**\_\_\_\_CS211 Fall 2013**

Dr. Kinga Dobolyi

**Exam 2, part 1**

**DO NOT START**

**Part 1: Short Answer.**

1. **Examine the following class:**

**import** java.util.ArrayList;

**public** **class** Thing{

**private** **static** **int** *count* = -1;

**public** **int** size;

**public** String name;

**public** Thing(**int** count, String n){

System.*out*.println("start Thing ctor");

//tricky!

size = count;

name = n;

System.*out*.println("in Thing ctor, size= " + size);

System.*out*.println("in Thing ctor, name= " + name);

System.*out*.println("in Thing ctor, count= " + **this**.*count*);

}

**public** **void** run(**int** times, ArrayList list){

**for** (**int** i = 0; i < times; i++)

**if** (i < *count*)

System.*out*.println("list["+i+"]= " + list.get(i).toString());

**else**

System.*out*.println("too large");

}

**public** **void** setSize(**int** s){

size = s;

}

**public** **static** **void** setCount(**int** c){

*count* = c;

}

**public** String toString(){

**return** name + " " + size + " " + *count*;

}

}

**CS211 Fall 2013**

Dr. Kinga Dobolyi

**Exam 2**

Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student G#: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student signature for Honor Code:

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

80 points total (will be recorded out of 100 on Blackboard)

75 minutes (for both parts of the exam together)

**1.A: Provide the output for the class below (21 points):**

OUTPUT:

**public** **class** Driver **extends** Thing{

**private** String driverName = "undefined";

**public** Driver(){

**super**(5,"none");

System.*out*.println("in Driver ctor");

}

**public** Driver(Thing thing, String d){

**super**(thing.size,thing.name);

driverName = d;

System.*out*.println("in Driver ctor2");

}

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

Thing t1 = **new** Thing(11, "thing1");

Thing t2 = **new** Thing(2, "thing2");

Driver d1 = **new** Driver();

Driver d2 = **new** Driver(t1,"driver2");

d1.*setCount*(14);

d2.setSize(7);

System.*out*.println("t2: " +

t2.toString());

System.*out*.println("d1: " +

d1.toString() + " " + d1.driverName);

System.*out*.println("d2: " +

d2.toString() + " " + d2.driverName);

}

}

**1.B: Provide the output for the class below (19 points):**

OUTPUT:

**import** java.util.ArrayList;

**public** **class** Driver2 **extends** Thing{

**private** String driverName = "undefined";

**public** Driver2(){

**super**(5,"none");

System.*out*.println("in Driver ctor");

}

**public** Driver2(Thing thing, String d){

**super**(thing.size,thing.name);

driverName = d;

System.*out*.println("in Driver ctor2");

}

**public** **void** run(ArrayList list){

**for** (**int** i = 0; i < list.size(); i++)

System.*out*.println("list["+

i+"]= " + list.get(i).toString());

}

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

Thing t1 = **new** Thing(11, "thing1");

Driver2 d2 = **new** Driver2();

t1.*setCount*(2);

ArrayList li = **new** ArrayList();

li.add(t1);

li.add(d2);

li.add("hello");

t1.run(5,li);

d2.run(5,li);

}

}

1. Examine the following three files, all saved in the same directory. None of them compile as written. Then, circle the line number in the file and BRIEFLY state the compilation error next to the line number you circled. Do this for all classes in the following order: 1) all compilation errors for **Animal** (saved in a file called Animal.java). Then, 2) assume those compilation errors in **Animal** have been fixed with the minimal amount of changes necessary to make the program compile, but have not changed any of the method signatures or the types or visibilities of any attributes/methods, nor have they added the abstract keyword anywhere. Now list the compilation errors for **Lion** (saved in Lion.java). Then, 3) assume those compilation errors in **Lion** have been fixed with the minimal amount of changes necessary to make the program compile, but have not changed any of the method signatures or the types or visibilities of any attributes/methods, nor have they added the abstract keyword anywhere. Now list the compilation errors for **Exam2** (saved in Exam2.java). You MUST circle both the line number and write a correct explanation of what the compilation error is. If there is no obvious line number for the compilation error, you should write FILE for line number at the bottom of the page and write the error there. Assume that all compilation errors are independent in the same file. (25 points)

