We have [seen](https://www.eimacs.com/eimacs/mainpage?epid=E205325474&cid=162149#ClassStatic) that the presence of the static keyword distinguishes a class method from an instance method. The static keyword is also used to identify *class variables*. A class variable is available both to instance and to class methods and, as we demonstrate shortly, it may even be accessed outside the class in which it is declared.

Our first example illustrates the situation in which a class variable is referenced in a class method of the same class. In this case, the class variable today is initialized in the definition of the MainClass class and is referenced in the main method of the same class.

public class MainClass   
{   
  private static String today = "Monday";   
  
  public static void main( String[] args )   
  {   
    System.out.println( today );   
  }   
}

Monday

Our second example has a class variable, testN, that is referenced in an instance method of the same class (namely, the Test class). In this case, the instance method testMe increments the class variable each time the method is called.

public class Test   
{   
  private static int testN = 0;   
  
  public Test()   
  {   
  }   
  
  public void testMe()   
  {    
    System.out.println( testN );   
    testN++;   
  }   
}   
  
public class MainClass   
{   
  public static void main( String[] args )   
  {   
    Test t1 = new Test();   
    t1.testMe();   
    t1.testMe();   
  
    Test t2 = new Test();   
    t2.testMe();   
    t1.testMe();   
    t2.testMe();   
  }   
}

0   
1   
2   
3   
4

If you click the **Run** button, you will perhaps be surprised to discover that both instances, t1 and t2, are accessing the *same* class variable, testN. This is a distinguishing characteristic of class variables. There is just one copy of each class variable, and that one copy is shared by all the instances of the class. As you will recall, the situation is very different for instance variables. Each instance of a class has its own copy of each of the class's instance variables.

**Exercise 141**

1. Study the following code and predict what you think the output will be when it is run. Then click the **Run** button and see how good your prediction was.

public class Test   
{   
  private static int n = 0;   
  
  public Test()   
  {   
    n++;   
  }   
  
  public void getN()   
  {    
    System.out.println( n );   
  }   
}

public class MainClass

{

  public static void main( String[] args )

  {

    Test t1 = new Test();

    t1.getN();

    t1.getN();

    Test t2 = new Test();

    t2.getN();

    t1.getN();

  }

}

1   
1   
2   
2

The value of such a class variable is always equal to the number of times that the class constructor has been called. It therefore tells you how many instances have so far been created of the class in question.

On the previous page we learned how to declare and initialize class variables, and we saw examples in which the value of a class variable was changed as code execution proceeded. Sometimes, however, it is desirable to prevent modification of a class variable. That is, there are sometimes occasions when we want to ensure that, once a class variable has been initialized, its value can never be changed. The way to do this is to include — along with the modifier keyword static (which makes it a class variable) — the modifier keyword final. An unchangeable class variable is referred to as a class constant. Here is an example:

public class Test

{

  private static final int n = 5;

  public Test()

  {

  }

  public void getN()

  {

    System.out.println( n );

  }

}

public class MainClass

{

  public static void main( String[] args )

  {

    Test t1 = new Test();

    t1.getN();

    t1.getN();

    Test t2 = new Test();

    t2.getN();

    t1.getN();

  }

}

5   
5   
5   
5

Many Java programmers follow the convention that the name of a class constant uses only capital letters, as in this code fragment:

public class MainClass   
{   
  private static final String TODAY = "Saturday";   
  
  public static void main( String[] args )   
  {   
    System.out.println( TODAY );   
  }   
}

Saturday

The purpose of the convention is to make the variable stand out from the surrounding code. By being made to notice it in this way, you — and any other programmers who might touch your code in the future — will be more likely to remember that this is a variable whose value cannot be changed.

Class variables don't have to be private. By declaring a class variable using the modifier keyword public instead of private, we allow that variable to be accessed directly from outside the class in which it is declared.