

How to comment: methods

- Write a comment at the top of each of your methods that explains what the method does.
  - You do not need to describe the Java syntax and statements in detail, but merely provide a short English description of the observed behavior when the method is run.
  - Example:

```
// This method prints the lyrics to the first verse
// of the TV theme song to the Fresh Prince of Bellaire.
// Blank lines separate the parts of the verse.
public static void verse1() {
    System.out.println("Now this is the story all about how");
    System.out.println("My life got flipped turned upside-down");
    System.out.println();
    System.out.println("And I'd like to take a minute,");
    System.out.println("just sit right there");
    System.out.println("I'll tell you how I became the prince");
    System.out.println("of a town called Bel-Air");
}
```