SIT232

OBJECT ORIENTATED PROGRAMMING

Learning Summary Report

DALE ORDERS
219106283

Self-Assessment Details

The following checklists provide an overview of my self-assessment for this unit.

	Pass (D)	Credit (C)	Distinction (B)	High Distinction (A)
Self-Assessment				✓

Self-Assessment Statement

	Included
Learning Summary Report	✓
Pass tasks complete	✓

Minimum Pass Checklist

	Included
All Pass Tasks are Complete	✓

Minimum Credit Checklist (in addition to Pass Checklist)

	Included
All Credit Tasks are Complete	✓

Minimum Distinction Checklist (in addition to Credit Checklist)

	Included
All Distinction Tasks are Complete	✓

Minimum High Distinction Checklist (in addition to Distinction Checklist)

	Included
All High Distinction Tasks are Complete	✓

Declaration

I declare that this portfolio is my individual work. I have not copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part of this submission been written for me by another person.

Signature: **Dale Orders**

Portfolio Overview

This portfolio includes work that demonstrates that I have achieved all Unit Learning Outcomes for SIT232 to a **High Distinction** level.

Throughout this unit, I have consistently sought to challenge myself by applying the principles of Object Orientated Programming to a range of tasks. I have actively attempted to engage with the many features of Object Orientated Programming in both the design and development of programs that seek to explore the .Net framework. In this respect, I have come to appreciate how Object Orientated programming can serve to increase functional design and application, by reducing errors and bolstering the functionality of an otherwise simple program. In this, I have endeavored to adapt my programs to better engage with the underlying principles of and apply what it is we learnt in the lectures. In doing so, I have become better at being able to structure my programs in a logical sequence with attributes and methods designed to deliver the intended outcome.

I have submitted all of the tasks and continued to be active in the learning community online in the teams chat, and in the deakin discussion forum. I have sought to engage with other students in the unit and hope to continue this approach throughout SIT221.

Reflection

The most important things I learnt:

I learnt the importance of employing a SOLID approach when designing a program, which is an something that will be of benefit to me throughout my entire career. I appreciate the importance of having a solid framework with which to design and implement a working program.

The things that helped me most were:

I really enjoyed the opportunity to engage with other students, to see how a problem can be approached from various perspectives. Being able to think more broadly about a problem is a skill I hope to continue to develop. It is important to be able to engage with other people and to see how a problem can be solved in various ways.

I found the following topics particularly challenging:

I initially found the concept of inheritance difficult. However, I used the resources the lectures listed on the task sheet and also found some additional material online on the docs Microsoft website. I also found that the course workbook was very helpful in providing more examples to illustrate the concept.

I found the following topics particularly interesting:

I liked learning about implementing an interface and identifying how common attributes and properties can be written to enact across classes.

I feel I learnt these topics, concepts, and/or tools really well:

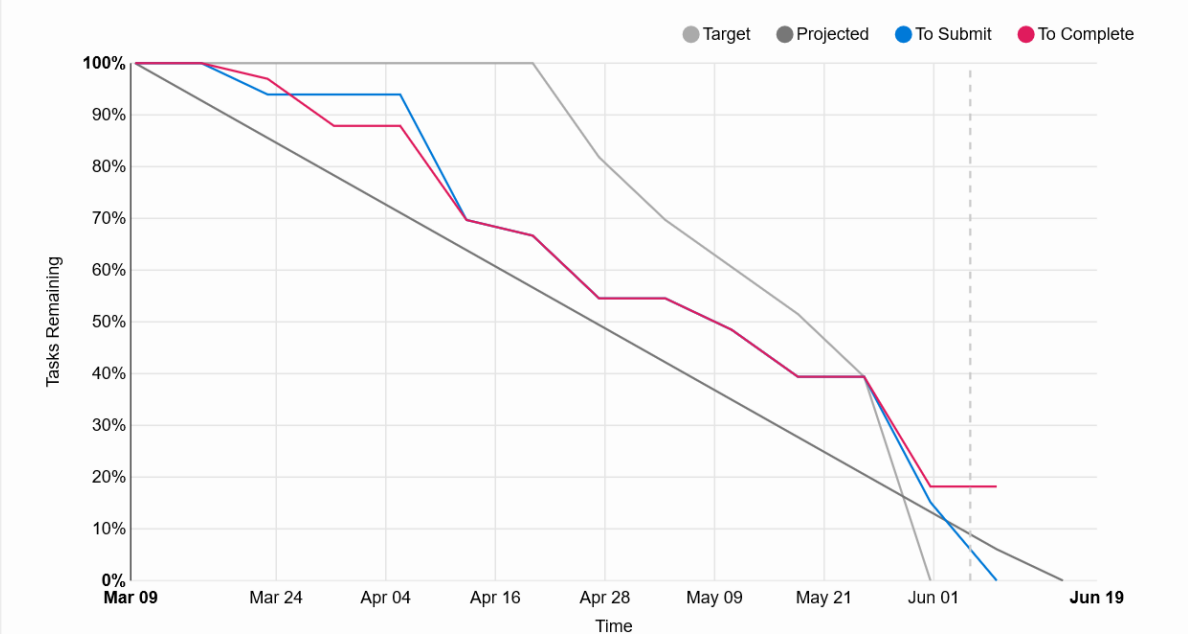
Whilst creating the Bank Program, I came to appreciate the importance of encapsulation and why we need to protect information within the system and ensure that it is not accessible to users by enacting a constructor and a read-only property.

I still need to work on the following areas:

I wish to develop more complicated real-world projects to showcase my skills and knowledge. This will give me more experience and allow me to build a portfolio I can show to a prospective employer.

My progress in this unit was …:

The screenshot of my progress is below. It shows that I have been consistent in my approach throughout the trimester.



This unit will help me in the future:

Being able to understand and apply the four principles of OOP, will enable me to more easily learn other such languages like Java. This will prepare me for being able to undertake more challenging projects in the future.

If I did this unit again I would do the following things differently:

I would have sought out more resources before I had started the course so I could be ready to start on getting through the tasks even earlier than I did. This would have given me more exposure to the design process.

Other…:

Deakin University

Object Oriented Development

Dale Orders

Portfolio Submission

Submitted By:
Dale Orders
dorders

Tutor:
Sergey Polyakovskiy

June 5, 2020

Contents

2 Overall Task Status

Task	Status	Times Assessed
C# Essentials: Selection and Casting	Complete	1
C# Essentials: Repetition	Complete	1
Helping Your Peers	Complete	1
C# Essentials: Classes and Objects	Complete	2
The Account Class	Complete	2
The MyTime Class	Complete	1
C# Essentials: Arrays and Lists	Complete	1
Validating Accounts	Complete	1
The MyPolynomial class	Complete	1
Bucket Sort	Complete	1
Exceptions and Error Handling	Complete	1
BuggySoft: Program Design and Class Composition	Complete	1
C# Essentials: Inheritance	Complete	1
Bank Transactions	Complete	1
A Simple Reaction-Timer Controller	Complete	1
An Enhanced Reaction-Timer Controller	Demonstrate	1
C# Essentials: Polymorphism	Complete	1
Multiple Bank Accounts	Complete	1
Documenting the Banking System	Time Exceeded	2
Abstract Transactions	Complete	1

3 Learning Outcomes

3.1 Evaluate Code

Evaluate simple program code for correct use of coding conventions, and use code tracing and debugging techniques to identify and correct issues.

Task	Rating	Status	Times Assessed
C# Essentials: Classes and Objects	♦◊◊◊◊	Complete	2
C# Essentials: Arrays and Lists	♦◊◊◊◊	Complete	1
The MyTime Class	♦◊◊◊◊	Complete	1
The MyPolynomial class	♦◊◊◊◊	Complete	1
Validating Accounts	♦◊◊◊◊	Complete	1
Bank Transactions	♦◊◊◊◊	Complete	1
Bucket Sort	♦♦◊◊◊	Complete	1
Exceptions and Error Handling	♦◊◊◊◊	Complete	1
C# Essentials: Inheritance	♦♦♦♦♦	Complete	1
BuggySoft: Program Design and Class Composition	♦♦♦♦◊	Complete	1
A Simple Reaction-Timer Controller	♦♦♦◊◊	Complete	1
An Enhanced Reaction-Timer Controller	♦◊◊◊◊	Demonstrate	1
C# Essentials: Polymorphism	♦♦♦◊◊	Complete	1
Abstract Transactions	♦◊◊◊◊	Complete	1

3.2 Principles

Apply and explain the principles of object oriented programming including abstraction, encapsulation, inheritance and polymorphism.

Task	Rating	Status	Times Assessed
C# Essentials: Classes and Objects	♦♦◊◊◊	Complete	2
C# Essentials: Arrays and Lists	♦♦◊◊◊	Complete	1
The MyTime Class	♦♦◊◊◊	Complete	1
The MyPolynomial class	♦♦◊◊◊	Complete	1
Validating Accounts	♦♦♦◊◊	Complete	1
Bank Transactions	♦♦◊◊◊	Complete	1
Bucket Sort	♦♦♦◊◊	Complete	1
Exceptions and Error Handling	♦♦♦◊◊	Complete	1
C# Essentials: Inheritance	♦♦♦♦♦	Complete	1
Multiple Bank Accounts	♦♦♦◊◊	Complete	1
BuggySoft: Program Design and Class Composition	♦◊◊◊◊	Complete	1
A Simple Reaction-Timer Controller	♦◊◊◊◊	Complete	1
An Enhanced Reaction-Timer Controller	♦♦♦◊◊	Demonstrate	1
C# Essentials: Polymorphism	♦♦◊◊◊	Complete	1
Abstract Transactions	♦♦♦◊◊	Complete	1

3.3 Build Programs

Implement, and test small object oriented programs that conform to planned system structures and requirements

Task	Rating	Status	Times Assessed
C# Essentials: Selection and Casting	♦♦◊◊◊	Complete	1
C# Essentials: Repetition	♦♦◊◊◊	Complete	1
C# Essentials: Classes and Objects	♦♦♦◊◊	Complete	2
C# Essentials: Arrays and Lists	♦♦♦◊◊	Complete	1
The MyTime Class	♦♦◊◊◊	Complete	1
The Account Class	♦♦◊◊◊	Complete	2
The MyPolynomial class	♦♦◊◊◊	Complete	1
Validating Accounts	♦♦◊◊◊	Complete	1
Bank Transactions	♦♦♦♦◊	Complete	1
Bucket Sort	♦♦♦◊◊	Complete	1
Exceptions and Error Handling	♦♦◊◊◊	Complete	1
C# Essentials: Inheritance	♦♦♦♦♦	Complete	1
Multiple Bank Accounts	♦♦♦♦◊	Complete	1
BuggySoft: Program Design and Class Composition	♦◊◊◊◊	Complete	1
A Simple Reaction-Timer Controller	♦♦♦◊◊	Complete	1
An Enhanced Reaction-Timer Controller	♦♦♦◊◊	Demonstrate	1
C# Essentials: Polymorphism	♦◊◊◊◊	Complete	1
Abstract Transactions	♦♦♦♦◊	Complete	1

3.4 Design

Design, communicate, and evaluate solution structures using appropriate diagrams and textual descriptions.

