# COMP 401 Midterm

## Tuesday, Oct 13, 2015, 11:00-12:15

## Instructions

1. Please spread out and try and sit in alternate seats.
2. This is a closed book exam. You will not be penalized for errors in Java syntax.
3. Write on the exam itself.
4. Write on a blank page if there is not enough space to solve a problem.
5. There are:

* 8 numbered pages including this one, and any marked blank pages.
* 3 questions.
* 75 possible points. Point values appear in brackets next to each question.

1. You have 75 minutes.
2. You are not required to comment any code you write, but may get partial credit if you write appropriate comments but incorrect code.
3. If you need to make any assumptions to clarify a problem, *write your assumptions down.* Only reasonable assumptions get full credit.
4. Please inform the proctor of anything in the exam that you think is a mistake.
5. Your code will be evaluated not only for correctness, but also for time and space efficiency and *style*.
6. You cannot use any Java capabilities not covered in class.
7. To answer questions about some piece of code given here, you can mark the code directly.
8. If you do not understand some English word, do not hesitate to ask the proctor. Naturally, you are expected to know the computer science terms defined in class.

Name (in Capitals) and Onyen

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pledge: I have neither given nor received unauthorized aid on this exam.

(signed)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For survey purposes, please indicate the time at which you turned in the exam.

\_\_\_\_\_\_\_\_\_\_\_

Please do not write below

1. \_\_\_\_\_/33 2. \_\_\_\_\_/14 3. \_\_\_\_\_/28

Total: \_\_\_\_\_/ 75

1. [33 pts.] Understanding Programs

All of the code related to this problem is located in the same package. Consider the follow code, which does not pretend do anything useful or follow style rules.

**public** **interface** SuperInterface {

**public** SuperInterface getThis();

**public** **int** getB();

**public** **void** setB(**int** newVal);

**public** String getC1();

**public** String getC2();

}

**public** **class** ASuperClass **implements** SuperInterface {

**int** a = 1;

**int** b = a - 2;

String s = "SuperC";

**public** ASuperClass() {b = b\*4;}

**public** SuperInterface getThis() {**return** **this**;}

**public** **int** getB() {**return** b;}

**public** **void** setB(**int** newVal) { b = newVal;}

**public** **int** getA(**int** anA) { **return** a + anA; }

**public** String getC1() {**return** s;};

**public** String getC2() {**return** s;};

}

**public** **class** ASubClass **extends** ASuperClass {

**int** d = a - b/2;

**public** String getC1() {**return** "SubC";};

**public** **static** **void** print(ASuperClass aSuper) {

System.***out***.println ("super:" + aSuper.getC1());

}

**public** **static** **void** print(ASubClass aSub) {

System.***out***.println ("sub:" + aSub.getC1());

}

}

**public** **static** **void** main (String[] args) {

// code manipulating aSuper

ASuperClass aSuper = **new** ASuperClass();

System.***out***.println (aSuper.getB());

aSuper.setB(4);

System.***out***.println(aSuper.getB());

// code manipulating aSub

ASubClass aSub = **new** ASubClass();

System.***out***.println (aSub.getC2());

System.***out***.println (aSub.getC1());

System.***out***.println (((ASubClass) aSub).getC1());

ASubClass.*print*(aSub);

ASubClass.*print*((ASuperClass) aSub);

}

1. [12 pts] Identify in the code below, the declaration of:

An instance variable

An instance method

A static method

A constructor

An inherited variable

An inherited method

An overridden method

An overloaded method

1. [21 pts] Give the output by each println() executed by the main method above. You can list each output below or next to the associated println(). There should be seven outputs.
2. [14 pts.] Logical Structure and Trees, Dags and Graphs

Draw the logical structure of the object: **new** ASuperClass(). Recall the logical structure is the one that is extracted from the (headers of the) instance methods of the object. Recall also that the logical structure is instance specific and depends on the property values of the instance. It can be a tree, DAG, or graph. The code for ASuperClass is reproduced below.

**public** **class** ASuperClass **implements** SuperInterface {

**int** a = 1;

**int** b = a - 2;

String s = "SuperC";

**public** ASuperClass() {b = b\*4;}

**public** SuperInterface getThis() {**return** **this**;}

**public** **int** getB() {**return** b;}

**public** **void** setB(**int** newVal) { b = newVal;}

**public** **int** getA(**int** anA) { **return** a + anA; }

**public** String getC1() {**return** s;};

**public** String getC2() {**return** s;};

}

// use this page if you run out of space for the next question

1. [28 pts] WRITING CODE

*Follow all the style principles you have learned in the writing of the code required in parts a) and b)*

1. [13] Consider the following table interface:

**public** **interface** Table {

**public** Object get(Object aKey);

**public** **void** put (Object aKey, Object aValue);

**public** Object[] keys();

}

Assume that an implementation, ATable, of this interface exists:

**public** **class** ATable **implements** Table{ … }

This class, *which is given to you*, extends the function of the table object required in assignment 7. The get and put methods are the same in the assignment and their explanations are reproduced here. The operation put (key, value) checks if key is already associated with a value. If it is, it associates key now with value. Otherwise it adds the new association (key, value) to the collection of associations. The put operation does nothing if key or value is null. The method get (key) returns null if key is not bound to any value; otherwise it returns the value to which key is bound. The keys() method returns a compact array of all the keys in the table, and has no empty slots. The order of the elements in the array is arbitrary.

Implement code that can be used to instantiate a *Set* object on which the following methods can be invoked:

**public** **void** add(Object anElement);

**public** Object[] elements();

Like the sets we saw in class, this object does not store duplicate items. Thus, the add method adds an item to the set if it does not already exist. The elements() method returns a compact array of all non-duplicate items added to the set. Again, the array has no empty slots and the order of the elements in the array is arbitrary. *To get full points you must use the Table and ATable types given to you.* You will get partial credit for using other data structures we have covered in class. You are free to choose the names of any classes or interfaces you implement*. Again, do not implement a table, it is given to you.*

*If you get stuck on how a table should be used, go to the next problem, and return to this problem after finishing it.* *You can get full points for the next part without getting this part correct.*

1. [15 pts] Your task now is to implement the method:

public static void printLettersUsed(String aString)

The method prints all letters in its argument, ignoring duplicates. Each letter is printed on a separate line, and the order of the letters is arbitrary. Thus, the call printLettersUsed ("hhe023llo!!!"), prints:

e

h

l

o

*To get full points you must use the Set code you wrote for 3(a).* You will get partial credit for using other data structures and writing code that contributes to the solution. You can, of course, use the Character.isLetter() method.