Objects, Methods, Parameters, Input

Lecture 5, Thu Jan 19 2006

based on slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/cpsc111-06-spr
Reading This Week

- Rest of Chap 2
  - 2.3-4, 2.6-2.10
- Rest of Chap 4
  - 4.3-4.7
Objectives

- Understand when to use parameters
- Understand how to use return values
- Understand how to handle keyboard input
Recap: Constants

- Things that do not vary
  - unlike variables
  - will never change

- Syntax:
  - `final typeName variableName;`
  - `final typeName variableName = value;`

- Constant names in all upper case
  - Java convention, not compiler/syntax requirement
Recap: Avoiding Magic Numbers

- magic numbers: numeric constants directly in code
  - almost always bad idea!
    - hard to understand code
    - hard to make changes
    - typos possible
  - use constants instead
Recap: Classes, Methods, Objects

- **Class**: complex data type
  - includes both data and operations
  - programmers can define new classes
  - many predefined classes in libraries

- **Method**: operations defined within class
  - internal details hidden, you only know result

- **Object**: instance of class
  - entity you can manipulate in your program
Recap: Declare vs. Construct Object

public static void main (String[] args) {
    String firstname;
    firstname = new String ("Kermit");
}

- Variable declaration does not create object
  - creates object reference
- Constructor and new operator creates object somewhere in memory
  - constructors can pass initial data to object
- Assignment binds object reference to created object
  - assigns address of object location to variable
Recap: Declare vs. Construct Object

firstname

expression on right side of assignment operator

bind variable to expression on right side of assignment operator

String object

“Kermit”

expression on right side of assignment operator
Recap: Objects vs. Primitives

- **References**
  - Frog favoriteFrog
  - String frogName
    - boolean isMuppet: true
  - int famousNum:
    - int favoriteNum: 42

- **vs. Direct Storage**
  - int famousNum: 42
  - String frogName
    - int favoriteNum: 42
Recap: Objects vs. Primitives

- **References**
  - `Frog` object `favoriteFrog`
  - `Frog` object `famousFrog`

- **Frog object**
  - `String frogName`
  - `boolean isMuppet`
  - `false`

- **String object**
  - `"Kermit"`
  - `int famousNum`
  - `42`

- **int**
  - `favoriteNum`
  - `999`

- vs. direct storage
Recap: API Documentation

- Online Java library documentation at http://java.sun.com/j2se/1.5.0/docs/api/
  - textbook alone is only part of the story
  - let’s take a look!

- Everything we need to know: critical details
  - and often many things far beyond current need

- Classes in libraries are often referred to as Application Programming Interfaces
  - or just API
Recap: Some Available String Methods

public String toUpperCase();
Returns a new string object identical to this object but with all the characters converted to upper case.

public int length();
Returns the number of characters in this string object.

public boolean equals(String otherString);
Returns true if this string object is the same as otherString and false otherwise.

public char charAt(int index);
Returns the character at the given index. Note that the first character in the string is at index 0.
More String Methods

public String replace(char oldChar, char newChar);
Returns a new string object where all instances of oldChar have been changed into newChar.

public String substring(int beginIndex);
Returns new string object starting from beginIndex position

public String substring(int beginIndex, int endIndex);
Returns new string object starting from beginIndex position and ending at endIndex position

up to but not including endIndex char:

    substring(4, 7)    “o K”

Hello K e r m i t F r o g
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname = new String ("Kermit");
        String lastname = new String ("theFrog");
        firstname = firstname.toUpperCase();
        System.out.println("I am not " + firstname + " " + lastname);
    }
}

- invoking methods
  - `objectName.methodName();`
  - remember identifiers can't have . in them
Methods and Parameters

- Class definition says what kinds of data and methods make up object
  - object is specific instance of class

```java
String firstname = "Alphonse";
char thirdchar = firstname.charAt(2);
```
Methods and Parameters

- Class definition says what kinds of data and methods make up object
  - object is specific instance of class
  - methods are how objects are manipulated

```java
String firstname = "Alphonse";
char thirdchar = firstname.charAt(2);
```
Methods and Parameters

- Class definition says what kinds of data and methods make up object
  - object is specific instance of class
  - methods are how objects are manipulated
  - pass information to methods with parameters
    - inputs to method call
    - tell `charAt` method which character in the String object we're interested in

```java
String firstname = "Alphonse";
char thirdchar = firstname.charAt(2);
```
Parameters

- Methods can have multiple parameters
- API specifies how many, and what type

```java
public String replace(char oldChar, char newChar);

String animal = "mole";
animal.replace('m', 'v');

public String substring(int beginIndex, int endIndex);

animal = "aardwolf";
String newanimal = animal.substring(4,8);
System.out.println(newanimal);  // wolf
```
Explicit vs. Implicit Parameters