Task	Rating	Status	Times Assessed
C# Essentials: Selection and Casting	♦♦◊◊◊	Complete	1
C# Essentials: Repetition	♦♦◊◊◊	Complete	1
C# Essentials: Classes and Objects	♦♦♦◊◊	Complete	2
C# Essentials: Arrays and Lists	♦♦◊◊◊	Complete	1
The MyTime Class	♦◊◊◊◊	Complete	1
The Account Class	♦♦♦◊◊	Complete	2
The MyPolynomial class	♦◊◊◊◊	Complete	1
Validating Accounts	♦♦◊◊◊	Complete	1
Bank Transactions	♦♦♦◊◊	Complete	1
Bucket Sort	♦♦♦◊◊	Complete	1
Exceptions and Error Handling	♦♦♦◊◊	Complete	1
C# Essentials: Inheritance	♦♦♦♦♦	Complete	1
Multiple Bank Accounts	♦♦♦♦◊	Complete	1
BuggySoft: Program Design and Class Composition	♦◊◊◊◊	Complete	1
A Simple Reaction-Timer Controller	♦♦♦◊◊	Complete	1
An Enhanced Reaction-Timer Controller	♦♦♦♦◊	Demonstrate	1
C# Essentials: Polymorphism	♦◊◊◊◊	Complete	1
Abstract Transactions	♦♦♦◊◊	Complete	1
Documenting the Banking System	♦♦♦◊◊	Time Exceeded	2

3.5 Justify

Justify meeting specified outcomes through providing relevant evidence and critiquing the quality of that evidence against given criteria.

Task	Rating	Status	Times Assessed
C# Essentials: Selection and Casting	♦♦◊◊◊	Complete	1
C# Essentials: Repetition	♦♦◊◊◊	Complete	1
Helping Your Peers	♦♦♦◊◊	Complete	1
C# Essentials: Classes and Objects	♦◊◊◊◊	Complete	2
C# Essentials: Arrays and Lists	♦◊◊◊◊	Complete	1
The Account Class	♦♦◊◊◊	Complete	2
The MyPolynomial class	♦◊◊◊◊	Complete	1
Validating Accounts	♦◊◊◊◊	Complete	1
C# Essentials: Inheritance	♦♦♦♦♦	Complete	1
Multiple Bank Accounts	♦◊◊◊◊	Complete	1
A Simple Reaction-Timer Controller	♦◊◊◊◊	Complete	1
Abstract Transactions	♦◊◊◊◊	Complete	1
Documenting the Banking System	♦♦◊◊◊	Time Exceeded	2

4 C# Essentials: Selection and Casting

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Build Programs	♦♦◊◊◊

These small programs were a good opportunity to revise some of the basic concepts we learnt last year in 'introduction to programming.' Overall, I think it offered a good introduction to c# for those who have yet to use the language before.

Outcome	**Weight**
Design	♦♦◊◊◊

These small programs were a good opportunity to revise some of the basic concepts we learnt last year in 'introduction to programming.' Overall, I think it offered a good introduction to c# for those who have yet to use the language before.

Outcome	**Weight**
Justify	♦♦◊◊◊

These small programs were a good opportunity to revise some of the basic concepts we learnt last year in 'introduction to programming.' Overall, I think it offered a good introduction to c# for those who have yet to use the language before.

Date	Author	Comment
2020/03/17 12:03	Dale Orders	Ready to Mark
2020/03/17 22:12	Sanjay Segu	Hi, I am Sanjay and I will be your tutor for this unit assisting you with all the help you would require to gain quality knowledge.
2020/03/17 22:12	Sanjay Segu	Since you are a cloud student we expect you to make a demonstration video (Demonstration of your code - Explaining why did you code it in such a way and what you think each components of your code does and so on).
2020/03/17 22:12	Sanjay Segu	The video can be uploaded to Deakin Air or YouTube. Once your upload is ready please share the link in here so that I can watch and provide you tailored comments.
2020/03/17 22:29	Sanjay Segu	audio comment
2020/03/17 22:29	Sanjay Segu	Discuss
2020/03/17 22:29	Sanjay Segu	audio comment
2020/03/21 20:41	Dale Orders	https://youtu.be/4P8kcyXXUbA
2020/03/21 20:43	Dale Orders	https://youtu.be/Stjq16sBBzI
2020/03/21 20:45	Dale Orders	https://youtu.be/p8A9orGGRXM
2020/03/21 20:48	Dale Orders	https://youtu.be/RFGN2MSGTn4
2020/03/24 20:13	Sanjay Segu	Hi, my apologies for the delayed response. I had some unforeseen tasks which made me real busy. Now that I am all set, will make sure to provide feedback often.
2020/03/24 20:15	Sanjay Segu	Thanks for your time in recording those videos
2020/03/24 20:16	Sanjay Segu	Unfortunately, I can't seem to have access any of those videos of yours
2020/03/24 20:16	Sanjay Segu	It says the video is private.
2020/03/24 20:37	Dale Orders	Hi Sanjay. I will have to set the videos to public
2020/03/24 20:37	Dale Orders	also on the task sheet it says we do not have to use try catch if we don't want to
2020/03/24 20:37	Dale Orders	" What happens if you enter a letter, e.g. 'a'? How can you add an error message to catch this? (HINT: We normallyuse a try-catch block to catch the error. Do not worry if you are unable to complete this now, we will look at it in more detail later on in the unit."
2020/03/24 20:43	Sanjay Segu	Easy, just wanted to push you a little.
2020/03/24 20:43	Sanjay Segu	Please 'NOTE' from now on it's my sincere request to upload the demonstration videos only in Deakin Air and not in YouTube or any other on demand platform. Also, please note 'Don't' share your solutions to any anybody as it's is plagiarism.
2020/03/24 20:43	Sanjay Segu	Please share your solutions for tasks 1.1.3 and 1.1.4 and I will sign this task off.
2020/03/24 21:22	Dale Orders	Hi Sanjay, I'm happy to do a catch block. I just haven't been taught it
2020/03/24 21:23	Dale Orders	If you are happy to teach it to me, I am happy to put it in
2020/03/24 21:23	Dale Orders	Otherwise I can take it to the next prac and see if I cna get some help
2020/03/24 21:26	Dale Orders	Hopefully you can access these videos
2020/03/24 21:27	Dale Orders	https://video.deakin.edu.au/media/t/0_cxauafvz
2020/03/24 21:29	Dale Orders	https://video.deakin.edu.au/media/t/0_38gjy2nd
2020/03/24 21:34	Dale Orders	https://video.deakin.edu.au/media/t/0_4a1c4tv6
2020/03/24 21:39	Dale Orders	Can you see the videos Sanjay?
2020/03/26 22:53	Sanjay Segu	I can access the videos Dale. Thanks very much for uploading it to Deakin Air.

Deakin University

Object Oriented Development

OnTrack Submission

C# Essentials: Selection and Casting

Submitted By:
Dale Orders
dorders
2020/03/17 12:03

Tutor:
Sanjay Segu

Outcome	**Weight**
Build Programs	♦♦◊◊◊
Design	♦♦◊◊◊
Justify	♦♦◊◊◊

These small programs were a good opportunity to revise some of the basic concepts we learnt last year in 'introduction to programming.' Overall, I think it offered a good introduction to c# for those who have yet to use the language before.

