**King Saud University**

**College of Computer & Information Science**

**CSC111 – Lab09**

**Objectss – IIII –**

**All Sections**

**-------------------------------------------------------------------**

# Instructions

Web-CAT submission URL:

<http://10.131.240.28:8080/Web-CAT/WebObjects/Web-CAT.woa/wa/assignments/eclipse>

# Objectives:

# To apply class abstraction to develop software.

# To design programs using the object-oriented paradigm.

# To use UML graphical notation to describe classes and objects.

# To demonstrate how to define classes and create objects.

# To create objects using constructors.

# To access objects via object reference variables.

# To define a reference variable using a reference type.

# To access an object’s data and methods using the object member access operator (.).

# To define data fields of reference types and assign default values for an object’s data fields.

# To distinguish between instance and static variables and methods.

# To learn how to use static constant data members.

# To define private data fields with appropriate getter and setter methods.

# To learn when and how to define private methods.

# To encapsulate data fields to make classes easy to maintain

# To develop methods with object arguments and differentiate between primitive-type arguments and object-type arguments

# To use the keyword *this* to refer to the calling object itself.

# To combine logic (conditionals and loops) with objects.

# To learn that most of program logic in OOP goes inside classes.

# To learn how to write private helper methods.

# To learn how a method of an object calls another method of the same object.

# Lab Exercise 1

Design a class named **Ball** class that models a moving ball. The class contains:

* Two properties **x, y** which maintain the position of the ball in a two dimensional space.
* A default constructor that sets position to (0, 0).
* A constructor that receives two parameters x and y that represent current position of the ball.
* Methods **getX** and **getY** that return the current position of the ball.
* A method **move**, which changes x and y by the given **xDisp** and **yDisp**, respectively. The ball must not move if it is going to finish its movement in a position occupied by another ball (Hint: use **lasX** and **lastY** to determine this. If movement is allowed then change the values of **lastX** and **lastY**).
* Two properties **distTraveledX** and **distTraveledY** that keep the total distance traveled by current ball throughout all of its moves on both x-axis and y-axis.
* Two methods **getDistTraveledX** and **getDistTraveledY** that return the distance traveled by the current ball throughout all of its moves on both x-axis and y-axis.
* Two static properties **totDistXAllBalls** and **totDistYAllBalls** that keep the total distance traveled by all balls throughout all of the moves on both x-axis and y-axis.
* Two static methods **getTotDistXAllBalls** and **getTotDistYAllBalls** that return the distance traveled by all balls throughout all of the moves on both x-axis and y-axis.
* Static properties **lastX** and **lastY** which store the last position of the most recent ball that has moved.
* A method **toString**, which returns the string "**Ball @ (x,y)**".

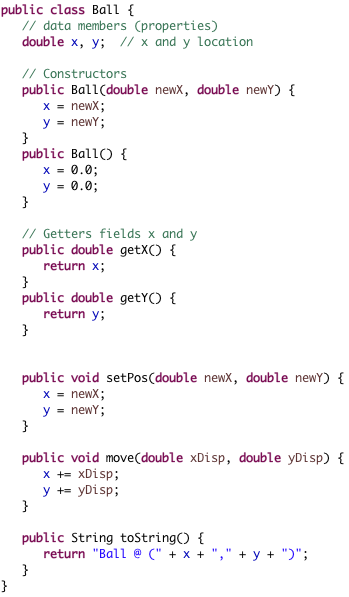
Start by drawing the UML for the class Ball. Then write a program that does the following:

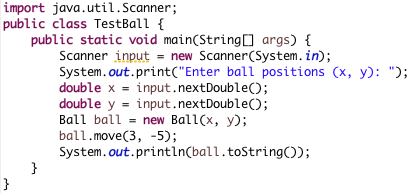
* It creates a new ball with a position (2, 2).
* Then it moves the ball by (3, -2).
* Then it moves the ball by (2, -7).
* Then it creates a new ball with a position (0,0).
* Then it moves the ball by (5, 5). Notice that the second ball must not move since it will otherwise occupy the same place occupied by the first ball.
* Then it moves the ball by (2, 4).
* Finally, it prints the last position of the two balls using **toString** method and the total distance traveled on both x-axis and y-axis by each ball and by all balls.

