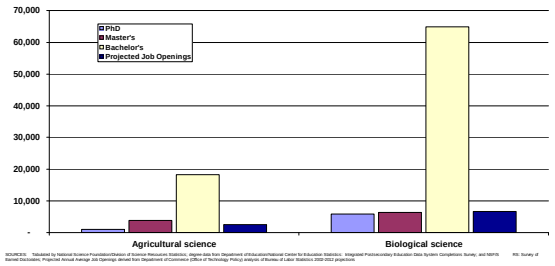
WELCOME TO CSE 142!

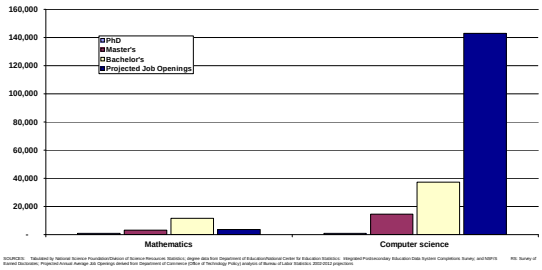
host: benson limketkai

University of Washington, Spring 2008

Who cares about computer science?



Who cares about computer science?



What is computer science?

- computers?
- science?
- programming?
- late lonely nights in front of the computer?

ALGORITHMIC THINKING

al·go·rithm:

a step by step procedure for solving a problem or accomplishing some end *especially by a computer*

- How does that relate to programming?

Programming is just like Legos…

Should you take this course?

- No
  - "I hate computers."
  - "I don't pay attention to details."
    - Programming is fairly detail-oriented.
  - "I refuse to think logically."
  - "I want to take an easy class."
    - Hard for those who find difficulty in logical thinking and who don't pay attention to details.

Should you take this course?

- Probably not
  - "I want free gourmet meals and to make lots of money by working for Google."
  - "World of Warcraft rocks hardcore!"
- Yes
  - "I have to take this class."
    - Is this the only reason? Are you pursuing the right major?
  - "I like to solve problems."
  - "Computers and robots are going to take over the world. I want to befriend them so that my life will be spared."

How to do well in this course

- Keep up with the assignments
  - The course material is cumulative
  - From a former student: "Procrastination will eventually come around to bite you in the ass!"
- If you don't understand something, ask questions (especially "WHY?").
  - "There's no such thing as a dumb question."
  - Computers are neither magical nor mysterious. Everything can be explained!

Basic Java programs

Readings: 1.1 – 1.3

Your first Java program!

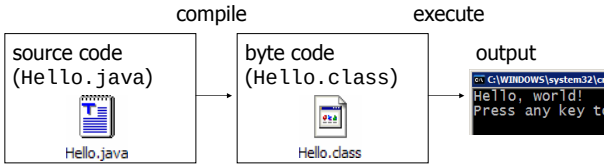
- Java is a *programming language*.

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

- File must be named `Hello.java`
- What does this code *output* (print to the user) when you *run* (execute) it?

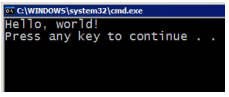
Running a program

- Before you run a program, you must *compile* it.
- **compiler:** Translates a computer program written in one language (i.e., Java) to another language (i.e., byte code)



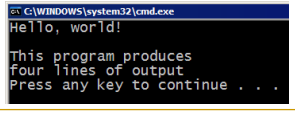
Program execution

- The *output* is printed to the *console*.
- Some editors pop up the console as another window.



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");
  }
}
```



Writing your own Java programs

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

- Every executable Java program consists of a **class**
  - that contains a **method** called `main`
    - that contains the **statements** (commands) to be executed

Syntax

- **syntax:** The set of legal structures and commands that can be used.
- Examples:
  - Every basic statement ends with a semi-colon.
  - The contents of a class occur between curly braces.

Syntax Errors

- **syntax error:** A problem in the structure of a program.

