**Question 1**

1a)

The Circle class provides nine specific methods. They consist of getter and setter methods for each of the four class instance variables and a @override toString()method.

1b) i)

private Circle body; // Circles body

private Circle tail; // Circles tail

1b) ii)

/\*\*

\* sets the colour of head, body and tail all to the argument colour.

\*/

private void setColour(OUColour colour)

{

head.setColour(colour); // provided

body.setColour(colour);

tail.setColour(colour);

}

1b) iii)

/\*\*

\* Constructor for objects of class Snake

\*/

public Snake(int startX, int startY, Circle aHead, Circle aBody, Circle aTail)

{

// initialise instance variables

head = aHead; // provided

head.setXPos(startX); // provided

head.setYPos(startY); // provided

head.setDiameter(10); // provided

body = aBody;

body.setXPos(startX);

body.setYPos(startY);

body.setDiameter(8);

tail = aTail;

tail.setXPos(startX);

tail.setYPos(startY);

tail.setDiameter(6);

this.setColour(OUColour.RED);

}

1c)

/\*\*

\* helper method to return y position of head

\*/

private int getHeadYPos()

{

return head.getYPos();

}

/\*\*

\* helper method to return x position of body

\*/

private int getBodyXPos()

{

return body.getXPos();

}

/\*\*

\* helper method to return y position of body

\*/

private int getbodyYPos()

{

return body.getYPos();

}

1d) i)

The condition (anXPos >= 0) tests to see if the received argument x-coordinate is at the minimum (zero) or greater than zero position. The result of this determines if the head position is still in bounds on the left side of the shapes window.

The condition (anXPos <= (250 - head.getDiameter())) tests to see if the received argument x-coordinate is at the maximum or less than maximum position. The maximum position available will be affected by the diameter of the snake’s head, this value is subtracted from the maximum x-coordinate value allowable of the shapes window. The result of this determines if the head position is still in bounds on the right side of the shapes window.

With the joining of the conditions using the logical AND operator (&&), both the operands have to be true for the if statement to true; in other words, the position of the x-coordinate has to be in the bounds of the of the shapes window. If the first operand is false, then the second operand would not be evaluated because the condition of true would not happen.

1d) ii)

/\*\*

\* return true if the argument represents a valid y-position for head

\*/

private boolean checkY(int anYPos)

{

return ((anYPos >= 0) && (anYPos <= (350 - head.getDiameter())));

}

1e)

/\*\*

\* calculate new position of the head, one head diameter to the

\* right of current position and move if it is possible

\*/

public void right()

{

this.moveTo(this.getHeadXPos() + head.getDiameter(), this.getHeadYPos());

}

/\*\*

\* calculate new position of the head, one head diameter to the

\* left of current position and move if it is possible

\*/

public void left()

{

this.moveTo(this.getHeadXPos() - head.getDiameter(), this.getHeadYPos());

}

/\*\*

\* calculate new position of the head, one head diameter up

\* of current position and move if it is possible

\*/

public void up()

{

this.moveTo(this.getHeadXPos(), this.getHeadYPos() - head.getDiameter());

}

/\*\*

\* calculate new position of the head, one head diameter down

\* of current position and if it is possible

\*/

public void down()

{

this.moveTo(this.getHeadXPos(), this.getHeadYPos() + head.getDiameter());

}

1f)

/\*\*

\* makes the snake move 100 random steps, each of

\* which is randomly up, down, left or right

\*/

public void randomWalk()

{

int number; // Stores current generated random number

for(int step = 0; step < 100; step++)

{

number = randomInteger();

if(number == 1)

{

this.right();

}

else if(number == 2)

{

this.left();

}

else if(number == 3)

{

this.up();

}

else // number is 4

{

this.down();

}

}

}

1g)

/\*\*

\* moves the snake horizontally one step at a time until the x position

\* of the head is within one head diameter of 125, and then vertically

\* one step at a time until the y position of the head is within

\* one head diameter of 175.

\*/

public void home()

{

// head x position left of home position

while(head.getXPos() < 125 - head.getDiameter())

{

this.right();

}

// head x position right of home position

while(head.getXPos() > 125)

{

this.left();

}

// head y position above home position

while(head.getYPos() < 175 - head.getDiameter())

{

this.down();

}

// head y position below home position

while(head.getYPos() > 175)

{

this.up();

