BUILDING JAVA PROGRAMS BY USING METHODS

Design of an algorithm

Algorithm: A list of steps for solving a problem

```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies2 {
    public static void main(String[] args) {
        // Step 1: Make the cookie dough.
        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.");
    }
}
```

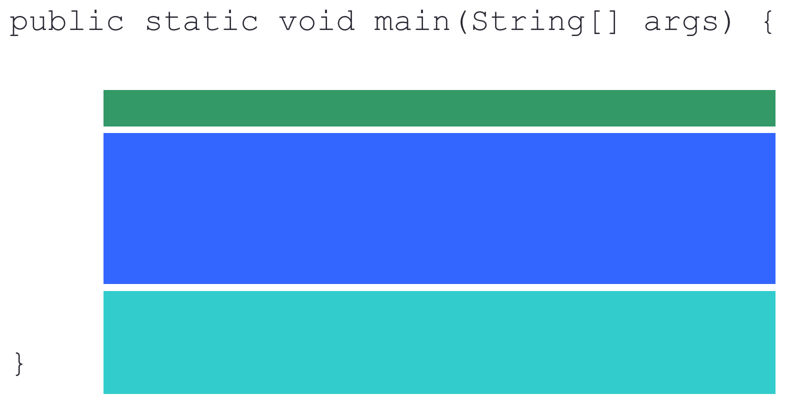
Cookie program

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

    // Step 1: Make the cookie dough.
    public static void makeDough() {
        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.");
    }
}
```

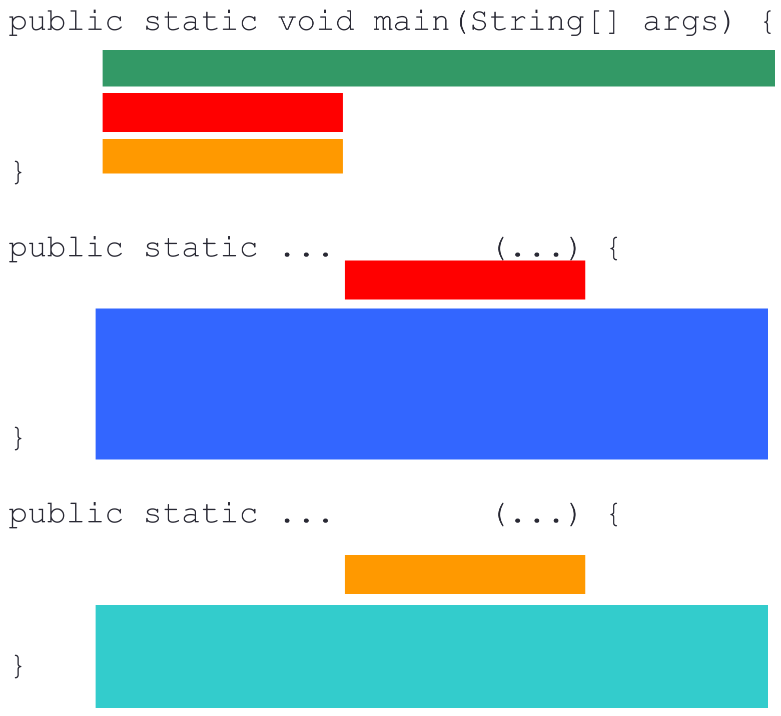
Why methods?

1) Makes code easier to read by capturing the structure of the program

- `main` should be a good summary of the program



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

}
```

