The final modifier keyword may also be used when declaring variables that are not class variables, that is, variables whose declarations do not include the static modifier. The effect of including the final modifier is to make it impossible to change the value of the variable, once it has been initialized. Furthermore, the absence of the static modifier has a profound effect upon the accessibility of the variable; it means that the variable is subject to the rules we listed [earlier](https://www.eimacs.com/eimacs/mainpage?epid=E1991300856&cid=162149#RulesVarDec), before we had told you about class variables. Each time the innermost block of code that contains the variable declaration is executed a new instance of the variable is created, and its value is only accessible from the point where it is initialized to the end of that block of code.

Consider the following code, for example:

public class Family   
{   
  ArrayList<FamilyMember> myMembers;   
  String    myFamilyName;   
  
  public Family( String lastName )   
  {   
    myFamilyName = lastName;   
    myMembers = new ArrayList<FamilyMember>();   
  }   
  
  public void add( FamilyMember famMem )   
  {   
    myMembers.add( famMem );   
  }   
  
  public String toString()   
  {   
    final int FAMILY\_SIZE = myMembers.size();   
  
    String s = "The " + myFamilyName + " family:\n";   
  
    for ( int i = 0 ; i < FAMILY\_SIZE ; i++ )   
      s += myMembers.get( i );   
  
    return s;   
  }   
}   
  
public class FamilyMember   
{   
  final String[] FIELD\_NAMES = { "First name", "Date of birth" };   
  String[] myData = new String[ FIELD\_NAMES.length ];   
  
  public FamilyMember( String firstName, String dob )   
  {   
    myData[ 0 ] = firstName;   
    myData[ 1 ] = dob;   
  }   
  
  public String toString()   
  {   
    final int NUM\_OF\_FIELDS = FIELD\_NAMES.length;   
  
    String s = "";   
  
    for ( int i = 0 ; i < NUM\_OF\_FIELDS ; i++ )   
    {   
      s += "  " + FIELD\_NAMES[ i ]  + ": ";   
      s += myData[ i ] + "\n";   
    }   
  
    return s + "\n";   
  }   
}

public class MainClass

{

  public static void main( String[] args )

  {

    Family f = new Family( "Smart" );

    f.add( new FamilyMember( "Farley", "10/31/1963" ) );

    f.add( new FamilyMember( "MerryAnn", "7/1/1965" ) );

    f.add( new FamilyMember( "Luke", "2/29/1988" ) );

    System.out.println( f );

  }

}

The Smart family:   
  First name: Farley   
  Date of birth: 10/31/1963   
  
  First name: MerryAnn   
  Date of birth: 7/1/1965   
  
  First name: Luke   
  Date of birth: 2/29/1988

In this code, the variable FAMILY\_SIZE is accessible only in the body of the toString method in the Family class, the variable FIELD\_NAMES is accessible throughout the definition of the FamilyMember class, and the variable NUM\_OF\_FIELDS is accessible only in the body of the toString method in the FamilyMember class. Notice that, just as we did in the case of class constants, we are using capital-letters-only names for these final block variables. Notice also that the initialization of FIELD\_NAMES is "hard-coded", whereas the values to which FAMILY\_SIZE and NUM\_OF\_FIELDS are initialized are calculated dynamically.

There are various reasons for using such variables. These include making it less troublesome for you (or another programmer) to update the code in light of changing circumstances. Suppose, for example, you decided to add another field to the FamilyMember class (a middle name field, for example). Then you would have to make a couple of small changes to the class constructor and change the (hard-coded) initialization of the FIELD\_NAMES variable. But all the rest of the code could remain untouched. Furthermore, if sensible and informative variable names are chosen, the use of final block variables can make the structure of your code much easier to grasp; in a way, the variable names provide a level of self-documentation within your code.