Chapter 7 Objects and Classes

Motivations
- We already know a lot about programming
- But Java is Object-Oriented language, so where are all the objects?
- For developing graphical user interfaces and large scale software systems we need to know OO programming techniques

Levels of Abstraction: Software Design
- Old times: computer programs manipulated primitive types such as numbers and characters
- Manipulating too many of these primitive quantities is too much for programmers and leads to errors
- Solution: Encapsulate routine computations to software black boxes

Objects and Classes
- Object-oriented programming (OOP) involves programming using objects
- Object: entity that you can manipulate in your programs
  - Data fields: state of an object
  - Methods: instructions that accesses or modifies the state of an object
- Class: construct that defines objects of the same type (set of objects with the same behaviour)
  - Definition of data fields: properties of defined objects
  - Definition of methods: behaviours of defined objects
- Each object belongs to a class
Programming with Objects and Classes

- Defining a class
- Creating objects using a class and using the objects
- Using existing classes in Java API

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()
Constructors

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

Circle() {  // No-args constructor
}

Circle(double newRadius) {
    radius = newRadius;
}

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

Declaring Object Reference Variables

To declare a reference variable, use the syntax:

Class Name objectRefVar;

Example:

Circle myCircle;

Creating Objects Using Constructors

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

- To reference an object, assign the object to a reference variable:

  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:
  objectRefVar.data
e.g., myCircle.radius

- Invoking the object’s method:
  objectRefVar.methodName{arguments}
e.g., 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

Primitive data types vs. object data types

- Variables of primitive data types store the actual value
- Variables of object types store the reference to the object

Copying Variables of Primitive Data Types and Object Types

- Primitive type assignment: i = j
  Before: i = 1
  After: i = 2

- Object type assignment: c1 = c2
  Before: c1: Circle
  c1: radius = 5
  After: c2: Circle
  c2: radius = 9

The null Value

- What if we forget to create an object and assign the reference to our object variable?
- If a data field of a reference type does not reference any object, the data field holds a special value: null
  Circle c;
  double r = c.getRadius();
  Compilation Error

Examples Labeled as: 19, 20, 21, 22, 23, 24
Create an object and initialize the object reference variable before trying to use it

```java
Circle c = new Circle(10);
```

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; // 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
}
```

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

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

Bank account example

- State (property) of bank account
  - current balance
  - ...
- Behaviour of bank account
  - deposit money
  - withdraw money
  - get balance
  - ...
Define and implement BankAccount class

- Instance fields
  - balance
- Constructors initialize instance fields
- Methods may nor may not change the instance fields, and may or may not return an output value
  - withdraw()
  - deposit()
  - getBalance()

Test the Class – BankAccountTester

- Create a test class: a class with a main method that contains statements to test another class.
  1. Construct one or more objects of the class that is being tested
  2. Invoke one or more methods
  3. Print out one or more results

BankAccount.java
BankAccountTester.java

Lab Today

- Creating and testing a Square class

Outline

- Review
  - Concepts of objects and classes
  - Defining a class
    - Data fields, constructors, methods
  - Creating objects from a class
- Today
  - Categories of variables
  - Using existing classes from Java API
  - Static variables and methods
  - Access modifiers
  - Using objects in methods and arrays

Review

- When you run the BankAccountTester program, how many objects of class BankAccount are constructed? How many objects of type BankAccountTester?

Categories of Variables

- Instance fields
  - Instance fields belong to an object
    - E.g. radius in Circle1
- Local variables
  - Local variables belongs to a method
    - E.g. area in Circle1.getArea() method
- Parameter variables
  - Parameter variables belong to a method
    - E.g. newRadius in Circle1 constructor
Implicit and Explicit Parameters

- **Explicit parameter**: defined in the method definition
  - Created upon method call, initialized with the values supplied
- **Implicit parameter**: the object on which the method is invoked
  - The `this` reference denotes the implicit parameter
  - Created upon method call, initialized to the object on which the method is invoked
  - Use of an instance field name in a method denotes the instance field of the implicit parameter

Categories of Variables

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Lifetime</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance variables</td>
<td>Created when object is constructed and die when object terminates</td>
<td>Initialized with default value or explicitly</td>
</tr>
<tr>
<td>Local variables</td>
<td>Created when method is invoked and die when method exits</td>
<td>Must be initialized explicitly</td>
</tr>
<tr>
<td>Parameter variables</td>
<td>Implicit parameter: the object the method is invoked on</td>
<td>Explicit parameter: the value supplied in the method call</td>
</tr>
</tbody>
</table>

Using this

- Use `this` explicitly when there is ambiguity

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

