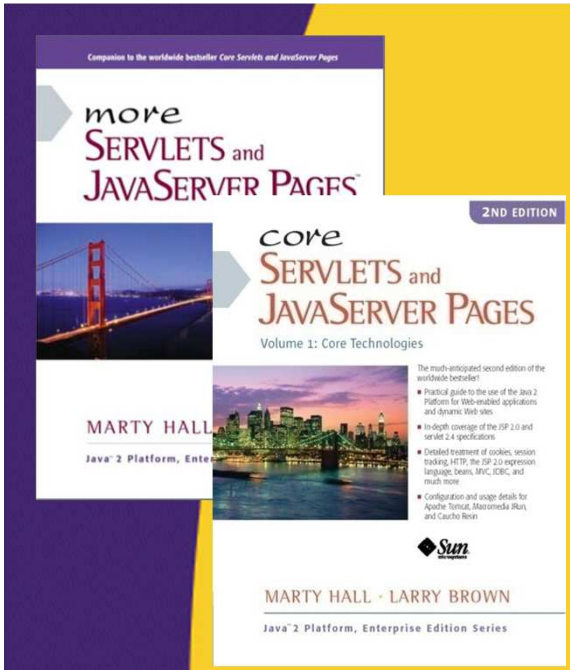
Course Material Usage Rules

- **PowerPoint slides for use only in full-semester, for-credit courses at degree-granting institutions**
  - Slides *not* permitted for use in commercial training courses except when taught by coreservlets.com (see http://courses.coreservlets.com).
- **Slides can be modified by instructor**
  - Please retain this notice and attribution to coreservlets.com
- **Instructor can give PDF or hardcopy to students, but should protect PowerPoint files**

  - *This slide is suppressed in Slide Show mode*

Basic Object-Oriented Programming in Java

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/java5.html


Companion to the worldwide bestseller Core Servlets and JavaServer Pages
more
SERVLETS and
JAVASERVER PAGES
MARTY HALL
Java 2 Platform, Enterprise
2ND EDITION
core
SERVLETS and
JAVASERVER PAGES
Volume 1: Core Technologies
Sun
MARTY HALL · LARRY BROWN
Java 2 Platform, Enterprise Edition Series

For live Java EE training, please see training courses at http://courses.coreservlets.com/.

Servlets, JSP, Struts, JSF 1.*x*, JSF 2.0, Ajax (with jQuery, Dojo, Prototype, Ext-JS, Google Closure, etc.), GWT 2.0 (with GXT), Java 5, Java 6, SOAP-based and RESTful Web Services, Spring, Hibernate/JPA, and customized combinations of topics.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact hall@coreservlets.com for details.

Topics in This Section

- Similarities and differences between Java and C++
- Object-oriented nomenclature and conventions
- Instance variables (fields)
- Methods (member functions)
- Constructors
- Example with four variations

"Object-oriented programming is an exceptionally bad idea which could only have originated in California." -- Edsger Dijkstra, 1972 Turing Award winner.

Basics

Object-Oriented Programming in Java

- **Similarities with C++**
  - User-defined classes can be used like built-in types.
  - Basic syntax
- **Differences from C++**
  - Methods (member functions) are the only function type
  - Object is the topmost ancestor for all classes
  - All methods use the run-time, not compile-time, types (i.e. all Java methods are like C++ virtual functions)
  - The types of all objects are known at run-time
  - All objects are allocated on the heap (always safe to return objects from methods)
  - Single inheritance only
- **Comparisons to C#**
  - C# very similar to Java in OOP. For details, see http://www.harding.edu/fmccown/java1_5_csharp_comparison.html

Object-Oriented Nomenclature

- **“Class” means a category of things**
  - A class name can be used in Java as the type of a field or local variable or as the return type of a function (method)
- **“Object” means a particular item that belongs to a class**
  - Also called an “instance”
- **Example**

```
String s1 = "Hello";
```

  - Here, String is the class, and the variable s1 and the value "Hello" are objects (or “instances of the String class”)

Instance Variables

Overview

- **Definition**
  - Data that is stored inside an object. “Instance variables” can also be called “data members” or “fields”.
- **Syntax**

```
public class MyClass {
    public SomeType field1, field2;
}
```

In any class that also has methods, it is almost always better to declare instance variables private. We will show how and why in the next tutorial section.