March 17, 2020

```
using System;

namespace IfStatement
{
    class IfStatement
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter the number (as an integer): ");
            int number=Convert.ToInt32(Console.ReadLine());
            if(number==1)
            {
                Console.WriteLine("One");
            }
            else if (number==2)
            {
                Console.WriteLine("Two");
            }
             else if (number==3)
            {
                Console.WriteLine("Three");
            }

              else if (number==4)
            {
                Console.WriteLine("Four");
            }
                   else if (number==5)
            {
                Console.WriteLine("Five");
            }
                    else if (number==6)
            {
                Console.WriteLine("Six");
            }
                    else if (number==7)
            {
                Console.WriteLine("Seven");
            }
                     else if (number==8)
            {
                Console.WriteLine("Eight");
            }
                 else if (number==9)
            {
                Console.WriteLine("Nine");
            }
                 else
            {
                Console.WriteLine("Please enter a number between 0 and 9");
            }

        }
```

```
    }
}
```

```
using System;

namespace task_1_2
{
    class SwitchStatement
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter a number (as an integer): ");
            int number=Convert.ToInt32(Console.ReadLine());

            switch(number)
            {
                case 1: Console.WriteLine("One"); break;
                case 2: Console.WriteLine("Two"); break;
                case 3: Console.WriteLine("Three"); break;
                case 4: Console.WriteLine("Four"); break;
                case 5: Console.WriteLine("Five"); break;
                case 6: Console.WriteLine("Six"); break;
                case 7: Console.WriteLine("Seven"); break;
                case 8: Console.WriteLine("Eight"); break;
                case 9: Console.WriteLine("Nine"); break;

                default: Console.WriteLine("Error: you must enter an integer between 1 and 9"); break;
            }
            Console.ReadLine();
        }
    }
}
```

```
using System;

namespace Microwave
{
    class Microwave
    {
        public static void Main(string[] args)
        {
            int heat=0;

            Console.WriteLine("Enter how many items you wish to heat: ");
            int number=Convert.ToInt32(Console.ReadLine());
            for (int i=0; i<number;i++)
                {
                    Console.WriteLine("What is the heating time of each individual
                    ↪  item: ");
                    heat+=Convert.ToInt32(Console.ReadLine());
                }

            if(number==2)
                {
                Console.WriteLine("Recommended heating time = " + heat*1.5);
                }
            else if(number==3)
                {
                Console.WriteLine("Recommended heating time = " heat*2);
                }
            else if(number>3)
                {
                Console.WriteLine("Heating three items is not recomeneded");
                }

        }
    }
}
```

```
using System;

namespace ConsoleApp1
{
    class DoCasting
    {
        static void Main(string[] args)
        {
            int sum = 17;
            int count = 5;
            int intAverage = sum / count;
            double doubleAverage = 2 * intAverage;
            Console.WriteLine(intAverage);
            Console.WriteLine(doubleAverage);
            Console.WriteLine(Convert.ToDouble(sum/count));
        }
    }
}
```

5 C# Essentials: Repetition

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Build Programs	♦♦◊◊◊

I think these tasks provided a great opportunity to explore the various features of a typical c# program. For me, it offered me the chance to build my own simple programs using selection to ensure certain conditions of the input were met. As such, it demonstrates the capability of the program.

Outcome	**Weight**
Design	♦♦◊◊◊

I think these tasks provided a great opportunity to explore the various features of a typical c# program. For me, it offered me the chance to build my own simple programs using selection to ensure certain conditions of the input were met. As such, it demonstrates the capability of the program.

Outcome	**Weight**
Justify	♦♦◊◊◊

I think these tasks provided a great opportunity to explore the various features of a typical c# program. For me, it offered me the chance to build my own simple programs using selection to ensure certain conditions of the input were met. As such, it demonstrates the capability of the program.

Date	**Author**	**Comment**
2020/03/17 12:39	Dale Orders	Ready to Mark
2020/03/17 22:30	Sanjay Segu	audio comment
2020/03/24 20:18	Sanjay Segu	Could you please share screenshots for the sub tasks which requires you to correct/identify syntactical errors in the program ?
2020/03/28 21:35	Sanjay Segu	Hi Dale
2020/03/28 21:36	Sanjay Segu	Could I please remind you on the screenshots for task 1.2.4 and 1.2.5 ?
2020/03/28 21:45	Sanjay Segu	Discuss
2020/03/29 03:00	Dale Orders	pdf document
2020/03/29 18:05	Sanjay Segu	Good.
2020/03/29 18:05	Sanjay Segu	Complete

Deakin University

Object Oriented Development

OnTrack Submission

C# Essentials: Repetition

Submitted By:
Dale Orders
dorders
2020/03/17 12:39

Tutor:
Sanjay Segu

Outcome	**Weight**
Build Programs	♦♦◊◊◊
Design	♦♦◊◊◊
Justify	♦♦◊◊◊

I think these tasks provided a great opportunity to explore the various features of a typical c# program. For me, it offered me the chance to build my own simple programs using selection to ensure certain conditions of the input were met. As such, it demonstrates the capability of the program.

March 17, 2020

```cs
using System;

namespace repetition
{
    class program
    {

        public static void Main(string[] args)
        {

            /* for loop will loop through a given interval

            int sum = 0;
            int upperbound = 100;

            for(int number=1; number<= upperbound; number++)
            {
                sum += number;
                Console.WriteLine("Current number: " + number + " the sum is " + sum);

            }
            double average = (sum / upperbound);
            Console.WriteLine(average);*/


            /* While loop will check conditions before it executes block
            int sum = 0;
            int upperbound = 100;

            int number = 1;
            while(number<=upperbound)
            {
                sum += number;
                Console.WriteLine("Current number: " + number + " the sum is " + sum);
                number++;
            }

            double average = (sum / upperbound);
            Console.WriteLine(average);*/




            //do while loop will loop through at least once

            int sum = 0;
            int upperbound = 100;

            int number = 1;
            do
            {
```

```
                sum += number;
                Console.WriteLine("Current number: " + number + " the sum is " +
                ↪  sum);
                number++;
            }while(number<=100);

            double average = (sum / upperbound);
            Console.WriteLine(average);

        }
    }
}
```

```
using System;

namespace GuessingNumber
{
    class Program
    {
        public static void Main(string[] args)
        {

            //do while statement to prompt user 2 until they guess correctly

            try
            {
                int guess;
                Console.WriteLine("User 1, enter a number: ");
                int number = Convert.ToInt32(Console.ReadLine());

                do
                {

                    Console.WriteLine("User 2, guess the number set by user 1: ");
                    guess = Convert.ToInt32(Console.ReadLine());

                    if (guess > number)
                    {
                        Console.WriteLine("Your guess is too high");
                    }

                    else if (guess < number)
                    {
                        Console.WriteLine("Your guess is too low");
                    }

                } while (guess != number);

                Console.WriteLine("You have guessed the number! Well done!");
            }

            catch (Exception e)
            {
                Console.WriteLine("Error, you entered in an unacceptable value");

            }

        }
    }
}
```

```
using System;

namespace task1_2DF
{
    class DivisibleFour
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter a number: ");
            int n = Convert.ToInt32(Console.ReadLine());

            for (int i = 1; i < n; i++)
            {
                if (i % 4 == 0 && i % 5 != 0)
                {
                    Console.WriteLine(i);
                }
            }
        }
    }
}
```

6 C# Essentials: Classes and Objects

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Evaluate Code	♦◊◊◊◊

I think this was a good opportunity to build three programs, using key methods to enact specific designs. Overall, I found it gave me greater ability to create programs and experiment with its functionality.

Outcome	Weight
Principles	♦♦◊◊◊

I think this was a good opportunity to build three programs, using key methods to enact specific designs. Overall, I found it gave me greater ability to create programs and experiment with its functionality.

Outcome	Weight
Build Programs	♦♦♦◊◊

I think this was a good opportunity to build three programs, using key methods to enact specific designs. Overall, I found it gave me greater ability to create programs and experiment with its functionality.

Outcome	Weight
Design	♦♦♦◊◊

I think this was a good opportunity to build three programs, using key methods to enact specific designs. Overall, I found it gave me greater ability to create programs and experiment with its functionality.

Outcome	Weight
Justify	♦◊◊◊◊

I think this was a good opportunity to build three programs, using key methods to enact specific designs. Overall, I found it gave me greater ability to create programs and experiment with its functionality.

Date	Author	Comment
2020/03/22 20:42	Dale Orders	Ready to Mark
2020/03/24 22:34	Sanjay Segu	This task will be reviewed shortly
2020/03/28 22:09	Sanjay Segu	audio comment
2020/03/28 22:10	Sanjay Segu	Discuss
2020/03/28 22:10	Sanjay Segu	discussion comment
2020/04/17 16:00	Dale Orders	Hi Sanjay, I finished my discussion
2020/04/19 19:59	Sanjay Segu	Well answered Dale.
2020/04/19 19:59	Sanjay Segu	Complete

Deakin University

Object Oriented Development

OnTrack Submission

C# Essentials: Classes and Objects

Submitted By:
Dale Orders
dorders
2020/04/07 22:18

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦◊◊◊
Build Programs	♦♦♦◊◊
Design	♦♦♦◊◊
Justify	♦◊◊◊◊

I think this was a good opportunity to build three programs, using key methods to enact specific designs. Overall, I found it gave me greater ability to create programs and experiment with its functionality.