```java
public class ThisTest1 {
    private int number;
    ThisTest1(int n) {
        number = n;
    }
    ThisTest1() {
        this(1);
    }
}
```

this - continued

```java
double getArea() {
    double area = radius * radius * Math.PI;
    // double area = this.radius * this.radius * Math.PI;
    return area;
}
```

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

this Keyword

- The `this` reference denotes the implicit parameter
- Compiler implicitly use this to reference the instance variables of the object on which the method is invoked

Programming with Objects and Classes

- Defining and testing classes by creating objects
  - Using existing classes in Java API library
    - Check API for the usage – required parameters and return values
Getting Input Using Scanner Class

0. Import Scanner class
   ```java
   import java.util.Scanner;
   ```

1. Create a Scanner object
   ```java
   Scanner scanner = new Scanner(System.in);
   ```

2. Use methods `next()`, `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, or `nextBoolean()` to obtain a string, byte, short, int, long, float, double, or boolean value.
   ```java
   System.out.print("Enter a double value: ");
   Scanner scanner = new Scanner(System.in);
   double d = scanner.nextDouble();
   ```

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.

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

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
  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.

The Random Class Example

  ```java
  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

  ```java
  TestFrame.java
  GUIComponents.java
  ```
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 constants** are final variables shared by all the instances of the class
- **Static methods** are not tied to a specific object

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

  - Circle2.java
  - TestCircle2.java

Caution

```java
double power = Math.pow(3, 2.5);
Can we do the following?
double area = Circle1.getArea();
```

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

```java
getArea() is non-static. It must be invoked from an object:
Circle1 circle = new Circle1(10);
double area = circle.getArea();
```

Visibility Modifiers

- 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.

```java
static int numberOfCircles;
```

```java
Math.PI
Math.random();
```
Accessor/Mutator Methods

- **Accessor methods**
  - Only access the data fields without modifying them
  - E.g. `getArea()` of `Circle1`

- **Mutator methods**
  - Modify data fields
  - E.g. `scale()` of `Square`

Visibility Modifiers and Accessor/Mutator Methods

```
package p1;
package p2;
public class C3 {
    void aMethod() {
        C1 o = new C1();
        can access o.x;
        cannot access o.y;
        cannot access o.z;
        can invoke o.m1();
        cannot invoke o.m2();
        cannot invoke o.m3();
    }
}
```

```
package p1;
package p2;
public class C3 {
    void aMethod() {
        C1 o = new C1();
        cannot access o.x;
        cannot access o.y;
        cannot access o.z;
        cannot invoke o.m1();
        cannot invoke o.m2();
        cannot invoke o.m3();
    }
}
```

Data Fields Should Be Always Private

- To protect data and to make the class easy to maintain
- Provide accessor and mutator methods for accessing and modifying the fields - conventions: `getX()`, `setX()`

```
class Thermometer {
    private int temperature;
    public void setTemperature(int temp) {
        temperature = temp;
    }
    public void getTemperature() {
        return temperature;
    }
}
```

Encapsulation Comic Book

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

```
public void setTemperature(int t) {
    if (t < 5) {
        //disable scale
    } else {
        this.temperature = t;
    }
}
```

```
I don't need any field encapsulation. If you want to change the temperature, use:
```
```
temperature = 10;
```
```
Billy
```
```
I will use field encapsulation. If you want to change the temperature, use:
```
```
t.setTemperature(10);
```
```
Jamie
```
```
Example of Data Field Encapsulation

- The - sign indicates private modifier

```
circle.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

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

Array of Objects, cont.
- Summarize the areas of the circles
- TotalArea.java

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

Array of Objects
- 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.

Passing Objects to Methods, cont.

Array of Objects
- Circle[] circleArray = new Circle[10];

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