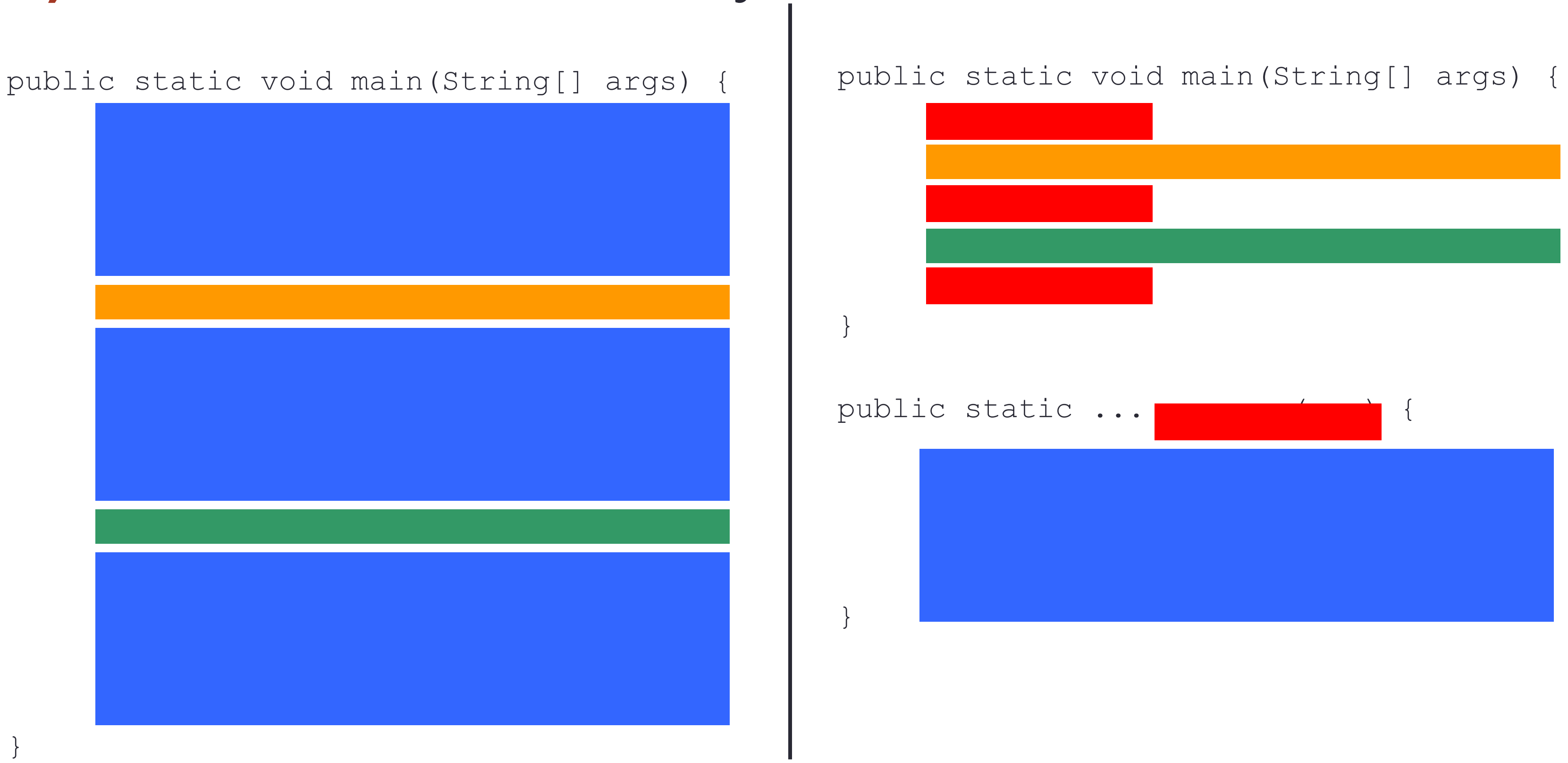


```
public static void main(String[] args) {

}

public static ...          (...) {

}

public static ...          (...) {

}
```

Note: Longer code doesn't necessarily mean worse code

Why else methods?

2) Eliminate redundancy



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

}
```

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

}

public static ...  ( ) {

}
```

Why else methods?

3) Easier to debug

Imagine if there was an error in the program on slide 2 in the code for baking. The first `println` statement should say “Set oven temperature to 350 degrees”. How many times would you have to correct the error?

Now imagine how much easier it would be to debug the program on slide 3. You would only have to correct the bug once! Moreover, you would know exactly where to look for the bug in the program.

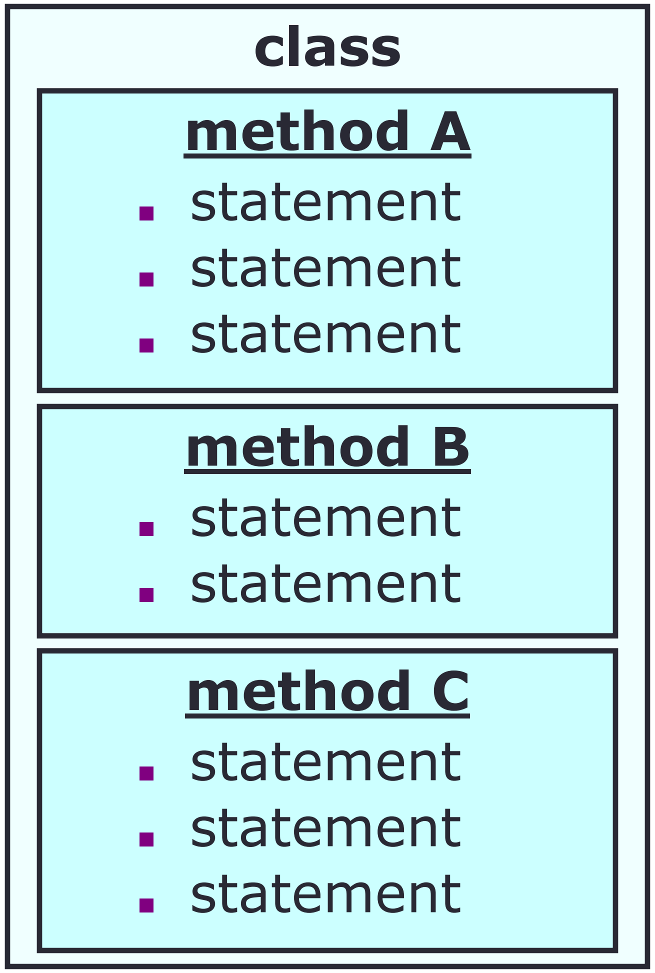
In an even longer program or one that contains many classes, the debugging issues are even more complicated.

