Example from last Class (almost):

```
public class VariableTester {
    public static void main ( String[] args ) {
        int perhaps;
        int maybe;
        int niceValue = 4;
        perhaps =  niceValue+1;
        maybe = 17 % niceValue;
        System.out.println("perhaps: " + perhaps);
        System.out.println("maybe: " + maybe);
    }
}
```

```
perhaps: 4
  maybe: 1
```

Computing Concepts

- Computers execute simple instructions known as machine code. Examples:

  "ADD 1 to value x"

  "MOVE value y to location z"

  "IF t = 0, then jump to instruction I"

- Computers only know about numbers—integer values (e.g., 1, -2, etc.), floating-point values (e.g., 3.1415), addresses of memory and instructions.

- Even machine instructions are represented as numbers.

Computer Counting: What is a bit?

- BBin t
- 8 bits = 1 byte
- 1024 bytes = 1 kilo byte (Kbyte)

= 2x2x2x2x2x2x2x2x2x2 bytes

$=2^{10}$ bytes

Computer Counting: Powers of 2

Not to be confused with:

- 2^{10} = 1024 = 1K (Kilo)

10^3 = 1000 = 1K

- 2^{20} = 1,048,576 = 1M (Mega)

10^6 = 1,000000 = 1M

- 2^{30} = 1,073,741,824 = 1G (Giga)

10^6 = 1,000,000,000 = 1G

- 2^{40} = 1,099,511,627,776 = 1T (Tera)

10^9 = 1,000,000,000,000 = 1T

Counting like a Computer

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001

Algorithms
&
Static methods

Algorithms

- **algorithm**: A list of steps for solving a problem.
- How does one bake sugar cookies?

  (what is 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.
  - ...

How to make twice as many?

A program with redundancy

- **redundancy**: Occurrence of the same sequence of commands multiple times in a program.

```
public class TwoMessages {
    public static void main(String[] args) {
        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("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
```

  - The same messages are printed twice.

Static methods

- **static method**: A group of statements given a name.

- using a static method requires two steps:
  1. **declare** it (writing 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

- static methods are useful for:
  - denoting the *structure* of a larger program in smaller pieces
  - eliminating *redundancy* through reuse

Declaring a static method

- D*eclaring* a static method (writing down the recipe):

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

- Example:

-
```
public static void printWarning() {
    System.out.println("This product is known to cause");
    System.out.println("cancer in lab rats and humans.");
}
```

Calling a static method

- *Calling* a static method (cooking using the recipe):
  - In another method such as `main`, write:

    <method name> `();`

```
public class TheWarnings {
  public static void main(String[] args) {
    printWarning();
    printWarning();
  }
  public static void printWarning() {
     System.out.println("This product is known to cause");
     System.out.println("cancer in lab rats and humans.");
  }
}
```

```
This product is known to cause
cancer in lab rats and humans.
This product is known to cause
cancer in lab rats and humans.
```

A program w/ static method

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

  public static void displayMessage()
    System.out.println("Now this is the story all about how");
    System.out.println("My life got flipped turned upside-down");
  }
}
```

When to use static methods?

- Place statements into a static method if:
  - The statements are related to each other and form a part of the program's structure, or
  - The statements are repeated in the program.

- You need not create static methods for:
  - Individual statements only occurring once in the program.
  - Unrelated or weakly related statements.

Structured algorithms

- **structured algorithm**: Split into coherent tasks.

  1 <u>Make the cookie batter.</u>
  - Mix the dry ingredients.
  - Cream the butter and sugar.
  - Beat in the eggs.
  - Stir in the dry ingredients.

  2 <u>Bake the cookies.</u>
  - Set the oven temperature.
  - Set the timer.
  - Place the cookies into the oven.
  - Allow the cookies to bake.

  3 <u>Add frosting and sprinkles.</u>
  - Mix the ingredients for the frosting.
  - Spread frosting and sprinkles onto the cookies.

  ...

Removing redundancy

- A well-structured algorithm can describe repeated tasks with less redundancy.

1 Make the cookie batter.
- Mix the dry ingredients.
- ...

2a Bake the cookies (first batch).
- Set the oven temperature.
- Set the timer.
- ...

2b Bake the cookies (second batch).

3 Decorate the cookies.
- ...

Program with static method

