

CS10, 09S, UCSB

H11: (Based on Horstman, Chapter 9, Interfaces) Total Points: 50 ([printable PDF](#))

Accepted: on paper, in lecture (1-1:50pm) on Friday, May 8th only.  
 No email submission allowed.

Name: (1 pts) \_\_\_\_\_ UCSBNetID (1 pts) \_\_\_\_\_

**Section (1 pts) Circle one:**      Thu 4pm                  Fri 10am                  Fri 11am                  Fri noon

1. Before reading Chapter 9, it will really help to re-read section 6.4, and familiarize yourself with the DataSet class. Once you've done that, read Section 9.1 then answer this question:

According to the textbook, "A Java Interface type declares a set of methods and their signatures".

- a. (4 pts) A *signature* for a method—in order to *be* a signature, has *only three* of the four things listed below. Which of the the following things does a method signature definitely not have? (Circle it)

implementation

name

list of parameters (or an empty list of parameters, if there aren't any)

return type (or void)

- b. (4 pts) *Why* does a Java Interface type *not* have any instance variables?

(The answer isn't directly in the textbook—you'll have to read what the textbook has to say on the subject, then do a bit of your own thinking. You can't just find the right sentence and copy it word for word.)

2. Read Sections 6.4 (about the DataSet class), and then Sections 9.1, 9.2, and 9.3 about interfaces and polymorphism—particularly the parts about the Measurable interface, and how it can apply both to the Coin class and the BankAccount class. Then answer these questions:

We've already established certain facts about references, constructors and objects—for instance, we know that:

- If I write `Coin c;` then `c` is a reference that can refer to a Coin object. I can make a new coin by writing `c = new Coin();` after which, `c` will refer to a `Coin` object.
- If I write `BankAccount b;` then `b` is a reference that can refer to a BankAccount object. I can make a new bank account object by writing `b = new BankAccount();` and then `b` refers to a new BankAccount object.

Given that both Coin and BankAccount implement the Measurable interface:

- a. (4 pts) Can I write this code?

```
BankAccount b = new Measurable();
```

- b. (4 pts) Can I write this code?

```
Measurable m1 = new BankAccount();
```

- c. (4 pts) Can I write this code?

```
Measurable m2 = new Measurable();
```

- d. (4 pts) Can I write this code?

```
Coin c = new Coin()
Measurable m = c;
```

**Please turn over for more problems**

3. Read section 9.4, and compare the approach used in Section 9.4 with that used in Sections 9.1 through 9.3—then answer this question.

In sections 9.1-9.3, we were able to reuse the DataSet class to find the maximum, minimum, count and average of both Coin and BankAccount objects by just adding the concept of a Measurable interface.

The solution in Section 9.4 is a bit more complex. To make this solution work, we have to add both a Measurerer interface, and a new class called a RectangleMeasurer.

- a. (4 pts) On what basis are we adding up rectangles, or finding the minimum or maximum? What about the rectangle are we measuring?
- b. (5 pts) My next question to you is: why was this extra complexity necessary? Why did the simple solution that we used with Coin and BankAccount not work for Rectangle objects?
4. In part (b) the previous question, you've just explained why the solution presented in Sections 9.1-9.3 could not be used for Rectangle objects, even though it applied just fine to Coin and BankAccount objects. There is an additional benefit of this solution—we could measure the Rectangle in more than one way.
- a. (4 pts) On what basis does the RectangleMeasurer class presented in Section 9.4 "measure" a rectangle? What exactly, about the rectangle is being measured?
- b. (10 pts) Suppose, instead, we wanted to use ALSO use width as a measure of a rectangle, and implement a RectangleWidthMeasurer. Fill in the body of that class below.

```
public class RectangleWidthMeasurer
{
```

```
}
```

---

End of H11