April 7, 2020

```
using System;

namespace task2_1
{
    class MobileProgram
    {
        static void Main(string[] args)
        {
            Mobile jimMobile = new Mobile("Monthly", "Samsung Galaxy S6", "07712223344");
            Mobile DaleMobile = new Mobile("Weekly", "iPhone", "03922449383");

            //Jim's Mobile Account
            Console.WriteLine("Account Type: " + jimMobile.getAccType() + "\nMobile Number: " +
            jimMobile.getNumber() + "\nDevice:  " + jimMobile.getDevice() + "\nBalance: " + jimMobile.getBalance());

            jimMobile.addCredit(10.0);
            jimMobile.makeCall(5);
            jimMobile.sendText(2);

            //Dale's Mobile Account
            Console.WriteLine("\n\nAccount Type: " + DaleMobile.getAccType() + "\nMobile Number: " +
            DaleMobile.getNumber() + "\nDevice:  " + DaleMobile.getDevice() + "\nBalance: " + DaleMobile.getBalance());


            DaleMobile.addCredit(15.0);
            DaleMobile.makeCall(10);
            DaleMobile.sendText(4);


            Console.WriteLine();

            Console.ReadLine();
        }
    }
}
```

```
using System;
namespace task2_1
{
    public class Mobile
    {
        //instance variables
        private String accType, device, number;
        private double balance;

        private const double CALL_COST = 0.245;
        private const double TEXT_COST = 0.078;

        //constructor to access instance variables.
        public Mobile(String accType, String device, String number)
        {
            this.accType = accType;
            this.device = device;
            this.number = number;
            this.balance = 0.0;
        }

        public String getAccType()
        {
            return this.accType;
        }


        public String getDevice()
        {
            return this.device;
        }

        public String getNumber()
        {
            return this.number;
        }

        public String getBalance()
        {
            return this.balance.ToString("C");
        }

        public void setAccType(String accType)
        {
            this.accType = accType;
        }

        public void setDevice(String device)
        {
            this.device = device;
        }

        public void setNumber(String number)
```

```
        {
            this.number = number;
        }

        public void setBalance(double balance)
        {
            this.balance = balance;
        }

        //add double amount to balance
        public void addCredit(double amount)
        {
            this.balance += amount;
            Console.WriteLine("Credit added successfully. New balance " + getBalance());
        }

        //deduct cost of making call by minutes
        public void makeCall(int minutes)
        {
            double cost = minutes * CALL_COST;
            this.balance -= cost;
            Console.WriteLine("Call made. New balance " + getBalance());
        }

        //deduct cost of sending a text
        public void sendText(int numTexts)
        {
            double cost = numTexts * TEXT_COST;
            this.balance -= cost;
            Console.WriteLine("Text sent. New balance " + getBalance());
        }

    }
}
```

```
using System;

namespace Task2_1E
{
    public class EmployeeProgram
    {
            static void Main(string[] args)
            {
               Employee Sarah=new Employee("Sarah", 140000);
               Employee Frank=new Employee("Frank", 84000);

            //Sarah's salary
            Console.WriteLine("Employee's name: " + Sarah.getName() + "\nSarah's salary: " + Sarah.getSalary());

            //Frank's Salary
            Console.WriteLine("\nEmployee's name: " + Frank.getName() + "\nFrank's salary: " + Frank.getSalary());

            //raises salary
            Console.WriteLine("\n");
            Sarah.RaiseSalary(50);
            Frank.RaiseSalary(50);

            //deducts tax according to pay scale
            Console.WriteLine("\n");
            Sarah.Tax(140000);
            Frank.Tax(84000);

            Console.WriteLine();
            }

    }
}
```

```
using System;

namespace Task2_1E
{
    class Employee
    {   //instance variables
        private string employeeName;
        private double currentSalary;

        //Constructor
        public Employee(string employeeName, double currentSalary)
        {
            this.employeeName= employeeName;
            this.currentSalary= currentSalary;
        }

        public String getName()
        {
            return this.employeeName;
        }

        public String getSalary()
        {
            return this.currentSalary.ToString("C");
        }

        public void RaiseSalary(double percentage)
        {
            double raise = ((percentage/100) * currentSalary);
            this.currentSalary += raise;
            Console.WriteLine("The salary after the raise is:  " + getSalary());

        }

        public void Tax(double currentSalary)
        {

            if(currentSalary<=18200)
            {
                this.currentSalary = currentSalary;
                Console.WriteLine("The salary after tax is  " + getSalary());
            }

            else if (currentSalary>=18201 && currentSalary <= 37000)
            {
                this.currentSalary = (currentSalary-18200)*0.19;
                Console.WriteLine("The salary after tax is  " + getSalary());
            }

            else if (currentSalary >= 37001 && currentSalary <= 90000)
            {
                this.currentSalary -= (3572 + (0.325 * (currentSalary - 37000)));
                Console.WriteLine("The salary after tax is  " + getSalary());
```

```
            }

            else if (currentSalary >= 90001 && currentSalary <= 180000)
            {
                this.currentSalary -= (20797 + (0.37 * (currentSalary - 90000)));
                Console.WriteLine("The salary after tax is  " + getSalary());
            }

            else if (currentSalary >= 180001)
            {
                this.currentSalary -= (54096 + (0.45 * (currentSalary - 180000)));
                Console.WriteLine("The salary after tax is  " + getSalary());
            }

        }
    }
}
```

```
using System;

namespace _2_1fuel
{
    class CarProgram
    {
            static void Main(string[] args)
        {
            //create objects
            Car ford = new Car(0.1, 40, 100);
            Car toyota = new Car(2, 40, 100);

            //refill car
            ford.addFuel(10.00);
            toyota.addFuel(5.00);


            //drive method is called as total miles <= gallons
            ford.drive(1000);

            //drive method produces error message as total miles >= gallons
            toyota.drive(100000);
        }
    }
}
```

```
using System;
namespace _2_1fuel
{
    public class Car
    {
        //instance variables
        private double efficiency;
        private double fuel;
        private double totalmiles;

        //constant variable
        public const double PetrolCost = 1.385;

        public Car(double efficiency, double fuel, double totalmiles)
        {
            this.efficiency = efficiency;
            this.fuel = 0;
            this.totalmiles = 0;

        }

        public double getFuel()
        {
            return this.fuel;
        }

        public double getTotalMiles()
        {
            return this.totalmiles;
        }


        public void setTotalMiles(double totalmiles)
        {
            this.totalmiles = totalmiles;
        }

        public string printFuelCost(double cost)
        {
            return (cost.ToString("C"));
        }

        //add refill
        public void addFuel(double refill)
        {
            this.fuel += refill;
            Console.WriteLine("The amount of fuel after refill:  " + getFuel());

        }

        //calculate cost by the cost of petrol
        public void calcCost(double fuel)
        {
```

```
            double cost = fuel * PetrolCost;
            Console.WriteLine("The cost of the fuel is:  " + printFuelCost(cost));

        }

        public void convertToLitres(double gallons)
        {
            this.fuel = gallons * 4.546;
            Console.WriteLine("The amount of fuel in litres is:  " + getFuel());
        }

        //set miles and calculate gallons needed.
        public void drive(double miles)
        {

            setTotalMiles(miles);
            double gallons = totalmiles / efficiency;

            if(totalmiles<=gallons)
            {
                convertToLitres(gallons);
                calcCost(this.fuel);
            }

            else
            {
                Console.WriteLine("You do not have enough fuel to drive this distance");
            }

        }

    }

}
```

7 The Account Class

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Build Programs	♦♦◊◊◊

This task helped me to better understand the features and design of a basic c# program. As such, it offered me opportunity to engage with the major components I will need to use in doing such a project in the future.

Outcome	**Weight**
Design	♦♦♦◊◊

This task helped me to better understand the features and design of a basic c# program. As such, it offered me opportunity to engage with the major components I will need to use in doing such a project in the future.

Outcome	**Weight**
Justify	♦♦◊◊◊

This task helped me to better understand the features and design of a basic c# program. As such, it offered me opportunity to engage with the major components I will need to use in doing such a project in the future.

Date	**Author**	**Comment**
2020/03/21 12:14	Dale Orders	Ready to Mark
2020/03/21 21:25	Dale Orders	https://youtu.be/u049CdF58Q8
2020/03/24 21:07	Sanjay Segu	Please ‘NOTE‘ from now on it’s my sincere request to upload the demonstration videos only in Deakin Air and not in YouTube or any other on demand platform. Also, please note ‘Don’t‘ share your solutions to any anybody as it’s plagiarism.
2020/03/26 23:01	Sanjay Segu	Can I please know the reason you have commented class ‘MobileProgram‘ ?
2020/03/26 23:21	Dale Orders	Apologies that was a previous task that I had tested. Forgot to remove it. Will do so and resubmit
2020/03/26 23:34	Sanjay Segu	No need to apologies Dale :)
2020/03/28 21:40	Sanjay Segu	discussion comment
2020/03/28 21:40	Sanjay Segu	Discuss
2020/04/07 22:19	Dale Orders	I’ve removed the commented sections and completed the discussion
2020/04/08 22:56	Sanjay Segu	Complete
2020/04/08 22:56	Sanjay Segu	Thanks.

Deakin University

Object Oriented Development

OnTrack Submission

The Account Class

Submitted By:
Dale Orders
dorders
2020/04/07 22:19

Tutor:
Sanjay Segu

Outcome	**Weight**
Build Programs	♦♦◊◊◊
Design	♦♦♦◊◊
Justify	♦♦◊◊◊

This task helped me to better understand the features and design of a basic c# program. As such, it offered me opportunity to engage with the major components I will need to use in doing such a project in the future.

April 7, 2020

```
using System;

namespace task2_2
{
    class TestAccount
    {
        static void Main(string[] args)
        {
            //objects created for two bank users.
            Account Mary = new Account(89078, "Mary");
            Account John = new Account(78078, "John");

            //examples of deposit and withdrawals from account
            Mary.Deposit(1000);
            John.Withdraw(1000);

            //print bank details
            Mary.Print();
            John.Print();
        }
    }
}
```

```
using System;

namespace task2_2
{
    public class Account
    {
        //instance variables declared
        private decimal _balance;
        private string _name;

        //constructor
        public Account(decimal _balance, string _name)
        {
            this._balance = _balance;
            this._name = _name;
        }

        //prints name and balance
        public void Print()
        {
            Console.WriteLine("The name of the account is " + getName() +
            ↪  "\nStarting Account Balance is " + getBalance());
        }

        //returns name
        public String getName()
        {
            return this._name;
        }

        //returns balance
        public decimal getBalance()
        {
            return this._balance;
        }

        //increases balance by adding deposit
        public void Deposit(decimal amount)
        {
            this._balance += amount;
            Console.WriteLine("Credit added successfully. New balance for " +
            ↪  this._name +" is " + this._balance);
        }

        //decreases balanace by withdrawing the giving input amount
        //does not guard against overdrawing from the account
        public void Withdraw(decimal amount)
        {
            this._balance -= amount;
            Console.WriteLine("Withdrawn undertaken successfully. New balance for "
            ↪  + this._name+ " is " + this._balance);
        }
    }
```

```
}
```

8 The MyTime Class

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊

I thought this was a good task that forced me to think about how I can manipulate data to produce certain outcomes

Outcome	**Weight**
Principles	♦♦◊◊◊

I thought this was a good task that forced me to think about how I can manipulate data to produce certain outcomes

Outcome	**Weight**
Build Programs	♦♦◊◊◊

I thought this was a good task that forced me to think about how I can manipulate data to produce certain outcomes

Outcome	**Weight**
Design	♦◊◊◊◊

I thought this was a good task that forced me to think about how I can manipulate data to produce certain outcomes

Date	**Author**	**Comment**
2020/04/07 22:21	Dale Orders	Ready to Mark
2020/04/08 22:56	Sanjay Segu	Complete
2020/04/08 22:56	Sanjay Segu	Good submission Dale

