Lecture 19: Objects

CS 170, Section 000
5 November 2009
Lecture Plan

- Logistics

  - Objects: data fields, methods, constructors (review)

- Using objects to build applications

  - Passing objects to/from methods
  - Static vs. instance methods
  - Access and visibility

- HW6 (due Wed, November 11)

- If time: preview of CS190: Web technologies
Logistics

• Homework 6: Due Wed Nov 11\textsuperscript{th}
  – Poker simulation

• Looks like we will have 9 homeworks total, not 10 (still will drop lowest).

• Midterm 2: Tuesday, Nov 17\textsuperscript{th}

• Final: Dec 10\textsuperscript{th}
Programming with Objects and Classes

• Define a class, or use existing classes in Java API
• Creating objects using a class and using the objects
Classes

- A Java class uses variables to define data fields
- A class uses methods to define behaviors which accesses or modifies the data fields
- A class provides a special type of methods, known as constructors, which are used to construct and initialize objects from the class
Circle example

- Circle
  - Properties: radius
  - Behavior: compute area
- Circle class
  - Data fields: radius
  - Constructors
  - Methods: getArea()
## UML design

### UML Class Diagram

<table>
<thead>
<tr>
<th>Circle</th>
<th>Class name</th>
<th>Data fields</th>
<th>Constructors and Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>radius: double</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circle()</td>
<td>Circle(newRadius: double)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getArea(): double</td>
<td></td>
</tr>
</tbody>
</table>

#### Objects

| circle1: Circle | radius: 10 |
| circle2: Circle | radius: 25 |
| circle3: Circle | radius: 125 |

### UML notation for objects
class Circle {
    /** The radius of this circle */
    double radius = 1.0;

    /** Construct a circle object */
    Circle() {
    }

    /** Construct a circle object */
    Circle(double newRadius) {
        radius = newRadius;
    }

    /** Return the area of this circle */
    double getArea() {
        return radius * radius * 3.14159;
    }
}

Data field
Constructors
Where is the main method?
Method
Constructors

```java
Circle() {
    // No-args constructor
}

Circle(double newRadius) {
    radius = newRadius;
    // Constructor that takes one double parameter
}
```

Constructors are a special kind of methods that are invoked to construct objects.
Creating and Using Objects

- Declaring object reference variables
- Creating objects
- Accessing objects
Declaring Object Reference Variables

To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
```

Example:
```
Circle myCircle;
```
Creating Objects Using Constructors

- Invoking a class constructor using the new operator to create an object
  ```java
  new ClassName();
  ```

- To reference an object, assign the object to a reference variable.
  ```java
  Example:
  Circle myCircle;
  myCircle = new Circle(5.0);
  ```
Declaring/Creating Objects in a Single Step

ClassName objectRefVar = new ClassName();

Example:
Circle myCircle = new Circle();
Accessing Objects

• Referencing the object’s data:
  \[ \text{objectRefVar.data} \]
  \[ e.g., \text{myCircle.radius} \]

• Invoking the object’s method:
  \[ \text{objectRefVar.methodName(arguments)} \]
  \[ e.g., \text{myCircle.getArea()} \]
Example: Circle Class and Tester Class

- Circle1.java – defines a Circle class which can be used to create Circle objects
- TestCircle1.java – a tester class that tests Circle class by creating Circle objects and invoking methods

- Trace the program of TestCircle1.java
Example: Circle step-by-step

```java
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```
Trace Code, cont.

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();
yourCircle.radius = 100;
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;

Create a new Circle object

myCircle reference value

: Circle
radius: 5.0

yourCircle no value

: Circle
radius: 0.0
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Assign object reference to yourCircle

myCircle

: Circle
radius: 5.0

yourCircle

: Circle
radius: 1.0
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();

yourCircle.radius = 100;
The null Value

• If a data field of a reference type does not reference any object, the data field holds a special value: null

```java
Circle c;
double r = c.getRadius();
```

![Compilation Error]

Reference value:
null
The null Value

• If a data field of a reference type does not reference any object, the data field holds a special value: null

```java
Circle c;
double r = c.getRadius();
Circle c = new Circle(10);
double d = c.getRadius();
```
Reference Data Fields

• The data fields can be of reference types
• For example, the following Student class contains a data field name of the String type
• String is a REFERENCE TYPE as well (it is a class)

```java
public class Student {
    String name; // name has default value null
    int age; // age has default value 0
    boolean isScienceMajor; // isScienceMajor has default value false
    char gender; // gender has default value (char)0
}
```
Default Value for a Data Field

• The default value of a data fields:
  – null for a reference type
  – 0 for a numeric type
  – false for a boolean type
  – (char)0 for a char type
Default Value for a Data Field

```java
public class Student {
    String name; // name has default value null
    int age; // age has default value 0
    boolean isScienceMajor; // isScienceMajor has default value false
    char gender; // gender has default value (char)0
}

public class Test {
    public static void main(String[] args) {
        Student student = new Student();
        System.out.println("name? " + student.name);
        System.out.println("age? " + student.age);
        System.out.println("isScienceMajor? " + student.isScienceMajor);
        System.out.println("gender? " + student.gender);
    }
}
```
Warning!

