**Q3.A**

The two object that I have chosen are a Book and a Bookshelf, and the book will have to fit on the bookshelf. The book and the bookshelf with have many of the same attributes including with and depth. The book will also have the attribute of title and the bookshelf will have the attributes of used space and remaining space will automatically be calculated. The book object will also have a method to test if the book object will fit on the bookshelf. The method putOnShelf will take the width of the book away from the remainingSpace on the bookshelf if the book will fit.

**Q3.B**

**Book Class**

/\*\*

\* Objects of the Book class have the attributes of width and depth which are integer numbers, as well as title and willFit with are strings.

\*

\* The class constructor will initialise objects with the attributes width, depth and title.

\* Throughout the class the attributes width and depth should be given as CM.

\*

\* The class also has setter and getter messages for the attributes.

\*

\* The method in the class is willFit, this method requres a a Bookshelf object to have been created

\* and will work out if the book object will fit on the bookshelf and return a statement stating if it will fit or not.

\*

\* The class also has the method toString which will return a statment giving the state of the object.

\*

\* The method putOnShelf with return a statement that the book is on the bookshelf and remove the books width from the remainingSpace

\* on the book shelf if it will fit. Else it will return a string stating that it will not fit.

\*

\* @author (Ross Barden)

\* @version (17.01.2016)

\*/

public class Book

{

// instance variables - replace the example below with your own

private int width;

private int depth;

private String title;

/\*\*

\* Constructor for objects of Book class.

\*

\* It it takes the argunments of width, and depth which should be given as integers. As well as title which should be given as a string.

\*/

public Book(int width, int depth, String title)

{

// initialise instance variables

super ();

this.width = width ; //The integer for width should be given in CM

this.depth = depth ; //The integer for depth should be given in CM

this.title = title ; //This attribute is important for the toString and willFit methods

}

/\*\*

\* Returns the width of the object, as an integer.

\*/

public int getWidth ()

{

return this.width;

}

/\*\*

\* A modifier method which sets the width of the object and takes the desired width given as an integer taken as an argument.

\*/

public void setWidth(int aWidth)

{

this.width = aWidth;

}

/\*\*

\* Returns the depth of the object, as an integer.

\*/

public int getDepth()

{

return this.depth;

}

/\*\*

\* A modifier method which sets the depth of the object and takes the desired width given as an integer taken as an argument.

\*/

public void setDepth(int aDepth)

{

this.depth = aDepth;

}

/\*\*

\* Returns the title of the object, as a string.

\*/

public String getTitle()

{

return this.title;

}

/\*\*

\* A modifier method which sets th title of the object and takes the desred title given as a string as an argument.

\*/

public void setTitle(String aTitle)

{

this.title = aTitle;

}

/\*\*

\* Returns a string describing the state of the of object.

\*/

@Override

public String toString()

{

return "The book " + this.getTitle() + " is " + this.getWidth() + "CM wide, and " + this.getDepth() + "CM deep.";

}

/\*\*

\* This method requires an object of the Bookshelf class to have been constructed.

\*

\* If the book will fit onto the bookshelf this method will return a string giving

\* the title and stating that it will fit into the shelf.

\*

\* If the book will not fit onto the bookshelf the method will return a string giving

\* the title and stating that it will not fit on the shelf.

\*

\*/

public String willFit(Bookshelf aBookshelf)

{

if (this.getWidth() <= aBookshelf.getRemainingSpace())// This method calls an object of the Bookshelf class.

{

return "The book " + this.getTitle() + " will fit on the bookshelf" ;

}

else

{

return "The book " + this.getTitle() + " will not fit on the bookshelf" ;

}

}

/\*\*

\* This method requires an object of the Bookshelf class to have been constructed.

\*

\* If the book will fit on the shelf, the width of the book will be taken away from the remainingSpace of the bookshelf.

\* It will also return a string stating that the book has been put onto the shelf.

\*

\* If the book will not fit on the self it will return a string stating that it will not fit.

