CS 100 Makeup Prelim
Summer 2001
Wednesday, August 1
6:00 – 7:15 p.m.

Name: _____

CUid: _____

Statement of Integrity:
I did not, and will not, break the rules of academic integrity on this exam.

_____
(Signature)

First, skim the entire test. Then carefully read all instructions before starting.

- This test is **closed-book**. No calculators, reference sheets, or other materials are allowed.
- Conciseness, clarity, and style count.
- Carefully comment each variable and major control structure.
- Do not use any MATLAB code.
- You may **not** alter, add, or remove code outside of the boxes and blanks.
- If you provide multiple answers, we will only grade one. We will pick which one. (In other words, it is to your advantage to give us just one answer!)

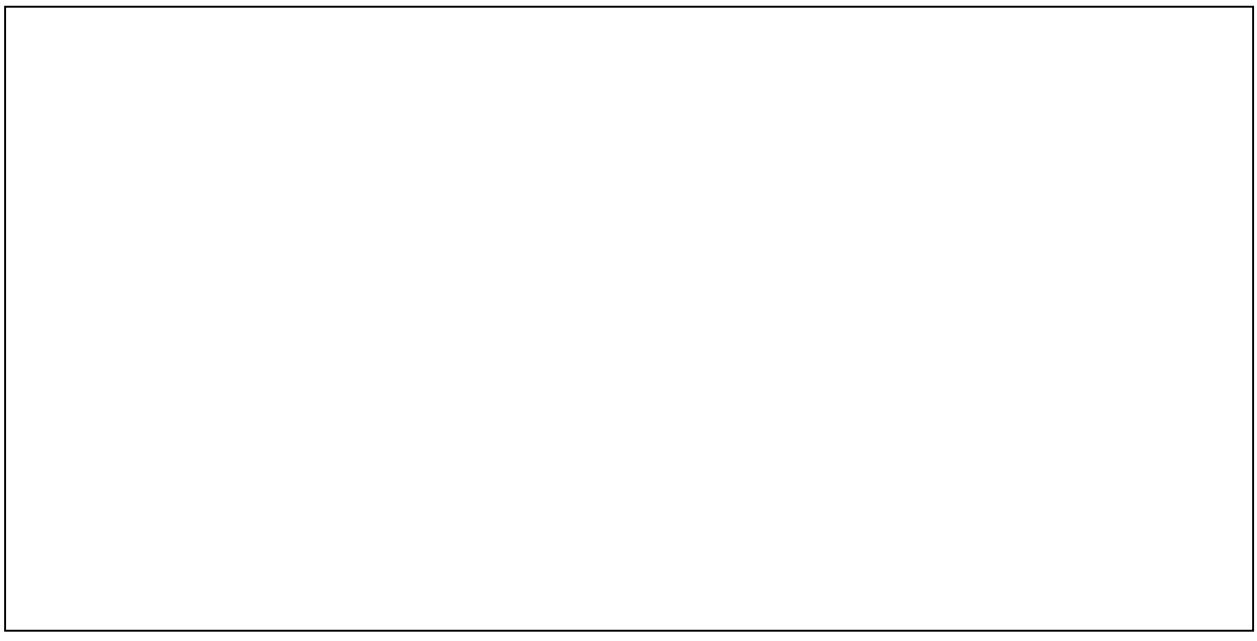
			Grader initials
Problem 1:	_____	**(10 possible)**	_____
Problem 2:	_____	**(10 possible)**	_____
Problem 3:	_____	**(5 possible)**	_____
Problem 4:	_____	**(10 possible)**	_____
Problem 5:	_____	**(15 possible)**	_____
Problem 6:	_____	**(20 possible)**	_____
Total:	_____	**(70 possible)**	_____

Problem 1: Inheritance Hierarchies (10 points)

