UNBC Laboratory Assignment
Computer Science 101—Winter 2024

Static and non-static Members

Due Date:

This assignment is due Friday, January 19 *by the beginning of lecture*.

Please submit via `moodle` See there for more instructions on what to submit.

Persons

Here is a description of the code to implement:

The `Person` class

⇒ Implement a `Person` class that makes all of the methods shown in Figure 1 work as described.

- More precisely, the `public` methods of your `Person` class must include **all** the methods show and **only** the methods shown in Figure 1
- In particular, the `Person` class does *not* have a `main` method.
- In order to make the `Person` class work, you will need to add member variables, both `static` and *non*-`staic`.
  All member variables must be private.
- You may use additional `private` methods.

The `PersonTester` class

⇒ Implement a `PersonTester` class whose `public static void main(...)` method tells a story about `Persons`. The story should exercise the methods of the `Person` class.

- A tiny `PersonTester` class is shown in Figure 3 on page 5. (The coding style has been contorted so that the code fits on one page.)
- Your `PersonTester` should test more thoroughly.
- Have fun inventing your story!

.

Method	*Meaning*
constructors	
`public Person(String n)`	Creates a living person with name `n`.
attributes	
`public Person murderer()`	Returns the `Person` that murdered this person. Returns `null` if this person has not been murdered.
`public String name()`	Returns this person's name. Should end with "`, deceased`" if the person is dead.
`public boolean isAlive()`	*obvious.*
actions	
`public void die()`	Causes `this` Person to die, unless they are already dead, when it has no effect.
`public void murder(Person victim)`	Causes `victim` to die, and the murderer to be known to the victim.
`public void sayHello()`	causes this `Person` to print "`Hello, I'm` *`name`*`.`" on `System.out`.
class attributes	
`public static int numberLiving()`	
`public static int numberDead()`	
class actions	
`public static void allSayHello()`	causes every living person to say hello as described above.

Figure 1: Properties and Actions of `Persons`

```
/**
 * This file is part of a solution to
 *     CPSC 110 Lab 7 Problem 2 Winter 2010
 *
 * Implements a Roman numeral calculator
 *
 * @author David Casperson
 * Student Number: 783030901
 * @version 1
 */
```

Figure 2: Format of file header comment

More about `Person` methods

The point of this lab assignment is to re-inforce your understanding of `public` versus `private` access, and to re-inforce your understanding of `static` versus non-static methods and member variables.

Please ask in tutorial about what should be public, and what should be static.

The `Person` class has one one-argument constructor. The constructor sets the `Person`'s name; there is no way to change a person's name once they are c onstructed. This is deliberate. Ask in tutorial about constructors and how they relate to non-static member variables. Determine what (non-static) member variables you need in order to implement the *attributes* and *actions* listed on page 2.

The *class attributes* and *class actions* need to access appropriate private static member variables. In particular, the `allSayHello` static member function should cause every currently living person to say hello. The tricky part of this is finding all of the currently living persons. To accomplish this use static member variable(s) to keep track of living people, and ensure that these variable(s) are updated by appropriate non-static functions. One technique is to use a member variable like

```
private static ArrayList<Person> thePeople;
```

A harder and more direct approach is to link each `Person` to the next. In either case, you require some kind of static member variable. Choose whatever you are comfortable with.

Coding Requirements

- All member variables **must be** `private`.
- The `Person` class and the `PersonTester` class must be implemented in separate `.java` files.
- The classes should be in package `lab1`. (If packages have not been covered, do your best.)
- Each (and every) `.java` file should have a header comment that looks substantially similar to that shown in Figure 2. In particular, the comment
  - should be a `javadoc` comment (start with "`/**`"),
  - should have an @author line,
  - should have your student number, and
  - should say why the file exists (lines 2–3)
  - and what it does (line 5).
- The `public` methods of the `Person` class must be exactly those described in Figure 1. You may add as many private methods and fields as you see fit.
- The `PersonTester` class must contain a `public static void main()` method that runs your program. Content is up to you, but the `main` method should tell a story, and the output of the story should indirectly confirm that the `Person` methods work correctly.

Checklist

Here are some things to check.

- are your class names, methods, and member variables capitalized correctly?
- Make sure that `die()` applied to a dead person doesn't cause the population to decrease.
- Make sure that a person's murderer's name prints correctly if the murderer herself is dead.
- Make sure that murdering a living person causes them to die.
- Decide what happens when either a murderer or a victim is already dead AND DOCUMENT YOUR CHOICE.

```
// Put a proper file header comment here!
public class PersonTester
{ private static void print  (String s) {System.out.print(s) ;}
  private static void println(String s) {System.out.println(s) ;}

  private static void printCount()
      {
      final int livingCount = Person.numberLiving() ;
      final int deadCount   = Person.numberDead() ;
      final int totalCount  = livingCount + deadCount ;
      print("The number of people is "+totalCount) ;
      if (deadCount>0)
          println(String.format(", but %d of them are dead!%n",deadCount)) ;
      else println(".") ;
      }

  public static void main(String [] args)
      {
      Person bob = new Person ("Bob") ;
      println("Bob says \"hi\"") ;
      bob.sayHello() ;
      printCount() ;

      Person bill = new Person ("Bill") ;
      println("Bill is born.") ;
      bill.sayHello() ;
      println("Everyone says hello") ;
      Person.allSayHello() ;
      printCount() ;

      println("Bob dies.") ;
      bob.die() ;
      printCount() ;
      Person.allSayHello() ;

      Person sue = new Person("Sue") ;
      println("Sue is born.") ;
      sue.murder(bill) ;
      Person.allSayHello() ;
      printCount() ;
      println("Oh Oh Oh!") ;
      print(bill.name()+" is ") ;
      println(bill.isAlive() ? "hiding." : "dead.") ;
      if (bill.murderer()!=null)
          {
          println(bill.name()+" is murdered!") ;
          println("The murderer of Bill is "+bill.murderer().name()+".  ") ;
          }
      print("\n\nOur exciting story ends suddenly.\n") ;
}     }
```

Figure 3: A tiny crampled PersonTester class