Pattern Name:
Name Use Pattern - Part 1 - Variables and Objects

Author:
Viera K. Proulx

Intent and Motivation:
Computer program works with different types of information - numbers, words, codes representing colors, files used by the program and many others. Each of these items needs to be uniquely identified within a program. The word *identifier* is used to describe any name used in most programming languages.

There are specific *grammar rules* how an identifier can be written - for example, it has to start with a letter of alphabet, can contain additional letters or digits or the character _ (underscore) though it is not a good idea to use underscore - as it is often used for compiler and system variables. The program may differentiate between capital letters and lower case letters (we say it is *case-sensitive*). In that case, we can use identifiers like *firstName, maxValue, etc.* Additionally, people working on a joint project may adapt *conventions* for selecting identifiers. For example, certain kind of names start must with a capital letter (names of classes), others must start with lower case (names of variables and object instances).

Every named item in Java needs to be *declared* and *defined* before it is *used*.

Problem Examples:
1. The program is painting rectangles. It will need the coordinates of two opposite corners - a total of four integer values. We need to declare and define four integer variables that record this information (typically we would name them *x1, y1, x2, y2*). Alternately, there may exist a class *Rectangle* that records these four values as its member data. In that case, we can define one instance of a *Rectangle* object (and we may name it *r*).

2. The program needs to read names, save them and print them later. To remember a name, we need to declare-define-use an object that represents a list of characters (letters, spaces, digits, and other symbols that can be typed in). In Java objects in the *String* class represent such strings of characters. A good choice of a name for this object would be *name*, or *firstName*, or *myName*.

3. The program is printing the weather record for the daily newspaper. It needs to remember the name of each city and the lowest and highest temperature.
For this we will need one string object (**city**) and two integer variables (**low**, **high**).

**Problem and Context:**

In a Java program a variable of a basic type represents a memory location that is used to store one item of data of the given type. So, for example, a variable of the type **int** in Java represents one integer value (within the specified range). Similarly, if we declare and name an instance of an object in a given class, the name will represent a reference to one object instance that refers collectively all the components of this object - its member data and member functions. We see, that before a name can be used, we need to specify what kind of information the name will represent: we need to **declare** the name and specify its usage.

We first need to declare that a name that represents memory storage area of an entity of a given type. At that time the compiler will know that this name can be used everywhere where an integer can be used - in computations, as output, etc. We now need to **define** the data that will initially be stored in the reserved memory storage area. Later we will see, that when a name represents a behavior or process, we will need to define the behaviors applicable to this name.

Finally, the name can be **used** - the value stored in the memory will be used in computations, will be printed, will guide the location or the color of graphic images, and can be modified the represent newly computed information. How the name can be used is determined by how it was defined - the definition specifies all possible uses of the name.

In some cases, we also need to worry about **invalidating** the name once it is no longer needed, so that the memory storage can be recycled, or to indicate that the behavior represented by this name is no longer applicable.

This pattern applies to all names used in a computer program. Some of the names we will consider in the future are: variables and objects, functions, files, streams, classes, constants, names of files (to include in a program).

**Required Elements and Structure:**

**Declaring an identifier**

To declare an identifier means to specify that a particular name will represent a quantity (information) of a given **type** or an object in a given **class**. For example, we may define a variable **x** to represent an integer quantity (type **int**) and variable **myName** to represent a sequence of letters (object in the class **String**), by including the following lines at the beginning of our program:
At this point, the compiler remembers the meaning of these names, but the names do not represent any values. Typically, the computer knows that \texttt{x} will represent an integer and \texttt{myName} will represent a \texttt{String}, but these names cannot be used, because they do not yet represent any value or any object. You can think of it as a piece of blackboard that has been reserved for you, but you have not written anything on it.

**Defining a value of an identifier**

There are several ways how an identifier can be assigned a value. We will learn only a few of them now - more will be shown as we learn more about the Java programming language.

We can assign a value to an integer variable in the following three ways:

1. Initialize the value of variable \texttt{x} when it is declared.

   Statements
   
   ```java
   int x = 5;
   String myName = "Peter Pan";
   ```

   combine the declaration of the name and the initialization of its value.

2. Change the value of an integer identifier anywhere in the program using the assignment statement.

   Statement
   
   ```java
   x = 4 + 9;
   ```

   will cause the computer to first compute the value of the expression on the right hand side of the equal sign (i.e. adding 4 and 9, resulting in 13), then assign the resulting value (13) to the identifier specified on the left (\texttt{x}).

   This can be used either to assign an initial value to an identifier, or to change its value later in the program.

   Notes:
   
   - the computer cannot understand the statement:
     ```java
     4 + 9 = x; //WRONG!!*****
     ```

     The left hand side \texttt{4 + 9} is not a name of an identifier and even if the computer could compute the value of the expression on the right hand side (in this case the value of the variable \texttt{x}), it cannot perform the assignment of this value to an identifier.
• The `String` class allows us to initialize or change the value of the `String` object `myName` in a similar way:

```java
myName = "Peter Pan";
```

3. Read user input or input from a file.

The statements and functions used to read data from the keyboard or from files are quite complex and are different in every language. The basic functions are not designed to deal with errors in the user input, and so most programmers use some tools and error checking filters when dealing with the input from keyboard. The issues related to reading input are quite complex and not related to the Name Use Pattern. For explanation of the examples shown below, the reader should consult the Reading Data Pattern.

The basic JPT statement that reads the input from the JPT `console` (the sequence of keystrokes typed in followed by the return key) is the following:

```java
x = console.in.demandInt("Type in a number");
```

The prompt for the user is displayed in the `console` and then the user input is converted to an integer. If user makes a mistake, the error is detected, and a demand for an integer input is repeated.

We prefer not to use the basic input functions provided by Java, because they require that the user converts the input represented as a `String` to its representation as an integer - and the built-in conversion functions do not provide any error checking of the user's input.

**Using an identifier**

Once an identifier has been properly declared and initialized, it can be used in computations, used in output and other operations appropriate for its type or class. Variables of the int and double type can take part in arithmetic expressions, can be printed, and we can assign new values to them. Besides reading and writing string objects, we can manipulate them in a number of ways. These will be discusses when we learn more about the string class.

If the identifier has not been initialized before it is used, the compiler will notice the error and notify the programmer.
Implementation Examples:

The following program segments show several uses of this pattern. All four program segments will result in identical output, namely the following two lines:

My name is Peter Pan
25 / 5 = 5

// example 1
String myName = "Peter Pan";
int x = 25;
console.out.println("My name is " + myName);
console.out.println("25 / 5 = " + x/25);

// example 2
String myName = "Peter Pan";
int x = 25;
int y;
y = x / 5;
console.out.println("My name is " + myName);
console.out.println("25 / 5 = " + y);

// example 3
String myName = "Peter Pan";
String title = "My name is "
int x = 25;
int y;
y = x / 5;
console.out.println( title + myName );
console.out.println( " + x + " / 5 = " + y );

// example 4
String myName;
myName = "Peter Pan";
int x = 25;
int y;
y = 5;
console.out.println( "My name is " + myName );
console.out.println( " + x + " / " << y + " = " + x/y );
Summary:

An identifier is the name of a Java item that the program needs to recognize. Each identifier needs to be declared, so that the computer knows how is it going to be used and assigns a space in the memory that will be used to store the value of the identifier (when appropriate). The identifier then needs to be initialized - that means we need to assign the actual value to the identifier. After that, when the identifier is used in a program it represents the value assigned to it most recently.

The first kinds of identifiers we use specify the type of a variable (for example int for integer variables, double for real numbers), or an object in a given class (such as String object).