CIS 110: Introduction to Computer Programming

Lecture 2

Decomposition and Static Methods

(§ 1.4)

Outline

- Structure and redundancy in algorithms
- Static methods
- Procedural decomposition

Reminders for the week

- Intro CIS 110/final exam surveys
- Sign up for and use Piazza!
- Lecture and lab content
- My office hours (GRW 260)
  - MW: one hr immediately after each lecture
  - Th: 5:30 – 6:30
  - Or email me for an appointment!

In Review: Why are We Here?

- Learning about *algorithmic thinking* via *computer programming!*
  1. Precision
  2. Decomposition
  3. Abstraction

Our focus for most of the semester!

Structure and Redundancy in Algorithms

Our Running Example

```
public class PBJ {
  public static void main(String[] args) {
    System.out.println(“Take out the ingredients and utensils.”);
    System.out.println(“Put ingredients and utensils on the table.”);
    System.out.println(“Remove the cap from the peanut butter.”);
    System.out.println(“Scoop out some peanut butter.”);
    System.out.println(“Spread it on a piece of bread.”);
    System.out.println(“Wash knife in the sink.”);
    System.out.println(“Wipe knife clean with a nupkin.”);
    System.out.println(“Remove the cap from the jelly.”);
    System.out.println(“Scoop out some jelly.”);
    System.out.println(“Spread it on a piece of bread.”);
    System.out.println(“Wash knife in the sink.”);
    System.out.println(“Wipe nife clean with napkin.”);
    System.out.println(“Put the two pieces of bread together.”);
    System.out.println(“Put the bread into your mouth and chew.”);
  }
}
```

Problem #1: Where's the Structure?

```
public class PBJ {
  public static void main(String[] args) {
    System.out.println("Take out the ingredients and utensils.");
    System.out.println("Put ingredients and utensils on the table.");
    System.out.println("Remove the cap from the peanut butter.");
    System.out.println("Scoop out some peanut butter.");
    System.out.println("Spread it on a piece of bread.");
    System.out.println("Wash knife in the sink.");
    System.out.println("Wipe knife clean with a nupkin.");
    System.out.println("Remove the cap from the jelly.");
    System.out.println("Scoop out some jelly.");
    System.out.println("Spread it on a piece of bread.");
    System.out.println("Wash knife in the sink.");
    System.out.println("Wipe nife clean with napkin.");
    System.out.println("Put the two pieces of bread together.");
    System.out.println("Put the bread into your mouth and chew.");
  }
}
```

Problem #2: We're Repeating Ourselves (Poorly)

```
public class PBJ {
  public static void main(String[] args) {
    System.out.println("Take out the ingredients and utensils.");
    System.out.println("Put ingredients and utensils on the table.");
    System.out.println("Remove the cap from the peanut butter.");
    System.out.println("Scoop out some peanut butter.");
    System.out.println("Spread it on a piece of bread.");
    System.out.println("Wash knife in the sink.");
    System.out.println("Wipe knife clean with a nupkin.");
    System.out.println("Remove the cap from the jelly.");
    System.out.println("Scoop out some jelly.");
    System.out.println("Spread it on a piece of bread.");
    System.out.println("Wash knife in the sink.");
    System.out.println("Wipe nife clean with napkin.");
    System.out.println("Put the two pieces of bread together.");
    System.out.println("Put the bread into your mouth and chew.");
  }
}
```

Structure and Redundancy

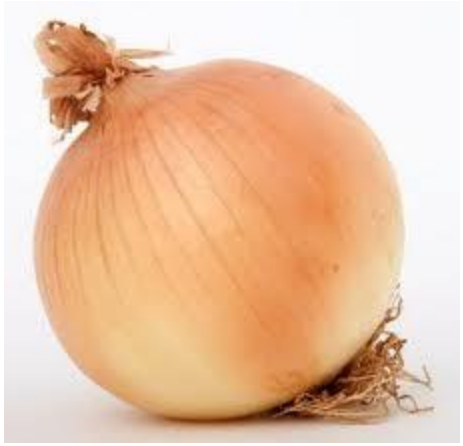
- Programs should reflect the structure of the problem at hand
  - Better understanding
- Programs should not contain redundancy
  - Better maintainability

What tool can we use to solve these problems?