1 public interface Animal extends Comparable{

2 public eat();

3 public void sleep();

4 public int drink(ArrayList list);

}

(over)

**5 import** java.util.ArrayList;

**6 public** **class** Lion **implements** Animal{

7 **public** **static** **int** *weight* = 8;

8 **public** **int** size = 0;

9 **private** String name = "none";

10 **private** **static** **int** *count*;

11 **private** **void** eat(){

12 System.*out*.println("sleeping")

13 }

14 **public** **void** sleep(){

15 **double** hours = 12 / size;

16 **int** f = hours;

17 }

18 **public** **int** drink(ArrayList list){

19 **return** *weight*;

20 }

21 **private** **static** **void** run(){

22 *count* += size;

23 }

24 **public** **static** **int** stop(){

25 **int** v = 5;

26 *count* = *count* - v;

27 }

28 }

(over)

**29 import** java.util.ArrayList;

**30 public** **class** Exam **extends** Lion{

31 **public** Exam2(String n){

32 **super**();

}

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

34 Animal A = **new** Animal();

35 Lion l1 = **new** Lion();

36 Lion l2 = **new** Lion("roar");

37 Exam2 e1 = **new** Exam2();

38 Exam2 e2 = **new** Exam2("purr");

39 System.*out*.println(Lion.*weight*);

40 System.*out*.println(Lion.size);

41 System.*out*.println(Lion.name);

42 System.*out*.println(Lion.*count*);

43 System.*out*.println(l1.*weight*);

44 System.*out*.println(l1.size);

45 System.*out*.println(l1.name);

46 System.*out*.println(l1.*count*);

47 System.*out*.println(Lion.eat());

48 System.*out*.println(Lion.sleep());

49 System.*out*.println(Lion.drink(**new** ArrayList()));

50 System.*out*.println(Lion.*run*());

51 System.*out*.println(Lion.*stop*());

52 System.*out*.println(l1.eat());

53 System.*out*.println(l1.sleep());

54 System.*out*.println(l1.drink());

55 System.*out*.println(l1.*run*());

56 System.*out*.println(l1.*stop*());

57 e2.compareTo(e1);

58 e2.compareTo("lion");

}

}

1. Fill in the Junit test cases for the following scenario. The test case MUST reveal the bug as described. (15 points)

Scenario: Imagine you are trying to write a method with the following signature:

public ArrayList sortDescending(ArrayList listIn)

This method will sort the incoming objects in descending order, based on their natural sorted ordering, as determined by the Comparable interface implementation. You may assume the method will only be called with objects that are Comparable, and you should not test for anything else.

The person who coded up this method did a good job, but they were distracted when writing their code, and they never consider the last element in the incoming argument for listIn. That is, the size of the incoming argument and the returned list are the same, but their algorithm does not consider the last element for sorting.

1. Write a Junit test case that would reveal this fault (assume all imports exist):

public void test1(){

//your code here

}

1. Now write a test case that would pass on the problem above, but would not reveal the fault we’re looking for:

public void test2(){

//your code here

}

1. TRUE/FALSE. Circle one and justify your answers for full credit. (5 points)

a. A parent’s public method can be accessed within the child using the super keyword. a. TRUE FALSE

b. A parent’s private method can be accessed within the child using the super keyword. b. TRUE FALSE

c. An abstract class must contain at least one abstract method.

c. TRUE FALSE

d. A try-catch block can have multiple catch clauses.

d. TRUE FALSE

e. A class can implement multiple interfaces.

d. TRUE FALSE