Chapter 9 Objects and Classes

CS1: Java Programming
Colorado State University

Original slides by Daniel Liang
Modified slides by Chris Wilcox

Motivations

After learning the preceding chapters, you are capable of solving many programming problems using selections, loops, methods, and arrays. However, these Java features are not sufficient for developing graphical user interfaces and large scale software systems.

Suppose you want to develop a graphical user interface as shown below. How do you program it?

Object-Oriented Programming Concepts

Object-oriented programming (OOP) involves programming using objects. 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. 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, and the behavior defines what the object does.

Classes

Classes are constructs that define objects of the same type. 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.
### Component Diagram

**Class Diagram**

```
class Circle {
    double radius = 1.0;

    Circle() { /* This is a no-arg constructor */
        // Constructors are special methods that are invoked to construct objects.
    }

    Circle(double newRadius) {
        radius = newRadius;
    }

    double getArea() {  // Method
        return radius * radius * 3.14159;
    }
}
```

**Example: Defining Classes and Creating Objects**

Objective: Demonstrate creating objects, accessing data, and using methods.

```
Circle circle1 = new Circle();
circle1.radius = 1.0;
```

### Constructors

Constructors are a special kind of methods that are invoked to construct objects.

```
Circle() {
}

Circle(double newRadius) {
    radius = newRadius;
}
```

### Constructors, cont.

A constructor with no parameters is referred to as a no-arg constructor.

- Constructors must have the same name as the class itself.
- 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

```java
new ClassName();
```

Example:

```java
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.

Declaring Object Reference Variables

To reference an object, assign the object to a reference variable.

To declare a reference variable, use the syntax:

```java
ClassName objectRefVar;
```

Example:

```java
Circle myCircle;
```

Declaring/Creating Objects in a Single Step

```java
ClassName objectRefVar = new ClassName();
```

Example:

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

Accessing Object’s Members

- Referencing the object’s data:
  ```java
  objectRefVar.data
  ```
  e.g., `myCircle.radius`

- Invoking the object’s method:
  ```java
  objectRefVar.methodName(arguments)
  ```
  e.g., `myCircle.getArea()`

Trace Code

```java
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;

myCircle
radius: 5.0

Create a circle

yourCircle
radius: 5.0

Assign object reference to myCircle

Declare yourCircle

Create a new Circle object

Assign object reference to yourCircle

Change radius in yourCircle
Caution

Recall that you use
Math.methodName(arguments) (e.g., Math.pow(3, 2.5))
to invoke a method in the Math class. Can you invoke getArea() using
SimpleCircle.getArea()? The answer is no. All the methods used before
this chapter are static methods, which are defined using the static
keyword. However, getArea() is non-static. It must be invoked from an
object using

objectRefVar.methodName(arguments) (e.g., myCircle.getArea()).

More explanations will be given in the section on “Static Variables,
Constants, and Methods.”

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.

```java
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'
}
```

The null Value

If a data field of a reference type does not
reference any object, the data field holds a
special 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, and '\u0000' for a char type.
However, Java assigns no default value to a local
variable inside a method.

```java
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);
    }
}
```

Example

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

```java
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

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

Created using new Circle()
Copying Variables of Primitive Data Types and Object Types

**Primitive type assignment: i = j**

Before:

```
  i = 2
```

After:

```
  j = 2
```

**Object type assignment: c1 = c2**

Before:

```
  c2: Circle
  radius = 9
```

After:

```
  c2: Circle
  radius = 9
```

---

Garbage Collection

As shown in the previous figure, after the assignment statement \( c1 = c2 \), \( c1 \) points to the same object referenced by \( c2 \). The object previously referenced by \( c1 \) is no longer referenced. This object is known as garbage. Garbage is automatically collected by JVM.

---

Garbage Collection, cont

TIP: If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object. The JVM will automatically collect the space if the object is not referenced by any variable.

---

The Date Class

Java provides a system-independent encapsulation of date and time in the `java.util.Date` class. You can use the `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.

**java.util.Date**

- `Date()` Constructs a `Date` object for the current time.
- `Date(elapseTime: long)` Constructs a `Date` object for a given time in milliseconds elapsed since January 1, 1970, GMT.
- `toString()`: String Returns a string representing the date and time.
- `getTime()`: long Returns the number of milliseconds since January 1, 1970, GMT. Sets a new elapsed time in the object.

---

The Date Class Example

For example, the following code

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


---

The Random Class

You have used `Math.random()` to obtain 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**

- `nextDouble()`: double Constructs a `Random` object with the current time as its seed.
- `nextInt(n: int)`: int Returns a random integer value.
- `nextInt()`: int Returns a random int value between 0 and \( n \) (exclusive).
- `nextLong()`: long Returns a random long value.
- `nextFloat()`: float Returns a random double value between 0.0 and 1.0 (exclusive).
- `nextBoolean()`: boolean Returns a random boolean value.
The Random Class Example

If two `Random` objects have the same seed, they will generate identical sequences of numbers. For example, the following code creates two `Random` objects with the same seed 3.

