Objects, Classes, and Methods

CSIS 2101

Objects and Classes

- Java is an example of an "object-oriented language"
- Two types of "object-oriented languages"
  - Pure
    - E.g., Smalltalk, Eiffel, Ruby, some put Java here
  - Hybrid (has some object-oriented properties)
    - E.g., C++, Python, some put Java here

Objects and Classes

- In an Object-Oriented language, key building block to writing a program is the object.
- **Object:** an entity that you manipulate in your programs, by calling methods
- We've seen some already. Any Guesses Where?
  - "Hello, World!", "Hello, Class!", etc
  - System.out
- **Note:**
  - Numbers such as 12, 3, 1.3, 1.2E6, etc are **not** objects
  - They are what we will call in chpt 4 a primitive data type

Objects and Classes

- **Class:** a set of objects with the same behavior
- Each object belongs to a **class**. For example, `System.out` belongs to the class `PrintStream`

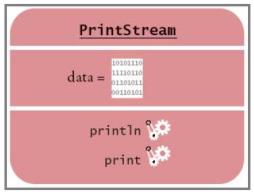

Figure 3:
Representation of the `System.out` object

Methods

- **Method:** Sequence of instructions that accesses the data of an object
- You manipulate objects by calling its methods
- Class: Set of objects with the same behavior
- Class determines legal methods

```
String greeting = "Hello";
greeting.println() // Error
greeting.length() // OK
```

Continued…

Syntax 1.1: Method Call

object.methodName(parameters)

Example:
```
System.out.println("Hello, Dave!");
```

Purpose:
To invoke a method of an object and supply any additional parameters

Methods

- **Public Interface:** Specifies what you can do with the objects of a class

String Methods

- `length:` counts the number of characters in a string

```
String greeting = "Hello, World!";
int n = greeting.length(); // sets n to 13
```

Continued...

String Methods

- `toUpperCase:` creates another String object that contains the characters of the original string, with lowercase letters converted to uppercase

```
String river = "Mississippi";
String bigRiver = river.toUpperCase();
// sets bigRiver to "MISSISSIPPI"
```

Continued...

String Methods

- When applying a method to an object, make sure method is defined in the appropriate class

```
System.out.length(); // This method call is an error
```

Implicit and Explicit Parameters

- **Parameter (explicit parameter):** Input to a method. Not all methods have explicit parameters.

```
System.out.println(greeting)
greeting.length() // has no explicit parameter
```

- **Implicit parameter:** The object on which a method is invoked

```
System.out.println(greeting)
```

Continued...

Return Values

- **Return value:** A result that the method has computed for use by the code that called it

```
int n = greeting.length(); // return value stored in n
```

Continued...

Passing Return Values

- You can also use the return value as a parameter of another method:

```
System.out.println(greeting.length());
```

- Not all methods return values. Example:

```
System.out.println("Hello!");
```

Note: the message printed to the terminal in this example is not a return value

Continued...

Passing Return Values

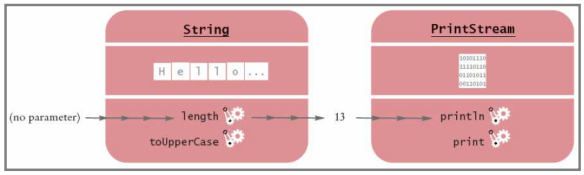

Figure 7:
Passing the Result of a Method Call to Another Method

A More Complex Call

- `replace` method carries out a search-and-replace operation

```
river.replace("issipp", "our")
  // constructs a new string ("Missouri")
```

- This method call has
  - one implicit parameter: the string **`"Mississippi"`**
  - two explicit parameters: the strings **`"issipp"`** and **`"our"`**
  - a return value: the string **`"Missouri"`**

Continued...

Method Definitions

- Method definition specifies types of explicit parameters and return value
- Type of implicit parameter = current class; not mentioned in method definition

Continued...

Method Definitions

- Example: `class String` defines

```
public int length()
   // return type: int
   // no explicit parameter
public String replace(String target, String replacement)
   // return type: String;
   // two explicit parameters of type String
```

Method Definitions

- If method returns no value, the return type is declared as `void`

```
public void println(String output) // in class PrintStream
```

- A method name is **overloaded** if a class has more than one method with the same name (but different parameter types)

```
public void println(String output)
public void println(int output)
```