NAMING CONVENTIONS AND DATA TYPES

CHAPTER 4

In the previous chapter, we understood Java by writing a couple of programs. In those programs, you might have observed that sometimes we have used small letters and sometimes capital letters. For example, we have written the class `System` starting with a capital letter. You cannot write this class name as `system` or `SYSTEM`. Since Java is a case sensitive programming language, it recognizes capital and small letters as different. So the programmer should take care of upper and lower case while writing a program in Java. But how to know where to use which case—upper or lower? For this purpose, certain conventions (rules) are followed by JavaSoft people while naming the variables, classes, methods, etc. These naming conventions should be followed by every programmer in his programs for maintaining uniformity and also for clarity of distinction between different elements of a program. These rules also reduce the possibility of any spelling mistakes while writing the names of variables, classes, etc. in the programs. For example, a Java program will not compile, if `main()` method is written as:

```
        Public static void Main(String args[])
```

The problem in the preceding statement is using capital `P` for public and capital `M` for `main()`. Such errors can be eliminated if we follow the naming conventions.

Naming Conventions in Java

Naming conventions specify the rules to be followed by a Java programmer while writing the names of packages, classes, methods, etc. Now, let us see some of the major naming conventions to be followed in Java.

- A *package* represents a sub directory that contains a group of classes and interfaces. Names of packages in Java are written in small letters as:

```
        java.awt
        java.io
        javax.swing
```

- A *class* is a model for creating objects. A class specifies the properties and actions of objects. An interface is also similar to a class. Each word of class names and interface names start with a capital letter as:

```
    String
    DataInputStream
    ActionListener
```

- A class and an interface contain methods and variables. The first word of a method name is in small letters; then from second word onwards, each new word starts with a capital letter as shown here:

```
    println()
    readLine()
    getNumberInstance()
```

- The naming convention for variables names is same as that for methods as given here:

```
    age
    empName
    employee_Net_Sal
```

Note

Now, the question that is commonly asked is that if same rule is applied for both variables and methods, how can we distinguish between them? The answer is that since a method's name ends with a pair of simple braces (), it can be distinguished easily from a variable whose name will not have any braces.

- *Constants* represent fixed values that cannot be altered. For example, PI is a constant whose value is `22/7` or `3.14159`, which is fixed. Such constants should be written by using all capital letters as shown here:

```
    PI
    MAX_VALUE
    Font.BOLD
```

Here, `BOLD` is a constant in `Font` class. This is the way most of the inbuilt constants in Java are referenced.

- All keywords should be written by using all small letters as follows:

```
    public
    void
    static
```

Data Types in Java

We know we need variables to store the data. Internally, a variable represents a memory locatio which holds data. When we want to use a variable in a program, we should first declare it as:

```
    int x;
```

Here, we are declaring that `x` is a variable, which can store `int` (integer) type data. This means `in` is representing the nature of data to be stored into `x`. `int` is also called a `data type`. For exampl `x` can store an integer number like `125` as:

```
    x = 125;
```

Here, x is a variable and = represents that the value 125 is stored into x. This value 125 stored into x is also called literal. There are various data types and literals defined in Java, which we will be discussing in this chapter.

Integer Data Types

These data types represent integer numbers, i.e. numbers without any fractional parts or decimal points. For example, 125, -225678, 0, 1022, etc. come under this category. Integer data types are again sub divided into `byte`, `short`, `int`, and `long` types. Table 4.1 lists the data types.

Table 4.1

Data type	Memory size	Minimum and Maximum values
Byte	1 byte	-128 to +127
Short	2 bytes	-32768 to +32767
Int	4 bytes	-2147483648 to +2147483647
Long	8 bytes	-9223372036854775808 to +9223372036854775807

Let us try to understand this through an example given here:

```
    byte rno=10;
```

In the preceding statement, we are declaring `byte` data type for the variable `rno` and the value `10`, which is stored into `rno`. `byte` represents any value between `-128` to `+127`.

`long x=150L;`

Here, `150` is stored into `x`, which is declared as `long` type. Notice the `L` at the end of the statement. If this `L` is not there, then JVM allots only 2 bytes of memory to `x` as against the usual 8 bytes memory that should be allotted to a `long` type. The reason for this is that 2 bytes are sufficient to store the value 150. But if we attach `l` or `L` at the end of the value as shown in the preceding example, then JVM will consider it as a `long` value and will allot 8 bytes to it.

Float Data Types

These data types are useful to represent numbers with decimal point. For example, `3.14, 0.0012, -123.11`, etc. are called floating point numbers. These are again classified as `float` (single precision floating point number) and `double` (double precision floating point number). The difference exists essentially in the number of digits, they can represent accurately after the decimal point. This accuracy is also called *precision*. Table 4.2 depicts the size of `float` and `double`.

Table 4.2

Data type	Memory size	Minimum and Maximum values
Float	4 bytes	-3.4e38 to -1.4e-45 for negative values and 1.4e-45 to 3.4e38 for positive values
Double	8 bytes	-1.8e308 to -4.9e-324 for negative values and 4.9e-324 to 1.8e308 for positive values

Important Interview Question

What is the difference between float and double?

`Float` can represent up to 7 digits accurately after decimal point, whereas `double` can represent up to 15 digits accurately after decimal point.

Let us now look at an example given here:

```
        float pi=3.142F;
```

Here, the variable `pi` is containing the value `3.142`. If `F` is not written at the end, then JVM would have allotted 8 bytes assuming the value to be `double`. The reason for this is that in `float` and `double` data types, the default is taken as `double`. By attaching `F` or `f`, we can ask the JVM to consider it as a `float` value and allot only 4 bytes.