```
1 public class Hello {
2     pooblic static void main(String[] args) {
3         System.owt.println("Hello, world!")
4     }
5 }
```

compiler output:

```
2 errors found:
File: Hello.java  [line: 2]
Error: Hello.java:2: <identifier> expected
File: Hello.java  [line: 3]
Error: Hello.java:3: ';' expected
```


Finding syntax errors

- Error messages do not always help us understand what is wrong:

```
File: Hello.java  [line: 2]
Error: Hello.java:2: <identifier> expected

pooblic static void main(String[] args) {
```

- Why can't the computer just say "*You misspelled 'public'*"?

First lesson in computer science

- Computers are stupid.
- Computers can't read minds.
- Computers don't make mistakes.
- If the computer is not doing what you want, it's because **YOU** made a mistake.

More on syntax errors

- Java is case-sensitive
  - `Hello` and `hello` are not the same

```
1 Public class Hello {
2   public static void main(String[] args) {
3     System.out.println("Hello, world!");
4   }
5 }
```

compiler output:

```
1 error found:
File: Hello.java  [line: 1]
Error: Hello.java:1: class, interface, or enum
expected
```


`System.out.println`

- `System.out.println`: A statement to print a line of output to the console.
  - pronounced "*print-linn*"
- Two ways to use `System.out.println`:

  `System.out.println("<message>");`
  - Prints the given message as a line of text to the console.

  `System.out.println();`
  - Prints a blank line to the console.

Strings

- **string:** A sequence of text characters.
  - Start and end with quotation mark characters
- Examples:

```
"hello"
"This is a string"
"This, too, is a string.    It can be very long!"
```

Details about strings

- A string may not span across multiple lines.

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

- A string may not contain a " character.
  - The ' character is okay.

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

- This begs the question…

Escape sequences

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

```
System.out.println("Hello!\nHow are \"you\"?");
```

- Output:

```
Hello!
How are "you"?
```

- This begs another question…

Questions

1. What is the output of each 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");
```

2. Write a `println` statement to produce the following line of output:

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


Answers

1.

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

2.

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


Questions

1. What `println` statements will generate the following output?

```
This program prints a
quote from the Gettysburg Address.

"Four score and seven years ago,
our 'fore fathers' brought forth on this continent
a new nation."
```

2. What `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 "
```


Answers

1.

```
System.out.println("This program prints a");
System.out.println("quote from the Gettysburg Address.");
System.out.println();
System.out.println("\"Four score and seven years ago,");
System.out.println("our 'fore fathers' brought forth on");
System.out.println("this continent a new nation.\"");
```

2.

```
System.out.println("A \"quoted\" String is");
System.out.println("'much' better if you learn");
System.out.println("the rules of \"escape sequences.\"");
System.out.println();
System.out.println("Also, \"\" represents an empty String.");
System.out.println("Don't forget: use \\\" instead of \" !");
System.out.println("'' is not the same as \"");
```

Procedural decomposition using static methods

Readings: 1.4 – 1.5

Algorithms

- **Recall:** An *algorithm* is a list of steps for solving a problem.

What is the algorithm to bake sugar cookies?

The "Bake sugar cookies" algorithm

- 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.

Structured algorithm

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 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.

- *Observation*: Structured algorithms are easier to understand.

How do we bake a double batch?

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 cookies into the oven.*
- *Allow the cookies to bake.*
- **Set the oven …**
- **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.

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*: Structured algorithms eliminate redundancy.

Redundancy in programs

```
public class FraggleRock {
  public static void main(String[] args) {
    System.out.println("Dance your cares away,");
    System.out.println("Worry's for another day.");
    System.out.println("Let the music play,");
    System.out.println("Down at Fraggle Rock.");
    System.out.println();
    System.out.println("Dance your cares away,");
    System.out.println("Worry's for another day.");
    System.out.println("Let the music play,");
    System.out.println("Down at Fraggle Rock.");
  }
}
```

How do we structure the program?

- **static method**: A group of statements given a name.
- To use a static method:
  1. **declare** it (write down the recipe)
     - Write a group of statements and give it a name.
  2. **call** it (cook using the recipe)
     - Tell our program to execute the method.

Declaring a static method

- The syntax for declaring a static method (writing down the recipe):