static methods

- **static method**: A collection of statements, belonging to a class, grouped together to perform an operation.
  - denotes the *structure* of a program
  - eliminates *redundancy* by code reuse
  - method can be called in another method

- **procedural decomposition**: dividing a problem into methods

- **Writing a static method is like adding a new command to Java.**



The `main` method is a `static` method

A Java Program

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

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

Java Static Method

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

- Example:

```
public static void printWarning() {
    System.out.println("This product causes cancer");
    System.out.println("in lab rats and humans.");
}
```

Methods calling methods

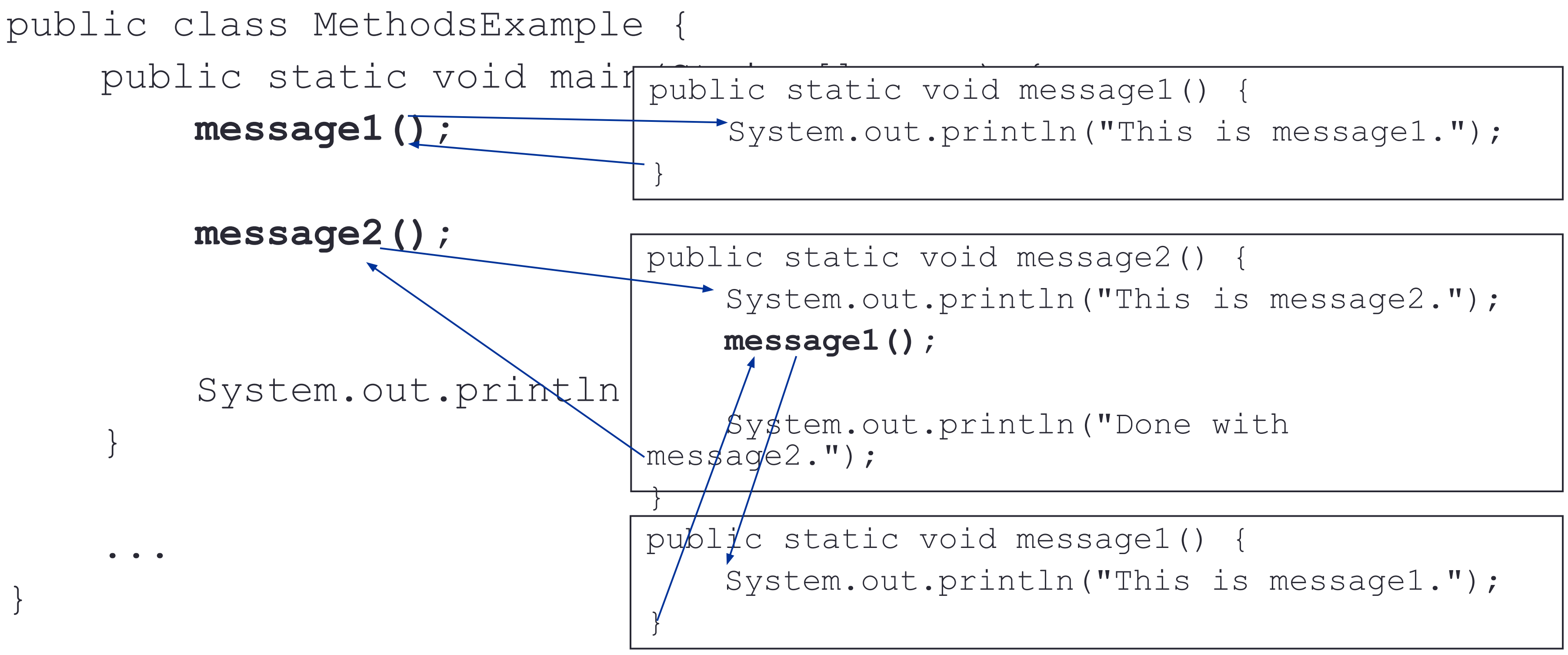
```
public class MethodsExample {
    public static void main(String[] args) {
        message1();
        message2();
        System.out.println("Done with main.");
    }
    public static void message1() {
        System.out.println("This is message1.");
    }
    public static void message2() {
 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

        message2();



        System.out.println
    }

    ...
}
```

```
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 NOT to use methods

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

Preconditions & Postconditions

Good programmers write preconditions and postconditions for their method in a multiline comment /* */ above header

Precondition: assumptions that must be true on entry into a method for it to work correctly

Postcondition: results expected at the exit of a method, assuming that the preconditions are met

```
/*
Precondition: parameters must be type int, first parameter is starting value, second parameter is value to add to start
Postcondition:  will calculate the sum of integers passed into method
*/
public static void main sumThis(int start, int end)
```

Method cohesion

cohesion - how well the statements in method accomplish the method's purpose

Good programmers *increase* cohesion

- operations should only contribute to a single task
- consider whether 2 methods should be combined if part of 1 task
- consider whether to split method into 2 methods if it does too much
- makes programs easier to understand, maintain, cohesive method is more functional
- subjective!

Examples: method that computes pay and prints it should be split into 2 methods; bake cookie program has with 20 statements should not have 20 methods!