Tutorial 4

Objectives

Writing constructors, getters, setters and overloading Object's methods.

Attendance Quiz

Please log on to cuLearn using on of the computer in the tutorial room and complete the attendance quiz. You can only access the quiz if you log in using one of the computers in the lab. You cannot use a laptop for this. This is a time limited quiz. Be sure to do this as soon as you arrive.

At the end of the tutorial a TA will assign you a grade, call it G, which is 0, 1 or 2, depending on the progress you make during the tutorial. If you spend your time reading mail and chatting with your friends you will receive 0. If you have completed the attendance quiz on time then G will be your tutorial grade. If you have not completed the attendance quiz on time, then your tutorial grade will be max(0, G - 1/2). Each tutorial grade will therefore be one of 0, 0.5, 1.5 or 2.

A Box class

Consider the class

```
public class Box extends Object{
   private String stuff;
}
```

Constructors

Write two constructors for the class. The first constructor will take zero input arguments and set the *stuff* attribute to null. The second will take a String as input and set the *stuff* attribute to input string.

Getter and Setter

Write a public getter method called *getStuff* that returns the string stored by the *stuff* attribute. If *stuff* is null, it should return the string "no stuff in here".

Write a public setter method called *setStuff* that takes a string as input. If the *stuff* attribute is currently null, the method sets it to the input string. If *stuff* is not null, then it replaces its value with the concatenation of the input string with the character "|" and then with the current string in *stuff*.

For example,

```
Box b = new Box();  // stuff is null
b.getStuff();       // returns "no stuff in here"
b.setStuff("cat");  // stuff is "cat"
b.setStuff("dog");  // stuff is "dog|cat"
```

toString()

The *Box* class inherits a

```
public String toString(){...}
```

method from the *Object* class.

Override this method so that the return string looks like a box that is empty (when *stuff* is null) and contains the *stuff* string when it is not.

For example,

```
Box b = new Box();
System.out.println(b);
b.setStuff("kitten");
System.out.println();
System.out.println(b);
```

should display

```
*****
*   *
*****

**********
* kitten *
**********
```

equals()

The *Box* class inherits the method

```
public boolean equals(Object other){
   return this == other;
}
```

from the *Object* class. Override this method in the *Box* class so that it returns true if the strings in two *Box* objects are the same and false otherwise.

Be sure you use the *@Override* annotation when you override the method. (That is, put the line *@Override* before you redefine the method.)

```
@Override
public boolean equals(Object o){
   ...
}
```

Recall from Friday's class that you will need to *cast* the input *Object* to a *Box* in order to access the *stuff* attribute of the input object. You can assume that a user of *Box* objects will only every call *equals* with another *Box* object as input.

More

merge()

If you did not complete the constructor problem from Tutorial 3 (Money object) go back and complete this now.

Even More

Add a method to the *Box* class

```
public static Box merge(Box[] boxes){...}
```

that returns a new *Box* object whose stuff is made up of all the strings in the input boxes stuff.

For example,

```
Box b1 = new Box("a");
b1.setStuff("x");
Box b2 = new Box("yak");
Box b3 = merge( new Box[]{ b1, b2 } );
System.out.println( b3 );
```

will output

```
***********
* a|x|yak *
***********
```

split()

Add a method to the *Box* class

```
public Box[] split(){...}
```

that returns an array of new *Box* objects, consisting of some of the *stuff* in the current box.

The way the *stuff* is broken up is illustrated in the following example.

```
Box b = new Box("one");
b.setStuff("two");
b.setStuff("three");
Box[] boxes = b.split();
System.out.println(boxes[0]);
System.out.println(boxes[1]);
System.out.println(boxes[2]);
System.out.println(b);
```

will output

```
*********
* eerht *
```

```
*********
*******
* owt *
*******
*******
* eno *
*******
*****************
* three|two|one *
*****************
```

Each new box has part of the original string (each word between the pipes, |) and that word is reversed.

program

Write a program to test your code. Note that given

```
Box[] boxes = {new Box{"a"}, new Box{"b"}, new Box{"c"} };
Box b = merge( boxes );
Box[] new_boxes = b.split();
```

the *boxes* and *new_boxes* arrays will be the "same". By "same", we mean each contains the same number of *Box* objects, and for every *Box* in *boxes* there is a *Box* in *new_boxes* that is *equals()* to it. Order does not matter.

As part of your program include a method

```
/** tests of two Box arrays contain the same boxes

    input: two arrays of Box objects
    output: -Returns true if each box contains the same
             boxes (using equals()). The order of the boxes in
             the arrays does not have be the same.
            -Returns false otherwise.
*/
public static boolean same(Box[] b1, Box[] b2)
```

that tests for this notion of "same".