1Q) When a message is sent to a class, it

(5M)

a) Looks for the corresponding methods among it's and it's superclass instance methods.
b) Looks for the corresponding methods among it's and it's superclass class methods.
c) Gives an error and halts the program at the calling method.

Ans: b

2Q) How are the state and the behavior for an object in an object oriented program defined?

(5M)

Ans:

State is defined by the values held by the instance variables of the object.
Behaviour is defined by the message protocol of the object.

3Q) Instance variables are inherited.

(5M)

True (or) False

Ans: true

4Q) Define an Abstract class? What is the utility of abstract classes in OO software design

(5M)

Ans: An abstract class is a class not intended to create instances, it is a repository for methods that are useful to two or more concrete classes, none of which is a logical subclass of the other.

5Q) Differentiate between class and instance variables? State whether each is private, public or protected in smalltalk.

(5M)

Ans: Class variable is used to store data that is useful to all instances of the class and it's subclasses.

Instance variable is a variable to store data of an instance, (that particular instance).

Usually by convention class variables start with a Capital and instance variables donot.

Both are private in smallTalk.

protected

6Q) We have a class named 'University' . The class University has a class method ,

(10M)

```
new: aString
Transcript show: aString; cr.
^ self.
```

We have the following code typed in the system workspace :-

```
| temp1  temp2  temp3 |

temp1 := University new.
temp3 := 'midterm'.
temp2 := University new: temp3.

( temp1 == temp2 )
        ifTrue: [Transcript show: ' HELLO'; cr.]
        ifFalse: [ Transcript show: 'GOODBYE'. ].
```

What is the result when this is executed?

Ans:

midterm
GOODBYE

7Q) Parse the following code fragment step by step and determine which message is executed at each step and with what arguments. (Do not compute the value)

(15M)

| temp |

temp := 5 factorial.
Transcript show: (Set new) add: temp + 3*5 sin) printString.

Ans: object and message.

First line create a temp variables. Next, the variables is assigned a value of (**5 factorial**) which is 120.

5 factorial	compute factorial 5 and assign to temp.
Set new	create an instance of set.
5 sin	computes the value of sin(5).
temp + 3	adds 3 to temp.
(temp + 3) *(5 sin)	does the multiplication
add:	adds the element to the Set.
printString	convert the set to a printable string
Transcript show:	Output the printable string to the system

8Q) Determine the expected output for the following code fragment. (10M)

```
| array1  array2  array3 |

array1 := Array new: 3 withAll: 12.
array2 := array1 copy.
array3 := array2.

array1  at: 1  put: 24.

Transcript show: 'array1—' .
Transcript show:  (array1)  printString; cr.
Transcript show:  'array2—'.
Transcript show:  (array2) printString; cr.
Transcript show:  'array3—'.
Transcript show:  (array3)  printString; cr.
```

Ans: array1--#(24 12 12)
array2--#(12 12 12)
array3--#(12 12 12)

9Q) Determine the expected output for the following code fragment. (10M)

```
| array1  array2  array3 |

array1 := Array new: 3 withAll: 12.
array2 :=  array1.
array3 := array2.

array3 at:3 put: 36.
array1 at:1 put: 24.

Transcript show: 'array1—' .
Transcript show:  (array1)  printString; cr.
Transcript show:  'array2—'.
Transcript show:  (array2) printString; cr.
Transcript show:  'array3—'.
Transcript show:  (array3)  printString; cr.
```

Ans: array1--#(24 12 36)

array2--#(24 12 36)
array3--#(24 12 36)