```
public class <class name> {
  ...
  public static void <method name>() {
    <statement>;
    <statement>;
    ...
    <statement>;
  }
}
```


Calling a static method

- The syntax for calling a static method (cooking using the recipe):

```
<method name>();
```


Static method example

- Declaring a static method

```
public static void printAffirmation() {
  System.out.println("I am good enough!");
  System.out.println("I am smart enough!");
  System.out.println("People like me!");
}
```

- Calling a static method (possibly multiple times)

```
printAffirmation();
printAffirmation();
```

- Output

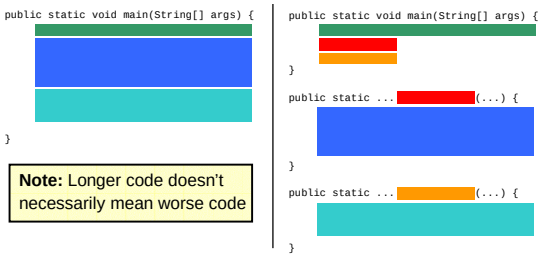
```
I am good enough!
I am smart enough!
People like me!
I am good enough!
I am smart enough!
People like me!
```


Worry's for another day!

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

  public static void singChorus() {
    System.out.println("Dance your cares away,");
    System.out.println("Worry's for another day.");
    System.out.println("Let the music play,");
    System.out.println("Down at Fraggle Rock.");
  }
}
```


Summary: Why write methods?

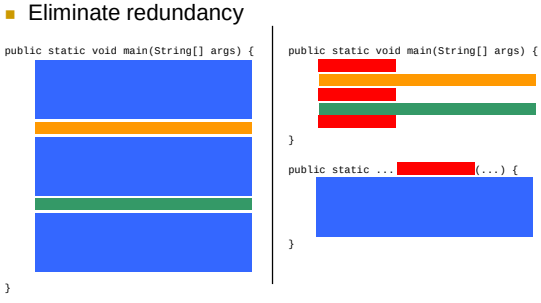
- Makes code easier to read by capturing the structure of the program
  - `main` should be a good summary of the program



Note: Longer code doesn't necessarily mean worse code

Summary: Why write methods?

- Eliminate redundancy



Methods calling methods

- One static method can call another:

```
public class MethodsExample {
  public static void main(String[] args) {
    message1();
    message2();
    System.out.println("Done with main.");
  }

  public static void message1() {
    System.out.println("This is message1.");
  }

  public static void message2() {
    System.out.println("This is message2.");
    message1();
    System.out.println("Done with message2.");
  }
}
```

Output:

```
This is message1.
This is message2.
This is message1.
Done with message2.
Done with main.
```

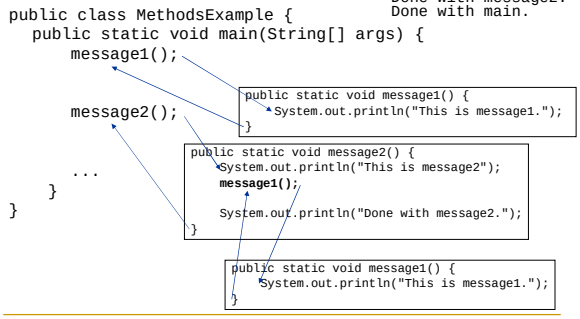

Control flow of methods

- When a method is called, the execution
  - "jumps" into that method
  - executes all of the method's statements
  - "jumps" back to the statement after the method call

Control flow of methods

Output:

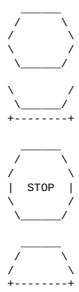
```
This is message1.
This is message2.
This is message1.
Done with message2.
Done with main.
```



Summary: To use or not to use…

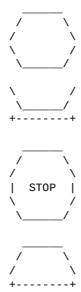
- Yes
  - Statements that are related to each other (*structure*).
  - Statements that are repeated (*redundancy*).
- No
  - Individual statements occurring only once and not related to other statements
  - Unrelated or weakly-related statements
    - Consider splitting the method into two smaller methods.
  - Blank lines
    - Blank `println` statements can go in the `main` method.