\*/

public String putOnShelf(Bookshelf aBookshelf)

{

if (this.getWidth() <= aBookshelf.getRemainingSpace())// This method calls an object of the Bookshelf class.

{

aBookshelf.usedSpace = aBookshelf.usedSpace + this.getWidth();

return "The book " + this.getTitle() + " is on the bookshelf." ;

}

else

{

return "The book " + this.getTitle() + " will not fit on the bookshelf." ;

}

}

}

**Bookshelf Class**

/\*\*

\* The Bookshelf class has the variables of shelfDepth, shelfWidth, usedSpace and SpaceRemaining which are all given as itegers.

\*

\* The class constructor will initialise the object with shelfDepth, shelfWidth and usedSpace, which should be given in CM.

\* The variable spaceRemaining will automatically be caculated by taking the usedSpace from the shelfWidth.

\*

\* There are both setter and getter messages for shelfDepth, shelfWidth and usedSpace.

\* The attribute spaceRemaining only has a getter message as it is caculated from the attributes shelfWidth and usedSpace.

\*

\* The toString method returns a string giving the state of the object.

\*

\*

\* @author (Ross Barden)

\* @version (17.01.2016)

\*/

public class Bookshelf

{

// instance variables - replace the example below with your own

private int shelfDepth;

private int shelfWidth;

public int usedSpace;

private int spaceRemaining;

/\*\*

\* Constructor for objects of class Bookshelf

\*/

public Bookshelf(int shelfDepth, int shelfWidth, int usedSpace)

{

// initialise instance variables

this.shelfDepth = shelfDepth; //This should be given in CM

this.shelfWidth = shelfWidth; //This should be given in CM

this.usedSpace = usedSpace; //This should be given in CM

spaceRemaining = shelfWidth - usedSpace;

}

/\*\*

\* Returns the depth of the shelf as an integer.

\*/

public int getShelfDepth()

{

return this.shelfDepth;

}

/\*\*

\* A modifier method which sets the depth of the object and takes the desired width given as an integer taken as an argument.

\*/

public void setShelfDepth(int depth)

{

this.shelfDepth = depth;

}

/\*\*

\* Returns the width of the shelf as an integer.

\*/

public int getShelfWidth()

{

return this.shelfWidth;

}

/\*\*

\* A modifier method which sets the width of the object and takes the desired width given as an integer taken as an argument.

\*/

public void setShelfWidth(int width)

{

this.shelfWidth = width;

}

/\*\*

\* Returns the space that has already been used on the bookshelf as an integer.

\*/

public int getUsedSpace()

{

return this.usedSpace;

}

/\*\*

\* A modifier method which sets the used space of the object and takes the desired width given as an integer taken as an argument.

\*/

public void setUsedSpace(int space)

{

this.usedSpace = space;

}

/\*\*

\* Returns the remaining space on the bookshelf as an integer.

\*/

public int getRemainingSpace()

{

return this.getShelfWidth() - this.getUsedSpace();

}

/\*\*

\* Returns a string giving the state of the bookshef.

\*/

@Override

public String toString()

{

return "The bookshelf is " + this.getShelfWidth() + " CM wide, and is " + this.getShelfWidth() + " and has " + this.getRemainingSpace() + " CM of free space remaining.";

}

}

**3.C**

**Code Panel**

Book aBook = new Book(5, 10, "How to code");

Bookshelf aBookshelf = new Bookshelf(10, 100, 0);

aBook.toString();

aBookshelf.toString();

**Display Pane**

The book How to code is 5CM wide, and 10CM deep.

The bookshelf is 100 CM wide, and is 100 and has 100 CM of free space remaining.

**4.A.**

import ou.\*;

/\*\*

\* The class RestrictedHoverFrog is a subclass of HoverFrog.

\* Unlike HoverFrog it can only move between the black stones 1 and 11.

\* RestrictedHoverFrog is unable to change its colour and will be green unless it

\* is hovering when it will be red.

