

Chapter 2

Objects at Work

Essential reading

¹*Ida, review this and add more chapters to read*

¹
Bishop, Judy *Java Gently*. (Addison-Wesley Publishing Company, 2001) third edition [ISBN 0-201-71050-1].
Chapter 8

Winder, Russel and Roberts, Graham *Developing Java Software*. (published by John Wiley & Sons Ltd., 1998) second edition [ISBN 0-471-60696-0].
Chapter 6

In this chapter we concentrate on the concept of *object programming*. Using the basic statements learnt in the previous chapter, we show how to write a program in an object-oriented approach.

Object oriented programming

Object oriented programming is a way of programming and software design. It is an approach to achieve a computational solution by identifying components. These components are defined as objects which in fact usually contain the combination of data and operations on the data. In this object-oriented approach, we try to model a program by something closer to the real world objects which have certain behaviour towards other objects.

There are three most important characteristics of object-oriented programming:

1. Encapsulation: data and operations on the data are wrapped up within an object.
2. Inheritance: classes can inherit properties from other classes.
3. Polymorphism: the behaviours (operations on data) of objects do not have to be fixed until execution time.

Object oriented languages

Java is a so-called object oriented language. This means that Java provides a lot of convenient ways to allow or even enforce the three characteristics above. For example, a Java program consists simply a collection of classes, A method can be included in a class or an object, A single class can have several methods with the same name but different parameter types, etc..

However, using an object-oriented language dose not guarentte a program written is object-oriented.

Objects: the main difference between an object-oriented program and a structural program

In a structural program, we often define some data structures and provide functions (procedures) to manipulate the data from somewhere else in the program. Even a tiny change in one part would lead to drastic consequences

for many other parts of the program.

Java, as an object-oriented programming language, provides standard tools and techniques for reducing dependencies between different parts of a program. An object contains both a structured set of data and a set of operations for inspecting and manipulating that data. The operations in an object would directly associate with its own data structure but would not spread throughout the whole program.

Classes

A class is used to define a type of objects. It defines both the data that the objects possesses (i.e. entities of the type of objects) and the methods for operating on those data (i.e. the behaviours of the objects). An object is an instance of a class. You may wonder, if it is objects that we are interested in constructing, why do we have to bother to first design a class, and then generate an object from the class by defining the object as an instance of the class?

The idea of classes actually comes from most big design activities, such as automobile design, architecture, construction or even fine art. A drafted plan, often on paper (sometimes called a blueprint), is designed first to specify the desired object completely for everyone involved including the new members or authorities for approval. Once designed, any number of identical objects such as cars, can be constructed in different places and different environments. Also, if there is a need to change models later, new models (or classes) can be derived from the old one without having to start from scratch all over again. In object-oriented programming, once we have specified a class, we can construct any number of objects with the same behaviours. It is also easier to design upgrade versions of these objects.

Methods

A method is the basic unit of functionality that belongs to a class and that can be applied to (invoked on) a specific object or the class itself. In this arrangement, we have grouped together operations with the data. So if there is a need to change an object's data, one has to request operations from the object rather than directly manipulating the object's data. One can change an object, which is an instance of a class carrying its own data, without changing the class or other parts of the program.

Object declaration

The syntax to declare an object can be one of the followings:

```
=====
MODIFIER CLASSNAME OBJECTNAME = new CLASSNAME ();
MODIFIER CLASSNAME OBJECTNAME =
    new CLASSNAME (PARAMETERS);
MODIFIER CLASSNAME OBJECTNAME;
=====
```

Visibility and access modifiers

Visibility is the term used for describing what can be accessed or be referred to by what. In Java the access to a part of the program is controlled by placing a key word called *modifier* to indicate the availability of an object, a method or even a variable.

Suppose that a class is in a particular package PACK. The options of access for a method or variable of the class are described by one of the following modifiers:

- `public` means accessible from anywhere, even from other packages
- `protected` means accessible from any class within this package and also from subclasses in other packages
- `default` (no description) means accessible from any class in this package
- `private` - accessible only from somewhere within this class.

This can be summarised in the following table:

Variable /method modifier	From same class	From any class in PACK	From any sub-classes	From any-where	Inherited by any class in PACK	Inherited by any subclasses
<code>public</code>	Y	Y	Y	Y	Y	Y
<code>protected</code>	Y	Y	Y	N	Y	Y
<code>default</code>	Y	Y	N	N	Y	N
<code>private</code>	Y	N	N	N	N	N

In fact, modifiers can apply to different classes as well as objects.