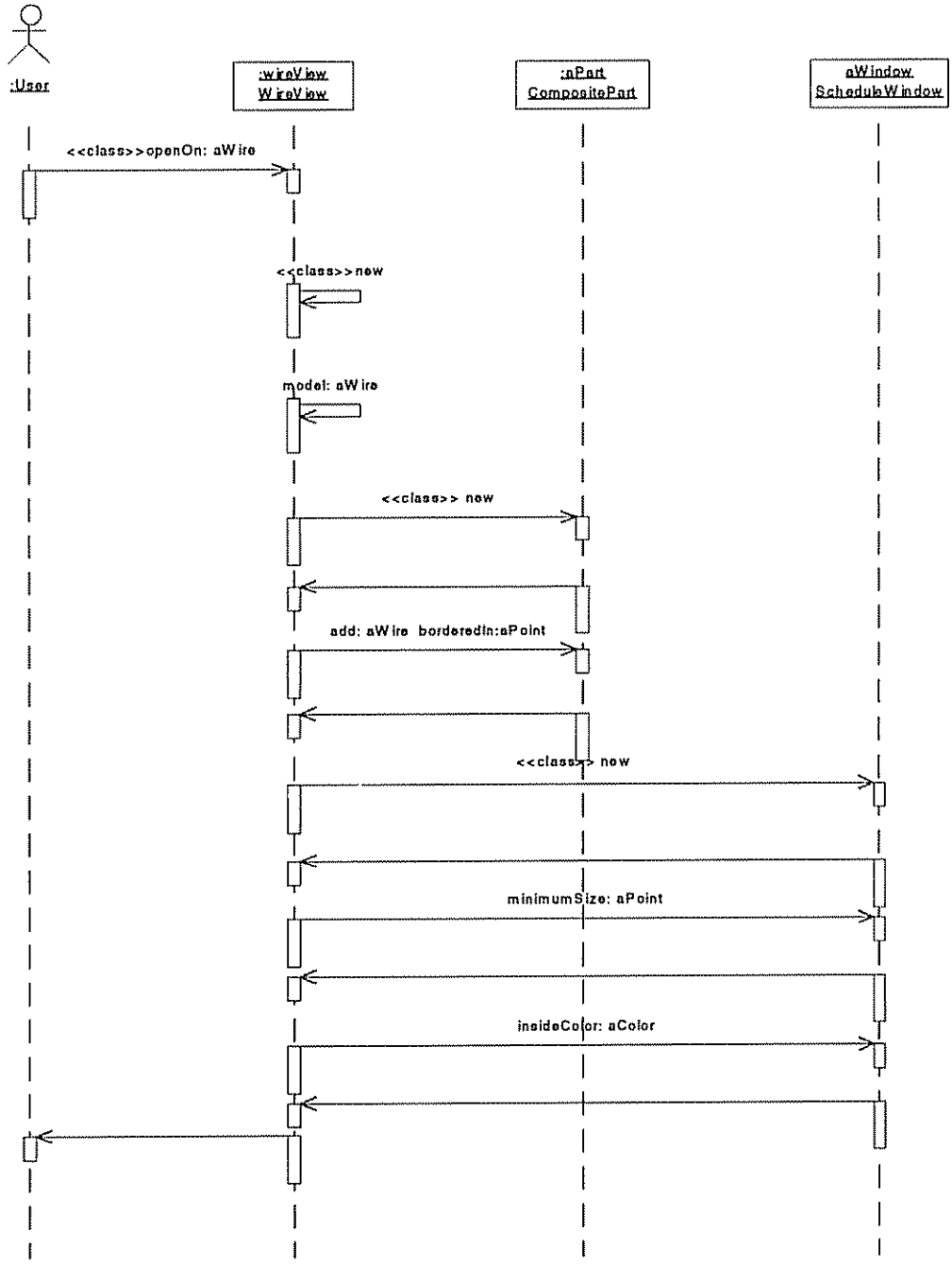
10Q)(a) The following is a class method that has been implemented for the class WireView. You are required to draw the sequence diagram corresponding to this method. Make sure that you distinguish between class methods and instance methods and include all parameters. You may assume **aPoint** and **aColor** are global variables. You may assume that all the methods return self by default.
(10M)

openOn: aWire

"Creates a new Wire View on aWire."

| wireView pinView window cPart |
wireView := WireView new.
wireview model: aWire.
cPart := CompositePart new.
cPart add: wireView borderedIn: aPoint.

window := ScheduledWindow new.
window minimumSize: aPoint.
window insideColor: aColor .


:User
:wireView
WireView
:nPart
CompositePart
aWindow
ScheduleWindow
<<class>>openOn: aWire
<<class>>new
model: aWire
<<class>> new
add: aWire borderedIn:aPoint
<<class>> new
minimumSize: aPoint
insideColor: aColor

10Q)(b) Draw a UseCase Diagram for the following problem specification. Make sure to specify the type of relation among the use cases, if there are any.