Read through the following Java code and draw the inheritance hierarchy for the Crop, Corn, and Wheat classes. You may omit constructors, but include all other variables and methods in the proper places.

```
public class Crop {
    private int timeToGrow; //stores the time left for the crop to grow
    private String name;    //the name of the crop

    //default constructor - nothing is planted
    public Crop() {
        this.name = "dirt";
        timeToGrow = 0;
    }

    //this constructor takes in a name for the crop and sets the timeToGrow
    //to 0. It should be used for non harvesting plants, i.e. grass
    public Crop(String name) {
        this.name = name;
        timeToGrow = 0;
    }

    //this constructor takes a crop name and a harvest time
    public Crop(String name, int timeToGrow) {
        this.name = name;
        this.timeToGrow = timeToGrow;
    }

    //this method reduces the time left until the Crop is ready by one
    public void grow() {
        timeToGrow--;
    }

    //returns the amount of time left to grow until harvest
    public int timeLeft() {
        return timeToGrow;
    }
}
```

Draw your inheritance hierarchy here:



```
class Corn extends Crop {
    private int earsOfCorn; //number of ears of corn on the corn plant

    //calls the super constructor and sets the number of ears to 0
    public Corn() {
        super("corn", 7);
        earsOfCorn = 0;
    }

    //this grow method calls the parent grow method and then adds one ear of
    //corn at random to the numbers of ears in the corn plant
    public void grow() {
        super.grow();
        if (timeLeft() >= 0) {
                earsOfCorn += (int) (Math.random() * 2.0);
        }
    }

    //converts the corn object into a string to be printed
    public String toString() {
        return (earsOfCorn + " ears of corn");
    }
}

class Wheat extends Crop {
    //named constants for the 3 types of wheat
    public static final int DURUM = 0;
    public static final int HARDWHITE = 1;
    public static final int SOFTWHITE = 2;

    private int wheatType; //stores the type of wheat

    //default constructor
    public Wheat() {
        super("wheat", 3);
        wheatType = SOFTWHITE;
    }

    //this constructor takes in a type of the wheat being grown
    public Wheat(int wheatType) {
        super("wheat", 3);
        this.wheatType = wheatType;
    }

    //convert the wheat object to a String
    public String toString() {
        //return the appropriate String
        switch(wheatType) {
        case DURUM:
            return "durum wheat";
        case HARDWHITE:
            return "hard white wheat";
        case SOFTWHITE:
            return "soft white wheat";
        default:
            return super.toString();
        }
    }
}
```

Problems 2, 3, and 4: Constructors and Methods

Skim the following Field class:

```
public class Field {
    private boolean isPlanted;   //is something planted?
    private Crop cropPlanted;    //what Crop is planted in the Field?

    //default constructor: sets the Crop to null
    public Field() {
        isPlanted = false;
        cropPlanted = null;
    }

    //constructor: specifies what Crop is planted in the Field
    public Field(Crop c) {
        isPlanted = true;
        cropPlanted = c;
    }

    //method that calls the grow() method of the Crop planted in the Field
    public void water() { cropPlanted.grow(); }

    //this method allows the user to change the cropPlanted value for the
    //field. If the field is already planted, it must be harvested first,
    //before a new crop can be planted, and the method returns 1.
    //Otherwise, the crop is planted, and the appropriate variables are
    //updated accordingly.
    public int plant(Crop c) {
        if (isPlanted) return 1;
        else {
            cropPlanted = c;
            isPlanted = true;
            return 0;
        }
    }

    //returns the crop that is planted in the field
    public Crop harvest() {
        cropPlanted = null;
        isPlanted = false;
        return cropPlanted;
    }

    //this method returns an integer for four different cases of the Crop
    //which is planted in the field. If there isn't a crop planted,
    //return 3. Otherwise, if the harvestTime is 0, it is ready, return 0.
    //If harvestTime is > 0, it is not ready, return 1.
    //If the timeLeft is < 0, the Crop has rotted, return 2.
    public int harvestTime() {
        if (isPlanted) {
            int harvestTime = cropPlanted.timeLeft();
            if (harvestTime == 0) return 0; // success
            else if (harvestTime > 0) return 1; // not ready yet
            else return 2; // crop has rotted!
        }
        else return 3; // no crop to harvest
    }
}
```