Static Methods

First: a Note About Learning Programming Languages

1. There are lots of layers to a language
2. As you peel away the layers, you'll cry
3. But it's not the layer that causes you to cry, it's the *cutting itself* that does it!
   - It gets better the sharper your *programming language knife* becomes!



Recall: Syntax, Syntax, Syntax

- *Syntax*: the rules to form legal programs

Class template

```
public class <name> {
  <method>
  <method>
  …
  <method>
}
```

Method template

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

An Example Static Method

```
public class PBJ {

  public static void printPreamble() {
    System.out.println("Take out the ingredients and utensils.");
    System.out.println("Put ingredients and utensils on the table.");
  }

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

Method *signature* or *header*

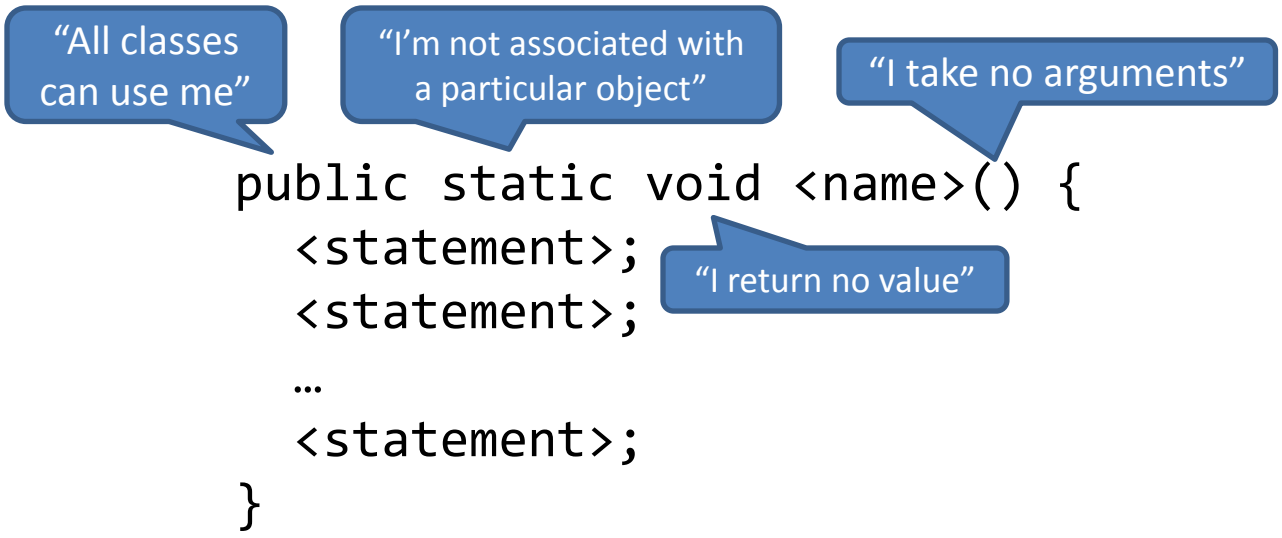
Method *body*

Method declaration

Method *call* or *invocation*

Static Method Declarations

- *(Static) methods* are named chunks of code that you can reuse.



- Methods can be declared in any order in a class.
- `main` is just another method (albeit special)!

Static Method Calls

- You use methods by *invoking* or *calling* them.

```
[Inside some method]
  …
  <name of method>();
  …
```

- Calling a method results in
  1. Executing the body of that method.
  2. Resuming execution right after you made the call.
- `System.out.println(…)` is just another method call!

Where Can We Call Methods?

- From `main`…

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

- But `main` is just another method, so really…