Example: Figure drawing



- Write a program to print the figures. Use static methods to capture structure and and eliminate redundancy.

Version 1: Unstructured

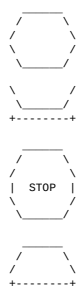


- Create an empty program with a skeletal header and `main` method.
- Copy the expected output into it, surrounding each line with `System.out.println` syntax.
- Run and verify that it produces the correct output.

Version 1: Unstructured

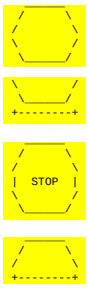
```
// Suzy Student, CSE 142, Autumn 2047
// This program prints several assorted figures.
//
public class Figures1 {
    public static void main(String[] args) {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println("+--------+");
        System.out.println();
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("|  STOP  |");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("+--------+");
    }
}
```


Version 2: Structured with redundancy



- Identify the overall structure of the output, and divide the `main` method into several static methods based on this structure.

Version 2: Structured with redundancy



- Identify the overall structure of the output, and divide the `main` method into several static methods based on this structure.

The structure of the output:

- initial "egg" figure
- second "teacup" figure
- third "stop sign" figure
- fourth "hat" figure

This structure can be represented by methods:

- `drawEgg`
- `drawTeaCup`
- `drawStopSign`
- `drawHat`

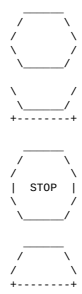
Version 2: Structured with redundancy

```
// Suzy Student, CSE 142, Autumn 2047
// Prints several assorted figures, with methods for structure.
//
public class Figures2 {
    public static void main(String[] args) {
        drawEgg();
        drawTeaCup();
        drawStopSign();
        drawHat();
    }

    // Draws a figure that vaguely resembles an egg.
    public static void drawEgg() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
    }

    // Draws a figure that vaguely resembles a teacup.
    public static void drawTeaCup() {
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println("+--------+");
        System.out.println();
    }
```


Version 2: Structured with redundancy

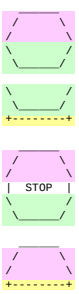
```
    // Draws a figure that vaguely resembles a stop sign.
    public static void drawStopSign() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("|  STOP  |");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
    }

    // Draws a figure that vaguely resembles a hat.
    public static void drawHat() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("+--------+");
    }
}
```

Version 3: Structured without redundancy



- Further divide the program to eliminate all redundancy.

Version 3: Structured without redundancy



- Further divide the program to eliminate all redundancy.

The redundancy:
- top half of egg (purple)
- bottom half of egg (green)
- divider line (yellow)

This redundancy can be fixed by methods:
- `drawEggTop`
- `drawEggBottom`
- `drawLine`

Version 3: Structured without redundancy

```
// Suzy Student, CSE 142, Autumn 2047
// Prints several figures, with methods for structure and redundancy.

public class Figures3 {
    public static void main(String[] args) {
        drawEgg();
        drawTeaCup();
        drawStopSign();
        drawHat();
    }

    // draws redundant part that looks like the top of an egg
    public static void drawEggTop() {
        System.out.println("      ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
    }

    // draws redundant part that looks like the bottom of an egg
    public static void drawEggBottom() {
        System.out.println("\\        /");
        System.out.println(" \\______/");
    }
```


