Assignment 10

Samuel Godwin, Computer Science BSc, K1630575

December 18, 2016

1 Introduction

The purpose of this assignment is to create a simulation in which a harvest takes place on a farm which contains multiple fields of crops. The farm, in my case, owns one harvester and one combine harvester. The result of the profit made from the harvest is to be printed on screen. This is possible since every crop is given a value.

(Pseudocode on p2).

2 Pseudocode

Crop Class

```
CLASS 'Crop':
        Create String field 'type'
        Create int field 'value'
        METHOD 'Crop'<type, value>:
                Set field 'type' to <type>
                Set field 'value' to <value>
        END METHOD.
        METHOD 'getValue':
                Return value
        END METHOD.
END CLASS.
```

Field Class

```
CLASS 'Field':
    Create Crop[] array 'crops'
    Create static final int field 'NUMBER_OF_CROPS' = 10
    Create int field 'totalValue'
    METHOD 'Field'<type, value>:
        Go to 'plant', set <type> = type, set <value> = value
    END METHOD.
    METHOD 'plant'<type, value>:
        Add array object of length NUMBER_OF_CROPS to 'crops'
        FOR i < NUMBER_OF_CROPS; increment i:
            Set crops[i] = new 'Crop' object
        ENDFOR.
    END METHOD.
    METHOD 'harvest':
        FOR i < NUMBER_OF_CROPS; increment i:
            IF crops[i] is not null THEN:
                totalValue += 'value' in crops[i]
                Set crops[i] = null
            ENDIF.
        ENDFOR.
    END METHOD.
    METHOD 'getTotalValue':
        Return totalValue
    END METHOD.
END CLASS.
```

Harvester Class

CLASS ‘Harvester’:

    Create int field ‘fuelTankSize’

    Create int field ‘topSpeed’

    METHOD ‘Harvester’<fuelTankSize, topSpeed>:

        Set field ‘fuelTankSize’ to <fuelTankSize>

        Set field ‘topSpeed’ to <topSpeed>

    END METHOD.

    METHOD ‘getHarvestingCapacity’:

        Return fuelTankSize + topSpeed

    END METHOD.

END CLASS.

CombineHarvester Class

CLASS ‘Harvester’:

    Create int field ‘length’

    METHOD ‘CombineHarvester’<fuelTankSize, topSpeed, length>:

        Call super(fuelTankSize, topSpeed)

        Set field ‘length’ to <length>

    END METHOD.

    METHOD ‘getHarvestingCapacity’:

        Return result of ‘getHarvestingCapacity’ in ‘Harvester’ * length

    END METHOD.

END CLASS.

Farm Class

```
Import 'ArrayList'

CLASS 'Farm':
    Create ArrayList<Field> 'fields'
    Create ArrayList<Harvester> 'harvesters'
    Create int field 'profit'
    Create int field 'totalHarvestingCapacity'
    METHOD 'Farm':
        Initialise ArrayList 'fields' as new ArrayList<Field>
        Initialise ArrayList 'harvesters' as new ArrayList<Harvester>
    END METHOD.
    METHOD 'addHarvester'<harvester>:
        Add <harvester> to 'harvesters' ArrayList
    END METHOD.
    METHOD 'addField'<type, value>:
        Add new 'Field' object of <type, value> to 'fields' ArrayList
    END METHOD.
    METHOD 'addFields'<number, type, value>:
        FOR i < number; increment i:
            Add new 'Field' object of <type, value> to 'fields' ArrayList
        ENDFOR.
    END METHOD.
    METHOD 'harvest':
        Call 'calcTotalHarvestingCapacity'
        IF totalHarvestingCapacity < size of 'fields' THEN:
            FOR i < totalHarvestingCapacity; increment i:
                Call 'harvest' through element at index i in 'fields'
            ENDFOR.
        ELSE IF totalHarvestingCapacity >= size of 'fields' THEN:
            FOR every field in 'fields':
                Call 'harvest' through object in 'fields'
```

```
                ENDFOR.
        ENDIF.
        Call ‘calculateProfit’
    END METHOD.
    METHOD ‘calcTotalHarvestingCapacity’:
        Set field ‘totalHarvestingCapacity’ = 0
        FOR every harvester in ‘harvesters’:
            totalHarvestingCapacity += result of ‘getHarvestingCapacity’ in harvester
        ENDFOR.
    END METHOD.
    METHOD ‘calculateProfit’:
        FOR i < totalHarvestingCapacity; increment i:
            profit += result of ‘getTotalValue’ through element at index i in ‘fields’
        ENDFOR.
    END METHOD.
    METHOD ‘getProfit’:
        Return profit
    END METHOD.
END CLASS.
```