```
public static void myMethod() {
  someOtherMethod();
}
```

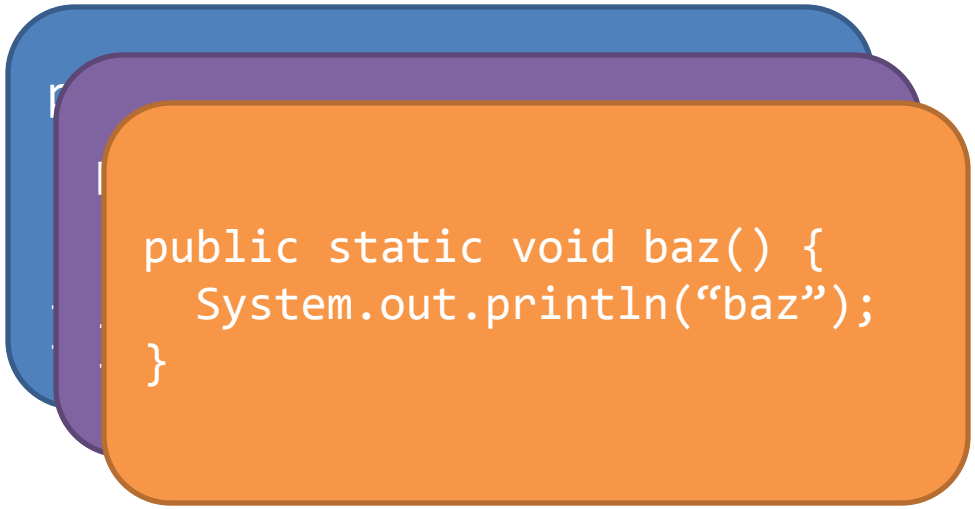
- *We can call any method from any other method!*

Control flow

```
class Foo {
  public static void
      main(String[] args) {
    System.out.println("main");
    bar();
    System.out.println("done");
  }

  public static void bar() {
    baz();
    System.out.println("bar");
  }

  public static void baz() {
    System.out.println("baz")
  }
}
```




```
Output:
main
baz
bar
done
```

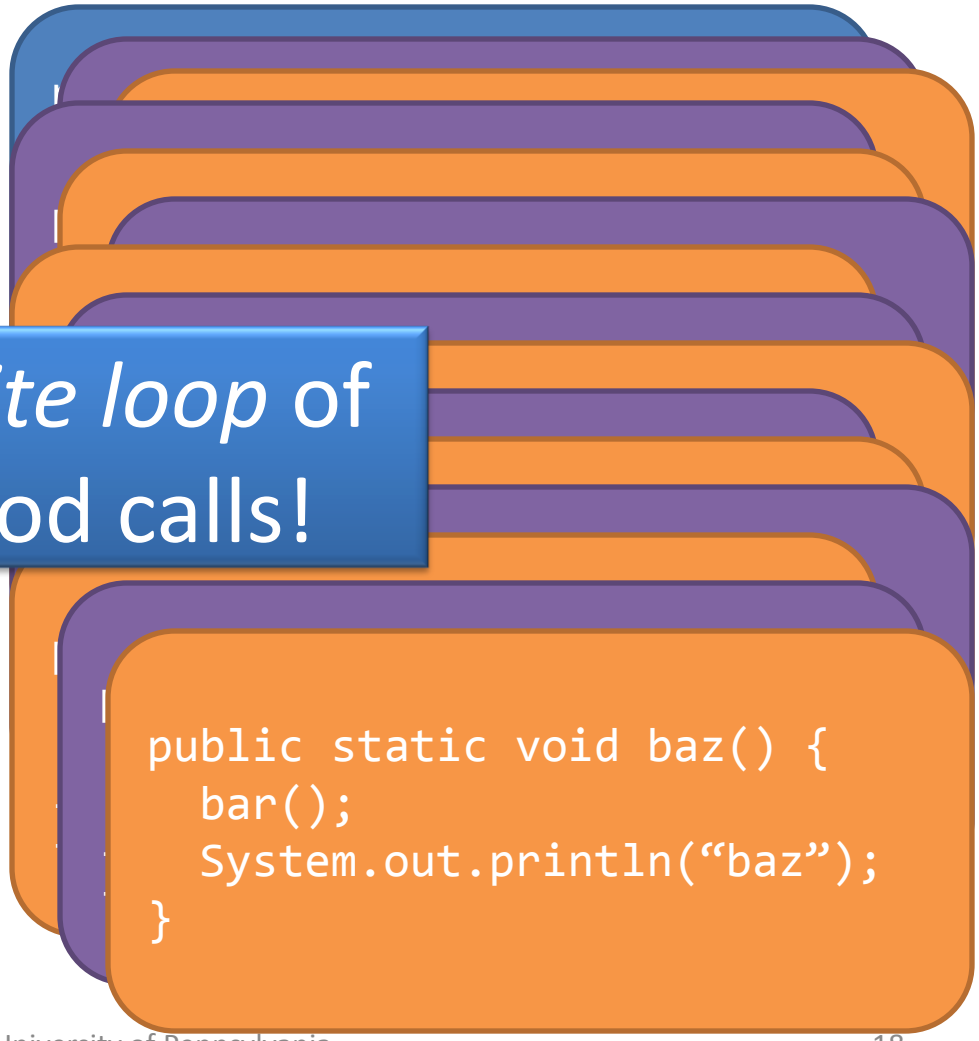
When Control Flow Goes Wild

