This short section deals with the concepts of *mutability* and *aliasing*. Although these are not explicitly mentioned as belonging to the Advanced Placement Java subset, they can be a source of great confusion to any programmer who is not aware of their existence.

If an object is such that the values of its instance variables can be changed after it is created, then the object is said to be *mutable*. If an object's instance variable values are unchangeable after it is created, then that object is said to be *immutable*.

For example, objects of the following PersonA class will be immutable:

public class PersonA   
{   
  private String myName;   
  
  public PersonA( String name )   
  {   
    myName = name;   
  }   
  
  public String getName()   
  {   
    return myName;   
  }   
}

This is because the only thing that sets the value of the instance variable myName is the class constructor. Furthermore, although the class has an accessor method that returns the value of the instance variable myName, that value is a String and Strings are constants — they cannot be changed.

On the other hand, objects of the following PersonB class will be mutable because the instance method setName makes it possible for us to change the value of the instance variable myName after a PersonB object is created:

public class PersonB   
{   
  private String myName;   
  
  public PersonB( String name )   
  {   
    myName = name;   
  }   
  
  public String getName()   
  {   
    return myName;   
  }   
  
  public void setName( String newName )   
  {   
    myName = newName;   
  }   
}

It can be quite challenging to determine whether or not the values of a class's instance variables can be changed. There are, however, a number of rules of thumb:

* If one or more instance variables have corresponding modifier set... instance methods, then objects of the class are mutable.
* If each instance variable has a primitive data type or is a String and the class has no modifier instance methods, then objects of the class are immutable.
* If there is an instance variable that is an instance of Object or one of its subclasses, if there is a corresponding accessor get... instance method, and if the object that is the value of the instance variable is mutable, then objects of the class are mutable. (For example, any class with an instance variable whose value is an ArrayList and that has a corresponding accessor method will have mutable objects, because the ArrayList can be changed by "getting" it using the accessor method and then modifying it using its add method.)

Clearly, our programs can be much more dynamic if the state of the objects we create can be changed while execution of the program proceeds. Significant dangers arise, however, if mutable objects are in play when *aliasing* is also involved. Aliasing occurs when two or more different variables store one and the same object.

But first, consider this code fragment, which involves a (mutable) PersonB object. Study the code and try to figure out what the output will be. Then run the program to see how accurate your prediction was.

public class MainClass   
{   
  public static void main( String[] args )   
  {   
    PersonB a, b;   
  
    a = new PersonB( "fred" );   
    b = new PersonB( "fred" );   
  
    System.out.println( a.getName() );   
  
    b.setName( "eric" );   
  
    System.out.println( a.getName() );   
  }   
}

[Show program details »](https://www.eimacs.com/eimacs/mainpage?cid=162149&epid=E2104719982)

fred   
fred

Here is almost exactly the same code again. The only difference is that this time, instead of creating two PersonB objects each having the name "fred", we simply assign to the second variable, b, the PersonB object we have just created and stored in the first variable, a. Once again we change b's name. Will the output be different this time? What do you think? Run the program to test your prediction.

public class MainClass   
{   
  public static void main( String[] args )   
  {   
    PersonB a, b;   
  
    a = new PersonB( "fred" );   
    b = a;   
  
    System.out.println( a.getName() );   
  
    b.setName( "eric" );   
  
    System.out.println( a.getName() );   
  }   
}

[Show program details »](https://www.eimacs.com/eimacs/mainpage?cid=162149&epid=E2104719982)

fred   
eric

You should find that as a result of changing b's name, a's name has been changed too. Of course, since a and b are actually the same object, you should have predicted that this would happen. The variables a and b are said to be *aliases*, since they store exactly the same object — they are two different labels for one and the same thing.

Aliasing can be very useful in certain circumstances, but when one or more of the objects involved is mutable, unexpected behavior can occur if we are not very careful.