- Java assigns no default value to a local variable inside a method

```java
public class Test {
    public static void main(String[] args) {
        int x; // x has no default value
        String y; // y has no default value
        System.out.println("x is " + x);
        System.out.println("y is " + y);
    }
}
```

Compilation error: variables not initialized
Differences between Variables of Primitive Data Types and Object Types

Primitive type
- `int i = 1`
- `i`:
  - `1`

Object type
- `Circle c`
- `c`:
  - Reference

- Created using `new Circle()`
  - `c: Circle`
  - `radius = 1`
Copying Variables of Primitive Data Types and Object Types

Primitive type assignment  \( i = j \)

Before:

- \( i \) = 1
- \( j \) = 2

After:

- \( i \) = 2
- \( j \) = 2

Object type assignment \( c1 = c2 \)

Before:

- \( c1 \):
  - Circle
  - radius = 5
- \( c2 \):
  - Circle
  - radius = 9

After:

- \( c1 \):
  - Circle
  - radius = 5
- \( c2 \):
  - Circle
  - radius = 9
The Date Class

- Use Date class to create an instance for the current date and time and use its toString method to return the date and time as a string.

<table>
<thead>
<tr>
<th>java.util.Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Date()</td>
<td>Constructs a Date object for the current time.</td>
</tr>
<tr>
<td>+Date(elapseTime: long)</td>
<td>Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.</td>
</tr>
<tr>
<td>+toString(): String</td>
<td>Returns a string representing the date and time.</td>
</tr>
<tr>
<td>+getTime(): long</td>
<td>Returns the number of milliseconds since January 1, 1970, GMT.</td>
</tr>
</tbody>
</table>

The + sign indicates public modifier.
The Date Class Example

For example, the following code

```java
import java.util.Date;

Date date = new Date();
System.out.println(date.toString());
```

The Random Class

- Math.random(): a random double value between 0.0 and 1.0 (excluding 1.0).
- A more useful random number generator is provided in the java.util.Random class.

```
java.util.Random
+Random()
+Random(seed: long)
+nextInt(): int
+nextInt(n: int): int
+nextLong(): long
+nextDouble(): double
+nextFloat(): float
+nextBoolean(): boolean
```

- Constructs a Random object with the current time as its seed.
- Constructs a Random object with a specified seed.
- Returns a random int value.
- Returns a random int value between 0 and n (exclusive).
- Returns a random long value.
- Returns a random double value between 0.0 and 1.0 (exclusive).
- Returns a random float value between 0.0F and 1.0F (exclusive).
- Returns a random boolean value.
Random random = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++) {
    System.out.print(random.nextInt(1000) + " ");
}
Displaying GUI Components

- GUI programs use Java classes such as JFrame, JButton, JRadioButton, JComboBox, and JList to create frames, buttons, radio buttons, combo boxes, lists, and so on
Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```

Declare, create, and assign in one statement
Trace Code

```java
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```
Trace Code

```java
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```
Trace Code

```java
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);

JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);

JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);

Declare, create, and assign in one statement
Trace Code

JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);

JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```java
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```
Trace Code

JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
Example: Adding GUI Components to Window

• Let’s see how we can add components to a window!
Instance Variables, and Methods

- Instance variables belong to a specific instance
- Instance methods are invoked by an instance of the class
Static Variables, Constants, and Methods

• **Static variables** are shared by all the instances of the class

• **Static methods** are not tied to a specific object

• **Static constants** are final variables shared by all the instances of the class

• To **declare** static variables, constants, and methods, use the `static` modifier
Using Instance and Class Variables and Method

• Add a class variable `numberOfObjects` (static variable `numberOfObjects`) to track the number of Circle objects created so far
Caution

double power = Math.pow(3, 2.5);

Can I do the following?
double area = Circle1.getArea()

All the methods we used in Math are static methods, which are defined using the static keyword.

getArea() is non-static. It must be invoked from an object:

Circle1 circle = new Circle1(10);
double area = circle.getArea();
Visibility Modifiers and Accessor/Mutator Methods

• By default, the class, variable, or method can be accessed by any class in the same package.
  
  public
  The class, data, or method is visible to any class in any package.

  private
  The data or methods can be accessed only by the declaring class.
Why Data Fields Should Be private?

• To protect data
• To make the class easy to maintain
• Let’s consider a Thermometer class