The following example shows the effect by the use of different access modifiers on classes, objects, methods and variables.

Example 1 In the left-hand column of the following chart, three packages Package 1, 2, 3 are shown exactly as they are written. Each package contains classes, variables and objects with different access modifiers. In the right-hand column, the packages are shown as how they are seen from other parts of the program.

Contents	As seen from other parts of the program
<pre> Package 1 variable field x private class A public class B </pre>	<pre> Package 1 class B </pre>
<pre> Package 2 variable field y public class C object objB /* inherited from class B in package 1 */ </pre>	<pre> Package 2 class C </pre>

<pre> Package 3 variable field z class C private class D public class E ----- </pre>	<pre> Package 3 class E ----- </pre>
----------------------------------------------------------------------------------------------	----------------------------------------

The reader is advised to write a few small classes, and conduct experiments with various kinds of access modifiers in order to understand different ways of controlling access from any part of the program.

Table 8.4, page 325 also gives a summary of the meanings of access modifiers. Example 8.1 on page 307 provides a good example for experimenting with the access including the inheritance.

Example 2 Exploring protection

The following simple program is purely for experiment with various levels of accessibility.²

²*Writing simple programs as an experiment is the most effective way to learn a computer language. The reader is encouraged to do so as much as possible.*

Suppose there are four classes A, B, C, E, each having one variable field a, b, c, e respectively, and E has an object objB as well, instantiated from B. A and B are in the same package. C and E are in the default package. The classes have inheritance relationships, which can stay within packages or cross package boundaries. Let E inherit from B which inherits from A (Note E is in a different package.).

```

/* In directory P, file A.java */
package P;

class A {
    int a;
    public A () {a=1;}
}

/* In directory P again, file B.java */
package P;

public class B extends A {
    int b;
    public B () {
        b = 1;
        a = 1;
    }
}

/* In any other directory, file E.java with classpath
including P */
import P.*;

class  extends B {
    private e;

    B objB = new B ();

    E () {
        objB.b = 2;
    }
}

class C {
    private c;
    C {c = 1;}
}

```

Class diagrams

There are many possible diagrammatic representations introduced in different books. The reader should be not surprised to see the different symbols in these books and one does not have to use them in order to write a good program.

For example, a notation was introduced on page 81 in *Java Gently* (second edition), where each of the following basic components is represented by simple blocks and lines:

- class
- method
- object
- variable
- interface

- package
- reference
- abstract class
- listener
- component
- private
- throws exception.

However, a different notation which is called *model diagrams* is used in the third edition of the book (see page 75 and Appendix A).

Visiting a package member

There are three ways to visit a package member. They are:

- inheritance and use of a member reference;
- use of `object.member`;
- for *static*, use of `class.member`.

Inheritance from a class

The statement used consists of a normal class statement and is followed by a keyword `extends` which is followed by the super class name.

Example 3 To declare a new class E which inherits an existed class B, we write:

```
class E extends B {
... // do more things
}
```

Properties of objects

Objects in Java have some good properties.

Special value

An object that has not been through a construction process is given a special value `null` by the Java virtual machine. This can be used as a checking condition.

Note The checking can be used with *`nullPointerException`*.

The `this` identifier

Key word `this` is used to specify the object that the method in which `this` occurs is currently working on. This can be useful in several cases:

- ³:
- in event handling³ we shall learn event handling later: this makes sure that the event handling mechanism knows which object to notify when an event actually happens.

- in current object: a program may need to invoke one of the methods within the current object.
- easy in reading the code: sometimes, the names of parameters are coincidentally the same as those of variables.

Example 4 An example for the use of this.

```
private int x, y;

public void test(int x, int y) {
    this.x = x;
    this.y = y;
}
```

Here this.x and this.y indicate the two private variables, and x and y mean the two parameter values.

Calling a member

All members of an object can be called by this.member.

Difference between a variable and an object

Objects differ from variables in many ways. One important difference is about the storage as listed in the table below:

	storage for	contents of storage
variable	value	value itself
object	values and methods	pointers to filled and methods

Note Because of the difference, comparison between two fields of two different objects cannot be fulfilled by using == of these fields. Because this would simply compare the addresses of two objects. Similarly, making a copy of an object cannot be done by a single assignment =. Because this would assign the address of an object to another object. The correct way to compare two fields of different objects is to define a boolean type of method equals for each new class.