Harvest Class

CLASS ‘Harvest’:

METHOD ‘main’:

Create object of ‘Farm’ called ‘farm’

Go to ‘addHarvester’ in ‘farm’, set <harvester> = new object of Harvester<1, 1>

Go to ‘addHarvester’ in ‘farm’, set <harvester> = new object CombineHarvester<2, 2, 3>

Go to ‘addFields’ in ‘farm’, set <number> = 5, set <type> = “corn”, set <value> = 20

Go to ‘addFields’ in ‘farm’, set <number> = 5, set <type> = “wheat”, set <value> = 20

Go to ‘addFields’ in ‘farm’, set <number> = 5, set <type> = “oats”, set <value> = 20

Go to ‘addFields’ in ‘farm’, set <number> = 5, set <type> = “barley”, set <value> = 20

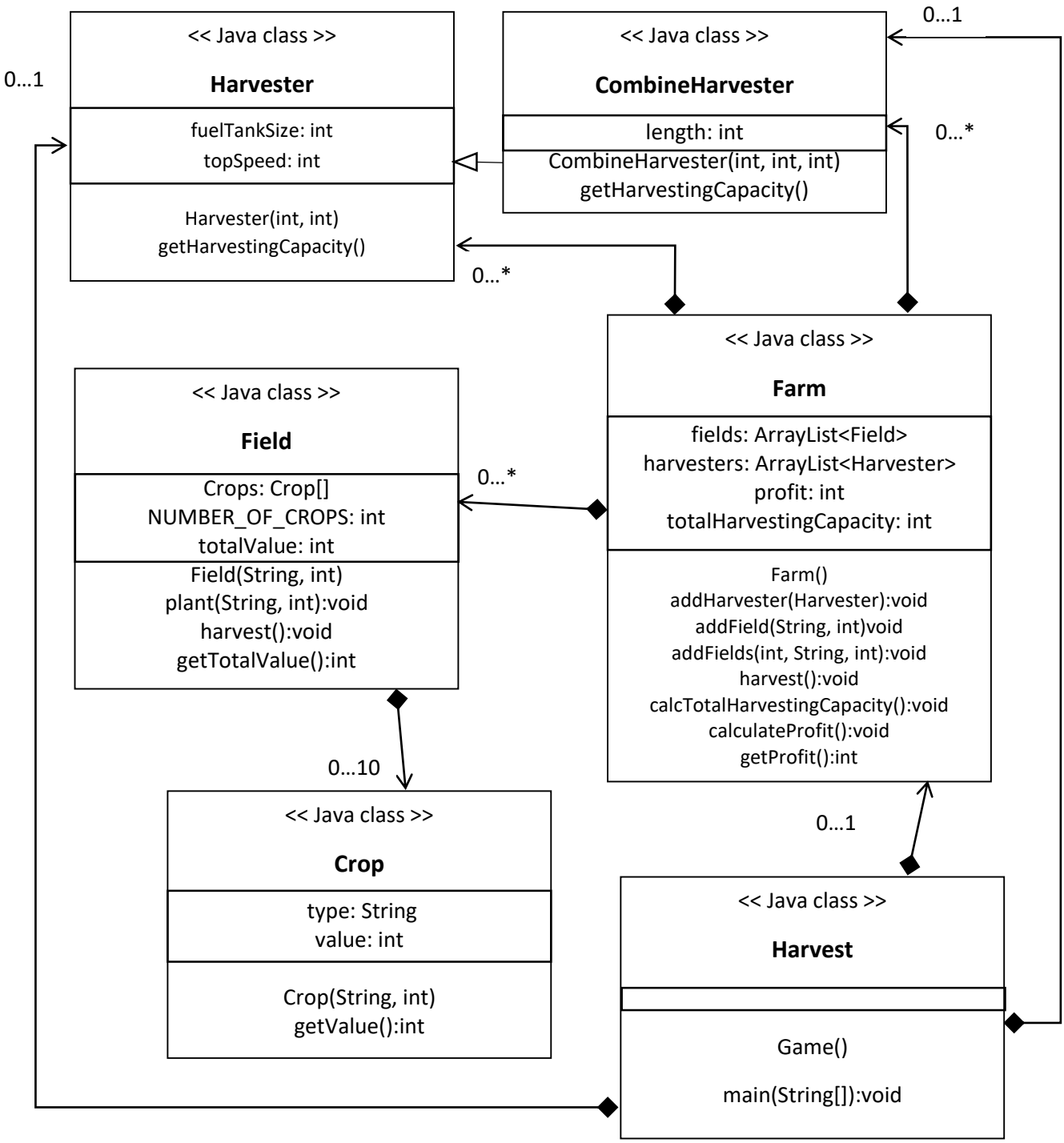
Call ‘harvest’ in ‘farm’