```
class Foo {
  public static void
      main(String[] args) {
    System.out.println("main");
    bar();
    System.out.println("done");
  }

  public static voi
    baz();
    System.out.prin
  }

  public static void baz() {
    bar();
    System.out.println("baz");
  }
}
```

An *infinite loop* of method calls!



Runtime and Logic Errors

- Remember: compilation is only step #2!
- Your program may compile, but it might (probably!) still has errors to fix:
  - **Runtime errors**, e.g., infinite method call chains.
  - **Logic errors**, e.g., incorrect output.
- Lesson: compilation isn't the end! Always *test and check* your programs before you're done!

Using Static Methods to Capture Structure

```
// Prints instructions to make a PBJ sandwich
public class PBJ {

  // Prints the PBJ preamble to the screen
  public static void printPreamble() { /* … */ }

  // Prints the peanut butter step to the screen
  public static void printPeanutButterStep() { /* … */ }

  // Prints the jelly step to the screen
  public static void printJellyStep() { /* … */ }

  // Prints the eating step to the screen
  public static void printEatStep() { /* … */ }

  public static void main(String[] args) {
    printPreamble();
    printPeanutButterStep();
    printJellyStep();
    printEatStep();
  }
}
```

Using Static Methods to Reduce Redundancy

```
// Prints instructions to make a PBJ sandwich
public class PBJ {

  // Prints the clean up step to the screen
  public static void printCleanupStep() {
    /* … */
  }

  // Prints the peanut butter step to the screen
  public static void printPeanutButterStep() {
    /* … */
    printCleanupStep();
  }

  // Prints the jelly step to the screen
  public static void printJellyStep() {
    /* … */
    printCleanupStep();
  }

  public static void main(String[] args) { /* … */ }
}
```

Procedural Decomposition

Decompose, Decompose, Decompose

- Our focus thus far: *procedural decomposition*
  - "Procedures" are (non-object oriented) methods
- Two development strategies arise:
  - **Top-down development**
    - Start with empty main, write skeletons for methods you believe you need, fill them in.
  - **Iterative refinement**
    - Write a (relatively) complete program in main, factor out existing functionality into methods.
- Both approaches focus on
  - *Keeping your program in a compliable state.*
  - *Constantly checking and testing your program.*
- Not mutually exclusive, neither one better than the other.

Example #1: MarathonTraining

- Week 1
  - Monday: Rest
  - Tuesday: 4 miles
  - Wednesday: Rest
  - Thursday: 1-hour run
  - Friday: Rest
  - Saturday: 4 miles
  - Sunday: 6 miles
- Week 2
  - Monday: Rest
  - Tuesday: 4 miles
  - Wednesday: Rest
  - Thursday: 1-hour run
  - Friday: Rest
  - Saturday: 4 miles
  - Sunday: 7 miles
- Week 3
  - Monday: Rest
  - Tuesday: 4 miles
  - Wednesday: Rest
  - Thursday: 6 miles
  - Friday: Rest
  - Saturday: Rest
  - Sunday: 8 miles

Example #2: SimpleFigure