Clone

Here the term is used to mean a genuine copy of all fields and the references of all the methods of an object. The technique used to copy an object to another is to define a method for each new class in the form below:

```
MODIFIER class CLASSNAME implements Clonable {
    public Object clone ( ) {
        CLASSNAME obj = new CLASSNAME (PARAMETERS);
        ... // statements to copy the fields to obj;
        return obj;
    }
}
```

Note Here `Object` is a Java superclass in which methods defined can be used by other objects, and `clone` is one of these methods.

In summary, the fundamental difference between assignments and cloning of objects is:

- assignments will copy the *references* (*addresses*) of an object
- cloning will copy the *values* including the addresses of methods of an object.

Lists of objects

In this section, we look at the implementation of a useful data structure *linked list*. Skipping the concepts and principles, we concentrate on how to define a linked structure in Java⁴.

⁴ Do some revisions if you have forgotten about linked lists.

As we know, a linked list structure consists of a sequence of nodes that connected one after another by 'links'. Each node contains at least two segments of normally different types, one is of some *data* and the other is the *link*, the address of the next node. The address of the first node of a linked list is stored in a special variable called *head* of pointer type.⁵

⁵ Note here *pointers*, *addresses* and *references* have the same meaning.

Constructing a list

Our approach is to first define a class for nodes and then each object derived from this class will be a node of the list. Of course, we shall store each datum into the data section and the address of the next datum in the *list* into the link section.

Let us look at an example below. Here is an outline of the main statements in construction of a linked list.

Example 5 1. Define a node class: this is to define a node type structure. The two parameters *d* and *n* are used to provide the *data* and the *link*.

```
class Node {
    Node (Object d, Node n) {
        data = d;
        link = n;
    }
    Object data;
    Node link;
}
```

2. Initialise the list: this is to construct the first node of the linked list. Suppose we construct the linked list from the *tail*, (i.e. by inserting a new node at the beginning of the list each time).

```
Node list = new Node (2, null);
```

As we can see, by providing two parameters 2 and `null`, we have constructed a linked list with 1 node, where `null` is a special value to indicate the end of the link, and `list`, the head of the list is the address of this only node.

3. Link the next node: the node values 3 and list are the data 3 and the address of the current head of the list.

```
list = new Node (3, list);
```

And the address of this new node becomes the head of the list, and so on. This process iterates until all the nodes are constructed.

At the end, we shall have a linked list in which the most recently added node is the first node, and address of the first node is in list.

⁶Note: this is not a well structured program but an easy test on the reference/link among several objects. See solution to Lab Exercise 4 for a better version later.

The following is a simple program ⁶ to construct a list of people's names (string).

Example 6 /* This is a test of the construction of a linked list.

```
*/
import java.io.*;

class Node {
    /* a Node consists of two fields: 1 data and 2 link
    */
    Node (String d, Node n) {
        data = d;
        link = n;
    }
    Object data;
    Node link;

    // Error! if "static" is missing as following.
    // void Assign () {
    static void Assign () {
        Node list = new Node ("Ida", null);
        list = new Node ("Fran", list);
        list = new Node ("Bob", list);

        for (Node l=list; l!=null; l=l.link) {
            System.out.println(l.data);
        }
    }

    // Error! if "static" is missing as following.
    // public void main (String [] args) {
    public static void main (String [] args) {
        Assign();
    }
}
}
```

Note String Bob is the last item to be inserted in the list and it is the first node of the constructed list and hence the first to be printed out.

Standard operations on ADT linked lists

Examples are *displaying the whole list, check if the list is empty, searching, removing a node, updating the whole list, sorting the list* etc.

We shall look into the implementation of some simple operations later.

Summary of Object-Oriented Programming(OOP)

- Programs consist of classes.
- Classes create objects.
- Classes consist of attributes and behaviours.
- Connecting classes by inheritance.
- Linking classes through packages and interfaces.

Solving problems

The general approach to solve a programming problem is:

1. Analyse the problems and decide what you wish to accomplish.
2. Design and construct a set of classes or identify classes available.
3. Create objects from the right set of classes.

Attributes: individual things that distinguish one class from another

Example 7 *Attributes*

- *Colour*
- *Sex*
- *Appetite.*

Objects, using methods, can:

- report a change to another object
- tell another object about some changes
- ask another object to do something.

Methods are groups of related statements in a class of objects.

Definitions:

inheritance A mechanism that enables one class to inherit all of the behaviours and attributes of another class.

subclass A class that inherits from another class.

superclass A class that gives the inheritance.

Learning outcomes

Having read this chapter and consulted related material you should:

- understand the advantages of *objects, classes and methods* in Java;
- have learnt further issues about *object-oriented programming*;
- have seen how to design a larger programs in Java.