Print result of ‘getProfit’ through ‘farm’

END METHOD.

END CLASS.

3 Class Diagram



Note, in '0…*':

* is an unspecified value.

4 Description

My solution to this assignment involves first creating an object of my Farm class 'farm' within my driver class, 'Harvest'. I add both a harvester and a combine harvester to my farm object, which I do via the 'addHarvester' method in the Farm class. This method takes an object of type 'Harvester' as a parameter. I construct my harvester objects in the main method and then pass them to 'addHarvester' in Farm.

Upon the creation of my farm object, the constructor in my Farm class creates two ArrayList objects; 'fields' and 'harvesters'. Each of these are previously declared as variables at the top of my Farm class.

My 'addHarvester' method within my Farm class adds either a new harvester or combine harvester to my farm, each of a specified fuelTankSize and topSpeed (and also 'length', in the case 'addCombineHarvester'). This results in either a new 'Harvester' or 'CombineHarvester' being added to my 'harvesters' ArrayList for the farm.

My 'CombineHarvester' class is a subclass of my superclass 'Harvester', and so extends it. It contains an additional field 'length' for the length of a cutter which is on the front of a combine harvester. With this, the harvesting capacity of a combine harvester is calculated slightly differently, as it takes 'length' into account. The constructor in my 'CombineHarvester' class makes a 'super' call to the constructor of my 'Harvester' class, passing it a 'fuelTankSize' and 'topSpeed' – *this* constructor method sets the fields 'fuelTankSize' and 'topSpeed' to the values of 'fuelTankSize' and 'topSpeed' in 'Harvester'. Then, in 'Harvester', I have an accessor method 'getHarvestingCapacity' which returns the sum of the values of the fields 'fuelTankSize' and 'topSpeed'. Back in the constructor of my 'CombineHarvester' subclass, I multiply this 'harvestingCapacity' result by the 'length' field in order to achieve the harvesting capacity of a combine harvester. At this point, I now have a Farm object 'farm' which contains 1 harvester and 1 combine harvester.

Next, in my driver class, I add 20 Fields to the Farm: 5 of corn, 5 of wheat, 5 of oats and 5 of barley. Each crop has a value of 20. I do this via my 'addFields' method within 'Farm', which expands upon my 'addField' method (intended for adding only a single field to a farm). 'addFields' simply uses a For loop to add multiple fields to a Farm, which it does by calling 'addField' a specified number of times (also passing it the parameters 'type' and 'value'). In my 'addFields' method, the number of fields to add is passed as the first of three parameters to the method. The other two parameters to the method remain the same as for the 'addField' method. This is an element of my code which I am particularly proud of, as it saves me calling 'addField' five times later on in my driver class – I can instead call 'addFields', passing it the parameter 5. This in turn makes my code more efficient.

At this point, I call my method 'harvest' through my 'farm' object. This method, in 'Farm', first calls a private class 'calcTotalHarvestingCapacity' which I made purely for calculating the total harvesting capacity of a farm, based on its harvester(s) and combine harvester(s). This method, each time it is called, sets the field 'totalHarvestingCapacity' to the correct total harvesting capacity of a farm. Next, in 'harvest', is an if statement and if-else statement. Each of these uses a condition which compares the number of 'Field' objects in my 'fields' ArrayList to the value inside of 'totalHarvestingCapacity'. If there are more fields than the total harvesting capacity of a farm, then not all of the fields can be harvested. Otherwise, all the fields can be harvested. In the second of these two cases, the 'harvest' method within 'field' is called for every field in the 'fields' ArrayList using a for-each loop. In the case that not all fields can be harvested, however, only the amount corresponding to 'totalHarvestingCapacity' can be harvested. This is done using a normal For loop, in

which the index ‘i’ must be less than ‘totalHarvestingCapacity’ for any further iterations through the loop. In either case, this ensures that only the number of fields which corresponds exactly to the total harvesting capacity of a farm, are harvested.

