Final Examination

Instructions

• You have 4 hours to complete the exam.

• This is an open book, open notes, closed computer examination.

• This examination consists of 3 questions worth 100 points. Question 1 counts for 50 points, question 2 for 25 point and question 3 for 25 points.

• Read each question completely before attempting to solve any part.

• Write your answers legibly on separate answer sheets. If you use the back of a sheet, indicate clearly that you have done so on the front.

1 Objects and Classes [50 points]

Suppose that we want to implement a small data base in Java for storing information about the items on stock in a furniture shop.

We will consider just three kinds of items: sofa, dinner table, coffee table. We want to store the following kind of information for each:

Sofa: price (an integer), number-on-stock (an integer), item-number (an integer), no-of-seats (an integer), color (a string), material (a string)

Dinner table: price (an integer), number-on-stock (an integer), item-number (an integer), height (an integer), width (an integer), length (an integer), material (a string)

Coffee table: price (an integer), number-on-stock (an integer), item-number (an integer), height (an integer), width (an integer), length (an integer), material (a string)

Problem 1

Draw a class diagram, which can be used to represent the data. The superclass in the class hierarchy should be called Item. Hint: note that we wish to store the same information for dinner tables and coffee tables, so consider using a superclass for dinner tables and coffee tables.
Problem 2

Implement the classes from Problem 1 in Java. That is, show the java code defining each class. Each class should have a constructor and a print method with signature public void print(), which prints out the information in a simple way.

We now wish to model the stock of items in the furniture shop. For simplicity, we decide to model the stock using a fixed-size array and just assume that the shop will never have more than 1000 items on stock. Thus we use the following class:

class Stock {
  Item[] items;
  private int n; % the number of items on stock
  public Stock() {
    items = new Item[1000];
    n = 0;
  }
  public void add_item(Item i) {
    /* add item i to the stock */
    ...
  }
  public void print() {
    /* print out all items on stock */
    ...
  }
  public int no_of_sofas() {
    /* return the number of sofas on stock */
    ...
  }
}

Problem 3

Implement the add_item method, which adds an item to the stock.

Problem 4

Implement the print method, which should print out all items on stock (you should only iterate through the number of items on stock).

Problem 5

Explain briefly in words (one or two sentences) why the print method type checks statically and why it prints out the right information for the different kinds of items.

Problem 6

Implement the no_of_sofas method, which should return the number of sofas on stock.
After using our mini-database, the furniture shoup realizes that it would like to be able to store more than 1000 items on stock. Thus we decide to change our implementation of the stock. Instead of using a fixed-size array as above, we could use one of the Java collection libraries, but we instead decide to implement our own data structure. Thus we define a class DStock as outlined below:

```java
class DStock {
    Item i;
    DStock next;
    public DStock(Item i, DStock n) { i = i; next = n; }
    public DStock add(Item i, DStock s) {
        return new DStock(i, s);
    }
    public void print() {
        ...
    }
    public int no_of_sofas() {
        ...
    }
}
```

The idea is of course that an object of class DStock contains an item and then a reference next to the rest of the stock. If the next reference is null then there are no more items in the stock.

**Problem 7**

Suppose i1, i1, i3 are objects of class Item. Consider the code

```java
s = add(i1,
    add(i2,
    add(i3,
    null)));
```

Draw the object diagram for s. You don’t need to show the details of the items objects.

**Problem 8**

Implement the print method, which prints out all items on stock.

**Problem 9**

Implement the no_of_sofas method, which should return the number of sofas on stock.

## 2 Inner Classes [25 points]

Consider the following situation at a harbor where containers are moved onto and from ships. Unfortunately the crane driver has called in sick today, but fortunately, in the main crane cabin, there is a remote control that one can take down and control the crane from the ground.
You are asked to model this “taking a remote control and giving it to the worker” in Java using inner classes.

Problem 1

1. What is an inner class?
2. What is an anonymous inner class?

Problem 2

The remote control will be an instance of interface Remote

```java
interface Remote {
    void up (double l);
    void down (double l);
    void left (double l);
    void right (double l);
}
```

where up (1) and down (1) model the usual up and down buttons to raise and lower a weight by 1 meters, and left (1) and right (1) move the hook from left to write. The center of crane is 0. We assume that the worker is responsible enough not to break the crane by using unreasonable values of l. Once the hook is in position, workers will attach and unattach it to and from the container.

In this exercise you are asked to implement a class MyCrane that implements the interface Crane with one single method that will allow us to retrieve the remote control from the crane.

```java
interface Crane {
    Remote getRemote ();
}
```
Hint: You will need to add some state, of course, to represent the position and height of the crane hook, best expressed as

```java
private double x = 0;
private double height = 0;
```
all of which are private and hence not defined in the interface.

### Problem 3

Consider now an implementation of a worker to whom you can give the remote control of the crane.

```java
class Worker {
    Remote r;
    void passRemote (Remote r) {
        this.r = r;
    }
    void doJob () {
        r.up (10);       // (1)
        r.left (90);     // (2)
        r.down (10);     // (3)
    }
}
```

and the following list of commands

```java
MyCrane k = new MyCrane ();
Worker w = new Worker ();
w.passRemote (k.getRemote ());
w.doJob ();
```

When the program runs, it will eventually execute (1), (2), (3) in the `doJob` method of `Worker`. Report the inner state of the crane object `k` after the respective lines are executed.

### 3 Design By Contract and Inheritance [25 points]

Consider the following statistical problem. You are throwing a dice `max` number of times. After every throw, you record the result in an an array `x`. To determine the quality of your dice, you need to compute the probability of a certain number showing and hope that it is close to 1/6. This probability can be computed in the following way. Traverse the array `x` and count how many times that number was shown and divide the sum by the number of trows `max`. A probability is always a number between 0 and 1, inclusive.

**Problem 1:**

What is a class invariant?
Problem 2:

Consider the following sketch of a class that describes that experiment. You are asked to implement the function prob that computes the probability outlined above, which satisfies the contract expressed by Pre and Post condition. Argue how and why your implementation satisfies the contract.

class Dice {
    // Class Invariant: x[i] is in {1,2,3,4,5,6} for all 0<i<max
    int [] x;
    int max;

    // Contract for method prob:
    // Pre: y \in {1,2,3,4,5,6}
    // Post: 0 \leq return \leq 1

    float prob (int y) {
        ...
    }
}

Problem 3:

Some unexperienced programmers have defined the following four subclasses of your Dice class. None work correctly, but some at least satisfy the contract. Analyze all four classes and either explain why they still satisfy the contract, or give a counter example in which way they violate the contract.

1. class DiceA extends Dice {
   float prob (int y) { return (super.prob ((y - 42) % 6)); }
}

2. class DiceB extends Dice {
   float prob (int y) { return (super.prob (y-1)); }
}

3. class DiceC extends Dice {
   float prob (int y) { return (12 * super.prob (y)); }
}

4. class DiceD extends Dice {
   float prob (int y) { return (super.prob (y) / 2); }
}