We wish to construct a wire (a multisegmented line) as an Ordered Collection of points as shown in the figure. The user needs to be able to create a wire, add a new point to the wire and remove a point from the wire. The user should also be able swap(exchange)any two points in the wire. A method which returns the length of the wire should be available to the user. Finally, the user should be able to call a routine to shorten the wire. This wire shortening method exchanges points in the wire and measures its length until the shortest configuration is found.

(10M)

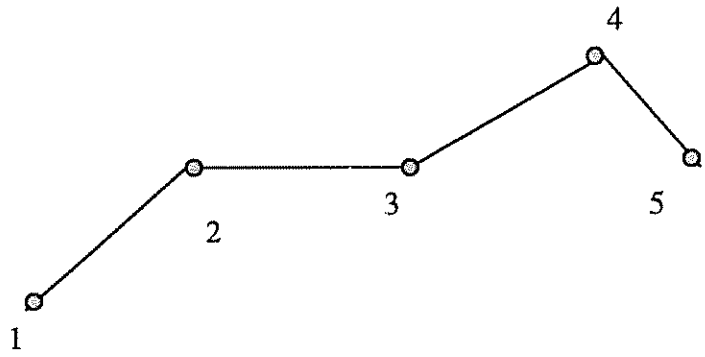
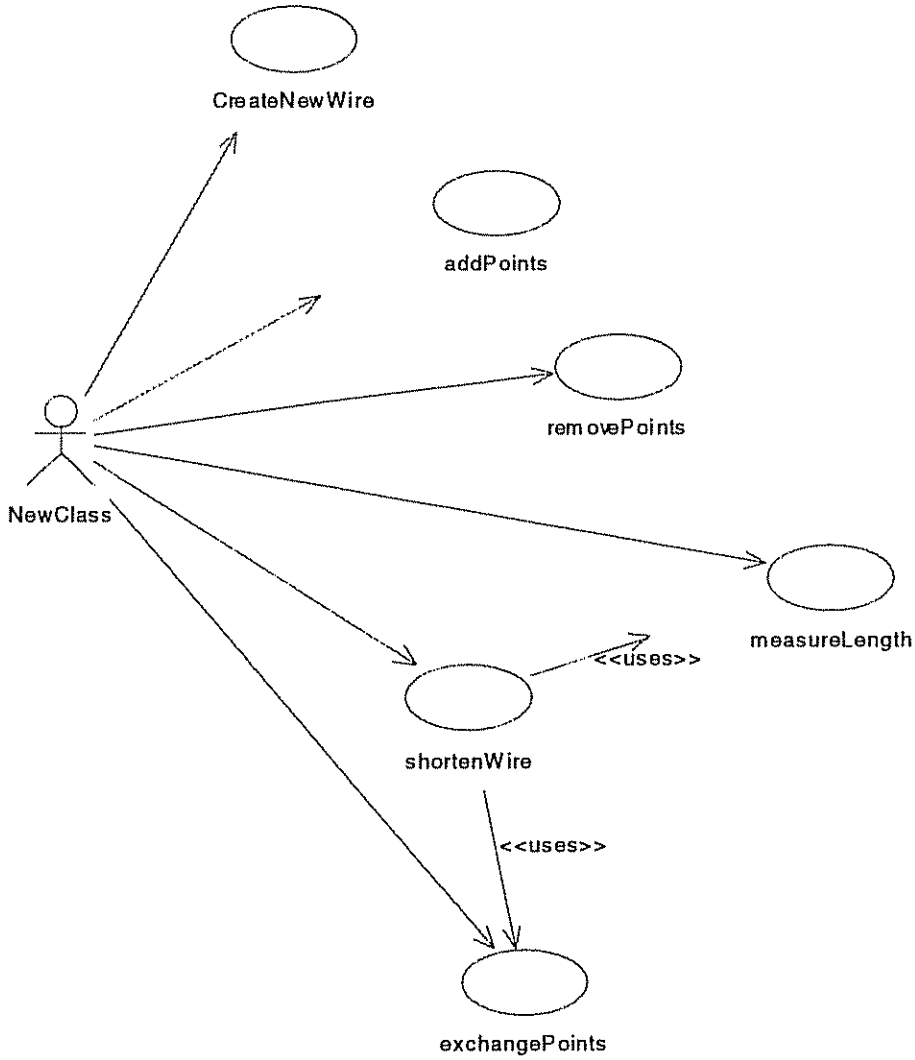


figure: wire Segment


CreateNewWire
addPoints
removePoints
NewClass
measureLength
<<uses>>
shortenWire
<<uses>>
exchangePoints

10Q)(c) The following code is a part of the implementation of a certain wire problem. The complete code implementation is not given. Draw the class diagram showing the proper hierarchy.