Deakin University

Object Oriented Development

OnTrack Submission

The MyTime Class

Submitted By:
Dale Orders
dorders
2020/04/07 22:21

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦◊◊◊
Build Programs	♦♦◊◊◊
Design	♦◊◊◊◊

I thought this was a good task that forced me to think about how I can manipulate data to produce certain outcomes

April 7, 2020

```
using System;

namespace task2_3
{
    class TestMyTime
    {
        static void Main(string[] args)
        {

            MyTime test_time1 = new MyTime();
            MyTime test_time2 = new MyTime(22, 59, 45);


            //test invalid time
            test_time1.SetHour(68);
            test_time2.SetTime(88, 43, 21);

            Console.WriteLine("Time: " + test_time2.ToString());

            //valid time
            test_time1.SetTime(4, 4, 4);
            Console.WriteLine("Time: " + test_time1.ToString());

            test_time1.SetHour(5);
            Console.WriteLine("New hour: " + test_time1.GetHour().ToString());

            test_time1.SetMinute(20);
            Console.WriteLine("New minute: " + test_time1.GetMinute().ToString());

            test_time1.SetSecond(49);
            Console.WriteLine("New Second: " + test_time1.GetSecond().ToString());


            //next second, minute, hour for test_time1
            Console.WriteLine("The next second for test_time1 will be: " + test_time1.NextSecond());

            Console.WriteLine("The next minute for test_time1 will be: " + test_time1.NextMinute());

            Console.WriteLine("The next hour for test_time1 will be: " + test_time1.NextHour());



            //previous second, minute, hour for test_time1
            Console.WriteLine("The previous for test_time1 second will be: " + test_time1.PreviousSecond());

            Console.WriteLine("The previous for test_time1 minute will be: " + test_time1.PreviousMinute());

            Console.WriteLine("The previous for test_time1 hour will be: " + test_time1.PreviousHour());
```

```
            //next second, minute, hour for test_time2
            Console.WriteLine("The next second for test_time2 will be: " + test_time2.NextSecond());

            Console.WriteLine("The next minute for test_time2 will be: " + test_time2.NextMinute());

            Console.WriteLine("The next hour for test_time2 will be: " + test_time2.NextHour());



            //previous second, minute, hour for test_time2
            Console.WriteLine("The previous second for test_time2 will be: " + test_time2.PreviousSecond());

            Console.WriteLine("The previous minute for test_time2 will be: " + test_time2.PreviousMinute());

            Console.WriteLine("The previous hour for test_time2 will be: " + test_time2.PreviousHour());


        }
    }
}
```

```
using System;
namespace task2_3
{
    public class MyTime
    {
        //instance variables
        private int hour;
        private int minute;
        private int second;

        //default constructor
        public MyTime()
        {
            hour = 0;
            minute = 0;
            second = 0;
        }
        //constructor with three arguements
        public MyTime(int hour, int minute, int second)
        {
            //conditional statement to assess validity of input
            if (hour >= 0 && hour <= 24 && minute >= 0 && minute <= 59 && second >= 0 && second <= 59)
            {
                this.hour = hour;
                this.minute = minute;
                this.second = second;
            }

            else
            {
                //print error message
                Console.WriteLine("Invalid Time");
            }
        }

        public void SetTime(int hour, int minute, int second)
        {
            //conditional statement to assess validity of input
            if (hour >= 0 && hour <= 24 && minute >= 0 && minute <= 59 && second >= 0 && second <= 59)
            {
                this.hour = hour;
                this.minute = minute;
                this.second = second;
            }

            else
            {
                Console.WriteLine("Invalid Time");
            }
        }

```

```


        public void SetHour(int hour)
        {
            //set hour parameters
            if (hour >= 0 && hour <= 23)
            {
                this.hour = hour;
            }
            else
            {
                Console.WriteLine("Invalid hour");
            }

        }

        public void SetMinute(int minute)
        {
            //set minute parameters
            if (minute >= 0 && minute <= 59)
            {
                this.minute = minute;
            }
            else
            {
                Console.WriteLine("Invalid minute");
            }

        }

        public void SetSecond(int second)
        {
            //set second parameters
            if (second >= 0 && second <= 59)
            {
                this.second = second;
            }
            else
            {
                Console.WriteLine("Invalid second");
            }

        }

        public int GetHour()
        {
            return this.hour;
        }

        public int GetMinute()
        {
            return this.minute;
        }
```

```

        public int GetSecond()
        {
            return this.second;
        }

        public override string ToString()
        {
            return String.Format("{0:00}:{1:00}:{2:00}", this.hour, this.minute,
             ↪  this.second);
        }


        //Next Second Method
        public MyTime NextSecond()
        {
            int next_second = this.second + 1;
            int next_minute = this.minute;
            int next_hour = this.hour;

            if (next_second >= 60)
            {
                next_second = 0;
                next_minute += 1;
            }

            if (next_minute >= 60)
            {
                next_minute = 0;
                next_hour += 1;
            }
            if (next_hour >= 24)
            {
                next_hour = 0;
            }

            MyTime nxtsec = new MyTime(next_hour, next_minute, next_second);
            return nxtsec;

        }

        public MyTime NextMinute()
        {
            int next_minute = this.minute + 1;
            int next_hour = this.hour;

            if (next_minute >= 60)
            {
                next_minute = 0;
                next_hour += 1;
            }
            if (next_hour >= 24)
            {
```

```
                next_hour = 0;
            }

            MyTime nxtmin = new MyTime(next_hour, next_minute, this.second);
            return nxtmin;

        }

        public MyTime NextHour()
        {
            int next_hour = this.hour + 1;

            if (next_hour >= 24)
            {
                next_hour = 0;

            }

            MyTime nxthour = new MyTime(next_hour, this.minute, this.second);
            return nxthour;

        }

        public MyTime PreviousSecond()
        {
            int previous_second = this.second - 1;
            int previous_minute = this.minute;
            int previous_hour = this.hour;

            if (previous_second < 0)
            {
                previous_second = 59;
                previous_minute -= 1;
            }

            if (previous_minute < 0)
            {
                previous_minute = 59;
                previous_hour -= 1;
            }
            if (previous_hour < 0)
            {
                previous_hour = 23;
            }

            MyTime prevsec = new MyTime(previous_hour, previous_minute,
            ↪ previous_second);
            return prevsec;

        }

        public MyTime PreviousMinute()
        {
```

```
            int previous_minute = this.minute - 1;
            int previous_hour = this.hour;

            if (previous_minute < 0)
            {
                previous_minute = 59;
                previous_hour -= 1;
            }
            if (previous_hour < 0)
            {
                previous_hour = 23;
            }

            MyTime prevmin = new MyTime(previous_hour, previous_minute,
            ↪  this.second);
            return prevmin;

        }

        public MyTime PreviousHour()
        {
            int previous_hour = this.hour - 1;

            if (previous_hour < 0)
            {
                previous_hour = 23;
            }

            MyTime prevhour = new MyTime(previous_hour, this.minute, this.second);
            return prevhour;

        }

    }
}
```

9 C# Essentials: Arrays and Lists

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊

I think this is the best task we have done so far, even though it was long. I like the way each sections builds on the one prior. I wish all tasks were like this.

Outcome	**Weight**
Principles	♦♦◊◊◊

I think this is the best task we have done so far, even though it was long. I like the way each sections builds on the one prior. I wish all tasks were like this.

Outcome	**Weight**
Build Programs	♦♦♦◊◊

I think this is the best task we have done so far, even though it was long. I like the way each sections builds on the one prior. I wish all tasks were like this.

Outcome	**Weight**
Design	♦♦◊◊◊

I think this is the best task we have done so far, even though it was long. I like the way each sections builds on the one prior. I wish all tasks were like this.

Outcome	**Weight**
Justify	♦◊◊◊◊

I think this is the best task we have done so far, even though it was long. I like the way each sections builds on the one prior. I wish all tasks were like this.

Date	**Author**	**Comment**
2020/04/07 22:23	Dale Orders	Ready to Mark
2020/04/08 22:55	Sanjay Segu	Demonstrate
2020/04/17 16:00	Dale Orders	Hi Sanjay. Deakin Air was not allowin me to upload today so I had to upload to youtube.
2020/04/17 16:00	Dale Orders	https://www.youtube.com/watch?v=RxOl8i6y5rg&feature=youtu.be
2020/04/19 20:00	Sanjay Segu	Complete
2020/04/19 20:00	Sanjay Segu	Liked it.

Deakin University

Object Oriented Development

OnTrack Submission

C# Essentials: Arrays and Lists

Submitted By:
Dale Orders
dorders
2020/04/07 22:23

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦◊◊◊
Build Programs	♦♦♦◊◊
Design	♦♦◊◊◊
Justify	♦◊◊◊◊

I think this is the best task we have done so far, even though it was long. I like the way each sections builds on the one prior. I wish all tasks were like this.