# Solution

|  |
| --- |
| Ball |
| - x: double  - y: double  - distTraveledX: double  - distTraveledY: double  - totDistXAllBalls: double  - totDistXAllBalls: double  - lastX: double  - lastY: double |
| + Ball()  + Ball(newX: double, newY: double)  + getX(): double  + getY(): double  + move(xDisp: double, yDisp: double): void  + getDistTraveledX (): double  + getDistTraveledY(): double  + getTotDistXAllBalls (): double  + getTotDistYAllBalls (): double  + toString(): String |

|  |
| --- |
| TestBall |
|  |
| main(): void |





# Lab Exercise 2

You have been asked by Saudi Wildlife Authority to design a class named **Species** to manage endangered species. Class details are as following:

* Three data fields (properties) which are:
  + **name** of type **String** which stores name of species,
  + **population** of type **int** which stores the current population of the species and it can not be negative, and
  + **growthRate** of type **double** which stores the growth rate of the current population.
* A method **readInput()** that reads the values of data members. The method must not allow the user to enter incorrect population value. If this happens, it should keep asking for correct value until the user enters it.
* A method **writeOutput()** that prints the data of the species.
* A method **predictPopulation(int years)** that returns the projected population of the species after the specified number of years (which must be a non-negative number).
* A method **setSpecies(String newName, int newPopulation, double newGrowthRate)** that sets values of receiving object to new data sent through parameters. The method should print an error message and exit the whole program if **newPopulation** is a negative number.
* Getter methods for **name, population** and **growthRate**.
* A method **equals(Species otherObject)** that compares this object to **otherObject**. Two species objects are equal if they have the same name ignoring the letter case.
* A method **isPopulationLargerThan(Species otherSpecies)** that returns true if population of this object is greater than the population of **otherSpecies**.
* A method **isExtinct()** that returns true if the population of this object is 0, otherwise returns false.

# Draw the UML diagram for the class and then implement the class. Write a test program that checks if the population of an input species chosen by the user could one day exceed that of Arabian Oryx.

* The program first reads the information of some species X.
* Then it checks if species X is extinct and if so it prints the message ”The species that you entered is extinct” and exits.
* If species X is not extinct, then it checks if, given the current population and growth rate of species X, its population will surpass that of Arabian Oryx, given that the current population of Arabian Oryx is 1000 and its growth rate is 0.25. If this could happen then it prints after how many years this will happen.
* If user enters Arabian Oryx then you should keep asking him to enter a different species.

Name your classes **Species** and **TestSpecies**. Use two separate files for each of the two classes.

**Sample Run 1**

What is the species' name?

Dinosaur **↵**

What is the population of the species?

0 **↵**

Enter growth rate (% increase per year):

0 **↵**

The species that you entered is extinct.

**Sample Run 2**

What is the species' name?

Houbara Bustard **↵**

What is the population of the species?

900 **↵**

Enter growth rate (% increase per year):

0.2 **↵**

Species Houbara Bustard has slower growth rate than Arabian Oryx.

**Sample Run 3**

What is the species' name?

African Lion **↵**

What is the population of the species?

16500 **↵**

Enter growth rate (% increase per year):

0.05 **↵**

Population of species Lion is already larger than population of Arabian Oryx.

**Sample Run 4**

What is the species' name?

Arabian Oryx **↵**

What is the population of the species?

1000 **↵**

Enter growth rate (% increase per year):

0.25 **↵**

You entered the Arabian Oryx species, please enter another one:

What is the species' name?

Arabian Oryx **↵**

What is the population of the species?

1000 **↵**

Enter growth rate (% increase per year):

0.25 **↵**

You entered the Arabian Oryx species, please enter another one:

What is the species' name?

Blue Whale **↵**

What is the population of the species?

500 **↵**

Enter growth rate (% increase per year):

0.38 **↵**

After 535 years, population of Blue Whale will surpass that of Arabian Oryx.

