Defining Classes and Methods (Savitch, Chapter 5)

TOPICS
• Java methods
• Java objects
• Static keyword
• Parameter passing
• Constructors

Methods

• A **method** (a.k.a. function, procedure, routine) is a piece of code that performs a useful action
  – You defined a method called ‘main’.
  – When you run a Java program, it always begins by running the main method.
• A **method** can also return a value to the program that called them
  – More details in a minute...

Mysteries Revealed

```java
public class Temperature {
    public static void main(String[] args) {
        // your code here
    }
}
```

In our recitations and assignments, you define classes (e.g. P1, R1).

You also define a method called ‘main’ that takes an array of Strings as its arguments

Terminology

• A **class** is a data type
  – Combines variables with methods
• An **object** is an instance of a class
  – Must be explicitly created in program
• Creating an object is called **instantiation**
  – This involves use of **new** operator
Data inside objects and classes

• They are of two types
  – They may belong to the class (and will take the same value for all the objects)
  – They may belong to the object (and can take different values for each object)
  – Objects of the former type must be marked as static to allow the compiler to differentiate

Another mystery: static

• Methods are called with an instantiated object of the type class:
  – The notation is `objectname.method()`
  – You must have a String variable called `word` to call `word.length()`
  – The `length()` method can access data in the instance it is called on
  – Such methods are called instance methods

• Exception: static methods can be called with only the class name, i.e. no instance:
  – The notation is `classname.method()`
  – Not all methods need to access data specific to objects
  – Static declares that a method will not access instance data
  – Static methods may access class data, but not instance data

public static void main

• Remember that magic incantation at the start of your program?
  – `main` is the name of your method
    • The main method is called by the OS at program startup.
  – `void` says that the main function does not return a value
    • What would the OS do with a return value?
  – `static` says that main will not access instance variables
    • Because the OS needs to call it without creating a class instance
  – `public` is destined to remain a mystery just a bit longer.

Static methods

• `main` is an example of a static method
• It can only access class variables (or static variables)
• Therefore `main` cannot access instance variables. To use instance variables, we will have main create an instance of its class...
• But first, let’s see some static methods
  – First we will see static methods that don’t share data
  – Then we will see static methods that can share data
import java.util.Scanner;
public class SnowRemoval {
    public static void main(String[] args) {
        System.out.println("Enter your address:");
        Scanner keyboard = new Scanner(System.in);
        String address = keyboard.nextLine();
        int delay = snowService(address);
        if(delay==0)
            System.out.println("My driveway is safe now");
        else // assume status is non-negative
            System.out.println("I have to wait for " + delay + " hours");
    }
}

public static int snowService(String home){
    System.out.println("Clearing driveway of %s...", home);
    System.out.println("...done");
    System.out.println("Clearing sidewalk");
    return 0;
}

Communication between calling and called methods
• Method parameters:
  − Method declares a parameter “formal parameter” to state what can be provided by the calling program.

public String reverseCase (String s1)
  − Indicates the calling program must specify a String

public int returnRandom()
  − Indicates the calling program specifies no params
Communication between calling and called methods

```java
public String reverseCase (String s1)
public int returnRandom()
```

- Calling method:
  - Supplies arguments that must match the type of the parameters in the method declaration
  - Uses the return value to do something
  - Return value must match type of variable

```java
System.out.println(reverseCase(strname));
int i = returnRandom();
```

Caution: Pass by value

- What do you expect this to print?

```java
public class PassByValue {
    public static void main(String[] args) {
      int num = 100;
      increment(num);
      System.out.println("After calling increment, num is "+ num);
    }
    public static void increment(int n) {
      n++;
    }
  }
```

• The value of the argument is copied. Any changes to the copy are not reflected in the original argument.

