1 Overview

Grading: This lab requires the use of the grading sheet for responses that must be checked by your instructor (marked as Question) AND the submission of some programs to WebCAT (marked as Lab). Be sure to have a copy of the grading sheet, acquire the appropriate signatures and submit the sheet to your instructor in addition to submitting the appropriate labs to WebCAT.

In an earlier lab, you learned how to use BlueJ as an IDE to debug a class, specifically a PersonName class, and how use jUnit to test the class.

In this lab you will extend your understanding of classes and objects by

- building a new PersonName class,
- incorporating the PersonName class into a Person class,
- doing some testing with jUnit, and
- investigating what static means.

2 Creating the PersonName class

Add the following private instance variables to PersonName.
Code:

```
int numberOfNames
String firstName
String middleName
String lastName
String suffixName
```

Then add the following methods.

1. A no-argument default constructor method initializes each name to "" and the number of names to zero.

2. Getter/Setter (accessor/mutator) methods for each of the names above (a pair for each of the four names). Each mutator should appropriately adjust the `numberOfNames` variable each time a name is modified. That is, if the name equals "" before setting, increment `numberOfNames` and if the name equals "" after setting, decrement `numberOfNames`. For example, consider what should happen if we call `setFirstName("")` when `firstName` equals "".

```
public void setFirstName(String name)
public String getFirstName()
public void setMiddleName(String name)
public String getMiddleName()
public void setLastName(String name)
public String getLastName()
public void setSuffixName(String name)
public String getSuffixName()
```

3. Accessor method `public int getNumberOfNames()`

As an example: to create a person name for Bobby Smith one would use the following segment.
Code:

    PersonName p = new PersonName();
    p.setFirstName("Bobby");
    p.setLastName("Smith");

**Question 1:** Build the new PersonName class in BlueJ and submit the PersonName class to WebCAT as Lab 9A for grading.

3 Enhancing the PersonName Class

The PersonName class should incorporate a few more elements before we consider it complete. One of those is to allow for a constructor to accept a string which could have up to 4 names. One could call the constructor as

**Code:**

    PersonName p = new PersonName("Bobby Smith");

In a previous lab you debugged a method that counted the number of names, but in this situation the objective is to extract each word, one at a time, and assign the words to the names as follows depending on the number of words in the name.

- 1 word implies a first name only,
- 2 words implies a first and last name only,
- 3 words implies a first, middle, and last name, and
- 4 words implies a first, middle, last and suffix.
3.1 Using `String.split(" ")`

Look up the JavaDoc for the `String` class and look at the `String[] split(String regex)` method. We note that the method processes a `String` object based on a rule called a regular expression and returns an array of `String` objects. Furthermore, if we pass " " to `String.split` it will create a new array of each word in the original `String` separated by at least one space. An example of its use is as follows.

Code:

```java
String yolo = "You only live once";
String[] words = yolo.split(" ");
for (int i = 0 ; i < words.length ; i++)
    System.out.println(words[i]);
```

The previous example will output the following.

Code:

```
You
only
live
once
```

Use the following code as an additional constructor in your `PersonName` class.
Code:

```java
public PersonName(String wholename){
    firstName=""; middleName=""; lastName=""; suffixName="";
    numberOfNames=0;

    // store the name passed in to the method
    String[] words = wholename.split(" ");
    int numnames = words.length;

    if (numnames == 1)
        setFirstName(words[0]);
    else if (numnames == 2){
        setFirstName(words[0]);
        setLastName(words[1]);
    }
    else if (numnames == 3){
        setFirstName(words[0]);
        setMiddleName(words[1]);
        setLastName(words[2]);
    }
    else if (numnames == 4){
        setFirstName(words[0]);
        setMiddleName(words[1]);
        setLastName(words[2]);
        setSuffixName(words[3]);
    }
}
```

In addition to this method, write two more methods called `EntireName` and `Initials` each of which returns a `String` with the values indicated.
1. Code:

   public String EntireName()
   This returns firstName + " " + middleName + " " + lastName + " " + suffixName
   with appropriate spacing (one space between each name ex: if middleName is ",
   then it returns firstName + " " + lastName +" "+suffixName).

2. Code:

   public String Initials()
   This returns firstInitial + middleInitial + lastInitial if each respective name
   exists. For example if middleName is ",
   then it should return firstInitial + lastInitial.

Take time to examine the testPersonName9B.java file to understand the exact requirements
as tested by the included test cases.