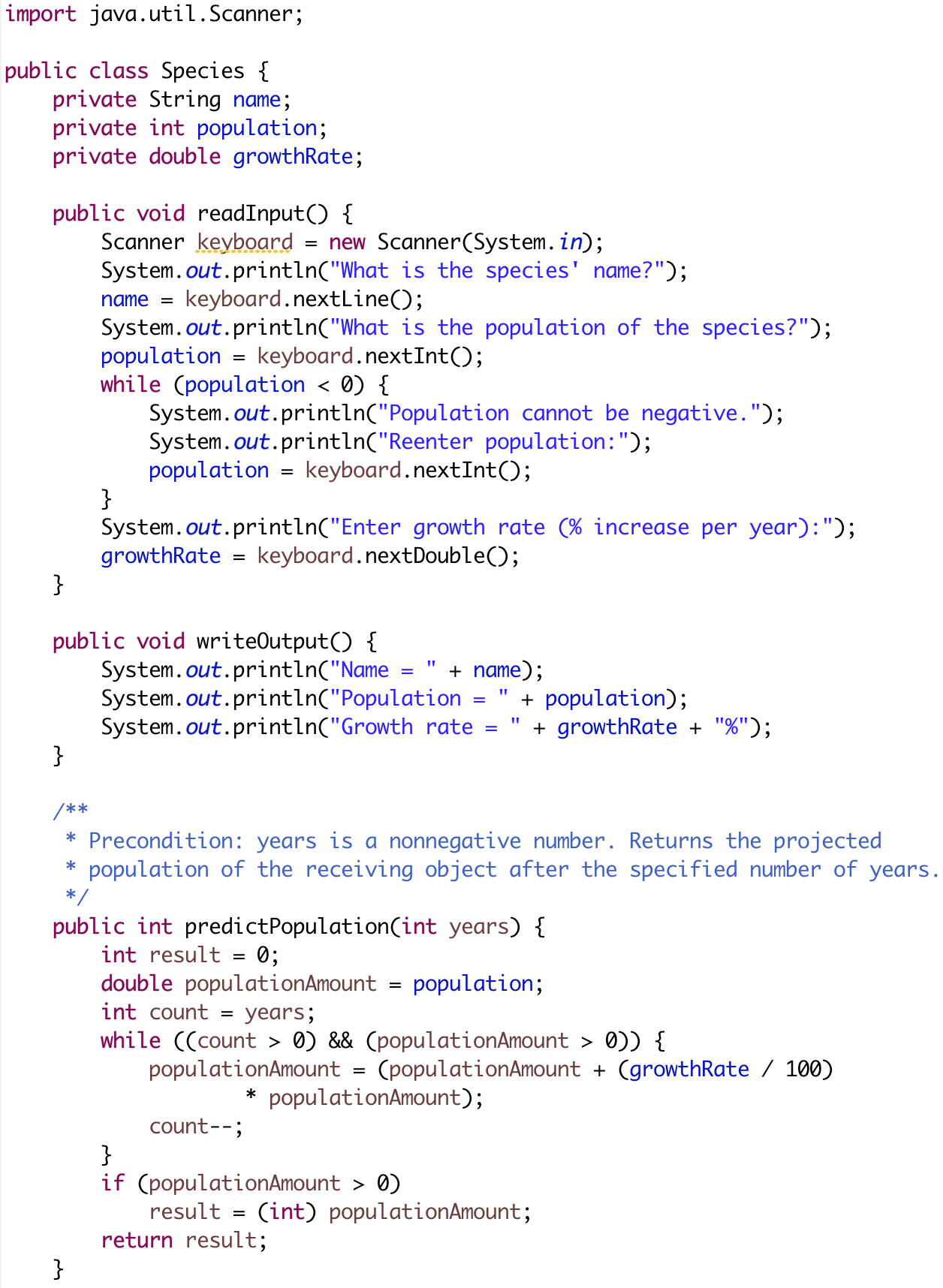
# Solution

1. First phase is to design your program as an OOP program. Draw UML diagrams for the two classes, **Species** and **TestSpecies**.

|  |
| --- |
| TestSpecies |
|  |
| + main(): void |

|  |
| --- |
| Species |
| - name: String  - population: int  - growthRate: double |
| + readInput(): void  + wrtieOutput(): void  + predictPopulation(years: int): int  + setSpecies(newName: String, newPopulation: int, newGrowthRate: double): void  + getName(): String  + getPopulation(): int  + getGrowthRate(): double  + equals(otherObject: Species): boolean  + isPopulationLargerThan(otherSpecies: Species): boolean  + isExtinct(): boolean |

1. Use previously created project **lab08**
2. Unlike in previous exercise, we will create two separate files for the two classes. Create a new class and name it **Species**.
3. Create a new class and name it **TestSpecies**. Make sure you choose the public static void main option.
4. Write the two program classes as shown in next pages (you can ignore comments)
5. When you are done, save your program and run it. Make sure it prints the output as shown above.
6. Submit your program to WebCAT through. Ask your TA for help.



Continue code of this class on next page



Class with main is on next page



# Lab Exercise 3

***(This exercise is extra and student should complete it at home)***

We would like you to program a simple game called Oracle. The oracle pretends that it knows the answer to any question. It starts by letting the user ask a question. Then, it asks the user for an advice on how to answer his question. Finally, it gives the user an answer to his question based on the advice he gave to previous question. Let me give you a sample run and then explain the program in detail.