- **Motivation**
  - Lets an object have persistent values.
    - It is often said that in OOP, objects have three characteristics: state, behavior, and identity. The instance variables provide the state.

Ship Example 1: Instance Variables

```
public class Ship1 {                    (In Ship1.java)
  public double x, y, speed, direction;
  public String name;
}

public class Test1 {                    (In Test1.java)
  public static void main(String[] args) {
    Ship1 s1 = new Ship1();
    s1.x = 0.0;
    s1.y = 0.0;
    s1.speed = 1.0;
    s1.direction = 0.0;   // East
    s1.name = "Ship1";
    Ship1 s2 = new Ship1();
    s2.x = 0.0;
    s2.y = 0.0;
    s2.speed = 2.0;
    s2.direction = 135.0; // Northwest
    s2.name = "Ship2";
    ...
```

Instance Variables: Example (Continued)

```
    ...
    s1.x = s1.x + s1.speed
           * Math.cos(s1.direction * Math.PI / 180.0);
    s1.y = s1.y + s1.speed
           * Math.sin(s1.direction * Math.PI / 180.0);
    s2.x = s2.x + s2.speed
           * Math.cos(s2.direction * Math.PI / 180.0);
    s2.y = s2.y + s2.speed
           * Math.sin(s2.direction * Math.PI / 180.0);
    System.out.println(s1.name + " is at ("
                       + s1.x + "," + s1.y + ").");
    System.out.println(s2.name + " is at ("
                       + s2.x + "," + s2.y + ").");
  }
}
```

Instance Variables: Results

- **Compiling and running manually**

```
> javac Test1.java
> java Test1
```

Output:

```
Ship1 is at (1,0).
Ship2 is at (-1.41421,1.41421).
```

Example 1: Major Points

- Java naming conventions
- Format of class definitions
- Creating classes with “new”
- Accessing fields with “variableName.fieldName”

Java Naming Conventions

- **Start classes with uppercase letters**
  - Constructors (discussed later in this section) must exactly match class name, so they also start with uppercase letters

```
public class MyClass {
    ...
}
```

Java Naming Conventions

- **Start other things with lowercase letters**
  - Instance vars, local vars, methods, parameters to methods

```
public class MyClass {
  public String firstName, lastName;

  public String fullName() {
    String name =
      firstName + " " + lastName;
    return(name);
  }
}
```

Objects and References

- **Once a class is defined, you can declare variables (object reference) of that type**

```
Ship s1, s2;
Point start;
Color blue;
```

- **Object references are initially `null`**
  - The `null` value is a distinct type in Java and is not equal to zero
  - A primitive data type (e.g., int) cannot be cast to an object (e.g., String), but there are some conversion wrappers

- **The new operator is required to explicitly create the object that is referenced**

```
ClassName variableName = new ClassName();
```

Accessing Instance Variables

- **Use a dot between the variable name and the field**

```
variableName.fieldName
```

- **Example**
  - For example, Java has a built-in class called `Point` that has `x` and `y` fields

```
Point p = new Point(2, 3); // Build a Point object
int xSquared = p.x * p.x;  // xSquared is 4
int xPlusY = p.x + p.y;    // xPlusY is 5
p.x = 7;
xSquared = p.x * p.x;      // Now xSquared is 49
```

- **Exceptions**
  - Methods can access fields of current object without varName
    - See upcoming method examples
  - It is conventional to make all instance variables private
    - In which case outside code can't access them directly

Methods

Overview

- **Definition**
  - Functions that are defined inside a class. “Methods” can also be called “member functions”.
- **Syntax**

```
public class MyClass {
   public myMethod(…) { … }
}
```

If you want code that uses your class to access the method, make it public. If your method is called only by other methods in the same class, make it private.

- **Motivation**
  - Lets an object calculate values or do operations, usually based on its current state (instance variables).
    - It is often said that in OOP, objects have three characteristics: state, behavior, and identity. The methods provide the behavior.

