

PART 1

1. Note: The answers here are **different** from Exam1, Fall2013. Syntactic differences change the actual output of the program.

a) Output:

```
start Thing ctor
in Thing ctor, size= 11
in Thing ctor, name= thing1
in Thing ctor, count= -1
start Thing ctor
in Thing ctor, size= 2
in Thing ctor, name= thing2
in Thing ctor, count= -1
start Thing ctor
in Thing ctor, size= 5
in Thing ctor, name= none
in Thing ctor, count= -1
start Thing ctor
in Thing ctor, size= 11
in Thing ctor, name= thing1
in Thing ctor, count= -1
in Driver ctor2
t2: thing2 2 14
d1: none 5 14 undefined
d2: thing1 7 14 driver2
```

b) Output:

```
start Thing ctor
in Thing ctor, size= 11
in Thing ctor, name= thing1
in Thing ctor, count= -1
start Thing ctor
in Thing ctor, size= 5
in Thing ctor, name= none
in Thing ctor, count= -1
in Driver ctor
list[0]= thing1 11 2
list[1]= none 5 2
too large
too large
too large
list[0]= thing1 11 2
list[1]= none 5 2
too large
too large
too large
```

2.

These answers and their changes are annotated in the folder pt1q2, in their Java files. Line numbers given by (line number in Test / line number in File). A Line number of 0 is Top of File.

1) Animal

Line 0 / 1	No import to ArrayList
Line 2 / 4	Eat() method lacks type

```
// No import to ArrayList
import java.util.ArrayList;
public interface Animal extends Comparable{
    //      public eat(); No method type. Picked void for Lion.java
```

```

public void eat();
public void sleep();
public int drink(ArrayList list);
}

```

2) Lion

Line 6 / 2	Lion does not implement Comparable, as required by Animal
Line 11 / 9	Eat() method is private, instead of public as per Animal.
Line 16 / 16	In sleep() method, int f is being assigned the value from double hours . Java doesn't allow implicit casting like this.
Line 22 / 23	In run() method, cannot use a non-static variable size in static method
Line 27 / 32	In stop() method, there is no return value

```

import java.util.ArrayList;
// There's no implementation of the Comparable class. Fix by implementing below.
public class Lion implements Animal{
    public static int weight = 8;
    public int size = 0;
    private String name = "none";
    private static int count;

    // private void eat(){ <-- Change visibility
    public void eat(){
        System.out.println("sleeping");
    }

    public void sleep(){
        double hours = 12/size;
        // int f = hours; <-- Improper casting
        int f = (int) hours;
    }

    public int drink(ArrayList list){
        return weight;
    }

    // private static void run(){ <-- Change static to non-static
    public void run(){
        count += size;
    }

    public static int stop(){
        int v = 5;
        count = count - v;
        // Forgot a return statement
        return count;
    }

    // As required by Comparable
    public int compareTo(Object o){
        return 1; // <-- DO NOT DO THIS IN REAL LIFE
        // DO THIS PROPERLY IN REAL LIFE.
        //http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html
    }
}

```

3) Exam2

Line 30 / 2	The name of the class is wrong.
Line 34 / 9	Animal cannot have an object
Line 36 / 11	Lion does not have a constructor that takes only a string
Line 37 / 12	Exam2 does not have a default constructor
Line 40 / 17	size is not a static variable. Cannot be accessed using Lion.size
line 41 / 18	name is a private variable. Cannot be accessed without getters
line 42 / 19	count is a private variable. Cannot be accessed without getters
line 45 / 23	name is a private variable. Cannot be accessed without getters
line 46 / 24	count is a private variable. Cannot be accessed without getters
line 47 / 26	a) eat() is not a static method. Cannot use Lion.eat() b) eat() is a void method. println cannot print void-types
line 48 / 27	a) sleep() is not a static method. Cannot use Lion.sleep() b) sleep() is a void method. println cannot print void-types
line 49 / 28	drink() is not a static method. Cannot use Lion.drink()
line 50 / 29	a) run() is not a static method. Cannot use Lion.run() b) run() is a void method. println cannot print void-types
line 52 / 32	eat() is a void type. println cannot print void-types
line 53 / 33	sleep() is a void type. println cannot print void-types
line 54 / 34	There does not exist a drink() method that takes no parameters.
line 56 / 35	run() is a void method. println cannot print void-types
* Note: (line 58 / 39) can cause a problem depending on how you implemented compareTo() in Lion.java	

```
import java.util.ArrayList;
// public class Exam extends Lion{
public class Exam2 extends Lion{
    public Exam2(String n){
```

```

        super();
    }

    public static void main(String[] args){
        //      Animal A = new Animal(); <-- Cannot make object using Animal
        Lion l1 = new Lion();
        //      Lion l2 = new Lion("roar"); <-- Lion does not have this ctor
        //      Exam2 e1 = new Exam2(); <-- Exam2 has no default ctor
        Exam2 e1 = new Exam2("bark"); // <-- Replaced e1
        Exam2 e2 = new Exam2("purr");

        System.out.println(Lion.weight);
        //      System.out.println(Lion.size); <-- size not static
        //      System.out.println(Lion.name); <-- name is private
        //      System.out.println(Lion.count); <-- count is private

        System.out.println(l1.weight);
        System.out.println(l1.size);
        //      System.out.println(l1.name); <-- name is private
        //      System.out.println(l1.count); <-- count is private

        //      System.out.println(Lion.eat()); <-- not static method. can't print void
        //      System.out.println(Lion.sleep()); <-- not static method. can't print void
        //      System.out.println(Lion.drink(new ArrayList())); <-- not static method.
        //      System.out.println(Lion.run()); <-- not static method. can't print void
        System.out.println(Lion.stop());

        //      System.out.println(l1.eat()); <-- can't print void
        //      System.out.println(l1.sleep()); <-- can't print void
        //      System.out.println(l1.drink()); <-- Method does not exist
        //      System.out.println(l1.run()); <-- can't print void
        System.out.println(l1.stop());

        e2.compareTo(e1);
        e2.compareTo("lion"); // <-- Depends on compareTo implementation
    }
}

```

PART 2

1.

a)

```

public void test1(){
    ArrayList x;
    x.add(0);
    x.add(1);
    x.add(2);
    assertEquals(0, sortDescending(x).get(2));
}

```

b)

```

public void test2(){
    ArrayList x;
    x.add(0);
    assertEquals(0, sortDescending(x).get(0));
}

```

2.

A parent's public method can be accessed within the child using the super keyword.	True. If the parent's willing to share it, then the child can have it.
A parent's private method can be accessed within the child using the super keyword.	False. What belongs privately to the parent stays with the parent. Like a child asking for the parent's private keys to the car. I wouldn't give it to them, would you?
An abstract class must contain at least one abstract method.	False. An abstract class does not contain abstract methods. That's the interface.
A try-catch block can have multiple catch clauses.	True. It's a series of catch clauses, curly brace after curly brace, like an if-else if statement. Or,
A class can implement multiple interfaces.	True. A class can implement as many as they choose, but can only inherit once.