CS 112 Introduction to Programming

Lecture 3: Java Methods

Yang (Richard) Yang

Computer Science Department
Yale University
208A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Preston, do you consider programming more of an art or a science?

Quiet! I'm trying to cut and paste 300 lines of code into 7 different places!

Never mind.
Outline

- Admin. and recap
- Java methods
Practice Slides at the end of slides for Lecture 2

Office hours to be posted on the help page
- Preference on time and location of office hours?

PS1
- PS1 submission site on classes v2 to open today
- Please check PS1 rubric before submission
- Getting started sessions: preferred time?
- Please use Piazza or cs112ta@cs to ask questions
- You have 9 discretionary late days across the semester, but can use at most 3 days per PSET
Recap: Java Programming Steps

- Programming in Java consists of three tasks
  - edit java source code (.java files)
  - compile java source code to generate bytecode (.class files)
  - execute/run/test bytecode using an interpreter
Recap: Top-Down Java Syntax Structure

A class:
- has a name, defined in a file with same name
  Convention we follow: capitalize each English word
- starts with {, and ends with }
- includes a group of methods

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

A method:
- has a name
  Convention we follow: lowercase first word, capital following
- starts with {, and ends with }
- includes a group of statements

statement:
- a command to be executed
  - end with ;
Recap: The `System.out.println` Statement

- **Two ways to use the statement:**
  - `System.out.println("string");`
    - You may need to use escape sequences in strings
  - `System.out.println();`

- **A related statement is**
  `System.out.print("string");`
  **It does not print a newline**
Basic Java syntax units
- white space and comments
- identifiers (words)
- symbols: { } " ()<> [] ; = …
- strings
- numbers

```java
// This is a one-line comment
public class Hello {
    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");
    }
}
```
Java Syntax: A Bottom-Up Look

- Basic Java syntax units
  - white space and comments
  - identifiers (words)
  - symbols: { } " ( ) <> [ ] ; = ...
  - strings
  - numbers

Java depends on the identifiers and symbols to understand your program
Syntax Error: Example

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

- **Compiler output:**

```
Hello.java:2: <identifier> expected
     pooblic static void main(String[] args) {
          ^
Hello.java:3: ';' expected
     }  ^
Hello.java:4: ';' expected
     }  ^
2 errors
```

- The compiler shows the line number where it found the error.
- The error messages sometimes can be tough to understand:
  - Why can’t the computer just say “You misspelled ‘public’”? 
  - Since the computer knows that a “;” is missing, can’t it just fix it??
Java Programming Steps and Errors

- **Compile-time errors**
  - the compiler may find problems with syntax and other basic issues
  - if compile-time errors exist, an executable version of the program is not created

- **Run-time errors**
  - a problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (crash)

- **Logical errors**
  - a program may run, but produce incorrect results
Professional programmers typically use an Integrated Development Environment (IDE)

- Example IDEs: Eclipse, IntelliJ, DrJava, etc.
- An IDE usually presents the user with a space for text (like an editor) but layers additional features on top of the text for the user's benefit.
  - Note: The underlying file contains pure text, just like a text editor.
- These features can be very useful and save time.
  - Example features are GUI compile, GUI execution, code completion, and syntax highlighting.
- IDEs take more time to get started than a simple text editor, and we will arrange sessions to review how to use the Eclipse IDE
Roadmap

any program you might want to write

- objects
- **methods** and classes
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- math
- text I/O
- primitive data types
- assignment statements
Outline

- Admin. And recap
- Java methods
  - Motivation: why methods?
**Algorithms**

- **Algorithm**: A list of steps for solving a problem.
- An example algorithm (recipe): "Bake sugar cookies"
An Example Algorithm Spec: "Bake two batches of sugar cookies"

1. Preheat oven temperature to 375F.
2. Mix the dry ingredients.
3. Cream the butter and sugar.
4. Beat in the eggs.
5. Stir in the dry ingredients.
6. Set the timer for 8 min.
7. Place 1st batch of cookies to oven.
8. Allow the cookies to bake.
9. Set the timer for 8 min.
10. Place 2nd batch of cookies to oven.
11. Allow the cookies to bake.
12. Mix ingredients for frosting.
13. Spread frosting and sprinkles.

Readability of the specification?
Problem 1: Lack of Structure

- **Lack of structure**: Many tiny steps; tough to remember.
  - A human being typically can only manage seven (plus or minus 2) pieces of information at one time

http://www.michaeljemery.com/nlp/your-conscious-minds-capacity-seven-plus-or-minus-two-chunks-of-information/
Problem 2: Redundancy

- Redundancy: unnecessary repeat

1. Preheat oven temperature to 375F.
2. Mix the dry ingredients.
3. Cream the butter and sugar.
4. Beat in the eggs.
5. Stir in the dry ingredients.
6. Set the timer for 8 min.
7. Place the first batch of cookies into the oven.
8. Allow the cookies to bake.
9. Set the timer for 8 min.
10. Place the second batch of cookies into the oven.
11. Allow the cookies to bake.
12. Mix ingredients for frosting.
13. Spread frosting and sprinkles.
Structured algorithm: Split into coherent tasks.