**Sample Run**

I am the oracle. I will answer any one-line question.

What is your question?

What time is it? **↵**

Hmm, I need some help on that.

Please give me one line of advice.

Seek and you shall find the answer **↵**

Thank you. That helped a lot.

You asked the question:

What time is it?

Now, here is my answer:

The answer is in your heart.

Do you wish to ask another question?

yes **↵**

What is your question?

Am I gonna pass this course :(? **↵**

Hmm, I need some help on that.

Please give me one line of advice.

Ask the professor **↵**

Thank you. That helped a lot.

You asked the question:

Am I gonna pass this course :(?

Now, here is my answer:

Seek and you shall find the answer

Do you wish to ask another question?

No **↵**

The oracle will now rest.

Design a class called **Oracle** that has the following members:

* Three instance data fields (we will see how to use these when we explain the method members):
  + **oldAnswer** of type **String** which keeps the old answer given by the user.
  + **newAnswer** of type **String** where we store new answer given by the user.
  + **question** of type **String** where we store the question given by the user
* Four static data fields:
  + **WELCOME\_MSG** of type **String** that stores the intro string given to user. The message is **“I am the oracle. I will answer any one-line question.”**
  + **ADVICE\_SEEKING\_MSG** of type **String**, which stores the advice-seeking message given to user after each question, he asks. The message is **“Hmm, I need some help on that.\nPlease give me one line of advice.”**.
  + **THANKS\_YOU\_MSG** of type **String** that stores the thank you message given to the user after he gives his advice. The value is **“Thank you. That helped a lot.”**.
  + **GOODBYE\_MSG** of type **String** that stores the goodbye message given to the user at the end of the program. The value is **“The oracle will now rest.”**.
* A method **chat()** which is the main method that runs the game logic. In this method, we start by giving the welcome message then we repeat the following tasks: (1) ask the user for his new question, (2) do the game trick to answer the question and then (3) check if the user wants to ask another question (i.e., repeat the game) quit. To ask, process and answer a new question (i.e., (1) and (2)), Method **chat()** delegates this task to method **answer()**. Finally, the method prints the goodbye message.
* A method **answer()** that receives the user’s question and tries to answer it. Method **answer()** first asks the user to enter his question then calls method **seekAdvice()** to get an advice from the user on how to answer the question. After that it uses the old advice (i.e., old answer) given by the user to previous question to answer the current question. At the first run, old answer is initialized to the string **“The answer is in your heart.”**. After that it calls the method **update()** to update the old answer with the advice given by the user when method **seekAdvice()** was called.
* A method **seekAdvice()** that asks the user for an advice on how to answer his question. It then stores this advice as the new answer and prints the thank you message. This advice/answer will be used later on to answer the user’s next question. You notice that this will achieve the game trick which is to answer the user’s question using his previous answer.
* A method **update()** which stores the advice given by the user as the old answer to be used later on to answer the next question.

Write a program called **OracleDemo** that runs the Oracle game by creating an object of type **Oracle** and then calling the **chat()** method. Notice that the program does not need to call any of the methods except method **chat()**. This means that, except for method **chat()**, all other methods should be **private** since they are called internally only. This is a clear example that shows how, in Object Oriented Programming, most of the program logic is implemented in the classes not the main method.