`double distance=1.98e8;`

Here, `e` or `E` represents `X 10 to the power`. Hence, `1.98e8` means `1.98X108`. This is also called *scientific notation* of representing numbers.

Character Data Type

This data type represents a single character like a, `P, &, *`, etc. Table 4.3 shows `char` data type details.

Table 4.3

Data type	**Memory size**	**Minimum and Maximum values**
Char	2 bytes	0 to 65535

Here is an example of `character` data type:

```
        char ch='X';
```

Here, we are storing the single character 'X' into the variable `ch`. Here, 'X' is also called *characte literal.* Whenever character literals are written, they should be enclosed inside the single quotes.

By observing the minimum and maximum values in the table earlier, we shall get a doubt regarding why the range is expressed in integer numbers (`0` to `65535`). We know that all the characters on the keyboard are translated into integer values called ASCII (American Standard Code for Information Interchange), which is a standard maintained by every keyboard manufacturer. The processo recognizes the character uniquely from its ASCII value. Hence, let us understand the range mentioned in the table is nothing but the ASCII value range only.

The ASCII value range, from `0` to `65535`, given in the table can uniquely represent a total of `6553` characters. This means a total of 65536 distinct characters can be recognized by Java. But we neve use these many characters since our keyboard contains English alphabets and some othe characters whose total does not exceed `256`. This means 1 byte is sufficient to represent all th available characters of the keyboard. Then why 2 bytes are used to represent the `char` data type i Java?

JavaSoft people wanted to provide a facility to include characters not only from English but als from all other human languages to be used in Java programs. This will enable the programmers t write and execute a Java program in any language, which becomes an advantage on Internet. Thi system is also called *Unicode system.* Unicode uses 2 bytes so that any character from any languag can be encoded successfully.

Important Interview Question

What is a Unicode system?

Unicode system is an encoding standard that provides a unique number for every character, no matter what the platform, program, or language is. Unicode uses 2 bytes to represent a single character.

String Data Types

A `String` represents a group of characters, like New Delhi, AP123, etc. The simplest way to create a `String` is by storing a group of characters into a `String` type variable as:

```
    String str = "New Delhi";
```

Now, the `String` type variable `str` contains "`New Delhi`". Note that any string written directly in a program should be enclosed by using double quotes.

There is a class with the name `String` in Java, where several methods are provided to perform different operations on strings. Any string is considered as an object of `String` class. But in C/C++, a string is considered as a character array containing some characters where the last character would be `\0`. This is not valid in Java, since in Java, we got strings and character arrays both separately.

Now the question arises that if `String` is a class, why are we taking it as a data type? The answer is that every class is a data type and is also called *user-defined data type.*

Boolean Data Types

`Boolean` data types represent any of the two values—true or false. JVM uses 1 bit to represent a boolean value internally, for example:

```
    boolean response=true;
```

As shown earlier, we should not enclose the boolean value `true` (or `false`) in any quotation marks. In C/C++, 0 represents false and any other number represents true. This is not valid in Java.

Literals

A *literal* represents a value that is stored into a variable directly in the program. See the following examples:

```
    boolean result = false;
    char gender = 'M';
    short s = 10000;
    int i = -1256;
```

In the preceding statements, the right hand side values are called literals because these values are being stored into the variables shown at the left hand side. As the data type of the variable changes, the type of the literal also changes. So we have different types of literals. These are as follows:

- Integer literals
- Float literals
- Character literals
- String literals
- Boolean literals

Read on to get familiar with them.

Integer Literals

Integer literals represent the fixed integer values like 100, -55, 123450, etc. All these numbers belong to decimal system, which uses 10 digits (from 0 to 9) to represent any number. Suppose, we want to write an integer in octal number system (octal number system uses 8 digits, from 0 to 7), then we should prefix *0* before the number. To represent hexadecimal number (hexadecimal number system uses 16 digits from 0 to 9 and from A to F), we should prefix *0x* before the value.

```
int decVal = 26;   // The number 26, in decimal
int octVal = 032;  // The number 26, in octal
int hexVal = 0x1a; // The number 26, in hexadecimal
```

Float Literals

Float literals represent fractional numbers. These are the numbers with decimal points like 2.0, -0.005, 3.14, 6.1e-22, etc. which should be used with float or double type variables. While writing these literals, we can use E or e for scientific notation, F or f for float literal, and D or d for double literal (this is the default and generally omitted).

```
double d1 = 123.4;
double d2 = 1.234e2; // same value as d1, but in scientific notation
float f1  = 123.4f;
```

Character Literals

Character literals indicate the following:

- General characters, like A, b, 9, etc.
- Special characters, like ?, @, etc.
- Unicode characters, like \u0042 (this represents *a* in ISO Latin 1 character set).
- Escape sequence (backslash codes) like \n, \b, etc.

Character literals should be enclosed in single quotation marks. The preceding unicode characters and escape sequence can also be represented as strings.

String Literals

String literals represent objects of String class. For example, Hello, Anil Kumar, AP1201, etc. wil come under string literals, which can be directly stored into a String object.

Boolean Literals

Boolean literals represent only two values—true and false. It means we can store either true or fals into a boolean type variable.

Conclusion

By following naming conventions, a programmer can avoid a lot of mistakes in writing the names c packages, classes, methods, variables, etc. This will improve readability as well as understandabilit of a program. So maintaining naming conventions is a good programming habit.

A program generally acts on data, processes it, and provides the results. So data is very importan element of a program. In this chapter, we also discussed different *data types,* in terms of type c data received, stored, or used in a program.