Caution: Pass by value

- Another example

```java
public class PassByValueString {
    public static void main(String[] args) {
      String word = new String("Good morning");
      changeGreeting(word);
      System.out.println("After calling changeGreeting, word is "+ word);
    }
    public static void changeGreeting(String w) {
      w = new String("Good night");
    }
  }
```

• Greeting remains unchanged

Incorrect Swapping

```java
public class Swapper {
    public static void main(String[] args) {
      String s1 = "Marvin";
      String s2 = "Scorcese";
      swap(s1, s2);
      System.out.println("main: After swap, s1=" + s1 + " and s2=" + s2);
    }
    public static void swap(String x, String y) {
      System.out.println("swap: Before swap, x= " + x + " and y= " + y);
      String temp = x;
      x = y;
      y = temp;
      System.out.println("swap: After swap, x= " + x + " and y= " + y);
    }
  }
```

• Nothing gets swapped!
Use methods for subtasks

- The general rule is:
  -- Break subtasks into tasks until tasks are trivial
  -- Every subtask is a method
  -- Some methods (subtasks) may call others

Objects

- An object in Java is
  -- A set of methods (think: functions)
  -- A set of members (think: variables)

- Fancy CS buzzwords:
  -- Objects encapsulate data and functionality
  -- Objects encapsulate behavior and state

Object Example: String

- You have been using objects all along
- String is an example of an object in Java
  -- The characters are the data in the object
  -- Methods include:
    - length(): how long is the string?
    - charAt(int): what character is at a given position?
- Syntax:
  -- You call an object’s method using ‘.’ and args()
    - E.g.: word.charAt(5); word.length()

Another example: Scanner

- Scanner is a more complex object
- Its data is a stream of characters
  -- May come from a file
  -- May come from the terminal (a stream)
  -- May come from a string
- Its actions are to parse and interpret the characters
  -- next() returns the next valid string
  -- nextInt() returns the next valid integer
  -- nextDouble() returns the next valid double
  -- ... and there are many more (see on-line Java reference)
Classes as data types

- Classes are data types (just like primitives):
  ```java
  int counter;
  String word;
  MyClass example;
  ```
- By convention, class names are capitalized
- Variables with object types still need names
  - e.g. counter, word, and example above
- Variables cannot be used until they are assigned values
  - True for both primitive and object types

Object Instances

- The value assigned to a variable of an object type is an **object instance**
- For example:
  ```java
  String word = “the”;
  ```
  is the same as
  ```java
  String word = new String(“the”);
  ```
- word is a variable of type String.
- String(“the”) creates an instance of String

Object constructors

- All Object instances are created using the keyword **new**.

  ```java
  String s1 = new String(“example”);
  ```

- This creates a new string and calls the string constructor passing it the value “example”
Scanner terminal = new Scanner(System.in);

Scanner is an object class that parses character streams so that they can be easily read as strings, ints or other data types

Initializes terminal to be a specific Scanner that reads from System.in

Declares a variable called ‘terminal’ of type Scanner

More Mysteries Revealed

Methods inside a class

• Order of writing methods is arbitrary
  – Generally constructors are written first
• Shared data problem: what if two methods need to share data?
  – One subtask reads input and creates a string of words
  – Another subtask checks each word in the string and does something with it

Solution #1

• Method1 for subtask 1 returns a value, v
• Method2 for subtask 2 uses the value, v
• Example:
  public static void main(String[] args) {
    String wordList = readInput();
    processWords(wordList);
  }

Solution #2

• Use instance variables
  – Define String wordList; as an instance variable
  – Any method of a class can access its variables
    • readInput() can create & write the string
    • processWords() can access it
Data Variables in Classes

• How does a method access data in a class?
  – Every method can access the class instance it is called on
    • Think of word.length();, it can access the data in the string "word"
    • Think of the class instance as a "hidden" argument to the method
  – Class variables look like any other variables in the code of a method
    • They do not need to be "re-declared"

Simple example

```java
public class Course {
    String department, number;

    public Course(String dept, String num) {
        department = dept;
        number = num;
    }

    public String getFullName() {
        return new String(department + " " + number);
    }

    public static void main(String[] args) {
        Course c1 = new Course("CS", "160");
        System.out.println(c1.getFullName());
    }
}
```