}

}

**Question 2**

2a)

|  |  |
| --- | --- |
| Requirement | My answer |
| 1(a) Composite class | FootballClub |
| 1(b) Component class | Manager |
| 1(c) Relationship between the two | A FootballClub has-a Manager |
| 2(a) Composite class instance variables and examples of use | manager of type Manager, the football club the manager is part of. (Provides the has-a relationship).  name of type String, the name of the football club, e.g. “Portsmouth FC”  yearFounded of type String, the year the football club came into existence, e.g. 1898 |
| 2(b) Component class instance variables and examples of use | name of type String, the name of the manager, e.g. “Kenny Jackett”  yearOfBirth of type String, the year the manager was born, e.g. “1962”  yearBecameManager of type String, the year they became manager of the club, e.g. 2017 |
| 3 Description of method (not toString()) that allows the composite class to communicate with the component class in order to compute some value | FootballClub has a method yearsManaged(). It computes the number of whole years the manager has been managing the football club. The Manager instance variable yearBecameManager is used in the computation. |
| 4 Other interesting features of the class | FootballClub provides getters and setters for its instance variables.  FootballClub has a method isOverHundred() that returns a boolean value of true if it is at least one hundred years old.  Manager provides getters and setters for its instance variables.  Both classes includes a toString() method to help describe objects of the class. |

2b)

Composite class

/\*\*

\* Class FootballClub is a composite class representing a real-world club.

\* It has attributes of name, the yearFounded and manager. The manager

\* attribute is an object of the Manager class.

\* TMA02 Q2

\*

\* @author Wayne Sandford

\* @version 12-01-2019

\*/

public class FootballClub

{

// instance variables

private String name;

private String yearFounded;

private Manager manager; // Provides has-a relationship - A FootballClub has-a Manager

/\*\*

\* Constructor for objects of class FootballClub

\* when all attributes are known.

\*/

public FootballClub(String aName, String aYearFounded, Manager aManager)

{

this.name = aName;

this.yearFounded = aYearFounded;

this.manager = aManager;

}

/\*\*

\* Constructor for objects of class FootballClub where name attribute

\* is only known. Minimum attribute needed to construct object.

\*/

public FootballClub(String aName)

{

this.name = aName;

this.yearFounded = null; // Indicates yearFounded is unknown

this.manager = new Manager(); // Empty Manager object

}

/\*\*

\* Setter for football club's name.

\*/

public void setName(String aName)

{

this.name = aName;

}

/\*\*

\* Setter for football club's yearFounded.

\*/

public void setYearFounded(String aYearFounded)

{

this.yearFounded = aYearFounded;

}

/\*\*

\* Setter for football club's managerName. managerName is attribute of

\* Manager object, so is forwarded to component class.

\*/

public void setManagerName(String aManagerName)

{

this.manager.setName(aManagerName);

}

/\*\*

\* Setter for football club's manager yearOfBirth. Manager yearOfBirth is

\* attribute of Manager object, so is forwarded to component class.

\*/

public void setManagerYearOfBirth(String aManagerYearOfBirth)

{

this.manager.setYearOfBirth(aManagerYearOfBirth);

}

/\*\*

\* Setter for football club's manager yearBecameManager. Manager

\* yearBecameManager is attribute of Manager object,

\* so is forwarded to component class.

\*/

public void setManagerYearBecameManager(String aManagerYearBecameManager)

{

this.manager.setYearBecameManager(aManagerYearBecameManager);

}

/\*\*

\* Getter for football club's name.

\*/

public String getName()

{

return this.name;

}

/\*\*

\* Getter for football club's yearFounded. Returns

\* "Unknown" if yearFounded has not been set.

\*/

public String getYearFounded()

{

if(this.yearFounded == null)

{

return "Unknown";

}

return this.yearFounded;

}

/\*\*

\* Getter for football club's manager name

\*/

public String getManagerName()

{

return this.manager.getName();

}

/\*\*

\* Getter for football club's manager yearOfBirth.

\*/

public String getManagerYearOfBirth()

{

return this.manager.getYearOfBirth();

}

/\*\*

\* Getter for football club's manager yearBecameManager.

\*/

public String getManagerYearBecameManager()

{

return this.manager.getYearBecameManager();

}

/\*\*

\* Computes the amount of whole years the manager has managed

\* at the football club. It uses the yearBecameManager attribute

\* from the Manager object in its calculation.

\*/

public void yearsManaged()

{

if(getManagerYearBecameManager() != "Unknown") // Valid value

{

System.out.println(this.getManagerName() + " has been manager for "

+ (2019 - (Integer.parseInt(this.manager.getYearBecameManager())))

+ " whole years.");

}

else

{

System.out.println("Years managed is unknown.");

}

}

/\*\*

\* Return true if the football club is one hundred or

\* more years old. Returns false if yearFounded is not set.

\*/

public boolean isOverHundred()

{

int year;

if(this.getYearFounded() == "Unknown")

{

return false;

}

year = (Integer.parseInt(this.getYearFounded()));

return (year + 100) <= 2019;

}

/\*\*

\* Returns a string describing the football club object.