\*

\* @author (Ross Barden)

\* @version (17.01.2016)

\*/

public class RestrictedHoverFrog extends HoverFrog

{

// instance variables - replace the example below with your own

/\*\*

\* Constructor for objects of class RestrictedHoverFrog

\*/

public RestrictedHoverFrog()

{

}

An import statement is written before the class comments, and it tells the Java compiler what public Java packages your code is using. There are some methods that are inherited from the Java Object class however there are not many of these. Importing a package means that you are able to use the methods in that package and will save you from having to re-write all of the code yourself.

**4.B.**

/\*\*

\* A modifier method that overrides the setPosition method of its super class, and takes an integer as an argument.

\* If the intgerer is between 1 and 11 (inclusive) it will set the RestrictedHoverFrog in that position.

\* If the integer is not in the range, it will do nothing.

\*/

@Override

public void setPosition (int aPosition)

{

if (aPosition > 0 && aPosition <=11)

{

super.setPosition(aPosition);

}

}

**4.C.**

/\*\*

\* A modifier method that overrides the setColour method its super class and does nothing.

\*/

@Override

public void setColour(OUColour aColour)

{

}

**4.D.**

/\*\*

\* A modifier method that overrights the setHeight of its super class.

\* It takes an interger as an argument.

\* If the height is set to 0 it will set the colour of the frog to Green, if the height is

\* more than 0 it will set the colour of the frog to Red.

\*/

@Override

public void setHeight(int aHeight)

{

super.setHeight(aHeight);

if (this.getHeight() > 0)

{

super.setColour(OUColour.RED);

}

else

{

super.setColour(OUColour.GREEN);

}

}

**4.E**

/\*\*

\* Constructor for the RestrictedHoverFrog class that takes two arguments given as integers.

\*

\* The first integer represents the position and if it is between 0 and 11 (inclusive) it will set the

\* RestrictedHoverFrog's position to the value of the interger. If the argument is not within this range it will

\* set the RestrictedHoverFrog's position to that set by the constructor of the Frog class.

\*

\* The second integer represents the height and if it is between 0 and 6 (inclusive) it will set the

\* RestrictedHoverFrog's height to the value of the interger. If the argument is not within the range it will set

\* the RestructedHoverFrog's height to that set by the constructor of the Frog class. If the height of the frog is 0 it

\* will set the colour to Green, and it it is between 1-5 (inclusive) it will set the colour to Red.

\*/

public RestrictedHoverFrog(int a, int b)

{

super();

if (a > 0 && a <=11)

{

this.setPosition(a);

}

if (b >= 0 && b <=6)

{

this.setHeight(b);

}

}

**4.F**

Constructor chaining works by calling the constructor of the classes superclass. In order to do this you must call super(); on the first line of the constructor. Using the class RestrictedHoverFrog as an example the constructor calls the constructor of HoverFrog which itself calls the constructor of the Frog class, and so on up the class hierarchy.

If the code was changed in the constructor of the Amphibian constructor it would mean that a newly created RestrictedHoverFrog object would have its colour set to null, because within the body of the RestrictedHoverFrog class it has overridden the setColour method so that it does nothing.

Overloading is when there is more than one constructor within a class. This is done because different constructors will initialise the objects with different states. The first constructor in the RestrictedHoverFrog class is a zero argument constructor because it creates the object without the user specifying any variable. The second constructor requires the user to put in to integer values and providing they are within the parameters laid out by the constructor they it will be created with those values.

**4.G.**

Classes that implement an interface can call other classes that implement the interface as arguments within their constructor. This will allow the classes to help break the class hierarchy and to save the developer from having to repeat code that will be re-used through different classes.

**Interface**

/\*\*

\* An interface which can be implemented by all classes.

\*

\* @author (Ross Barden)

\* @version (17.01.2016)

\*/

public interface Exercisable

{

/\*\*

\* Takes two arguments an integer and a boolean value.

\*

\* The interger dictates how many times the exercises should be performed.

\*

\* If the boolean value is true the exercise should begin in a leftwards direction (left, right, right, left).

\* If the boolean value is false the exercise should begin in rightwards direction, (right, left, left, right).