1. Preheat oven.
   - Set oven to 375 degrees

2. Make the cookie batter.
   - Mix the dry ingredients.
   - Cream the butter and sugar.
   - Beat in the eggs.
   - Stir in the dry ingredients.

3. Bake the cookies.
   - Set the timer for 8 min.
   - Place the cookies into the oven.
   - Allow the cookies to bake.

4. Decorate the cookies.
   - Mix the ingredients for the frosting.
   - Spread frosting and sprinkles onto the cookies.
Structured Algorithm?

// This program displays a delicious recipe for baking cookies.
public class BakeCookies2 {
    public static void main(String[] args) {
        // Step 1: preheat oven
        System.out.println("Preheat oven to 375F.");

        // Step 2: Make the cookie 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 3a: Bake cookies (first batch).
        System.out.println("Set the timer for 8 min.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");

        // Step 3b: Bake cookies (second batch).
        System.out.println("Set the timer for 8 min.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");

        // Step 4: Decorate the cookies.
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
Structured Algorithms

- **Structured algorithm** provides **abstraction** (hide/ignore the right details at the right time)
  1. **Preheat oven.**
     - Set oven to 375 degrees
  
  2. **Make the cookie batter.**
     - Mix the dry ingredients.
     - Cream the butter and sugar.
     - Beat in the eggs.
     - Stir in the dry ingredients.
  
  3. **Bake the cookies.**
     - Set the timer.
     - Place the cookies into the oven.
     - Allow the cookies to bake.
  
  4. **Decorate the cookies.**
     - Mix the ingredients for the frosting.
     - Spread frosting and sprinkles onto the cookies.
Structured Algorithms

- **Structured algorithm** provides abstraction (hide/ignore the right details at the right time)
  1. Preheat oven.
  2. Make the cookie batter.
  3. Bake the cookies.
  4. Decorate the cookies.
Removing Redundancy

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

  1. Preheat oven.

  2. Make the cookie batter.

  3a. Bake the cookies (first batch).

  3b. Bake the cookies (second batch).

  4. Decorate the cookies.
Outline

- Admin. and recap
- Java methods
  - Motivation
  - Syntax: declaring method
Static Methods

- Arrange statements into groups and give each group a name.

- Each such named group of statements is a static method.

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

Gives your method a name so it can be referred to.

- **Syntax:**

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

- **Example:**

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

Executes the method's code

- Syntax:
  
  `<name>() ;`

  - You can call the same method many times if you like.

- Example:
  
  `printWarning();`

  - Output:
    
    This product causes cancer in lab rats and humans.
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");
    }
}
Example

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

    // Step 1: Preheat oven
    public static void preheatOven() {
        System.out.println("Preheat Oven to 375F.");
    }

    // Step 2: 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 3: Bake a batch of cookies.
    public static void bake() {
        System.out.println("Set the timer for 8 min.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
    }

    // Step 4: Decorate the cookies.
    public static void decorate() {
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
Examples: Modifying BakeCookies

- Bake three batches
- Change timer from 8 to 10 min
Summary: Why Methods?

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

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

```java
public static void main(String[] args) {
}
```
Summary: Why Methods?

- Eliminate redundancy

```java
public static void main(String[] args) {
    public static void main(String[] args) {
        ...
        ...
    }
}
```
Outline

- Admin. and recap
- Java methods
  - Motivations
  - Syntax: declaring method
  - Method control flow
Method Calling Flow

- When a method A calls another method B, the program's execution...
  - "jumps" into method B, executing its statements, then
  - "jumps" back to method A at the point where the method was called.
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.");
    }
}
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.
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.");
    }
    public static void message1() {
        System.out.println("This is message1.");
    }
}
Methods Calling Methods

Example: What is the output of Lullaby?
Outline

- Admin. and recap
- Java methods
  - Why methods?
  - Syntax: declaring method
  - Method control flow
  - Designing methods
Example

- Write a program to print these figures using methods.
public class Figures1 {
    public static void main(String[] args) {
        System.out.println("  ______");
        System.out.println(" / \\
        \\
        /");
        System.out.println(" | STOP | \\
        \\
        /");
        System.out.println("+--------+");
    }
}
A basic approach of designing methods, with consideration of structure and removing redundancy, is called top-down decomposition.

- Dividing a problem into subproblems to be solved using methods.
Top-Down Decomposition
Top-Down Decomposition (egg)
Top-Down Decomposition (teaCup)
Top-Down Decomposition (stopSign)
Q: What is a good order to implement/test the methods?
// 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 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();
    }
}
// Draws a line of dashes.
public static void line() {
    System.out.println("+--------+");
}

// 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();
    line();
}
A Word about Style

- **Structure** your code properly
- **Eliminate redundant** code

- Use comments to describe code behavior
- Use spaces judiciously and **consistently**
- Indent properly

- Follow the naming conventions
Why 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!
Foundational Programming Concepts

any program you might want to write

- objects
- methods and classes
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- Math
- text I/O
- primitive data types
- assignment statements
Backup Slides
Top-Down Decomposition