```
public class FreshPrince {
    public static void main(String[] args) {
        rap();                  // Calling (running) the rap method
        System.out.println();
        rap();                  // Calling the rap method again
    }

    // This method prints the lyrics to my favorite song.
    public static void rap() {
        System.out.println("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
```

Output:

```
Now this is the story all about how
My life got flipped turned upside-down

Now this is the story all about how
My life got flipped turned upside-down
```

Those Cookies

17

A program with redundancy

```
public class BakeCookies {
    public static void main(String[] args) {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
```

Design of an algorithm

```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies2 {
    public static void main(String[] args) {
        // Step 1: Make the cake batter.
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");

        // Step 2a: Bake cookies (first batch).
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");

        // Step 2b: Bake cookies (second batch).
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");

        // Step 3: Decorate the cookies.
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
```

Final cookie program

```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies3 {
    public static void main(String[] args) {
        makeBatter();
        bake();         // 1st batch
        bake();         // 2nd batch
        decorate();
    }

    // Step 1: Make the cake batter.
    public static void makeBatter() {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
    }

    // Step 2: Bake a batch of cookies.
    public static void bake() {
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
    }

    // Step 3: Decorate the cookies.
    public static void decorate() {
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
```

Methods calling methods

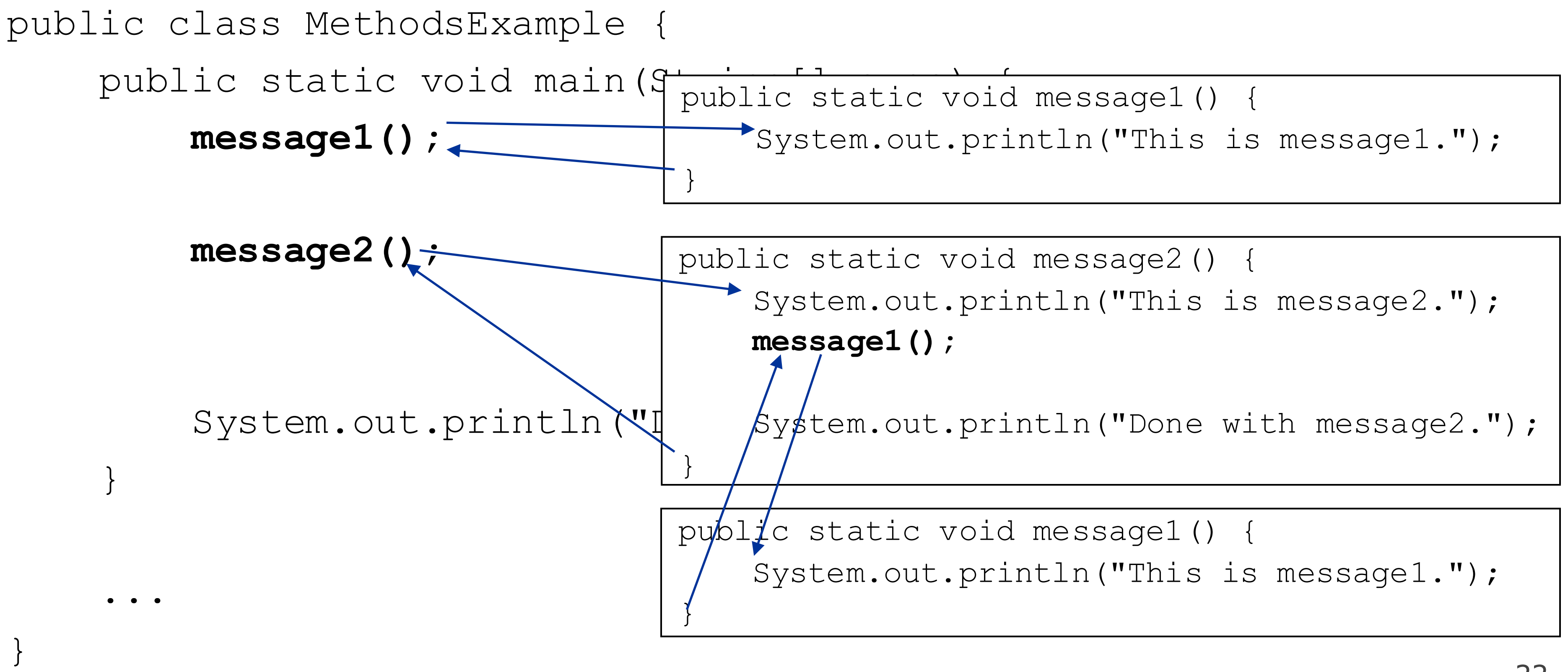
```
public class MethodsExample {
    public static void main(String[] args) {
        message1();
        message2();
        System.out.println("Done with main.");
    }

