MEMBER VARIABLES IN JAVA

A class can include other things besides subroutines. In particular, it can also include variable declarations. Of course, you can declare variables **inside** subroutines. Those are called local variables. However, you can also have variables that are not part of any subroutine. To distinguish such variables from local variables, we call them member variables, since they are members of a class.

Just as with subroutines, member variables can be either static or non-static. In this chapter, we'll stick to static variables. A static member variable belongs to the class itself, and it exists as long as the class exists. Memory is allocated for the variable when the class is first loaded by the Java interpreter. Any assignment statement that assigns a value to the variable changes the content of that memory, no matter where that assignment statement is located in the program. Any time the variable is used in an expression, the value is fetched from that same memory, no matter where the expression is located in the program. This means that the value of a static member variable can be set in one subroutine and used in another subroutine. Static member variables are "shared" by all the static subroutines in the class. A local variable in a subroutine, on the other hand, exists only while that subroutine is being executed, and is completely inaccessible from outside that one subroutine.

The declaration of a member variable looks just like the declaration of a local variable except for two things: The member variable is declared outside any subroutine (although it still has to be inside a class), and the declaration can be marked with modifiers such as `static`, `public`, and `private`. Since we are only working with static member variables for now, every declaration of a member variable in this chapter will include the modifier `static`. They might also be marked as `public` or `private`. For example:

```
static String usersName;
public static int numberOfPlayers;
private static double velocity, time;
```

A static member variable that is not declared to be `private` can be accessed from outside the class where it is defined, as well as inside. When it is used in some other class, it must be referred to with a compound identifier of the form **class-name**.**variable-name**. For example, the *System* class contains the public static member variable named `out`, and you use this variable in your own classes by referring to`System.out`. Similarly, `Math.PI` is a public member variable in the *Math* whose value is the mathematical constant π. If `numberOfPlayers` is a public static member variable in a class named `Poker`, then subroutines in the `Poker` class would refer to it simply as `numberOfPlayers`, while subroutines in another class would refer to it as `Poker.numberOfPlayers`.

As an example, let's add a static member variable to the `GuessingGame` class that we wrote earlier in this section. This variable will be used to keep track of how many games the user wins. We'll call the variable`gamesWon` and declare it with the statement "`static int gamesWon;`". In the `playGame()` routine, we add 1 to `gamesWon` if the user wins the game. At the end of the `main()` routine, we print out the value of`gamesWon`. It would be impossible to do the same thing with a local variable, since we need access to the same variable from both subroutines.

When you declare a local variable in a subroutine, you have to assign a value to that variable before you can do anything with it. Member variables, on the other hand are automatically initialized with a default value. For numeric variables, the default value is zero. For boolean variables, the default is `false`. And for char variables, it's the unprintable character that has Unicode code number zero. (For objects, such

as `Strings`, the default initial value is a special value called `null`, which we won't encounter officially until later.)

Since it is of type int, the static member variable `gamesWon` automatically gets assigned an initial value of zero. This happens to be the correct initial value for a variable that is being used as a counter. You can, of course, assign a different value to the variable at the beginning of the `main()` routine if you are not satisfied with the default initial value.

Here's a revised version of `GuessingGame.java` that includes the `gamesWon` variable. The changes from the above version are shown in red:

```
public class GuessingGame2 {

    static int gamesWon;      // The number of games won by
                              //    the user.

    public static void main(String[] args) {
       gamesWon = 0;  // This is actually redundant, since 0 is
                      //                         the default initial
value.
       TextIO.putln("Let's  play  a  game.    I'll  pick  a  number
between");
       TextIO.putln("1 and 100, and you try to guess it.");
       boolean playAgain;
       do {
          playGame();  // call subroutine to play one game
          TextIO.put("Would you like to play again? ");
          playAgain = TextIO.getlnBoolean();
       } while (playAgain);
       TextIO.putln();
       TextIO.putln("You won " + gamesWon + " games.");
       TextIO.putln("Thanks for playing.  Goodbye.");
    } // end of main()
```

```
    static void playGame() {
        int computersNumber;  // A random number picked by the
computer.
        int usersGuess;         // A number entered by user as a
guess.
        int guessCount;         // Number of guesses the user has
made.
        computersNumber = (int)(100 * Math.random()) + 1;
                //  The value assigned to computersNumber is a
randomly
                //    chosen integer between 1 and 100, inclusive.
        guessCount = 0;
        TextIO.putln();
        TextIO.put("What is your first guess? ");
        while (true) {
           usersGuess = TextIO.getInt();  // Get the user's guess.
           guessCount++;
           if (usersGuess == computersNumber) {
              TextIO.putln("You got it in " + guessCount
                      +   "   guesses!      My   number   was   "   +
computersNumber);
              gamesWon++;    //  Count  this  game  by  incrementing
gamesWon.
              break;       // The game is over; the user has won.
           }
           if (guessCount == 6) {
              TextIO.putln("You   didn't   get   the   number   in   6
guesses.");
              TextIO.putln("You   lose.     My   number   was   "   +
computersNumber);
              break;  // The game is over; the user has lost.
           }
           // If we get to this point, the game continues.
           // Tell the user if the guess was too high or too low.
           if (usersGuess < computersNumber)
              TextIO.put("That's too low.  Try again: ");
           else if (usersGuess > computersNumber)
              TextIO.put("That's too high.  Try again: ");
```

```
        }
        TextIO.putln();
    } // end of playGame()

} // end of class GuessingGame2
```

Source : http://math.hws.edu/javanotes/c4/s2.html