\*/

void exercise(int duration, boolean isStartingLeft);

}

**Method**

/\*\*

\*This method calles the method from the interface Exercisable.

\*It takes two arguments and interger and a boolean value.

\*

\*If the boolean value is true, the object moves left, right, right, left according to the interger value.

\*If the boolean value is false, the object moves right, left, left, right accorinding to the interger value.

\*/

public void exercise(int duration, boolean isStartingLeft)

{

if (isStartingLeft == true)

{

while (duration > 0)

{

this.left();

this.right();

this.right();

this.left();

duration--;

}

}

else

{

while (duration > 0)

{

this.right();

this.left();

this.left();

this.right();

duration--;

}

}

**Whole class**

import ou.\*;

/\*\*

\* The class RestrictedHoverFrog is a subclass of HoverFrog.

\* Unlike HoverFrog it can only move between the black stones 1 and 11.

\* RestrictedHoverFrog is unable to change its colour and will be green unless it

\* is hovering when it will be red.

\*

\* @author (Ross Barden)

\* @version (17.01.2016)

\*/

public class RestrictedHoverFrog extends HoverFrog implements Exercisable

{

// instance variables - replace the example below with your own

/\*\*

\* Constructor for objects of class RestrictedHoverFrog

\* Zero argument constructor

\*/

public RestrictedHoverFrog()

{

}

/\*\*

\* Constructor for the RestrictedHoverFrog class that takes two arguments given as integers.

\*

\* The first integer represents the position and if it is between 0 and 11 (inclusive) it will set the

\* RestrictedHoverFrog's position to the value of the interger. If the argument is not within this range it will

\* set the RestrictedHoverFrog's position to that set by the constructor of the Frog class.

\*

\* The second integer represents the height and if it is between 0 and 6 (inclusive) it will set the

\* RestrictedHoverFrog's height to the value of the interger. If the argument is not within the range it will set

\* the RestructedHoverFrog's height to that set by the constructor of the Frog class. If the height of the frog is 0 it

\* will set the colour to Green, and it it is between 1-5 (inclusive) it will set the colour to Red.

\*/

public RestrictedHoverFrog(int a, int b)

{

super();

if (a > 0 && a <=11)

{

this.setPosition(a);

}

if (b >= 0 && b <=6)

{

this.setHeight(b);

}

}

/\*\*

\* A modifier method that overrides the setPosition method of its super class, and takes an integer as an argument.

\* If the intgerer is between 1 and 11 (inclusive) it will set the RestrictedHoverFrog in that position.

\* If the integer is not in the range, it will do nothing.

\*/

@Override

public void setPosition (int aPosition)

{

if (aPosition > 0 && aPosition <=11)

{

super.setPosition(aPosition);

}

}

/\*\*

\* A modifier method that overrides the setColour method its super class and does nothing.

\*/

@Override

public void setColour(OUColour aColour)

{

}

/\*\*

\* A modifier method that overrights the setHeight of its super class.

\* It takes an interger as an argument.

\* If the height is set to 0 it will set the colour of the frog to Green, if the height is

\* more than 0 it will set the colour of the frog to Red.

\*/

@Override

public void setHeight(int aHeight)

{

super.setHeight(aHeight);

if (this.getHeight() > 0)

{

super.setColour(OUColour.RED);

}

else

{

super.setColour(OUColour.GREEN);

}

}

/\*\*

\*This method calles the method from the interface Exercisable.

\*It takes two arguments and interger and a boolean value.

\*

\*If the boolean value is true, the object moves left, right, right, left according to the interger value.

\*If the boolean value is false, the object moves right, left, left, right accorinding to the interger value.

\*/

public void exercise(int duration, boolean isStartingLeft)

{

if (isStartingLeft == true)

{

while (duration > 0)

{

this.left();

this.right();

this.right();

this.left();

duration--;

}

}

else

{

while (duration > 0)

{

this.right();

this.left();

this.left();

this.right();

duration--;

}

}

}

}