In my ‘harvest’ method within my ‘Field’ class, I first build a For loop that runs only for the number of crops in a field. Since this number of crops is always 10, this class contains a static final int field ‘NUMBER_OF_CROPS’, which is equal to 10. The reason this field is also static is in case any future implementation may require it. I use this variable within my For loop condition, ensuring that the correct number of iterations are made for harvesting the number of crops in a whole field. Inside of this For loop is an if statement. This tests whether the item at the current index in the loop is null – if it is, then it has already been harvested, or the array (‘crops’) contained no ‘Crop’ objects. Therefore, if it is not, there is a crop in this position ready to be harvested, and so I do this by setting the value at the position in the array to null. First, however, I increase my ‘totalValue’ field in this class by the value of the current crop. This ensures that for every crop harvested, the ‘totalValue’ field (which increments during harvests and plays a role in keeping track of overall profit) updates accordingly.

As seen, my ‘Field’ class involves the creation of an array ‘crops’, of ‘Crop’ objects. The constructor to this class calls one of my methods, ‘plant’ and this method adds an array object to my ‘crops’ field before filling it with a new ‘Crop’ object, until the number of crops in a field reaches the maximum amount: ‘NUMBER_OF_CROPS’. A ‘Crop’ object simply consists of two fields; ‘type’ (of type String) and ‘value’ (of type int), which are set when a Crop is constructed.

Finally, in my driver class, I print the result of my ‘getProfit’ method through my farm object, which prints the profit made from previous harvests. The role of this method, in ‘Farm’, is to return the field ‘profit’. This field, however, is calculated in my ‘calculateProfit’ class, which is a private class since it is only used directly in my ‘Farm’ class. What this method does is uses a For loop which loops while the index ‘i’ is less than ‘totalHarvestingCapacity’ - which I discussed earlier. Within this loop, my ‘profit’ field increments by the result of the ‘getTotalValue’ field through each of the ‘Field’ objects in my ‘fields’ ArrayList. What this method does is simply return ‘totalValue’, which has been incrementing every harvest and which I talked about earlier. This process correctly prints the amount of profit made from the number of harvests in my program: £2800.

An element of my code which I implemented and then implemented again, to make my solution more efficient was within my ‘Farm’ and ‘Harvest’ classes. The initial implementation of my solution involved both an ‘addHarvester’ method and an ‘addCombineHarvester’ method, which each added either a new ‘Harvester’ object or ‘CombineHarvester’ object respectively, the details of which would be passed as parameters and then used in the construction of the objects. To make my code more efficient, however, I instead use a single method to add a harvester to a farm, constructing my harvester objects in the main method and then passing them to Farm. Thus, my single ‘addHarvester’ method now contains just one parameter of type ‘Harvester’, which can be either a harvester or a combine harvester, and adds this object to the ‘harvesters’ ArrayList.

An element of my program which I struggled to implement involved a simple mistake on my part; in my ‘plant’ method in ‘Field’, I had initially been creating my ‘crops’ array with the code `Crop[] crops = new Crop[NUMBER_OF_CROPS]`. This created an entirely exclusive local array ‘crops’ inside of my plant method. Since I already have a crops field created in my class, I realised I would only need the code `crops = new Crop[NUMBER_OF_CROPS]` in my ‘plant’ method, as the variable for my crops array has already been created (as a field). I therefore removed the '`Crop[]`' part of my code in my ‘plant’ method, leaving only the code `crops = new Crop[NUMBER_OF_CROPS]`. This fixed the NullPointerException I was previously getting, which

was initially extremely confusing to me, since this sort of error occurs when an array has not been properly filled – and from a pseudo code point of view, my array was being filled correctly – I simply made a mistake in terms of syntax.