COMP 401
Spring 2015
Midterm 1

I have not received nor given any unauthorized assistance in completing this exam.

Signature: ____________________________

Name: ________________________________

PID: _________________________________

Please be sure to put your PID at the top of each page.
This page left intentionally blank. Do not write on it.
Part I: True / False (40 points)

Directions:
Answer these questions on your Scantron answer form. Select A for true and B for false.

1. `unsigned int` is a value type in Java for representing unsigned integer numeric values.

2. `String` is a built-in value type in Java.

3. An instance of a class is always reference type in Java.

4. A class name usually starts with a capital letter.

5. `switch` is a valid variable name.

6. An expression can always be evaluated to produce a value.

7. An object reference is not a valid expression.

8. A variable can be assigned the result of any expression.

9. An `if` statement is a type of loop.

10. A `continue` statement within the body of a loop exits the loop.

11. A local variable declared within a loop is not available outside of the body of the loop.

12. Several case statements within a switch statement may be associated with the same block of statements to be executed.

13. A method must return a value that can be used in an expression.

14. To invoke (i.e., call) a method, you must provide a value for each parameter declared in the method's signature.

15. Invoking a class method (i.e., a static method) requires a reference to a particular instance (i.e., object) of the class.

16. When a method (call it method A) is called from within another method (call it method B), the code within method A can access local variables within method B at the point where the call to method A was made.

17. The length of an array in Java cannot be changed after the array is created.

18. A specific array can hold values of any type.

19. The `==` operator should not be used to compare whether two `String` objects are the same on a character-by-character basis.

20. The individual characters of a `String` object can be changed.

21. The principle of encapsulation encourages object fields to be marked as private.
22. Every method of a class should be included in at least one of the interfaces that the class implements.

23. Overloading a method is an example of the principle of encapsulation.

24. Setters should never be provided for an immutable object.

25. To adhere to the principle of encapsulation, a public method should not make use of (i.e., invoke or call) a private method declared in the same class.

26. It is possible to create an instance of a class that does not explicitly define a constructor.

27. Defining multiple constructors that accept different types and number of parameters is an example of polymorphism.

28. A constructor that chains to another constructor must do so as the first statement of the constructor body.

29. Two different versions of the same method can be declared to accept the same number and type of parameters as long as the parameters have been given different names.

30. All of the methods declared in an interface must be implemented as public methods.

31. An interface may only extend (i.e., inherit from) a single parent interface.

32. Every class must implement at least one interface.

33. An interface may declare several different versions (i.e., overload) the same method name.

34. An interface can only be implemented by one class.

35. The name of an interface is a valid type name that can be used as the type of a variable.

36. A class that implements an extended interface must implement all of the methods declared in the parent interface of the extended interface.

37. A subclass that extends (i.e., inherits from) a parent class automatically implements any interfaces that the parent class implements.

38. An instance field marked as private can be directly accessed by code outside of that class.

39. An instance of a subclass can always be used wherever a reference to an object with the parent class type is required.

40. A subclass can extend from more than one parent class.
Part II: Short answer / calculation (5 points per part A-E, 25 points total)

A) What is the value of the variable “result” after the code below executes?

```java
int[] a = new int[] {1, 1, 2, 3, 5};
int[] b = new int[] {a.length, a[0]+a[1], a[1]+a[2], 10};
int[] c = a;
int[] d = b;
c[0] = d[2];
a[1] = b[3];
a = b;
d = c;
b = d;
b[3] = a[b[0]];
int result = a[0] + b[1] + c[2] + d[3];
```

Answer: _______

B and C) What are the values of the variables j and k after the following code executes:

```java
int j=0;
int k=0;
for (int i=1; i<10; i += j) {
    j = 5;
    while (j > i) {
        k += j;
        j--;
    }
}
```

Answer: j = _______, k=_________

D and E) Given the following definition of foo:

```java
public static int foo(int a, int b) {
    if (b < a) {
        return foo(b, a);
    }
    if ((b/a)*a == b) {
        return a;
    }
    return foo(a, b-a);
}
```

What is the value of the expression foo(2, 7)? Answer: _______

Including the original call to the function foo in the expression above, how many times is foo invoked when the expression is evaluated?