\*/

@Override

public String toString()

{

return "Name: " + this.name + "(Founded:" + this.getYearFounded() + ") "

+ "Manager: " + this.getManagerName() + "(Born:"

+ this.manager.getYearOfBirth() + ") Club hundred or more years old: "

+ this.isOverHundred();

}

}

Component class

/\*\*

\* Class Manager is a component class. It has attributes

\* of name, yearOfBirth and yearBecameManager.

\* TMA02 Q2

\*

\* @author Wayne Sandford

\* @version 12-01-2019

\*/

public class Manager

{

// instance variable

private String name;

private String yearOfBirth;

private String yearBecameManager;

/\*\*

\* Constructor for objects of class Manager when all attributes are known.

\*/

public Manager(String aName, String aYearOfBirth, String aYearBecameManager)

{

this.name = aName;

this.yearOfBirth = aYearOfBirth;

this.yearBecameManager = aYearBecameManager;

}

/\*\*

\* Zero-argument constructor for object of class Manager.

\*/

public Manager()

{

this.name = null; // Indicates name is unknown

this.yearOfBirth = null; // Indicates yearOfBirth is unknown

this.yearBecameManager = null; // Indicates yearBecameManager is unknown

}

/\*\*

\* Setter for manager's name.

\*/

public void setName(String aName)

{

this.name = aName;

}

/\*\*

\* Setter for manager's yearOfBirth.

\*/

public void setYearOfBirth(String aYearOfBirth)

{

this.yearOfBirth = aYearOfBirth;

}

/\*\*

\* Setter for manager's yearBecameManager.

\*/

public void setYearBecameManager(String aYearBecameManager)

{

this.yearBecameManager = aYearBecameManager;

}

/\*\*

\* Getter for manager's name. Returns "Unknown" if

\* name has not been set.

\*/

public String getName()

{

if(this.name == null)

{

return "Unknown";

}

return this.name;

}

/\*\*

\* Getter for manager's yearOfBirth. Returns "Unknown" if

\* yearOfBirth has not been set.

\*/

public String getYearOfBirth()

{

if(this.yearOfBirth == null)

{

return "Unknown";

}

return this.yearOfBirth;

}

/\*\*

\* Getter for manager's yearBecameManager. Returns "Unknown" if

\* yearBecameManager has not been set.

\*/

public String getYearBecameManager()

{

if(this.yearBecameManager == null)

{

return "Unknown";

}

return this.yearBecameManager;

}

/\*\*

\* Returns a string describing the Manager object.

\*/

@Override

public String toString()

{

return "Manager: " + this.getName() + "(Born:" + this.getYearOfBirth() +

") Year became manager: " + this.getYearBecameManager();

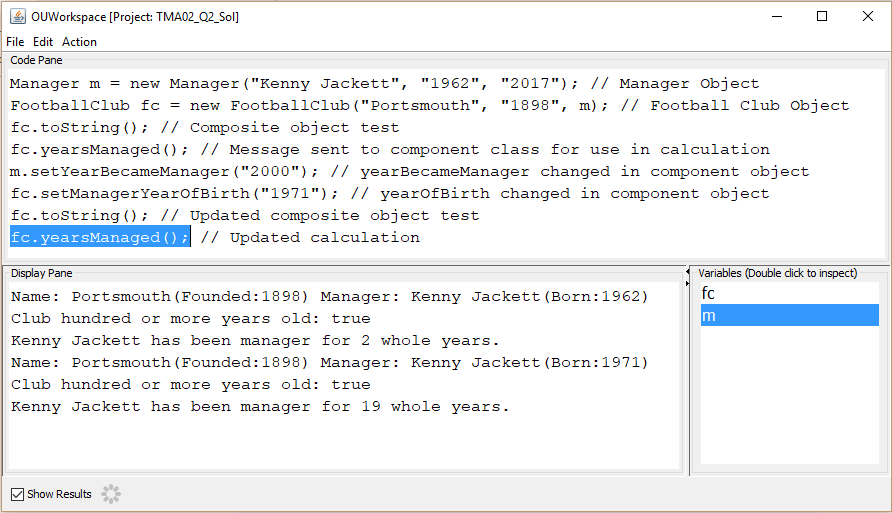
}

}

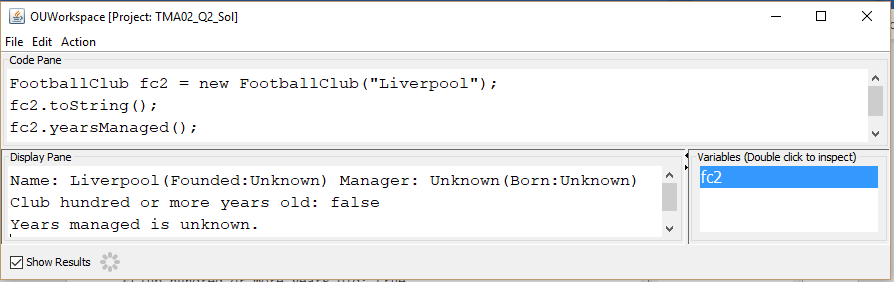
2c)

Test code

* Creates Manager and FootballClub objects.
* Displays FootballClub object toString() information.
* Sends message to component class in method fc.yearsManaged(). The resultant value used in calculation.
* Change yearBecameManager in component class.
* Change yearOfBirth in component class from composite class.
* Display updated toString() and yearsManaged() methods.