Version 3: Structured without redundancy

```
    // Draws a figure that vaguely resembles an egg.
    public static void drawEgg() {
        drawEggTop();
        drawEggBottom();
        System.out.println();
    }

    // Draws a figure that vaguely resembles a teacup.
    public static void drawTeaCup() {
        drawEggBottom();
        System.out.println("+--------+");
        System.out.println();
    }

    // Draws a figure that vaguely resembles a stop sign.
    public static void drawStopSign() {
        drawEggTop();
        System.out.println("|  STOP  |");
        drawEggBottom();
        System.out.println();
    }

    // Draws a figure that vaguely resembles a hat.
    public static void drawHat() {
        drawEggTop();
        System.out.println("+--------+");
    }
}
```


Exercise

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

```
I do not like my email spam,
I do not like them, Sam I am!
I do not like them on my screen,
I do not like them to be seen.
I do not like my email spam,
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
```

Exercise

```
BBBBB
B    B
BBBBB
B    B
BBBBB

 AAAA
A    A
AAAAAA
A    A

N    N
NNN  N
N  NNN
N    N

 AAAA
A    A
AAAAAA
A    A

N    N
NNN  N
N  NNN
N    N

 AAAA
A    A
AAAAAA
A    A
```

- Write a program to print the block letters spelling "banana". Use static methods to capture structure and and eliminate redundancy.

Identifiers: Say my name!

- **identifier**: A name given to an entity in a program such as a class or method.
  - Identifiers allow us to refer to the entities.
- Examples (in **bold**):
  - `public class Hello`
  - `public static void main`
  - `public static void drawEgg`
- Conventions for naming in Java (which we will follow):
  - *classes*: capitalize each word (`ClassName`)
  - *methods*: capitalize each word after the first (`methodName`)

Identifiers: Syntax

- First character must be a letter, underscore (_) or $
- Following characters can be any of those or a number
- Examples:
  - legal:

```
susan       second_place     _myName
TheCure     ANSWER_IS_42     $variable
method1     myMethod         name2
```

  - illegal:

```
me+u        49er         question?
side-swipe  hi there     ph.d
jim's       2%milk       suzy@yahoo.com
```

- **Remember:** Java is case-sensitive (`name` is different from `Name`)

Identifiers: 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
```

- NB: Because Java is case-sensitive, you could technically use `Class` or `cLaSs` as identifiers, but this is very confusing and thus **strongly discouraged.**

Comments

- **comment**: A note written in the source code to make the code easier to understand.
  - Comments are not executed when your program runs.
  - Most Java editors show your comments with a special color.
- 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.
```


Comments: Where do you go?

- … at the top of each file (also called a "comment header"), naming the author and explaining what the program does
- … at the start of every method, describing its behavior
- … inside methods, to explain complex pieces of code

Comments: Why?

- **Comments provide important documentation.**
- Later programs will span hundreds or thousands of lines, split into many classes and methods.
- Comments provide a simple description of what each class, method, etc. is doing.
- When multiple programmers work together, comments help one programmer understand the other's code.

Comments: Example

```
/* Suzy Student
   CS 101, Fall 2019
   This program prints lyrics from my favorite song! */
public class MyFavoriteSong {
  /* Runs the overall program to print the song
     on the console. */
  public static void main(String[] args) {
    sing();

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

    sing();
  }

  // Displays the first verse of the theme song.
  public static void sing() {
    System.out.println("Now this is the story all about how");
    System.out.println("My life got flipped turned upside-down");
  }
}
```


Comments: How-to

- Do not describe the syntax/statements in detail.
- Instead, 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 my favorite TV theme song.
// 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");
}
```


Style guidelines

- Structure your code properly
- Eliminate redundant code
- Use spaces judiciously and consistently
- Indent properly
- Follow the naming conventions
- Use comments to describe code behavior

The importance of proper style

- Programmers build on top of other's code all the time.
  - You shouldn't waste time deciphering what a method does.
- You should spend time on thinking or coding. You should **NOT** be wasting time looking for that missing closing brace.
- So code with style!