Objects and Classes
(Part 1)

Introduction to Programming and Computational Problem Solving - 2
CSE 8B
Lecture 6
Announcements

• Assignment 2 is due today, 11:59 PM
• Quiz 2 is Oct 15
• Assignment 3 will be released today
  – Due Oct 20, 11:59 PM
• Educational research study
  – Oct 15, weekly survey
• Reading
  – Liang
    • Chapter 9
Object-oriented programming

• Object-oriented programming (OOP) involves programming using objects

• This is the focus of CSE 8B
  – The previous four lectures have been “double speed”
  – Beginning with this lecture, they will be “half speed”
Objects and classes

• An object represents an entity in the real world that can be distinctly identified
  – For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects
  – An object has a unique identity, state, and behaviors

• Classes are constructs that define objects of the same type
Objects

• An object has a unique identity, state, and behaviors
• The state of an object consists of a set of data fields (also known as properties) with their current values
• The behavior of an object is defined by a set of methods
Objects

• An object has both a state and behavior
  – The state defines the object
  – The behavior defines what the object does

Class Name: Circle
Data Fields:
  radius is ______
Methods:
  getArea

Circle Object 1
Data Fields:
  radius is 10

Circle Object 2
Data Fields:
  radius is 25

Circle Object 3
Data Fields:
  radius is 125

A class template
Three objects of the Circle class
Classes

• A Java class uses variables to define data fields and methods to define behaviors
• Additionally, a class provides a special type of methods, known as constructors, which are invoked to construct objects from the class
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
Method
Unified Modeling Language (UML)

UML Class Diagram

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

<table>
<thead>
<tr>
<th>circle1: Circle</th>
<th>UML notation for objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius = 1.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>circle2: Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius = 25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>circle3: Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius = 125</td>
</tr>
</tbody>
</table>
Constructors

• Constructors must have the same name as the class itself
• A constructor with no parameters is referred to as a no-arg constructor
  – It is a best practice to provide (if possible) a no-arg constructor for every class (we’ll cover why in two weeks)
• Constructors do not have a return type
  – Not even void
• Constructors are invoked using the new operator when an object is created
• Constructors play the role of initializing objects
Creating objects using constructors

new ClassName();

• For example
  
  new Circle();
  new Circle(5.0);
Default constructor

• A class may be defined without constructors
• In this case, a no-arg constructor with an empty body is *implicitly* defined in the class
• This constructor, called a *default constructor*, is provided automatically *only if no constructors are explicitly defined in the class*
  – It is a best practice to provide (if possible) a no-arg constructor for every class (we’ll cover why in two weeks)
Declaring object reference variables

• To reference an object, assign the object to a reference variable
• To declare a reference variable, use the syntax `ClassName objectRefVar;`
• For example
  `Circle myCircle;`
Declaring and creating in one step

ClassName objectRefVar = new ClassName();

For example
Circle myCircle = new Circle();
Accessing an object’s members

• Use the object member access operator
  – Also called the dot operator (.)
• Reference the object’s data using objectRefVar.data
  – For example
    myCircle.radius
• Invoke the object’s method using objectRefVar.methodName( arguments)
  – For example
    myCircle.getArea()

Member variables and methods do not use the dot operator to access other member variables and methods within the same class (but, when method formal parameters have the same name as a member, then member variables and methods must be accessed a special way; covered next lecture).
Trace code

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

Declare myCircle

myCircle

no value
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Create a new Circle object

myCircle

no value

: Circle
radius: 5.0
Trace code

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

Assign object reference to myCircle

myCircle reference value

radius: 5.0
Trace code

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

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

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

Assign object reference to yourCircle

myCircle reference value
: Circle
radius: 5.0

yourCircle reference value
: Circle
radius: 1.0
Trace code

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

Change radius in yourCircle

myCircle
reference value

yourCircle
reference value

: Circle
radius: 5.0

: Circle
radius: 100.0
Reference data fields and `null`

- The data fields can be of reference types
- For example, the following `Student` class contains a data field name of the `String` type

```java
public class Student {
    String name;
    int age;
    boolean isScienceMajor;
    char gender;
}
```

- If a data field of a reference type does not reference any object, then the data field holds the special Java literal value `null`
Default value for a data field

• The default value of a data field is null for a reference type
  0 for a numeric type
  false for a boolean type
  '\u0000' for a char type

```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; // c has default value '\u0000'
}
```
Default values

- **Note:** 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);
    }
}
```

Compile error: variable not initialized
Differences between variables of primitive data types and object types

- A variable of a primitive type holds a value of the primitive type
- A variable of a reference type holds a reference to where an object is stored in memory

<table>
<thead>
<tr>
<th>Primitive type</th>
<th>int i = 1</th>
<th>i</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object type</td>
<td>Circle c</td>
<td>c</td>
<td>reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Created using new Circle()

C: Circle
radius = 1
Differences between variables of primitive data types and object types

• Variable assignment

Primitive type assignment \( i = j \)

Before:

\[
\begin{align*}
1 & \quad j \\
2 & \quad i
\end{align*}
\]

After:

\[
\begin{align*}
2 & \quad j \\
2 & \quad i
\end{align*}
\]

Object type assignment \( c1 = c2 \)

Before:

\[
\begin{align*}
& c1 \\
& c2
\end{align*}
\]

\[
\begin{align*}
& \text{c1: Circle} \\
& \text{radius} = 5 \\
& \text{c2: Circle} \\
& \text{radius} = 9
\end{align*}
\]

After:

\[
\begin{align*}
& c1 \\
& c2
\end{align*}
\]

\[
\begin{align*}
& \text{c1: Circle} \\
& \text{radius} = 5 \\
& \text{c2: Circle} \\
& \text{radius} = 9
\end{align*}
\]
Garbage and its collection

• If an object is no longer referenced, then it is considered *garbage*

• Garbage occupies memory space

• Garbage collection
  – The JVM will automatically detects garbage and reclaims the space it occupies

• If you know an object is no longer needed, then you can explicitly assign `null` to the object reference variable
Using classes from the Java library

• The Java API contains a rich set of classes for developing Java programs

• Some commonly used ones
  – The String class
  – The java.util.Date class
  – The Math class
  – The java.util.Random class
    • More capable than Math.random method
Instance methods vs static methods

• An instance method can only be invoked from a specific instance of an object
  – The syntax to invoke an instance method is `referenceVariable.methodName(arguments)`

• A static method (i.e., a non-instance method) can be invoked without using an object (i.e., they are not tied to a specific object instance)
  – The syntax to invoke a static method is `ClassName.methodName(arguments)`
Next Lecture

• Objects and classes
• Reading
  – Liang
    • Chapter 9