April 7, 2020

```
using System;
using System.Collections.Generic;

namespace task3_1
{
    class Program
    {

        static void Main(string[] args)
        {

            Console.WriteLine("-------------------------------------------------------------------------------");
            Console.WriteLine("------------------------------Task One-----------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------");
            double[] myArray = new double[10];

            myArray[0] = 1.0;
            myArray[1] = 1.1;
            myArray[2] = 1.2;
            myArray[3] = 1.3;
            myArray[4] = 1.4;
            myArray[5] = 1.5;
            myArray[6] = 1.6;
            myArray[7] = 1.7;
            myArray[8] = 1.8;
            myArray[9] = 1.9;


            Console.WriteLine("The first element in myArray is " + myArray[0]);
            Console.WriteLine("The second element in myArray is " + myArray[1]);
            Console.WriteLine("The third element in myArray is " + myArray[2]);
            Console.WriteLine("The fourth element in myArray is " + myArray[3]);
            Console.WriteLine("The fifth element in myArray is " + myArray[4]);
            Console.WriteLine("The sixth element in myArray is " + myArray[5]);
            Console.WriteLine("The seventh element in myArray is " + myArray[6]);
            Console.WriteLine("The eighth element in myArray is " + myArray[7]);
            Console.WriteLine("The ninth element in myArray is " + myArray[8]);
            Console.WriteLine("The tenth element in myArray is " + myArray[9]);

            Console.WriteLine("\n\n-----------------------------------------------------------------------------");
            Console.WriteLine("------------------------------Task Two-----------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------");
            int[] my_Array = new int[10];

            for (int i = 0; i < my_Array.Length; i++)
            {
                my_Array[i] = i;
```

```
            }
            for (int i = 0; i < my_Array.Length; i++)
            {
                Console.WriteLine("The element at position " + i + " in the array is " + my_Array[i]);
            }
            Console.WriteLine("\n\n-------------------------------------------------------------------------------------");
            Console.WriteLine("------------------------------Task Three-----------------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------------");



            int[] studentArray = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
            int total = 0;

            for (int i = 0; i < studentArray.Length; i++)
            {
                total += studentArray[i];
            }

            Console.WriteLine("The total marks for the student is " + total);
            Console.WriteLine("This consists of " + studentArray.Length + " marks");
            Console.WriteLine("Therefore the average mark is " + (total / studentArray.Length));

            Console.WriteLine("\n\n-------------------------------------------------------------------------------------");
            Console.WriteLine("------------------------------Task Four------------------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------------");

            //assigns names to the array
            string[] studentNames = new string[6];

            for (int i = 0; i < studentNames.Length; i++)
            {
                try
                {
                    Console.WriteLine("Enter in the name of a student: ");
                    studentNames[i] = Console.ReadLine();
                }
                catch (FormatException)
                {
                    Console.WriteLine("You need to enter a name.");
                }
            }

            //prints names stored within the array
            for (int i = 0; i < studentNames.Length; i++)
```

```
            {
                Console.Write(studentNames[i] + "  ");
            }

            Console.WriteLine("\n\n-----------------------------------------------------------------------------------");
            Console.WriteLine("------------------------------Task Five-----------------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------------");
            double[] numbers = new double[10];
            double currentLargest, currentSmallest;

            try
            {
                for (int i = 0; i < numbers.Length; i++)
                {
                    Console.WriteLine("Enter in a double: ");
                    numbers[i] = Convert.ToDouble(Console.ReadLine());
                }

            }

            catch (FormatException)
            {
                Console.WriteLine("That is not a number");
            }

            currentLargest = numbers[0];

            for (int i = 0; i < numbers.Length; i++)
            {
                if (currentLargest < numbers[i])
                {
                    currentLargest = numbers[i];
                    Console.WriteLine();
                }
            }


            Console.WriteLine("\nThe largest value in the array is " + currentLargest);

            currentSmallest = numbers[0];

            for (int i = 0; i < 10; i++)
            {
                if (currentSmallest > numbers[i])
                {
                    currentSmallest = numbers[i];
                }
            }

```

```
            Console.WriteLine("\nThe smallest value in the array is " + currentSmallest);

            Console.WriteLine("\n\n-------------------------------------------------------------------------------------");
            Console.WriteLine("-------------------------------Task Six-----------------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------------");

            int[,] myArray1 = new int[3, 4] { { 1, 2, 3, 4 }, { 1, 1, 1, 1 }, { 2, 2, 2, 2 } };

            for (int i = 0; i < myArray1.GetLength(0); i++)
            {
                for (int j = 0; j < myArray1.GetLength(1); j++)
                {
                    Console.Write(myArray1[i, j] + "\t");
                }
                Console.WriteLine();
            }

            Console.WriteLine("\n");
            Console.WriteLine("-------------------------------------------------------------------------------------");


            List<String> myStudentList = new List<String>();
            Random randomValue = new Random();
            int randomNumber = randomValue.Next(1, 12);
            Console.WriteLine("You now need to add " + randomNumber + " students to your class list");

            for (int i = 0; i < randomNumber; i++)
            {
                Console.WriteLine("Please enter the name of Student " + (i + 1) + ": ");
                myStudentList.Add(Console.ReadLine());
                Console.WriteLine();
            }


            Console.Write("The name of the students in my list are ");
            for (int i = 0; i < myStudentList.Count; i++)
            {
                Console.Write(myStudentList[i] + " , ");
            }
            Console.WriteLine("\n\n-------------------------------------------------------------------------------------");
            Console.WriteLine("-------------------------------Task Seven---------------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------------");
```

```

            Console.WriteLine("How many elements would you like to enter");
            int elements = Int32.Parse(Console.ReadLine());
            int[] number = new int[elements];

            Console.WriteLine("Enter integers in your array: ");
            //populate Array

            for (int i = 0; i < number.GetLength(0); i++)
            {

                number[i] = Convert.ToInt32(Console.ReadLine());
            }


            int FuncOne(int[] number)
            {
                int total_even = 0;
                int total_odd = 1;
                int number_FuncOne = 1;
                if (number.Length > 10)
                {
                    for (int i = 0; i < number.Length; i++)
                    {
                        if (number[i] % 2 == 0)
                            total_even++;
                    }
                    Console.Write("The total of the even numbers is " + total_even);
                    return total;
                }

                else if (number.Length < 10)
                {
                    for (int i = 1; i < number.Length; i++)
                    {
                        if (number[i] % 2 == 1)
                            total_odd *= number[i];
                    }
                    Console.Write("The product of the odd numbers is " +
                    ↪ number_FuncOne);
                    return number_FuncOne;
                }

                else
                {
                    throw new ArgumentException("Invalid entry");
                }

            }

            FuncOne(number);
            Console.WriteLine("\n\n-----------------------------------------------
            ↪ -------------------------------------");
```

```
            List<double> list_one = new List<double> { 1.2, 3.4, 6.5, 7.2 };

            double FuncTwo(List<double> list_one)
            {
                double total_list = 0;
                for (int i = 0; i < list_one.Count; i++)
                {
                    total_list += list_one[i];
                }
                double average = total_list / list_one.Count;
                Console.WriteLine("The average is " + average);
                Console.WriteLine("The calcuation of each element minus the average is: ");

                for (int i = 0; i < list_one.Count; i++)
                {
                    list_one[i] = list_one[i] - average;
                    Console.Write(list_one[i] + "\t");
                }

                return 1;
            }

            FuncTwo(list_one);

            Console.WriteLine("\n\n-------------------------------------------------------------------------------------");
            Console.WriteLine("-------------------------------Task Eight-------------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------------------------------------------------");



            int[,] FuncTree = new int[3, 4];
            Console.WriteLine("Enter elements for a 3 x 4 2 dimensional array");

            //populate Array
            for (int i = 0; i < FuncTree.GetLength(0); i++)
            {
                for (int j = 0; j < FuncTree.GetLength(1); j++)
                {
                    FuncTree[i, j] = Convert.ToInt32(Console.ReadLine());
                }
            }

            //assign multiples of three to 1 dimensional array
            for (int i = 0; i < FuncTree.GetLength(0); i++)
            {
                for (int j = 0; j < FuncTree.GetLength(1); j++)
                {
```

```
                    int element = FuncTree[i, j];
                    if (element % 3 == 0)
                    {
                        List<int> list = new List<int>();
                        list.Add(element);
                        int[] array = new int[element];
                        Console.WriteLine("The number {0} in this array is a multiple of 3: ", element);

                    }

                }

            }


            Console.WriteLine("\n\n-------------------------------------------------------------------------------");
            Console.WriteLine("------------------------------Task Nine-----------------------------------------------");
            Console.WriteLine("-------------------------------------------------------------------------------------");




            int[,] FuncFour(int[] input)
            {

                int[,] range = new int[input.Length, 5];
                for (int i = 0; i < input.Length; i++)
                {
                    for (int j = 1; j <= 5; j++)
                    {
                        range[i, j - 1] = input[i] * j;
                    }
                }
                return range;

            }

            int[] NewArray = { 1, 2, 3, 4, 5 };
            int[,] range = FuncFour(NewArray);

            Console.WriteLine(" \t1\t2\t3\t4\t5");
            for (int i = 0; i < range.GetLength(1); i++)
            {

                Console.Write(NewArray[i] + "\t");
                for (int j = 0; j < 5; j++)
                {
                    Console.Write(range[i, j] + "\t");
                }
```

```
                Console.WriteLine();
            }
            Console.WriteLine();
        }

    }

    }
```

10 Validating Accounts

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊

I like the way in which we are continuing to develop the bank system program. It shows increasing complexity.

Outcome	**Weight**
Principles	♦♦♦◊◊

I like the way in which we are continuing to develop the bank system program. It shows increasing complexity.

Outcome	**Weight**
Build Programs	♦♦◊◊◊

I like the way in which we are continuing to develop the bank system program. It shows increasing complexity.

Outcome	**Weight**
Design	♦♦◊◊◊

I like the way in which we are continuing to develop the bank system program. It shows increasing complexity.

Outcome	**Weight**
Justify	♦◊◊◊◊

I like the way in which we are continuing to develop the bank system program. It shows increasing complexity.

Date	**Author**	**Comment**
2020/04/07 22:29	Dale Orders	Ready to Mark
2020/04/11 16:34	Sanjay Segu	Complete
2020/04/11 16:34	Sanjay Segu	Good submission Dale

Deakin University

Object Oriented Development

OnTrack Submission

Validating Accounts

Submitted By:
Dale Orders
dorders
2020/04/07 22:29

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦♦◊◊
Build Programs	♦♦◊◊◊
Design	♦♦◊◊◊
Justify	♦◊◊◊◊

I like the way in which we are continuing to develop the bank system program. It shows increasing complexity.

