Java: A System Programming Language

by David G. Messerschmitt

Supplementary section for Understanding Networked Applications: A First Course, Morgan Kaufmann, 1999.

A system programming language is used to implement objects and components in application development. Some features of these languages is described in "Programs and Languages" in Chapter 11, and the distinction between implementation and assembly is described in "Visual Architecture Modeling" in Chapter 10.

Java is a representative object-oriented systems programming language [Arn96][Fla96]. Some of the key language elements of Java that support object interfaces are listed in Table 1 (which summarizes only small fraction of the features of Java).

Table 1 Key elements of a system programming language, with examples from Java.

Element	Description	Example
Keywords	Words that have specific meaning to and are reserved by the language.	The word **`interface`** is used to specify the interface to an object class. In the remainder of this Appendix, keywords are in a bold font to make them easy to distinguish from programmer-defined names.

Specify object class interface	Assign the class a name, and specify the methods, parameters, and return values.	In the following specification,
		`interface` `Account {`
		`//Description of class interface`
		`}`
		`Account` is a name assigned to the class by the programmer. The "//" at the beginning of a line makes it a *comment* that is ignored.

Specify structure of data	Declaration of the data types passed as method parameters or return values in an object interface. A data type is a specification the range of values and allowable operations on data (see below).	All data must have a name and a declaration of type, like **`int`** `amount;` The statement **`// boolean`** `Withdrawal (`**`int`**` amount);` specifies a method at the interface with a single parameter named `amount`, which describes data representing an integer, and the single unnamed return value is **`true`** if there was a sufficient balance to make the withdrawal.

Table 1 Key elements of a system programming language, with examples from Java.

Element	Description	Example

Instantiate a class	Create a new object of a specified class.	An object with class Account would be created like this:
		`Account my_account;`
		`my_account = new Account;`
		Afterward, `my_account` is a reference to an object with class `Account`.

Invoke method of an object	Interact with an object by invoking one of its methods, passing parameters and with a return value (both are data with a specified structure).	With a reference to an Account, its `Balance()` method can be invoked as follows:
		`boolean OK;`
		`// Let's withdraw $34.00`
		`// in my_account`
		`OK = my_account.Deposit(3400);`
		The `my_account.Deposit` is Java's way of specifying that the `Deposit()` method of `my_account` is to be invoked. Now, `OK` will be **`true`** if my balance was $34.00 or more; otherwise, it will be **`false`** and the balance will not be changed.

In Table 1, however, many key concepts discussed in Chapter 11 are displayed. One of them is the data type. In Java, the type of all data must be specified before that data can be manipulated.

Example... Data type **`int`** specifies the representation of an integer represented by 32 bits. The range of values—determined by the 32 bit representation—is between -2147483648 and +2147483647. Typical allowed operations on integers are to add two integers or multiply them.

Data type **`char`** specifies a *Unicode* character, which is represented by 16 bits and has a sufficient range of values to represent all the world's major languages.

Data type **`boolean`** has just two values and is represented by one bit, but is represented symbolically in the language by **`true`** and **`false`**.

In the notation of Chapter 6, an action might be written

`Withdrawal: amount` $\rightarrow$ `status`

for the withdrawal form an account, where `amount` is the amount of funds to be withdrawn and status is an indication of the result (were there sufficient funds?). In a Java class interface this would be written as a method

```
boolean Withdrawal (int amount)
```

The major difference is that Java requires not only a descriptive name for data parameters and return values, but also a specification of the data types. Also, the return value is not named—it is just data of a specified type. There is only a single return value.

Bank account example

Here is an example of a simple Java program specifying the implementation of an object class `Account` that manages a financial account balance: First, Java provides a way to specify the interface of the class without telling anything about what might be encapsulated or how the class

is implemented:

```
interface Account{
      // Return the balance in pennies
      int Balance();

      // The return value is true if the deposit is
      // accepted, false otherwise
      boolean Deposit (int amount);

      // The return value is true if there was sufficient
      // balance, false otherwise
      boolean Withdrawal (int amount);
      }
```

Having specified the interface, its implementation can be specified as follows:

```
public class Bank_account implements Account {

      // Must store account balance
      private int balance;

      public int Balance() {return balance;}

      public void Deposit (int amount) {
            balance = balance + amount;
            }

      public boolean Withdrawal (int amount) {
            if (amount <= balance) {
                  balance = balance - amount;
                  return true;
                  }
            else return false;
            }
      }
```

The keywords **public** and **private** control the encapsulation of internal details. The internal data `balance` is private, meaning that it isn't visible outside the class, but can be accessed by the methods of the class `Bank_account`.

This implementation illustrates in a small way the interpretation of data, turning it into information through data processing. For example, the statement

```
balance = balance + amount;
```

expresses the interpretation of `balance` as the money in a account, and `amount` as the money being deposited in the account. The resulting change in `balance` is information, in the sense that it influences the behavior (spending) of the owner of the `Bank_account`.

The way another object can make use of a `Bank_account` can be illustrated. Here is an interface to a `Person`, which represents information about a citizen. This interface allows the Person, among many other things, to accept and hold funds. At the interface, what the `Person` does with those funds is appropriately abstracted—some may choose to hold it as cash (stuffed under a mat-

tress), others may open a bank account and deposit it, and others may spend it immediately.

```
interface Person {

      // Many methods omitted for brevity
      // ...

      // Accept funds from another
      // Returns true if the money is accepted, false otherwise
      boolean accept_funds (int amount);
            }
```

A particular type of `Person`, a `Bank_customer`, may deposit these funds in a `Bank_account`, although this fact and other details are encapsulated:

```
class Bank_customer implements Person {
      // This Person likes to store money in the
      // bank rather than under a mattress

      // This is the Person's Bank_account, which is
      // encapsulated--hidden from other classes
      private Bank_account my_account;

      // Many methods omitted for brevity
      // ...

      public boolean accept_funds (int amount) {
            // Only accept funds if amount is positive!!
            if (amount > 0) {
                  my_account.Deposit(amount);
                  return true;
            else return false;
            }
      }
```

An important point to note about this example is the interpretation of `amount` that is reflected in the processing performed within the `accept_funds()` method. It is treated as funds to be deposited in a `Bank_account`, and that interpretation is expressed by the program code encapsulated within the class.