Announcements

• Course web: http://www.cs.stonybrook.edu/~alee/cse114/

• Send me your survey ASAP

• Why CSE 101 is strongly recommended?
  • Pace of CSE 114 assumes that you have done some programming
  • HTML is not a programming language!

• First programming assignment will be assigned on Wed

• Reading assignment for this week: Chapter 2 of Liang
  • The order of topics in my lecture notes are not identical to that in the textbook.
Static methods

- In Java, a named block of statements is known as a *method*
  - Also known as a *function* (or *subroutine*)

- You can think of methods as new Java commands (for now at least)

- A *static* method is a special kind of method that can be executed or run just by *calling* it
Side note: Java applets

- Java bytecodes are designed to be platform independent, so they could be embedded into webpages!
  - If there’s a JVM installed, a browser could run bytecodes as an “applet”
  - Actually, Java was designed specifically to allow this kind of programming
- In theory, Java applets are safe since the JVM runs applets in a secure “sandbox” which prevents applets from escaping and accessing the operating system directly
- In reality, sporadic security issues with the Java/JVM have made running unknown Java applets problematic
  - Java applets from known trusted sources are OK in general
- We are not addressing applets in this class
Method (from last lecture)

public class MyClass {
  public static void main (String[] args) {
    statement1;
    statement2;
    ...
  }
}

method name

method header

method body is a block of statements
- starts with a left brace {
- ends with a right brace }
Declaring a method

- **Give your method a name so it can be executed as a unit**
- **Syntax:**

```
public static void methodName() {
    statement;
    . . .
    statement;
}
```

- **Example:**

```
public static void printReply() {
    System.out.println("No, now go away or I shall");
    System.out.println("taunt you a second time.");
}
```
Calling a method

• *Execute the method’s code*

• Syntax:

```
methodName ( );
```

  parentheses are used to indicate
  methodName is a method

• Example:

```
printReply();
```

• Output:

  No, now go away or I shall
  taunt you a second time.
The printRowYourBoat() method

- Since the song had lot of redundancy, we can encapsulate the refrain into a method block by declaring a method!

  ```java
  public static void printRowYourBoat() {
      System.out.println("Row row row your boat");
      System.out.println("Gently down the stream");
      System.out.println("Merrily merrily merrily merrily");
      System.out.println("Life is but a dream");
  }
  ```

- Now, we can call (run) this method as many times as needed:

  ```java
  public static void main(String[] args) {
      printRowYourBoat();
      printRowYourBoat();
      printRowYourBoat();
      printRowYourBoat();
  }
  ```
Example: printing lyrics (bad one!)

- Suppose you need to print lyrics for schoolchildren to sing:

```java
public static void main(String[] args) {
    System.out.println("Row row row your boat");
    System.out.println("Gently down the stream");
    System.out.println("Merrily merrily merrily merrily");
    System.out.println("Life is but a dream");

    System.out.println("Row row row your boat");
    System.out.println("Gently down the stream");
    System.out.println("Merrily merrily merrily merrily");
    System.out.println("Life is but a dream");

    System.out.println("Row row row your boat");
    System.out.println("Gently down the stream");
    System.out.println("Merrily merrily merrily merrily");
    System.out.println("Life is but a dream");
}
```
When and why should you create a method?

• If you find yourself repeating a series of statements more than once, consider putting them in a method
  • *Procedural decomposition*: breaking a task down into smaller tasks
  • Helps with eliminating redundancy

• Methods help with:
  • Simplifying your code, making it more readable
  • Maintainability: fix a bug in one place as opposed to fixing the same bug in multiple places
  • *Abstraction*: others who use your method do not need to understand the details of your method – just that it works!
Methods calling methods

```java
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.

**Flow of control:**
1. Starts with the first statement of `main`
2. Follow method calls executing each method until it reaches the last statement of `main`
Common method declaration errors

- Don’t name your method the same as the class name, yet.
  - A method with the same name as the class has special meaning.
- Don’t declare your method inside another method declaration

```java
public class MethodError {
    public static void main(String[] args) {
        // some statements here
    }
    public static void method1() {
        System.out.println("This is a method1.");
        public static void method2() {
            System.out.println("This is method2.");
        }
    }
}
```
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 and continue

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

- Methods are defined or declared inside of a class and “belong” to that class
- Methods add functionality of abilities to classes
- Classes usually “group” similar or related methods together. For example, math functions in a Math class
- More on classes when we talk about objects later
Variables

• Continue in the next slide set