(10M)

Note: Italicized lines are the actual code in the method.
aWire – is an instance of the class Wire

```
OrderedCollection variableSubclass: #Wire
        instanceVariableNames: 'selectedPin view '
        classVariableNames: 'Generator '
        poolDictionaries: ''
        category: 'Wiring'

<instance methods>
        shorten
        pinNear: appoint
replace: aPoint with: newPoint
edit
 "Edit the Wire"
        WireView openOn: self.


<class methods>

        initialize
make: anInteger on: aWindow


Controller subclass: #WireController
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'Wiring'


<instance methods>

        movePin
        controlActivity
"Allows us to shorten aWire dynamically and graphically position pins in the model."

                sensor redButtonPressed
                ifTrue: [self movePin]
```

ifFalse: [aWire shorten]. " aWire is the instance of the class Wire"

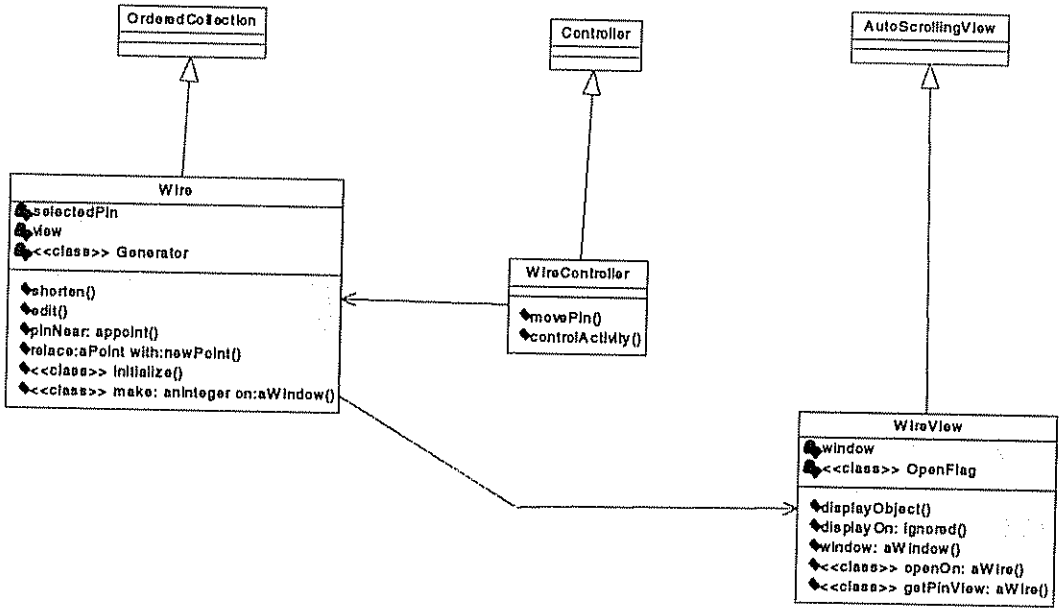
AutoScrollingView subclass: #WireView
instanceVariableNames: 'window '
classVariableNames: 'OpenFlag '
poolDictionaries: ''
category: 'Wiring'

<instance methods>

displayObject
displayOn: ignored
window: aWindow

<class methods>

openOn: aWire
getPinViewOn: aWire



```
System.out.println(e2.toString());
//Ben makes 84375.0 as the Senior Engineer
```

