**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

---

**Reading This Week**
- Rest of Chap 2
  - 2.3-4, 2.6-2.10
- Rest of Chap 4
  - 4.3-4.7

---

**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
Recap: Declare vs. Construct Object

- Variable declaration does not create object
- 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

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

Recap: Objects vs. Primitives

- References
- vs. direct storage

```java
Frog favoriteFrog
Frog famousFrog
String frogName
"Kermit"
boolean isMuppet
true
int famousNum
42
int favoriteNum
42
```

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

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

```java
public int length();
Returns the number of characters in this String object.
```

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

```java
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"

String Method Example

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);
    }
}

Methods and Parameters

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

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

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

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

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

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

Parameters

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

Parameters

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

Parameters

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

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

```
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!

```
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

```
Method Summary
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charAt</td>
<td>Returns the character at the specified index.</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

```
Constructor Summary
<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String()</td>
<td>Instantly creates a new string object so that it represents an empty string.</td>
</tr>
<tr>
<td>String(String original)</td>
<td>Instantly creates a new string object so that it represents the same sequence of characters as the argument; in other words, newly created object is a copy of the argument string.</td>
</tr>
</tbody>
</table>
```

```
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

#### 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 + ">");
    }
}
```

#### 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
- 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) {
        Scanner scan = new Scanner ((System.in));
        System.out.println ("Enter a line of text: ");
        String message = scan.nextLine();
        System.out.println ("You entered: \"" + message + \"\\n\"");
    }
}
```

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

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 + \"\\n\"");
    }
}
```

- Prompt user for input

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 + \"\\n\"");
    }
}
```

- `nextLine` method reads all input until end of line
- returns it as one long string of characters

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 + \"\\n\"");
    }
}
```

- Print out the message on the display

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 + \"\\n\"");
    }
}
```

Questions?