April 7, 2020

```
using System;
using task2_2;
//using task2_2;

namespace task3_2
{


    class BankSystem
    {

        enum MenuOption
        {
            Withdraw,
            Deposit,
            Print,
            Quit
        }

        static void DoDeposit(Account account)
        {
            Boolean result;
            Console.WriteLine("Enter deposit amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            result = account.Deposit(amount);

            if(result==true)
            {
                Console.WriteLine("Your transaction was successful");
            }
            else
            {
                Console.WriteLine("Your transaction was unsuccessful");
            }
        }

        static void DoWithdrawal(Account account)
        {
            Boolean result;
            Console.WriteLine("Enter withdrawal amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            result = account.Withdraw(amount);

            if (result == true)
            {
                Console.WriteLine("Your withdrawal was successful");
            }
            else if (result==false)
            {
                Console.WriteLine("Your withdrawal was unsuccessful");
            }
        }

```

```

        static MenuOption ReadUserOption()
        {
            int? option = null;
            do
            {
                Console.WriteLine("Please select from the following options");
                Console.WriteLine("MENU \n1. Withdraw \n2. Deposit \n3. Print \n4. Quit");

                try
                {
                    option = Convert.ToInt32(Console.ReadLine());
                    if (option > 4 || option < 1)
                    {
                        option = null;
                        Console.WriteLine("Please enter a number from 1 to 4 from the menut");
                    }
                }
                catch(FormatException)
                {
                    Console.WriteLine("Invalid input. Enter an integer from 1-4");
                    }

            } while (option == null);

            return (MenuOption)option;
        }


        static void DoPrint(Account account)
        {

            account.Print();

        }




        static void Main(string[] args)
        {


            Account person_one = new Account(400, "Lucy");
            Account person_two = new Account(1000, "Michael");
            MenuOption option;

            do
            {
                option = ReadUserOption()-1;

```

```
                switch (option)
                {
                    case MenuOption.Withdraw:
                        Console.WriteLine("You have selected withdraw");
                        Console.WriteLine("---Lucy's Account---");
                        DoWithdrawal(person_one);
                        Console.WriteLine("---Michael's Account---");
                        DoWithdrawal(person_two);
                        break;
                    case MenuOption.Deposit:
                        Console.WriteLine("You have selected deposit");
                        Console.WriteLine("---Lucy's Account---");
                        DoDeposit(person_one);
                        Console.WriteLine("---Michael's Account---");
                        DoDeposit(person_two);
                        break;
                    case MenuOption.Print:
                        Console.WriteLine("You have selected print");
                        Console.WriteLine("---Lucy's Account---");
                        DoPrint(person_one);
                        Console.WriteLine("---Michael's Account---");
                        DoPrint(person_two);
                        break;
                    case MenuOption.Quit:
                        Console.WriteLine("Goodbye");
                        break;

                }
            } while (option != MenuOption.Quit);
        }



    }
}
```

```
using System;

namespace task2_2
{
    public class Account
    {
        //instance variables declared
        private decimal _balance;
        private string _name;

        //constructor
        public Account(decimal balance, string name)
        {

            this._name = name;
            if (balance <= 0)
                return;
            this._balance = balance;

        }

        //prints name and balance
        public void Print()
        {
            Console.WriteLine("The name of the account holder is " + getName() +
            ↪  "\nCurrent Account Balance is " + getBalance());
        }

        //returns name
        public String getName()
        {
            return this._name;
        }

        //returns balance
        public decimal getBalance()
        {
            return this._balance;
        }

        //increases balance by adding deposit
        //boolean ensure that no 0 or a negative value can not be input
        public Boolean Deposit(decimal amount)
        {

            if (amount <= 0)
                return false;

                this._balance += amount;
                //Console.WriteLine("Credit added successfully. New balance for " +
                ↪  this._name + " is " + this._balance);
                return true;

```

```
        }


        //decreases balanace by withdrawing the giving input amount
        //boolean values protect against overdrawing from account
        public Boolean Withdraw(decimal amount)
        {
            if (amount > this._balance || amount < 0)
                return false;


                this._balance -= amount;
                //Console.WriteLine("Withdrawal successfull. New balance for " + this._name + " is " + this._balance);
                return true;

        }

    }
}
```

11 The MyPolynomial class

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊

This was a challenging task, but it shows how important it is to plan the program out in advance.

Outcome	**Weight**
Principles	♦♦◊◊◊

This was a challenging task, but it shows how important it is to plan the program out in advance.

Outcome	**Weight**
Build Programs	♦♦◊◊◊

This was a challenging task, but it shows how important it is to plan the program out in advance.

Outcome	**Weight**
Design	♦◊◊◊◊

This was a challenging task, but it shows how important it is to plan the program out in advance.

Outcome	**Weight**
Justify	♦◊◊◊◊

This was a challenging task, but it shows how important it is to plan the program out in advance.

Date	**Author**	**Comment**
2020/04/07 22:31	Dale Orders	Ready to Mark
2020/04/11 16:34	Sanjay Segu	Demonstrate
2020/04/11 16:35	Sanjay Segu	A demonstration video would be helpful to provide you with a tailored feedback
2020/04/18 17:53	Dale Orders	https://video.deakin.edu.au/media/t/0_bobp0dhz
2020/04/19 20:07	Sanjay Segu	Complete
2020/04/19 20:07	Sanjay Segu	Good efforts Dale
2020/04/19 20:07	Sanjay Segu	Your code looks solid

Deakin University

Object Oriented Development

OnTrack Submission

The MyPolynomial class

Submitted By:
Dale Orders
dorders
2020/04/07 22:31

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦◊◊◊
Build Programs	♦♦◊◊◊
Design	♦◊◊◊◊
Justify	♦◊◊◊◊

This was a challenging task, but it shows how important it is to plan the program out in advance.

April 7, 2020

```
using System;
using System.Collections.Generic;
using System.Text;

namespace task3_3
{
    class TestmyPolynomial
    {
        static void Main(string[] args)
        {
            MyPolynomial polynomial_1 = new MyPolynomial(new double[] { 6, 0, 4, 2, 6, 5, 5 });
            MyPolynomial polynomial_2 = new MyPolynomial(new double[] { 5, 3, 3, 2, -1, 0, 2 });

            Console.WriteLine("Polynomial 1 is: " + polynomial_1.ToString());
            Console.WriteLine("Polynomial 1 is: " + polynomial_2.ToString());
            Console.WriteLine("The degree of polynomial 1 is: " + polynomial_1.GetDegree());
            Console.WriteLine("The degree of polynomial 2 is: " + polynomial_2.GetDegree());
            Console.WriteLine("The evaluation of polynomial 1 at 4 : " + polynomial_1.Evaluate(4));
            Console.WriteLine("The evaluation of polynomial 1 at 4 : " + polynomial_2.Evaluate(4));
            Console.WriteLine("The multiplication of the two polynomials is: " + polynomial_1.Multiply(polynomial_2));
            Console.WriteLine("The addition of the two polynomials is: " + polynomial_1.Add(polynomial_2));

        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
namespace task3_3
{
    public class MyPolynomial
    {
        private double[] _coeffs;

        public MyPolynomial(double[] array_coeffs)
        {

            _coeffs = array_coeffs;

        }

        public int GetDegree()
        {
            return _coeffs.Length - 1;
        }

        // Create a string representation
        public override string ToString()
        {
            //null string
            string poly = null;

            for (int i = _coeffs.Length - 1; i >= 0; i--)
            {
                if (_coeffs[i] == 0) continue;

                //if _coeffs[i]> greater than zero use "+"
                //if _coeffs[i]> less than zero use "-"
                poly += $"{(_coeffs[i] > 0 ? "+" : "-")}";
                if (i == 0 || _coeffs[i] != 1)
                {
                    poly += Math.Abs(_coeffs[i]);
                }

                if (i == 1)
                {
                    poly += "x";
                }
                //raise to the value of i
                else if (i > 0)
                {
                    poly += $"x^{i}";
                }
            }
            //remove addition from the start of the polynomial
            return poly.Trim(' ', '+');
        }

```

```

        //evaluate polynomial
        public double Evaluate(double x)
        {

            double result=0.0;

            for (int i = 0; i < _coeffs.Length; i++)
            {
                result += _coeffs[i] * Math.Pow(x, i);
            }

            return result;
        }


        public MyPolynomial Multiply(MyPolynomial another)
        {
            int degree = GetDegree() + another.GetDegree();
            var result = new double[degree + 1];
            for(int i=0; i<_coeffs.Length;i++)
            {
                if (_coeffs[i] == 0) continue;
                for(int j=0;j<another._coeffs.Length;j++)
                {
                    if (another._coeffs[j] == 0) continue;
                    result[i + j] += _coeffs[i] + another._coeffs[j];
                }
            }
            _coeffs = result;
            return this;
        }

        public MyPolynomial Add(MyPolynomial another)
        {

            double[] result = _coeffs;
            double[] array = another._coeffs;
            if (another.GetDegree() > GetDegree())
            {
                result = another._coeffs;
                array = _coeffs;
            }

            for (var i = 0; i < array.Length; i++)
            {
                result[i] += array[i];
            }

            _coeffs = result;
            return this;


```

```
            }


        }



}
```

12 Bucket Sort

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Evaluate Code	♦♦◊◊◊

This task really tested my ability to think through a problem in a logical manner. I how this task challenged me.

Outcome	Weight
Principles	♦♦♦◊◊

This task really tested my ability to think through a problem in a logical manner. I how this task challenged me.

Outcome	Weight
Build Programs	♦♦♦◊◊

This task really tested my ability to think through a problem in a logical manner. I how this task challenged me.

Outcome	Weight
Design	♦♦♦◊◊

This task really tested my ability to think through a problem in a logical manner. I how this task challenged me.