- Explicit parameters given between parentheses
- Implicit parameter is object itself
- Example: `substring` method needs
  - `beginIndex, endIndex`
  - but also the string itself!

```java
animal = "aardwolf";
System.out.println(animal);                   // aardwolf
String newanimal = animal.substring(4,8);
System.out.println(newanimal);               // wolf
```

- All methods have single implicit parameters
  - can have any number of explicit parameters
    - none, one, two, many…
Parameters

- Most of the time we'll just say parameters, meaning the explicit ones
Return Values

- Methods can have \textit{return values}
- Example: \texttt{charAt} method result
  - \texttt{return value}, the character 'n', is stored in \texttt{thirdchar}

```java
String firstname = "kangaroo";
char thirdchar = firstname.charAt(2);
```

Return Values

- Methods can have return values
- Example: `charAt` method result
  - return value, the character 'n', is stored in `thirdchar`

```java
String firstname = "kangaroo";
char thirdchar = firstname.charAt(2);
```

- Not all methods have return values
- Example: `println` method does not return anything
  - prints character 'n' on the monitor, but does not return that value
  - printing value and returning it are not the same thing!

```java
System.out.println(thirdchar);
```
Return Values

- Again, API docs tell you
  - how many explicit parameters
  - whether method has return value
  - what return value is, if so

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- No return value indicated as **void**
Constructors and Parameters

- Many classes have more than one constructor, taking different parameters
  - use API docs to pick which one to use based on what initial data you have

```
animal = new String();
animal = new String("kangaroo");
```
Accessors and Mutators

- Method that only retrieves data is **accessor**
  - read-only access to the value
  - example: `charAt` method of String class
- Method that changes data values internally is **mutator**
  - Stay tuned for examples of mutators, we haven't seen any yet
  - String class has no mutator methods
- Accessor often called getters
- Mutators often called setters
  - names often begin with get and set, as in `getWhatever` and `setWhatever`
Keyboard Input

- Want to type on keyboard and have Java program read in what we type
  - store it in variable to use later
- Want class to do this
  - build our own?
  - find existing standard Java class library?
  - find existing library distributed by somebody else?

- Scanner class does the trick
  - java.util.Scanner
  - nicer than System.in, the analog of System.out
import java.util.Scanner;

public class Echo
{
    public static void main (String[] args)
    {
        String message;
        Scanner scan = new Scanner (System.in);
        System.out.println("Enter a line of text: ");
        message = scan.nextLine();
        System.out.println("You entered: \\
                          + message + \\
                          ");
    }
}
import java.util.Scanner;

public class Echo {
    public static void main(String[] args) {
        String message;
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a line of text: ");
        message = scan.nextLine();
        System.out.println("You entered: \\
                         + message + \\
                         ");
    }
}

Import Scanner class from java.util package
Importing Packages

- Collections of related classes grouped into packages
  - tell Java which packages to keep track of with import statement
  - again, check API to find which package contains desired class
- No need to import String, System.out because core java.lang packages automatically imported
import java.util.Scanner;

public class Echo {
    public static void main (String[] args) {
        String message;
        Scanner scan = new Scanner (System.in);
        System.out.println("Enter a line of text: ");
        message = scan.nextLine();
        System.out.println("You entered: " + message + \\
                          + message + \\
                          + message + \\
                          + message + "\"");
    }
}

- Declare string variable to store what user types in
Scanner Class Example

```java
import java.util.Scanner;

public class Echo {
    public static void main (String[] args) {
        String message;
        Scanner scan = new Scanner (System.in);
        System.out.println("Enter a line of text: ");
        message = scan.nextLine();
        System.out.println("You entered: \
        + message + \
        ");
    }
}
```

- Use Scanner constructor method to create new Scanner object named `scan`
- could be named anything, like `keyboardStuff` or `foo`
import java.util.Scanner;

public class Echo
{
    public static void main (String[] args)
    {
        String message;
        Scanner scan = new Scanner (System.in);
        System.out.println ("Enter a line of text: ");
        message = scan.nextLine();
        System.out.println ("You entered: " + message + "\n");
    }
}

- Prompt user for input
import java.util.Scanner;

public class Echo {
    public static void main (String[] args) {
        String message;
        Scanner scan = new Scanner (System.in);
        System.out.println ("Enter a line of text: ");
        message = scan.nextLine();
        System.out.println ("You entered: \\
                             + message + \\
                             \\
        } }

- `nextLine` method reads all input until end of line
- returns it as one long string of characters
import java.util.Scanner;

public class Echo
{
    public static void main (String[] args)
    {
        String message;
        Scanner scan = new Scanner (System.in);
        System.out.println("Enter a line of text: ");
        message = scan.nextLine();
        // Print out the message on the display
        System.out.println("You entered: \\
                           + message + \\
                           ");
    }
}

■ Print out the message on the display
Scanner Class Example

- Let’s try running it
Questions?