Ship Example 2: Methods

(In Ship2.java)

```
public class Ship2 {
  public double x=0.0, y=0.0, speed=1.0, direction=0.0;
  public String name = "UnnamedShip";

  private double degreesToRadians(double degrees) {
    return(degrees * Math.PI / 180.0);
  }

  public void move() {
    double angle = degreesToRadians(direction);
    x = x + speed * Math.cos(angle);
    y = y + speed * Math.sin(angle);
  }

  public void printLocation() {
    System.out.println(name + " is at ("
                       + x + "," + y + ").");
  }
}
```

Methods (Continued)

(In Test2.java)

```
public class Test2 {
  public static void main(String[] args) {
    Ship2 s1 = new Ship2();
    s1.name = "Ship1";
    Ship2 s2 = new Ship2();
    s2.direction = 135.0; // Northwest
    s2.speed = 2.0;
    s2.name = "Ship2";
    s1.move();
    s2.move();
    s1.printLocation();
    s2.printLocation();
  }
}
```

- **Compiling and Running:**

```
javac Test2.java
java Test2
```

- **Output:**

```
Ship1 is at (1,0).
Ship2 is at (-1.41421,1.41421).
```

Example 2: Major Points

- Format of method definitions
- Methods that access local fields
- Calling methods
- Static methods
- Default values for fields
- public/private distinction

Defining Methods (Functions Inside Classes)

- **Basic method declaration:**

```
public ReturnType methodName(Type1 arg1,
                             Type2 arg2, ...) {
  ...
  return(somethingOfReturnType);
}
```

- **Exception to this format: if you declare the return type as void**
  - This special syntax that means “this method isn’t going to return a value – it is just going to do some side effect like printing on the screen”
  - In such a case you do not need (in fact, are not permitted), a `return` statement that includes a value to be returned

Examples of Defining Methods

- **Here are two examples:**
  - The first squares an integer
  - The second returns the faster of two **Ship** objects, assuming that a class called **Ship** has been defined that has a field named **speed**

```
// Example function call:
//   int val = square(7);

public int square(int x) {
  return(x*x);
}

// Example function call:
//   Ship faster = fasterShip(someShip, someOtherShip);

public Ship fasterShip(Ship ship1, Ship ship2) {
  if (ship1.speed > ship2.speed) {
    return(ship1);
  } else {
    return(ship2);
  }
}
```

Calling Methods

- **The term “method” means “function associated with an object” (I.e., “member function”)**
  - The usual way that you call a method is by doing the following:

```
variableName.methodName(argumentsToMethod);
```

- **For example, the built-in `String` class has a method called `toUpperCase` that returns an uppercase variation of a `String`**
  - This method doesn’t take any arguments, so you just put empty parentheses after the function (method) name.

```
String s1 = "Hello";
String s2 = s1.toUpperCase(); // s2 is now "HELLO"
```

Accessing External and Internal Methods

- **Accessing methods in other classes**
  - Get an object that refers to instance of other class
    - Ship s = new Ship();
  - Call method on that object
    - s.move();
- **Accessing instance vars in same class**
  - Call method directly (no variable name and dot in front)
    - move();
    - double d = degreesToRadians()
      - For local methods, you can use a variable name if you want, and Java automatically defines one called "this" for that purpose. See constructors section.
- **Accessing static methods**
  - Use ClassName.methodName(args)
    - double d = Math.cos(Math.PI/2);

Calling Methods (Continued)

- **There are two exceptions to requiring a variable name for a method call**
  - Calling a method defined inside the current class definition
    - Use “methodName(args)” instead of “varName.methodName(args)”
  - Functions (methods) that are declared “`static`”
    - Use “ClassName.methodName(args)”
- **Calling a method of the current class**
  - You don’t need the variable name and the dot
  - For example, a **`Ship`** class might define a method called **`degreeesToRadians`**, then, within another function in the same class definition, do this:

    ```
    double angle = degreesToRadians(direction);
    ```

    - No variable name and dot is required in front of **`degreesToRadians`** since it is defined in the same class as the method that is calling it

Static Methods

- **Also “class methods” (vs. “instance methods”)**
  - Static functions do not access any non-static methods or fields within their class and are almost like global functions in other languages
- **You call a static method through the class name**

```
ClassName.functionName(arguments);
```

  - For example, the **`Math`** class has a static method called **`cos`** that expects a **`double`** precision number as an argument
    - So you can call **`Math.cos(3.5)`** without ever having any object (instance) of the **`Math`** class
- **Note on the `main` method**
  - Since the system calls **`main`** without first creating an object, **`static`** methods are the only type of methods that **`main`** can call directly (i.e. without building an object and calling the method of that object)

Method Visibility

- **public/private distinction**
  - A declaration of private means that "outside" methods can't call it – only methods within the same class can
    - Thus, for example, the **`main`** method of the **`Test2`** class could not have done