Question 2: When you have this finished, submit the new PersonName
class with the additional constructor and methods for Initials and
EntireName to WebCAT as Lab

4 Creating the Person Class

Later in the course, we will use a Person class for another lab, so here we will be begin the
process of creating this class. The Person class that you are to create should be defined as:

- the Person should have a PersonName and an age field;
- a public Person() constructor that creates a newly born person (metaphorically
  speaking), one with "no name" as its first name, and age equal to zero;
- a public Person(String aName, int anAge) constructor that receives a String
containing the name and an int containing the age to be stored as the Person’s age;

• a public String getName() method that returns a String containing the Person’s entire name;

• a public void setName(String aName) method that sets the name to aName;

• a public int getAge() method that returns the age;

• a public void setAge(int anAge) method that sets age to anAge;

• and a public boolean canDrive() method that returns true if the Person is over 16 years old and false otherwise.

5 Testing the Person Class with jUnit

Create a JUnit test class which minimally creates three different people, one under 16, one equal to 16 and one over 16. This type of testing does boundary condition testing as it investigates how your program functions as the data transitions from one state to another; here from being too young to drive to being eligible to drive.

To do this right-click on Person and select “Create Test Class”. This creates a JUnit tester called PersonTest. Open PersonTest and delete everything between the brackets of public class PersonTest{...}. Add the following code for a new test method.

Code:

@Test public void testPerson()

Inside this method add the three Person objects described above. For testing, we will use the assertTrue(boolean statement) and assertFalse(boolean statement) methods that jUnit provides. The assertTrue takes a boolean as its input and evaluates whether or not it is true. If it is true, then it passes the test; otherwise it will fail. For example

Code:

    assertTrue(1==1)
would pass since 1 does indeed equal 1. On the flip side, `assertFalse` will pass if the boolean is false and will fail if it is true. Use `assertTrue()` and `assertFalse()` when calling `canDrive()` for each of your `Person` objects. Run the tester by right-clicking `PersonTest` and selecting “Test all”.

**Question 3:** When you have finished coding the jUnit class for testing `Person`, show your instructor.

## 6 static vs. non-static

Let’s investigate two important concepts with a simple exercise. Take the `canDrive` method and add a second method which is `static` and has an integer input parameter. In other words, add the following method declaration to the `Person` class while leaving the current `canDrive` in place.

**Code:**

```java
public static boolean canDrive(int age)
```

And now you have two `canDrive` methods in the `Person` class and we have now *overloaded* the `canDrive` method. Instead of testing an object’s age, this second version of `canDrive` grabs the age it will test from the parameter. Furthermore, we will not be able to access any fields within `Person` from this method since it is a `static` method. Static methods are independent of instances of each class and can be called without an attached object.

Another way of viewing this is to right-click the class and actually invoke `Person`’s `canDrive` method. It will prompt you with a request for the parameter value (an age) that it can use to answer the question. Note that this method does not use the age of a specific object, but can be used independent of the creation of objects.

On the other hand, right-click on the `Person` and use the constructor to create a `Person` object. Name it `bob` and give the `name` value as “Bob”, and choose an age to enter. You should see an object named `bob` in the lower left corner. Now right-click the object and
invoke the `canDrive` method. Note that this method is going to use `bobs` age to determine if he can drive, and does not prompt you for an age.

Two conclusions to observe:

1. You are **overloading** the method, defining two different versions of the method, only differing by the interface. Same as with the constructor for this class.

2. **Static** methods do not require you to create an object in order to use it. In this case, the `canDrive` method is available to use as long as you pass it the age. If you want it to respond based on the `Person` object, you must create it and then invoke as you did in the previous step.

In this case we have examined how to invoke these methods graphically, but how do you call this `static` method? A `static` method can be called without creating an object. In order to call the method, you must use the class name.

For example, the following code snippet is `static` as shown by calling `Person.canDrive(17).`  
**Code:**

```java
if (Person.canDrive(17))
    System.out.println("This person can drive");
```

Since `Person` is a class, `Person.canDrive(17)` invokes a `static` method. Whereas the following code is `nonstatic`

**Code:**

```java
Person myPerson = new Person("Bobby Paul Smith", 15);
If (myPerson.canDrive())
    System.out.println("This person can drive");
```

since `myPerson` is an object.

**Question 4:** When you have finished coding and testing the `Person` class, submit your `Person` class to WebCAT as Lab 9C.