Answer: ________
Part III: Reading and understanding code (15 points total)

Directions:
All of the questions in this section concern the code for the Phone interface and SmartPhone class that implements that interface given below. Each question in this part is worth 3 points.

```java
public interface Phone {
    String getNumber();
    String getLastNumberDialed();
    double currentBatteryLevel();
    void dial(String number);
    void dial();
} // End of interface Phone

public class SmartPhone implements Phone {
    private double battery_life;
    private String number;
    private String last_number_dialed;
    private final static String DEFAULT_NUMBER = "867-5309";

    public SmartPhone(String number) {
        this.number = number;
        battery_life = 1.0;
        last_number_dialed = null;
    }

    public SmartPhone() {
        this(SmartPhone.DEFAULT_NUMBER);
    }

    public String getNumber() {
        return number;
    }

    public String getLastNumberDialed() {
        if (last_number_dialed == null) {
            throw new RuntimeException("No last number. ");
        }
        return last_number_dialed;
    }

    public double currentBatteryLevel() {
        return battery_life;
    }

    public void dial(String number) {
        if (currentBatteryLevel() < 0.1) {
            throw new RuntimeException("Not enough battery");
        }
        battery_life *= 0.9;
        last_number_dialed = number;
    }
    // Continued on next page.
```
A) Which methods, if any, act as getters that follow the JavaBeans convention?

B) Which methods, if any, are overloaded?

C) How many instance fields does a Smartphone object have?

D) What will the following code snippet print?

```java
Phone a = new Smartphone();
Phone b = new Smartphone(8675309);
if (a == b) {
    System.out.println("Go!");
} else {
    System.out.println("Heels!");
}
```

E) Suppose I make the claim that because the code above does not provide any setters that follow the JavaBeans convention, a Smartphone object is immutable. Am I right? Why or why not?
Part IV: Writing Code (20 points, at discretion of grader)

Directions:
Below is the code for a class called LibraryImpl. It stores information about books (title, length, and author) in arrays. The class is not particularly well written. You will rewrite this class as described on the next page adhering to best practice as much as possible.

```java
public class LibraryImpl {
  String[] titles;
  int[] lengths;
  String[] authors;

  public LibraryArray(int capacity) {
    titles = new String[capacity];
    lengths = new int[capacity];
    authors = new String[capacity];
  }

  int findFirstOpenIndex() {
    for (int i=0; i<titles.length; i++) {
      if (titles[i] == null) {
        return i;
      }
    }
    return -1;
  }

  void addToLibrary(String title, int length, String author) {
    int first_open_index = findFirstOpenIndex();
    if (first_open_index == -1) {
      throw new RuntimeException("No room in the library");
    }
    titles[first_open_index] = title;
    lengths[first_open_index] = length;
    authors[first_open_index] = author;
  }

  void removeFromLibrary(String title) {
    for (int i=0; i<titles.length; i++) {
      if (titles[i].equals(title)) {
        titles[i] = null;
        lengths[i] = 0;
        authors[i] = null;
      }
    }
  }

  int getTotalPageCount() {
    int page_count = 0;
    for (int i=0; i<lengths.length; i++) {
      page_count += lengths[i];
    }
    return page_count;
  }
}
```

// End of LibraryImpl
First, create a class called BookImpl which encapsulates a String title, integer length, and String author. This class should implement the Book interface provided below that defines standard JavaBeans getters for this information.

```java
public interface Book {
    String getTitle();
    int getLength();
    String getAuthor();
}
```

Your implementation of BookImpl should go here:
Now rewrite LibraryImpl so that it makes use of the Book interface and your BookImpl class defined above. The new version should implement the Library interface defined below. Note, this interface has an overloaded definition for the method addToLibrary that your new version of LibraryImpl must also implement.

```java
public interface Library {
    void addToLibrary(Book book);
    void addToLibrary(String title, int length, String author);
    void removeFromLibrary(String title);
    int getTotalPageCount();
}
```

Your new version of LibraryImpl below:
Extra page for your code if needed.