      ```
      double x = s1.degreesToRadians(2.2);
      ```

      - Attempting to do so would have resulted in an error at compile time
  - Only say public for methods that you *want to guarantee your class will make available to users*
  - You are free to change or eliminate private methods without telling users of your class
- **private instance variables**
  - In next lecture, we will see that you almost always make instance vars private and use methods to access them

Declaring Variables in Methods

- **Format**
  - When you declare a local variable inside of a method, the normal declaration syntax looks like:

```
Type varName = value;
```

- **The value part can be:**
  - A constant
  - Another variable
  - A function (method) call
  - A constructor invocation (a special type of function prefaced by **new** that builds an object)
  - Some special syntax that builds an object without explicitly calling a constructor (e.g., strings)

Declaring Variables in Methods: Examples

```
int x = 3;
int y = x;

// Special syntax for building a String object
String s1 = "Hello";

// Building an object the normal way
String s2 = new String("Goodbye");

String s3 = s2;
String s4 = s3.toUpperCase(); // Result: s4 is "GOODBYE"

// Assume you defined a findFastestShip method that
// returns a Ship
Ship ship1 = new Ship();
Ship ship2 = ship1;
Ship ship3 = findFastestShip();
```

Constructors

Overview

- **Definition**
  - Code that gets executed when “new” is called
- **Syntax**
  - “Method” that exactly matches the class name and has no return type.
    - public class MyClass {
    -     public MyClass(…) { … }
    - }
- **Motivation**
  - Lets you build an instance of the class, and assign values to instance variables, all in one fell swoop
  - Lets you enforce that all instances have certain properties
  - Lets you run side effects when class is instantiated

Example: No User-Defined Constructor

- **Person**

```
public class Person1 {
  public String firstName, lastName;
}
```

- **PersonTest**

```
public class Person1Test {
  public static void main(String[] args) {
    Person1 p = new Person1();
    p.firstName = "Larry";
    p.lastName = "Ellison";
    // doSomethingWith(p);
  }
}
```

It took three lines of code to make a properly constructed person. It would be possible for a programmer to build a person and forget to assign a first or last name.

Example: User-Defined Constructor

- **Person**

```
public class Person2 {
  public String firstName, lastName;

  public Person2(String initialFirstName,
                 String initialLastName) {
    firstName = initialFirstName;
    lastName = initialLastName;
  }
}
```

- **PersonTest**

```
public class Person2Test {
  public static void main(String[] args) {
    Person2 p = new Person2("Larry", "Page");
    // doSomethingWith(p);
  }
}
```

It took one line of code to make a properly constructed person. It would not be possible for a programmer to build a person and forget to assign a first or last name.

Ship Example 3: Constructors

(In Ship3.java)

```
public class Ship3 {
  public double x, y, speed, direction;
  public String name;

  public Ship3(double x, double y,
               double speed, double direction,
               String name) {
    this.x = x; // "this" differentiates instance vars
    this.y = y; //  from local vars.
    this.speed = speed;
    this.direction = direction;
    this.name = name;
  }

  private double degreesToRadians(double degrees) {
    return(degrees * Math.PI / 180.0);
  }
  ...
```

Constructors (Continued)

```
  public void move() {
    double angle = degreesToRadians(direction);
    x = x + speed * Math.cos(angle);
    y = y + speed * Math.sin(angle);
  }
  public void printLocation() {
    System.out.println(name + " is at ("
                       + x + "," + y + ").");
  }
}

public class Test3 {                         (In Test3.java)
  public static void main(String[] args) {
    Ship3 s1 = new Ship3(0.0, 0.0, 1.0,   0.0, "Ship1");
    Ship3 s2 = new Ship3(0.0, 0.0, 2.0, 135.0, "Ship2");
    s1.move();
    s2.move();
    s1.printLocation();
    s2.printLocation();
  }
}
```

Constructor Example: Results

- Compiling and running manually

```
> javac Test3.java
> java Test3
```

- Output

```
Ship1 is at (1,0).
Ship2 is at (-1.41421,1.41421).
```

Example 3: Major Points

- Format of constructor definitions
- The “this” reference
- Destructors (not!)

