**public and private**

[A little while ago](https://www.eimacs.com/eimacs/mainpage?epid=E2232603872&cid=162149), we saw that the keywords public and private may be used to control the visibility of class variables and constants. In particular, a class variable that is declared as private is only visible to the methods of its own class. On the other hand, a public class variable is visible and may be used anywhere in the program in which it is declared.

In exactly the same way, the keywords public and private can be used to control the visibility of instance variables. However, it is the convention in the Advanced Placement examination — and it is often considered good programming practice — to declare instance variables as private, thereby restricting access to their values to the data accessor methods of the class in which they are declared.

Similarly, the keywords public and private may also be used to control the visibility of both the instance and the class methods of a class. A public instance or class method is visible throughout the program in which it is defined, whereas a private instance or class method may only be referenced within the class in which it is defined.

In general, it is considered good practice to declare as public only those methods that are specifically designed to be used outside the class, and to declare all other methods to be private. A particularly striking example of this occurs in the context of multiple constructors, where it is quite common for the various constructors to differ solely in the extent to which they assign default values to instance variables. In such circumstances, it is common in many programming languages — including Java — to use a single private method — often called initialize — to perform instance variable initialization and to call that method from each of the different constructors. Consider the following class, for example:

public class FastFood   
{   
  private String myType;   
  private String myFriesSize;   
  private String myDrink;   
  
  public FastFood()   
  {   
    initialize( "hamburger", "no", "no drink" );   
  }   
  
  public FastFood( String type )   
  {   
    initialize( type, "no", "no drink" );   
  }   
  
  public FastFood( String type, String friesSize )   
  {   
    initialize( type, friesSize, "no drink" );   
  }   
  
  public FastFood( String type, String friesSize, String drink )   
  {   
    initialize( type, friesSize, drink );   
  }   
  
  private void initialize( String type, String friesSize, String drink )   
  {   
    myType = type;   
    myFriesSize = friesSize;   
    myDrink = drink;   
  }   
  
  /\* other methods \*/   
  
  public String toString()   
  {   
    return myType + " with " + myFriesSize + " fries and " + myDrink;   
  }   
}   
    
public class MainClass   
{   
  public static void main( String[] args )   
  {   
    FastFood orderA = new FastFood();   
    FastFood orderB = new FastFood( "fish sandwich" );   
    FastFood orderC = new FastFood( "cheeseburger", "medium" );   
    FastFood orderD = new FastFood( "garden salad", "no", "small lemonade" );   
    System.out.println( orderA );   
    System.out.println( orderB );   
    System.out.println( orderC );   
    System.out.println( orderD );   
  }   
}

hamburger with no fries and no drink   
fish sandwich with no fries and no drink   
cheeseburger with medium fries and no drink   
garden salad with no fries and small lemonade

In a case like this, it is clear that the sole purpose of the initialize method is to avoid the need for duplicating several similar code statements in each of the constructors. Its usefulness is strictly internal to this class, and therefore it is natural to designate it as a private method.

Java provides an alternative method of avoiding code duplication in a multiple constructor context such as the FastFood class. It uses the keyword this to refer to the FastFood class itself, as follows:

public class FastFood   
{   
  private String myType;   
  private String myFriesSize;   
  private String myDrink;   
  
  public FastFood()   
  {   
    this( "hamburger", "no", "no drink" );   
  }   
  
  public FastFood( String type )   
  {   
    this( type, "no", "no drink" );   
  }   
  
  public FastFood( String type, String friesSize )   
  {   
    this( type, friesSize, "no drink" );   
  }   
  
  public FastFood( String type, String friesSize, String drink )   
  {   
    myType = type;   
    myFriesSize = friesSize;   
    myDrink = drink;   
  }   
  
  /\* other methods \*/   
  
  public String toString()   
  {   
    return myType + " with " + myFriesSize + " fries and " + myDrink;   
  }   
}   
    
public class MainClass   
{   
  public static void main( String[] args )   
  {   
    FastFood orderA = new FastFood();   
    FastFood orderB = new FastFood( "fish sandwich" );   
    FastFood orderC = new FastFood( "cheeseburger", "medium" );   
    FastFood orderD = new FastFood( "garden salad", "no", "small lemonade" );   
    System.out.println( orderA );   
    System.out.println( orderB );   
    System.out.println( orderC );   
    System.out.println( orderD );   
  }   
}

This use of the keyword this is *not* part of the Advanced Placement Java subset. However, there are other uses of the keyword that *are* in the subset. We meet them [in a little while](https://www.eimacs.com/eimacs/mainpage?epid=E201976518&cid=162149#This).

So far in this course, we have only seen public constructors. It is possible, however, to declare a constructor as private. This is useful in circumstances that occur in certain kinds of practical applications where it is important that only one instance of a class exists while a program is executing. Here is a trivial example:

public class OnlyOne   
{   
  private static OnlyOne myInstance;   
  private static String myName;   
  
  private OnlyOne()   
  {   
  }    
  
  public static OnlyOne getInstance( String name )   
  {   
    if ( myInstance == null)   
    {   
      myInstance = new OnlyOne();   
      myName = name;   
    }   
    return myInstance;   
  }   
  
  public String toString()   
  {   
    return "OnlyOne instance: " + myName;   
  }   
}   
  
public class MainClass   
{   
  public static void main( String[] args )    
  {   
    OnlyOne oo1 = OnlyOne.getInstance( "Adam" );   
    OnlyOne oo2 = OnlyOne.getInstance( "Eve" );     
    System.out.println( oo1 );   
    System.out.println( oo2 );   
  }   
}

OnlyOne instance: Adam   
OnlyOne instance: Adam

Here, by declaring the default constructor to be private we make it invisible and inaccessible outside the OnlyOne class. In fact, the *only* way to create OnlyOne objects outside the OnlyOne class is to use the getInstance class method, and the conditional statement in that method prevents a new instance being created if one already exists. Thus, the second statement in the above main method does not create a new OnlyOne object with name "Eve"; instead, it creates oo2 as an "alias" for the already-in-existence object oo1. (We have more to say about [object aliasing](https://www.eimacs.com/eimacs/mainpage?epid=E2316510320&cid=162149) later on.)

Using the keyword this in the way that we remarked on the previous page is *not* in the Advanced Placement Java subset, we can also combine public and private constructors to enforce the provision of arguments when class instances are created. Here is a trivial example:

public class Trivial   
{   
  private int myIndex;   
  
  private Trivial()   
  {   
    System.out.println( "One more triviality!" );   
  }   
  
  public Trivial( int index )   
  {   
    this();   
    myIndex = index;   
  }   
  
  public String toString()   
  {   
    return "Trivial #" + myIndex;   
  }   
}   
  
public class MainClass   
{   
  public static void main( String[] args )   
  {   
    Trivial t1 = new Trivial( 1 );   
    Trivial t2 = new Trivial( 2 );   
    System.out.println( t1 );   
    System.out.println( t2 );   
  }   
}

One more triviality!   
One more triviality!   
Trivial #1   
Trivial #2

Here, because the default constructor is private, the only way to create instances of the Trivial class is to use the public constructor, which requires the provision of an integer argument.

It is important to note that, just as in the case of super when creating subclasses of a class, the call to this in this context *must* be the first statement in the constructor code block.

The keywords public and private are the only access modifiers in the Advanced Placement Java subset. In addition to these, Java also provides two other types of visibility: protected and package.