```java
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
    System.out.print(random1.nextInt(1000) + " ");
Random random2 = new Random(3);
System.out.print("nFrom random2: ");
for (int i = 0; i < 10; i++)
    System.out.print(random2.nextInt(1000) + " ");
```

From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961

---

The Point2D Class

Java API has a convenient `Point2D` class in the `javafx.geometry` package for representing a point in a two-dimensional plane.

```java
Point2D point1 = new Point2D(3.0, 4.0);
Point2D point2 = new Point2D(5.0, 6.0);
```

---

Instance Variables, and Methods

Instance variables belong to a specific instance.

Instance methods are invoked by an instance of the class.

Instance variables and methods are specified by omitting the `static` keyword.

---

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.

---

Static Variables, Constants, and Methods, cont.

To declare static variables, constants, and methods, use the `static` modifier.
Example of Using Instance and Class Variables and Method

Objective: Demonstrate the roles of instance and class variables and their uses. This example adds a class variable `numberOfObjects` to track the number of Circle objects created.

```java
package p1;
public class C1 {
    public int x;
    int y;
    private int z;
    public void m1() {
        void m2() {
            private void m3() {
            }
        }
    }
}
```

The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.

```java
package p2;
public class C3 {
    void method1() {
        if (p == null) {
            cannot access p;
        }
        can invoke o.m1();
        cannot invoke o.m3();
    }
}
```

The default modifier on a class restricts access to within a package, and the public modifier enables unrestricted access.

NOTE

An object cannot access its private members, as shown in (b). It is OK, however, if the object is declared in its own class, as shown in (a).

```java
public class C2 {
    private boolean x;
    public static void main(String[] args) {
        C c = new C();
        System.out.println(c);
        System.out.println(c.convert());
    }
    private int convert() {
        return x ? 1 : 0;
    }
}
```

Why Data Fields Should Be private?

To protect data.

To make code easy to maintain.
**Example of Data Field Encapsulation**

<table>
<thead>
<tr>
<th>Circle</th>
<th>The radius of this circle (default: 1.0).</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius: double</td>
<td>The number of circle objects created.</td>
</tr>
<tr>
<td>(Circle)</td>
<td>Constructs a 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>getNumberOfObjects(): 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>

- The + sign indicates private modifier.

**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)

**Array of Objects**

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

An array of objects is actually an *array of reference variables*. So invoking `circleArray[1].getArea()` involves two levels of referencing as shown in the next figure. `circleArray` references to the entire array. `circleArray[1]` references to a Circle object.

**Array of Objects, cont.**

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

Summarizing the areas of the circles.

```java
TotalArea
```
Immutable Objects and Classes

If the contents of an object cannot be changed once the object is created, the object is called an immutable object and its class is called an immutable class. If you delete the set method in the Circle class in Listing 8.10, the class would be immutable because radius is private and cannot be changed without a set method.

A class with all private data fields and without mutators is not necessarily immutable. For example, the following class Student has all private data fields and no mutators, but it is mutable.

```java
public class Student {
    private int id;
    private BirthDate birthDate;

    public Student(int id, int year, int month, int day) {
        this.id = id;
        this.birthDate = new BirthDate(year, month, day);
    }

    public int getId() {
        return this.id;
    }

    public BirthDate getBirthDate() {
        return this.birthDate;
    }
}
```

What Class is Immutable?

For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.

Scope of Variables

- The scope of instance and static variables is the entire class. They can be declared anywhere inside a class.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be initialized explicitly before it can be used.

The this Keyword

- The **this** keyword is the name of a reference that refers to an object itself. One common use of the this keyword is reference a class’s hidden data fields.
- Another common use of the this keyword to enable a constructor to invoke another constructor of the same class.

```java
public class F {
    private int i = 5;
    private static double k = 0;

    void setI(int i) {
        this.i = i;
    }

    static void setK(double k) {
        F.k = k;
    }
}
```

Reference the Hidden Data Fields

```java
public class F {
    private int i = 5;
    private static double k = 0;

    void setI(int i) {
        this.i = i;
    }

    static void setK(double k) {
        F.k = k;
    }
}
```

Suppose that f1 and f2 are two objects of F.
- If f1 = new F(); f2 = new F();
- Invoking f1.setI(10) is to execute this.i = 10, where this refers f1.
- Invoking f2.setI(45) is to execute this.i = 45, where this refers f2.
public class Circle {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }
    public Circle() {
        this(1.0);
    }
    public double getArea() {
        return this.radius * this.radius * Math.PI;
    }
}

Every instance variable belongs to an instance represented by this, which is normally omitted.

This must be explicitly used to reference the data field radius of the object being constructed.

This is used to invoke another constructor.