Constructors

- **Constructors are special functions called when a class is created with new**
  - Constructors are especially useful for supplying values of fields
  - Constructors are declared through:

```
public ClassName(args) {
  ...
}
```

  - Notice that the constructor name must exactly match the class name
  - Constructors have no return type (not even `void`), unlike a regular method
  - Java automatically provides a zero-argument constructor if and only if the class doesn't define it's own constructor
    - That's why you could say

```
Ship1 s1 = new Ship1();
```

      in the first example, even though a constructor was never defined

The this Variable

- **The `this` object reference can be used inside any non-static method to refer to the current object**
- **The common uses of the `this` reference are:**
  1. To pass a reference to the current object as a parameter to other methods

     ```
     someMethod(this);
     ```

  2. To resolve name conflicts
     - Using **`this`** permits the use of instance variables in methods that have local variables with the same name

  – Note that it is only necessary to say **`this.fieldName`** when you have a local variable and a class field with the same name; otherwise just use **`fieldName`** with no **`this`**

Destructors

This Page Intentionally Left Blank

Example: Person Class

Idea

- **Goal**
  - Make a class to represent a person’s first and last name
- **Approach: 4 iterations**
  - Person with instance variables only
    - And test case
  - Add a getFullName method
    - And test case
  - Add a constructor
    - And test case
  - Change constructor to use “this” variable
    - And test case
    - Also have test case make a Person[]

Iteration 1: Instance Variables

Person.java

```
public class Person {
  public String firstName, lastName;
}
```

PersonTest.java

```
public class PersonTest {
  public static void main(String[] args) {
    Person p = new Person();
    p.firstName = "Larry";
    p.lastName = "Ellison";
    System.out.println("Person's first name: " +
                              p.firstName);
    System.out.println("Person's last name: " +
                              p.lastName);
  }
}
```

Iteration 2: Methods

Person.java

```
public class Person {
  public String firstName, lastName;

  public String getFullName() {
    return(firstName + " " + lastName);
  }
}
```

PersonTest.java

```
public class PersonTest {
  public static void main(String[] args) {
    Person p = new Person();
    p.firstName = "Bill";
    p.lastName = "Gates";
    System.out.println("Person's full name: " +
                                  p.getFullName());
  }
}
```

Iteration 3: Constructors

Person.java

```
public class Person {
  public String firstName, lastName;

  public Person(String initialFirstName,
                         String initialLastName) {
    firstName = initialFirstName;
    lastName = initialLastName;
  }

  public String getFullName() {
    return(firstName + " " + lastName);
  }
}
```

PersonTest.java

```
public class PersonTest {
  public static void main(String[] args) {
    Person p = new Person("Larry", "Page");
    System.out.println("Person's full name: " +
                                 p.getFullName());
  }
}
```

Iteration 4: Constructors with the “this” Variable (and Arrays)

Person.java

```
public class Person {
  public String firstName, lastName;

  public Person(String firstName,
                          String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  public String getFullName() {
    return(firstName + " " + lastName);
  }
}
```

PersonTest.java

```
public class PersonTest {
  public static void main(String[] args) {
    Person[] people = new Person[20];
    for(int i=0; i<people.length; i++) {
      people[i] =
         new Person(NameUtils.randomFirstName(),
                              NameUtils.randomLastName());
    }
    for(Person person: people) {
      System.out.println("Person's full name: " +
                                     person.getFullName());
    }
  }
}
```

Helper Class for Iteration 4

```
public class NameUtils {
  public static String randomFirstName() {
    int num = (int)(Math.random()*1000);
    return("John" + num);
  }

  public static String randomLastName() {
    int num = (int)(Math.random()*1000);
    return("Smith" + num);
  }
}
```

To Do: Later Iterations

- **Use accessor methods**
  - Make instance variables private and use getFirstName, setFirstName, getLastName, setLastName
- **Document code with JavaDoc**
  - Add JavaDoc-style comments so that online API for Person class will be useful
- **Use inheritance**
  - Make a class (Employee) based on the Person class. Don’t repeat the code from the Person class.
- **Next lecture**
  - Covers all of these ideas, then shows updated code

Wrap-Up

Summary

- **Conventions**
  - Class names start with upper case
  - Method names and variable names start with lower case
  - Indent nested blocks consistently
- **Example class**

```
public class Circle {
  public double radius; // We'll make this private next lecture
  public Circle(double radius) { this.radius = radius; }
  public double getArea() { return(Math.PI*radius*radius); }
}
```

- **Example usage**

```
Circle c1 = new Circle(10.0);
double area = c1.getArea();
```