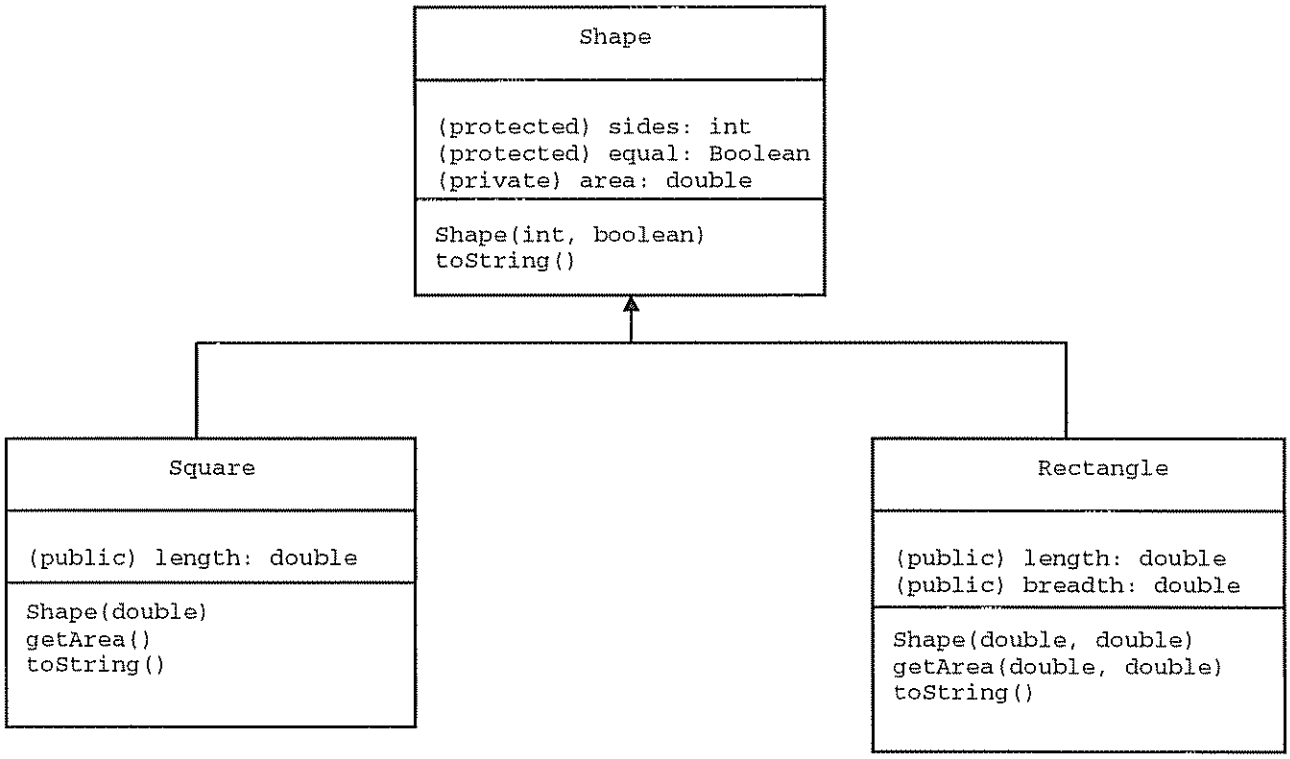
Solution: 2 files: EmployeeInterface.java, Employee.java

Pts to consider (2 pts per method)

1) Employee must *implement* EmployeeInterface
2) All the functions declared in the EmployerInterface must be defined in Employee
3) toString method must be present in the Employee class

Question 3:

(5 pts) Consider the following class diagram to answer the questions given below:



a) (1 pt) List the instance variables inherited by class `Square`:

b) (2 pts) Write the constructor function for class `Rectangle`. Give one statement to initialize all the instance variables inherited from class `Shape`.

c) (2 pts) Suppose the `toString` method of class Square is defined as follows:

```
public String toString( )
{
```

```
        String result;
        result = "Square\n" + "Number of Sides = " + this.sides;
        result += "\nAre they equal?" + equal + "\nLength: " + length;
        result += "\nArea: " + getArea();
        return result;
}
```

Now give the output of the following statements: (the `getArea` method returns a normal result for area of a square (length*length))

```
Square s = new Square(5);
System.out.println(s.toString());
```

Solutions:

a) sides, equal

b)
```
public Rectangle(double l, double b)
{
        super(4, false);
        length = l;
        breadth = b;
}
```

c)
```
Square
Number of Sides = 4
Are they equal? true
Length: 5
Area: 25
```