```java
class Thermometer {
    private int temperature;

    public void setTemperature(int temp) {
        temperature = temp;
    }

    public int getTemperature() {
        return temperature;
    }
}
```
Encapsulation Comic Book

Billy

I don’t need any field encapsulation. If you want to change temperature, use:

t.temperature = 10;

The thermometer does not work for less than 5 degrees. We need to disable the scale when temperature is set below 5 degrees…

I need to cancel my Spring Break plans!!
The thermometer does not work for less than 5 degrees. We need to disable the scale when temperature is set below 5 degrees...

I will use field encapsulation. If you want to change the temperature, use:

```java
t.setTemperature(10);
```

OK boss…
```java
public void setTemperature(int t) {
    if (t < 5) {
        //disable scale
    } else {
        this.temperature = t;
    }
}
```
## Example of Data Field Encapsulation

The - sign indicates private modifier.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius: double</td>
<td>The radius of this circle (default: 1.0).</td>
</tr>
<tr>
<td>numberOfObjects: int</td>
<td>The number of circle objects created.</td>
</tr>
<tr>
<td>Circle()</td>
<td>Constructs a default circle object.</td>
</tr>
<tr>
<td>Circle(radius: double)</td>
<td>Constructs a circle object with the specified radius.</td>
</tr>
<tr>
<td>getRadius(): double</td>
<td>Returns the radius of this circle.</td>
</tr>
<tr>
<td>setRadius(radius: double): void</td>
<td>Sets a new radius for this circle.</td>
</tr>
<tr>
<td>getNumberOfObject(): int</td>
<td>Returns the number of circle objects created.</td>
</tr>
<tr>
<td>getArea(): double</td>
<td>Returns the area of this circle.</td>
</tr>
</tbody>
</table>

*Circle.java*  
*TestCircle3.java*
Passing Objects to Methods

- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

TestPassObject.java
Passing Objects to Methods, cont.

Stack
- Space required for the printAreas method
  - int times: 5
  - Circle c: reference

Space required for the main method
- int n: 5
- myCircle: reference

Heap
- A circle object

Pass by value (here the value is 5)
Pass by value (here the value is the reference for the object)
The meaning of this

- **this** is a keyword
- **this** serves as a proxy for the object

```java
public class ThisTest {
    private int number;
    public void setNumber(int number) {
        number = number;  // number = number;
    }
}
```
The meaning of `this`

- **this** is a keyword
- **this** serves as a proxy for the object

```java
public class ThisTest {
    private int number;
    public void setNumber(int number)
    {
        number = number;
    }
}
```

```java
public class ThisTest {
    private int number;
    public void setNumber(int number) {
        this.number = number;
    }
} 
```
public class ThisTest1 {
    private int number;

    ThisTest1(int n) {
      number = n;
    }

    ThisTest1() {
      this(1);
    }
}

Array of Objects

```java
Circle[] circleArray = new Circle[10];
```

- An array of objects is actually an array of reference variables
- Invoking `circleArray[1].getArea()` involves two levels of referencing
- `circleArray` references to the entire array
- `circleArray[1]` references to a Circle object.
Array of Objects, cont.

```java
Circle[] circleArray = new Circle[10];
```
Array of Objects, cont.

- Compute total area covered by all the circles

- TotalArea.java
HW6: Poker Simulation

• Decide value of hand given list of Card objects
  – Hand object (provided)
  – Card object: need to implement some of the methods
    • isStraightFlush()
  – Implement PokerSim class that randomly draws 5 cards from a Deck (class provided), and uses Card methods to evaluate each of the hands.
Practice problems

• Basic concept of objects and classes
  – 7.5
• Static variables and methods
  – 7.10, 7.12
• Visibility modifiers, accessor and mutator methods
  – 7.15
• Using Objects in Arrays and methods
  – 7.17, 7.20
Preview of CS190: Web Concepts and Technologies (Spring 2010)

• “Freshman seminar” on science & engineering of Web:
  – Concepts, ideas, and technology under the hood of the Web (focus on Web2.0 – web services, social tools)
  – Technology behind web search, Facebook(s), web services.
  – Individual and group programming projects + (final project or paper). There are no exams.

• First offered last semester (Spring 2010):
  http://www.mathcs.emory.edu/~eugene/cs190/

• Assumes no previous programming background (all of you are already over-qualified) 😊