Test code when only FootballClub name attribute known:



**Question 3**

3a)

/\*\*

\* class Dog - is a subclass of Pet

\* TMA02 Q3

\*

\* @author Wayne Sandford

\* @version 12/01/2019

\*/

public class Dog extends Pet

{

// instance variables

/\*\*

\* Constructor for objects of class Dog

\*/

public Dog()

{

}

}

3b)

// instance variables

private int happiness;

private int timesFedToday;

/\*\*

\* Returns the happiness of the receiver.

\*/

public int getHappiness()

{

return this.happiness;

}

/\*\*

\* Returns the timesFedToday of the receiver.

\*/

public int getTimesFedToday()

{

return this.timesFedToday;

}

3c) i)

/\*\*

\* Constructor for objects of class Dog

\*/

public Dog(String aName, String aDescription)

{

super(aName, aDescription);

this.happiness = 2;

}

3c) ii)

It is not necessary to set the instance variable timesFedToday explicitly in the constructor because the default value of type int is 0; this is the initial value required. However, to be clear of the intentions of this instance variable it would be good practice to initialise explicitly in the constructor.

3d)

/\*\*

\* Helper method that decrements happiness by one down

\* to a minimum of zero.

\*/

public void decrementHappiness()

{

if(this.getHappiness() > 0)

{

this.happiness--;

}

}

/\*\*

\* Helper method that decrements timesFedToday by one down

\* to a minimum of zero.

\*/

public void decrementTimesFedToday()

{

if(this.getTimesFedToday() > 0)

{

this.timesFedToday--;

}

}

/\*\*

\* Helper method that increments happiness by one.

\*/

public void incrementHappiness()

{

this.happiness++;

}

/\*\*

\* Helper method that increments timesFedToday by one.

\*/

public void incrementTimesFedToday()

{

this.timesFedToday++;

}

3e) i)

/\*\*

\* Repeatedly will :

\* - display walking message and add one to happiness.

\* - display hungry message and subtract one from timesFedToday.

\* While timesFedToday is greater than zero.

\*/

public void walkies()

{

while(this.getTimesFedToday() > 0)

{

System.out.println("I'm going for a walk!");

this.incrementHappiness();

System.out.println("I'm getting hungry");

this.decrementTimesFedToday();

}

}

3e) ii)

/\*\*

\* Returns true if happiness and timesFedToday are above zero,

\* otherwise returns false. Displays appropriate can't sleep

\* message for either or both happiness and timesFedToday when

\* zero.

\*/

public boolean sleep()

{

if(this.getHappiness() == 0)

{

System.out.println("Not happy, can't sleep");

}

if(this.getTimesFedToday() == 0)

{

System.out.println("Hungry, can't sleep");

}

return ((this.getHappiness() > 0) && (this.getTimesFedToday() > 0));

}

3e) iii)

/\*\*

\* Displays no walkies message and decrements happiness.

\*/

public void noWalkies()

{

System.out.println("No walkies :-(");

this.decrementHappiness();

}

3f) i)

public class Dog extends Pet implements Lovable

public class Fish extends Pet implements Lovable

public class PondFish extends Fish is a subclass of class Fish, therefore it will inherit the Lovable interface from class Fish. However, to explicable implement it would be:

public class PondFish extends Fish implements Lovable

3f) ii)

/\*\*

\* interface Lovable TMA02 Q3

\*

\* @author Wayne Sandford

\* @version 12-01-2019

\*/

public interface Lovable

{

/\*\*

\* Performs an arbitrary action in response to message stroke().

\*/

public void stroke();

/\*\*

\* Performs an arbitrary action in response to message canStroke().

\* Returns a boolean value of true or false.

\*/

public boolean canStroke();

/\*\*

\* Performs an arbitrary action in response to message feed().

\*/

public void feed();

}

3f) iii)

The Lovable interface’s three methods are not required in the PondFish subclass because they are inherited from its Fish superclass. In PondFish constructor the line super(aName, aDescription); sends the arguments to its superclass Fish to initialise the instance variables. The Fish class implemented the Lovable interface methods, so every subclass can use these method versions as they are.