Now read through the following Farm class. Then implement the indicated methods.

```
//class Farm - allows user to create a Farm object that has a square field
public class Farm
{
    private Field [][] fields; //array to store the Fields

    //default constructor for Farm object, makes 5x5 empty fields
    public Farm()
    {
        fields = new Field [5][5];
        for (int i = 0; i < fields.length; i++) {
            for (int j = 0; j < fields.length; j++) {
                fields[i][j] = new Field();
            }
        }
    }

    //Main constructor for class Farm, takes in a size argument for the
    //field size and initializes them to empty fields.
    //Note that the field is in 2 dimensions, with the number of rows and
    //the number of columns equal to the sizeOfFields parameter. It also
    //assumes the value passed in is > 0.
    public Farm(int sizeOfFields) {
        fields = new Field [sizeOfFields][sizeOfFields];
        for (int i = 0; i < fields.length; i++) {
            for (int j = 0; j < fields[i].length; j++) {
                fields[i][j] = new Field();
            }
        }
    }

    //                PROBLEM 2 (Constructors) 10 pts                //
    //---------=========implement this Constructor=========---------//
    //this constructor takes in a size but also takes in a boolean parameter
    //that determines the layout of the field. The fields are again laid out
    //in a 2 dimensional field with sizeOfFields rows and columns.
    //If isRows is true, then the fields should be laid out with alternating
    //rows of Crops: In the first row, plant Corn. In the second row,
    //plant Wheat. Repeat this until all of the rows are filled.
    //If isRows is false, plant DURUM Wheat in the whole field.
    //Note that the constructor assumes the value for sizeOfFields is > 0.
    public Farm(int sizeOfFields, boolean isRows) {









    }
```

```
    //             PROBLEM 3 (Update method) 5 pts             //
    //---------=========implement this method=========---------//
    //waters the specified field in the array of fields,
    //should check for validity of x and y coordinates
    public void waterField(int x, int y) {









    }

    //         PROBLEM 4 (Update/access method) 10 pts         //
    //---------=========implement this method=========---------//
    //this method attempts to harvest Crop planted in the Field
    //specified by the coordinates received. The coordinates should
    // be checked for validity. It should then check the
    //harvest condition of the crop and print out an appropriate
    //error message if the Crop is not ready to harvest. Also, if
    //the Crop is ready, return it, if not, return null.
    public Crop harvestField(int x, int y) {









    }


    // ASSUME THIS METHOD HAS BEEN IMPLEMENTED.
    //plants the specified field in the array of fields with the
    //supplied Crop
    //also checks for validity of x and y coordinates
    public void plantField(int x, int y, Crop c) { }

}
```

Problem 5: Testing (15 points)

Write a **main()** method for the Farm class that tests the methods you just implemented. This means that it should use the **Farm() constructor** you wrote, the **waterField()** method, and the **harvestField()** method. You may also make use of other methods from any class provided to help you test your methods.

For the Farm() constructor, you should have **at least two** tests: one that will lay out the Farm in alternating rows, and one that will make all of the Fields hold wheat.

For waterField(), you should have **at least two** tests: one that uses valid inputs, and one that uses invalid inputs.

For harvestField(), you should have **at least three** tests: one that uses valid inputs and returns a valid Crop, one that uses valid inputs but returns a null Crop, and one that uses invalid inputs.

ALL OF YOUR TESTS SHOULD BE COMMENTED.

```
    //---------=========implement this method=========---------//
    //you should test the methods you have just written here
    public static void main(String[] args) {









    }
```

Problem 6: Tracing (20 points)

Trace the following testing method by drawing the current state of the specified objects and references in the blanks between lines.

```
public static void tester() {
  Crop wheat = new Wheat();
  Field myField = new Field();
  // Draw wheat and myField.




  myField.plant(wheat);
  // Draw wheat and myField.




  myField.water();
  // Draw wheat.




  myField.water();
  // Draw wheat.




  myField.water();
  // Draw wheat.




  wheat = myField.harvest();
  // Draw wheat and myField.




}
```