    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

- When a method is called, the program's execution...
  - "jumps" into that method, executing its statements, then
  - "jumps" back to the point where the method was called.



```
public class MethodsExample {
    public static void main(S
        message1();

        message2();


        System.out.println("I
    }

    ...
}
```

```
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.");
}
```

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

When to use methods

- Place statements into a static method if:
  - The statements are related structurally, and/or
  - The statements are repeated.

- You should not create static methods for:
  - An individual `println` statement.
  - Only blank lines. (Put blank `println`s in `main`.)
  - Unrelated or weakly related statements.
    (Consider splitting them into two smaller methods.)

Drawing complex figures with static methods

Static methods question

- Write a program to print these figures using methods.

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

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

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

Development strategy

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

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

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

First version (unstructured):

- Create an empty program and `main` method.
- Copy the expected output into it, surrounding each line with `System.out.println` syntax.
- Run it to verify the output.

Program version 1

```
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("+--------+");
    }
}
```

Development strategy 2

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

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

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

<u>Second version (structured, with redundancy):</u>

- Identify the structure of the output.
- Divide the `main` method into static methods based on this structure.

Output structure

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

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

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

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:

- `egg`
- `teaCup`
- `stopSign`
- `hat`

Program version 2

```
public class Figures2 {
    public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    public static void egg() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
    }

    public static void teaCup() {
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println("+--------+");
        System.out.println();
    }
    ...
```

Program version 2, cont'd.

```
    ...

    public static void stopSign() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("|  STOP  |");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
    }

    public static void hat() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("+--------+");
    }
}
```

Development strategy 3

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

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

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

Third version (structured, without redundancy):

- Identify redundancy in the output, and create methods to eliminate as much as possible.

- Add comments to the program.

Output redundancy

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

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

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

The redundancy in the output:

- egg top: reused on stop sign, hat
- egg bottom: reused on teacup, stop sign
- divider line: used on teacup, hat

This redundancy can be fixed by methods:

- `eggTop`
- `eggBottom`
- `line`

Program version 3

```
// Suzy Student, CSE 138, Spring 2094
// Prints several figures, with methods for structure and redundancy.
public class Figures3 {
    public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    // Draws the top half of an an egg figure.
    public static void eggTop() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
    }

    // Draws the bottom half of an egg figure.
    public static void eggBottom() {
        System.out.println("\\        /");
        System.out.println(" \\______/");
    }

    // Draws a complete egg figure.
    public static void egg() {
        eggTop();
        eggBottom();
        System.out.println();
    }

    ...
```

Program version 3, cont'd.

```
    ...
    // Draws a teacup figure.
    public static void teaCup() {
        eggBottom();
        line();
        System.out.println();
    }

    // Draws a stop sign figure.
    public static void stopSign() {
        eggTop();
        System.out.println("|  STOP  |");
        eggBottom();
        System.out.println();
    }

    // Draws a figure that looks sort of like a hat.
    public static void hat() {
        eggTop();
        line();
    }

    // Draws a line of dashes.
    public static void line() {
        System.out.println("+--------+");
    }
}
```