King Saud University
College of Computer & Information Science
CSC111 – Lab09
Objects – IV –
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

Part 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.
- 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.
- 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.
- 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.

Start by drawing the UML for the class Ball.

```
public class Ball {
     // data members
     // define instance variables x, y
     /*        */
     //distTraveledX,distTraveledY
     /*        */
     /* modifier datatype  variable name*/
     /* modifier datatype  variable name*/
     /* modifier datatype  variable name*/
     /* modifier datatype  variable name*/

     // Getter for x getX()
```

```
      /*          */


      //Getter for y getY()
      /*          */


   // method move that moves x to the xDisp and y to yDisp
   /*modifier */ /* returntype */ move(/* parameter xDisp
*/, /* parameter yDisp*/) {
      // add xDisp to x and yDisp to y
      /*          */
      distTraveledX += /* the difference between the old
position and the new position */
      // update distTraveledY
      /*          */
   }

}
```

Part 2

Add the following constructors to the class:

- A default constructor that sets position to (0, 0). **<u>This constructor should call the next one</u>**.
- A constructor that receives two parameters x and y that represent current position of the ball.

```
public class Ball {
   // data members
      /*          */

   public Ball() {
      // set x and y to zeros
      /*          */
   }
   public Ball(double newX, double newY) {
       //set x to the newX
      /*          */
       //set y to the newY
```

```
      /*          */
   }



    // Getter fro x getX()
     /*          */

    //Getter fro y getY()
     /*          */

    // move() function
     /*          */
}
```

Part 3

In this part, you will add static members which will help you to keep track of the position and distance travelled by all the balls. Add the following static properties and methods:

- 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.
- Static properties **`lastX`** and **`lastY`** which store the last position of the most recent ball that has moved.
- 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.

```
public class Ball {
   // data members
   // non-static data members
   /*        */
   // define static variables totDistXAllBalls
   // , totDistYAllBalls, lastX and lastY
   /* modifier↑static datatype  variable name*/
   /* modifier↑static datatype  variable name*/
   /* modifier↑static datatype  variable name*/
   /* modifier↑static datatype  variable name*/
```

```
    public Ball() {
        // set x and y to zeros
      /*          */
    }
    public Ball(double newX, double newY) {
       //set x to the newX
      /*          */
       //set y to the newY
      /*          */
    }

    // Getter fro x getX()
      /*          */


    //Getter fro y getY()
      /*          */

    // Static getter getTotDistXAllBalls()
      /*          */
    // Static getter getTotDistYAllBalls()
      /*          */

    // move() function
      /*          */

}
```

Part 4

In this part you will define a new method to print the information of the ball and modify the move method:

- A method `toString`, which returns the string "`Ball @ (x,y)`".
- In the move method, modify the method so that a ball must not move if it is going to finish its movement in a position occupied by last moving ball (Hint: use static members `lastX` and `lastY` to

determine this. If movement is allowed, then change the values of `lastX` and `lastY`).

After that 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.

```
import  java.util.Scanner;

public class TestBall {
   public static void main(String[] args) {
   // create a scanner
   /*          */
   System.out.print("Enter ball positions (x, y): ");
   // read x
   /*          */
   // read y
   /*          */
   // create a new ball with postion x=2 and y=2
   /*          */
   // move the ball by x=3 & y=-2
   /*          */
   // move the ball by x=2 & y=-7
   /*          */
   // create a new ball with a position(0, 0)
   /*          */
   // move it by x=5 & y=5
```

```
    /*          */

    System.out.println(/* print the two balls' info using
     the toString() */);
    }
}
```

Lab Exercise 2

You have been asked by Saudi Wildlife Authority to design a class named `Species` to manage endangered species.

Part 1

Add the following requirements to the :

- 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.
- Two constructors one with no arguments and one with new arguments to be assigned. In the one with no arguments assign all the properties to their default values (i.e. integers to 0, doubles to 0.0 and Strings to "") and **call the other constructor to do this**.
- 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.

```
public class Species {
   /* define the three data members name
      , population and growthRate */

   public Species() {
      // assign each property to its default value
       /*       */
   }
   public Species(/* name, population and growthRate */)
   {
      // assign each property to its corresponding value
       /*       */
   }
   public void readInput() {
      // create a new Scanner
       /*       */
      /* print a message prompting the user to enter
         the values: name, population and growthRate */
      name = keyboard.nextLine();
      // read population from the user
       /*       */
      while(population < 0) {
      /* keep asking and reading the population
         from the user and tell him that his input
         for the population is invalid untill he enters
         a valid input*/
      }
      // read growthRate from the user
       /*         */
   }
}
```

Sample Run for method readInput

```
What is the species' name?
Dinosaur ↲
What is the population of the species?
-200 ↲
You entered an invalid population, please renter the
population of the species again?
200 ↲
Enter the growth rate (% increase per year):
0 ↲
```

Part 2

Now you have a class called species Add the following method to the previous class:

- 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).
- Getter methods for `name`, `population` and `growthRate`.
- 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.

```
public class Species {
   /* define the three data members name
      , population and growthRate */
   public Species() {
      // assign each property to its default value
       /*          */
   }
   public Species(/* name, population and growthRate */)
   {
      // assign each property to its corresponding value
       /*          */
   }

   // readInput() function
       /*          */

   public /* return type */ writeOutput() {
      // return a summary of the species info
```

```
        /*          */
    }

   public /* return type */ predictPopulation(int years)
   {
       // check the value to make sure that it is not a

        // negative number If not return the population
       // after "years"
        /*          */
    }

    // getter methods for name, population and growthRate
        /*          */

    //setter for the name
    public /* return type */ setSpecies(String newName,
    int newPopulation, double newGrowthRate) {
       // assign each property to its new corresponding
        // value
       // if the newPopulation is print error and exit
       // the program
        /*          */
    }
}
```

Part 3

- 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.

```
public class Species {
   /* define the three data members name
      , population and growthRate */
   public Species() {
      // assign each property to its default value
          /*         */
   }
   public Species(/* name, population and growthRate */) {
      // assign each property to its corresponding value
          /*         */
   }

   // readInput() function
          /*         */

   // writeOutput() function
          /*         */

   // predictPopulation() function
          /*         */

   // getter methods for name, population and growthRate
          /*         */

  // setSpecies() function
          /*         */

   public boolean equals(Species otherObject) {
        return (this.name.equals(otherObject.name) && /*
          population comparison&&growthRate comparison */)
  }

   public /* return type */ isPopulationLargerThan(Species
       otherSpecies) {
      /* return true if population of this object is longer
       than the population of the otherObject's population
       or false otherwise */
      }

      public /* return type */ isExtinct() {
      // return true if population is less than 0 or false
      // otherwise
          /*         */
   }
}
```

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.
- Check if the population of specie X is surpasses that of Arabian Oryx print after how many years this will happen.
- check if the population of specie X prediction is less than

Name your test class **TestSpecies**. Use a new separate file for the test class.

```
public class TestSpecies {
   public static void main(String[] args){
     Species speciesX = new Species();
     // read inputs using readInputs() method
     /* check if species X is extinct and if so print the
     message "The species that you entered is extinct" and
     exits */
```

```
      // create a new species call it arabianOryx
      /* set its properties using the setSpecies method for
      arabianOryx
      to name: arabian Oryx, population: 1000 and
      growthRate: 0.25 */
      while(/* species X equals arabianOryx*/) {
          /* print a message saying that the values the user
          entered is equal to arabianOryx's values and ask
          them to re-enter it again */
      }
      if(/* speciesX poplation is larger than arabianOryx's
         */){
         // print a message indicating that specie
         //  population is larger
           /*         */
      }else{
         if(speciesX.getGrowthRate() <= arabianOryx) {
                  /* print a message saying that speciesX
                  has a lower growthRate than arabianOryx
                  growthRate */
               }else{
              int years = 1;
              while(/* while speciesX prediction is less
                    thanarbianOryx prediction*/) {
                /* increase the number of "years" i.e. the
                 argument of the function predictPopulation*/

                 //print how many years it will take species
                 //x to surpass Arbian Oryx
                /*         */
              }
           }
         }

   }
}
```

Sample Run

```
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.
```

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.

```
import java.util.Scanner

public Oracle{
   //data fields
   private String oldAnswer = "The answer is in your
heart";
   // intilize the other two Strigns newAnswer and question

   private static final String WELCOME_MSG = "I am the
oracle. " +
   "I will answer any one-line question. ";
   // intilize the other three static final Strings
   // THANKS_SEEKING_MSG, THANK_YOU_MSG and GOODBYE_MSG
     /*         */
```

```
/* modifier */ /* return type */ chat() {
// print out the welcome message
  /*         */
// intilize a new scanner "keyboard"
  /*         */
String = response;
do {
   // call answer method
  /*         */
   // ask the user for if they wish to ask a new
   // question
  /*         */
   // read the question and store it in response
  /*         */
}while(/* as long as response is equal to yes */);
// print out GOODBYE_MSG
  /*         */
}

/* modifier */ /* return type */ answer() {
// prompt the user for the question
  /*         */
// intilize a new scanner "keyboard"
  /*         */
question = keyboard.nextLine();
// call seekAdvice()
  /*         */
// print out the question in the given sample format
  /*         */
// call the function update
  /*         */
}
```

```
   /* modifier */ /* return type */ seekAdvice() {
      // print out ADVICE_SEEKING_MSG
     /*          */
      // intilize a new scanner "keyboard"
     /*          */
      newAnswer = /* read from keyboard */
      // print out THANK_YOU_MSG
     /*          */
   }

   /* modifier */ /* return type */ update() {
     // assign old value to newAnswer
     /*          */
   }
}
class OracleDemo{
  public static void main(String[] args){
     // create a new Orcle object
     /*          */
     // call chat function
     /*          */
   }
}
```