# Solution

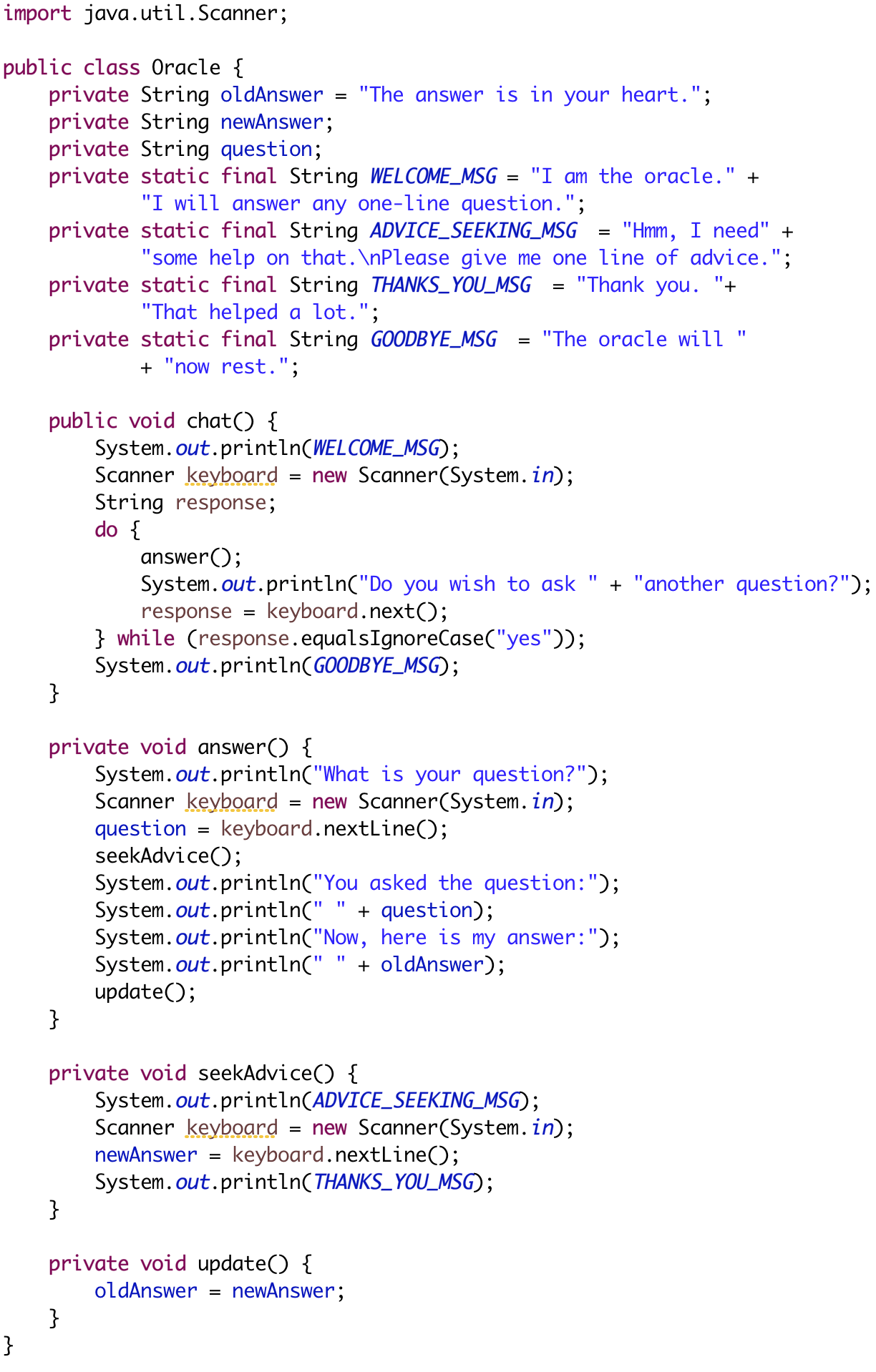
1. First phase is to design your program as an OOP program. Draw UML diagrams for the two classes, **Oracle** and **OracleDemo**.

|  |
| --- |
| Oracle |
| - oldAnswer: String  - newAnswer: String  - question: String  - WELCOME\_MSG: String  - ADVICE\_SEEKING\_MSG: String  - THANK\_YOU\_MSG: String  - GOODBYE\_MSG: String |
| + chat(): void  - answer(): void  - seekAdvice(): void  - update(): void |

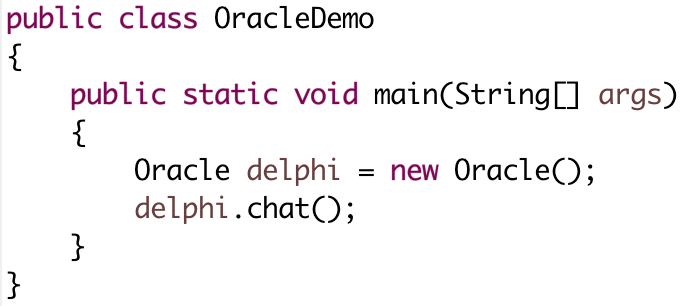
1. Use previously created eclipse project **lab08**

|  |
| --- |
| OracleDemo |
|  |
| + main(): void |

1. Similar to previous exercise, we will create two separate files for the two classes. Create a new class and name it **Oracle**.
2. Create a new class and name it **OracleDemo**. Make sure you choose the public static void main option.
3. Write the two program classes as shown in next pages (you can ignore comments)
4. When you are done, save your program and run it. Make sure it prints the output as shown above.
5. Submit your program to WebCAT through. Ask your TA for help.



Class with main is on next page

****

**Done…**