Date	Author	Comment
2020/04/23 02:16	Dale Orders	Ready to Mark
2020/04/25 22:30	Sanjay Segu	Demonstrate
2020/04/25 22:30	Sanjay Segu	Code looks ok Dale
2020/04/25 22:30	Sanjay Segu	Can I request you to make a demonstration video for this one ?
2020/05/05 20:28	Dale Orders	https://youtu.be/GNaCwZEpz7I
2020/05/05 21:41	Sanjay Segu	Discuss
2020/05/05 21:42	Sanjay Segu	What determines the size of bucket.
2020/05/08 18:26	Dale Orders	the createbucket method. It takes (int b) as a parameter then creates a list of size, b,. It iterates through the length of the list and creates individual buckets. It return the list as buckets
2020/05/10 19:18	Sanjay Segu	Complete
2020/05/10 19:18	Sanjay Segu	:+1:

Deakin University

Object Oriented Development

OnTrack Submission

Bucket Sort

Submitted By:
Dale Orders
dorders
2020/04/23 02:16

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦♦◊◊◊
Principles	♦♦♦◊◊
Build Programs	♦♦♦◊◊
Design	♦♦♦◊◊

This task really tested my ability to think through a problem in a logical manner. I how this task challenged me.

April 23, 2020

```
using System;
using System.Collections.Generic;
using System.Linq;


namespace task3_4
{


    class Program
    {

        enum MenuOption
        {
            Withdraw,
            Deposit,
            Print,
            Quit
        }

        static void DoDeposit(Account account)
        {
            Boolean result;
            Console.WriteLine("Enter deposit amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            result = account.Deposit(amount);

            if (result == true)
            {
                Console.WriteLine("Your transaction was successful");
            }
            else
            {
                Console.WriteLine("Your transaction was unsuccessful");
            }
        }

        static void DoWithdrawal(Account account)
        {
            Boolean result;
            Console.WriteLine("Enter withdrawal amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            result = account.Withdraw(amount);

            if (result == true)
            {
                Console.WriteLine("Your withdrawal was successful");
            }
            else if (result == false)
            {
                Console.WriteLine("Your withdrawal was unsuccessful");
            }
        }
```

```


        static MenuOption ReadUserOption()
        {
            int? option = null;
            do
            {
                Console.WriteLine("Please select from the following options");
                Console.WriteLine("MENU \n1. Withdraw \n2. Deposit \n3. Print \n4. Quit");

                try
                {
                    option = Convert.ToInt32(Console.ReadLine());
                    if (option > 4 || option < 1)
                    {
                        option = null;
                        Console.WriteLine("Please enter a number from 1 to 4 from the menut");
                    }
                }
                catch (FormatException)
                {
                    Console.WriteLine("Invalid input. Enter an integer from 1-4");
                }

            } while (option == null);

            return (MenuOption)option;
        }


        static void DoPrint(Account account)
        {

            account.Print();

        }

        static void PrintAccountArray(Account[] accounts)
        {
            foreach (Account account in accounts)
                account.Print();
        }




        static void Main(string[] args)
        {


            Account[] accounts_Array = new Account[3];
```

```
            accounts_Array[0] = new Account(Convert.ToDecimal(8394.43), "Mary Stevens");
            accounts_Array[1] = new Account(Convert.ToDecimal(8492.20), "David Marks");
            accounts_Array[2] = new Account(Convert.ToDecimal(4843.90), "Jack Todd");
            Console.WriteLine("Accounts before sorting");
            PrintAccountArray(accounts_Array);
            Console.WriteLine("Accounts after sorting");
            AccountsSorter.Sort(accounts_Array, 3);
            PrintAccountArray(accounts_Array);

            Console.WriteLine("----------------------------------");
            List<Account> accounts_List = new List<Account>();
            accounts_List.Add(new Account(Convert.ToDecimal(37493.50), "Sam Newton"));
            accounts_List.Add(new Account(Convert.ToDecimal(84084.70), "James Evans"));
            accounts_List.Add(new Account(Convert.ToDecimal(484.70), "Kylie Denis"));

            Console.WriteLine("Accounts before sorting");
            PrintAccountArray(accounts_List.ToArray());

            Console.WriteLine("Accounts after sorting");
            AccountsSorter.Sort(accounts_List, 3);
            PrintAccountArray(accounts_List.ToArray());

            Account person_one = new Account(400, "Lucy");
            Account person_two = new Account(1000, "Michael");
            MenuOption option;

            do
            {
                option = ReadUserOption() - 1;

                switch (option)
                {
                    case MenuOption.Withdraw:
                        Console.WriteLine("You have selected withdraw");
                        Console.WriteLine("---Lucy's Account---");
                        DoWithdrawal(person_one);
                        Console.WriteLine("---Michael's Account---");
                        DoWithdrawal(person_two);
                        break;
                    case MenuOption.Deposit:
                        Console.WriteLine("You have selected deposit");
                        Console.WriteLine("---Lucy's Account---");
                        DoDeposit(person_one);
                        Console.WriteLine("---Michael's Account---");
                        DoDeposit(person_two);
                        break;
```

```
                    case MenuOption.Print:
                        Console.WriteLine("You have selected print");
                        Console.WriteLine("---Lucy's Account---");
                        DoPrint(person_one);
                        Console.WriteLine("---Michael's Account---");
                        DoPrint(person_two);
                        break;
                    case MenuOption.Quit:
                        Console.WriteLine("Goodbye");
                        break;

                }
            } while (option != MenuOption.Quit);
        }



    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;


namespace task3_4
{
    static class AccountsSorter
    {

    private static decimal MaximumBalance(Account[] accounts)
        {
            return accounts.Max(a => a.Balance);
        }

    private static List<Account>[] CreateBuckets(int b)
        {
            List<Account>[] buckets = new List<Account>[b];
            for(int i=0;i<buckets.Length;i++)
            {
                buckets[i] = new List<Account>();
            }
            return buckets;
        }

        private static void DistributeAccounts(Account[] accounts, List<Account>[] buckets)
        {
            decimal maximum = MaximumBalance(accounts);
            foreach (Account account in accounts)
            {
                int bucket = (int)(Math.Floor(buckets.Length * account.Balance / maximum));
                if (bucket == buckets.Length)
                    bucket -= 1;
                buckets[bucket].Add(account);

            }
        }

        private static void SortBuckets(List<Account>[] buckets)
        {
            for (int i = 0; i < buckets.Length; i++)
            {
                buckets[i] = buckets[i].OrderBy(a => a.Balance).ToList();
            }
        }

        public static void Sort(Account[] accounts, int b)

        {
            if(accounts==null)
            {
```

```
                throw new NullReferenceException("Accounts can not be null");
            }
            if(b<=1)
            {
                throw new ArgumentOutOfRangeException("buckets can not be less than 2");
            }


            List<Account>[] buckets = CreateBuckets(b);
            DistributeAccounts(accounts, buckets);
            SortBuckets(buckets);

            int idx = 0;
            for (int i = 0; i < buckets.Length; i++)
            {
                foreach (Account account in buckets[i])
                {
                    accounts[idx] = account;
                    idx++;

                }

                //Console.WriteLine();

            }


        }

        public static void Sort(List<Account> accounts, int b)
        {
            if(accounts==null)
            {
                throw new NullReferenceException("Accounts cannot be null");
            }
            Account[] accountsArray = accounts.ToArray();
            Sort(accountsArray, b);

            for(int i=0; i<accounts.Count;i++)
            {
                accounts[i] = accountsArray[i];
            }

           // Console.WriteLine();
        }

    }
}



//class BucketSort
```

```
//{
//    public static List<int> Sort(params int[] x)
//    {
//        List<int> sortedArray = new List<int>();

//        int numOfBuckets = 10;

//        //Create buckets
//        List<int>[] buckets = new List<int>[numOfBuckets];
//        for (int i = 0; i < numOfBuckets; i++)
//        {
//            buckets[i] = new List<int>();
//        }

//        //Iterate through the passed array and add each integer to the appropriate bucket
//        for (int i = 0; i < x.Length; i++)
//        {
//            int bucket = (x[i] / numOfBuckets);
//            buckets[bucket].Add(x[i]);
//        }

//        //Sort each bucket and add it to the result List
//        for (int i = 0; i < numOfBuckets; i++)
//        {
//            List<int> temp = InsertionSort(buckets[i]);
//            sortedArray.AddRange(temp);
//        }
//        return sortedArray;
//    }

//    //Insertion Sort
//    public static List<int> InsertionSort(List<int> input)
//    {
//        for (int i = 1; i < input.Count; i++)
//        {
//            //2. Store the current value in a variable
//            int currentValue = input[i];
//            int pointer = i - 1;

//            //3. As long as we are pointing to a valid value in the array...
//            while (pointer >= 0)
//            {
//                //4. If the current value is less than the value we are pointing at...
//                if (currentValue < input[pointer])
//                {
//                    //5. Move the pointed-at value up one space, and insert the current value at the pointed-at position.
//                    input[pointer + 1] = input[pointer];
//                    input[pointer] = currentValue;
//                }
//                else break;
```

```
//            }
//        }

//        return input;
//    }
//    static void Main(string[] args)
//    {
//        int[] array = new int[] { 43, 17, 87, 92, 31, 6, 96, 13, 66, 62, 4 };

//        Console.WriteLine("Bucket Sort");

//        CommonFunctions.PrintInitial(array);

//        List<int> sorted = Sort(array);

//        CommonFunctions.PrintFinal(sorted);
//        Console.ReadLine();
//    }
//}
```

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace task3_4
{
    public class Account
    {
        //instance variables declared
        private decimal _balance;
        private string _name;

        //read only properties
        public decimal Balance { get => _balance; }
        public string Name { get => _name; }

        //constructor
        public Account(decimal balance, string name)
        {

            _name = name;
            if (balance <= 0)
                return;
            _balance = balance;

        }

        //prints name and balance
        public void Print()
        {
            Console.WriteLine("The name of the account holder is " + getName() +
            ↪  "\nCurrent Account Balance is " + getBalance());
        }

        //returns name
        public String getName()
        {
            return this._name;
        }

        //returns balance
        public decimal getBalance()
        {
            return this._balance;
        }

        //increases balance by adding deposit
        //boolean ensure that no 0 or a negative value can not be input
        public Boolean Deposit(decimal amount)
        {

            if (amount <= 0)
                return false;
```

```

            this._balance += amount;
            //Console.WriteLine("Credit added successfully. New balance for " + this._name + " is " + this._balance);
            return true;

        }


        //decreases balanace by withdrawing the giving input amount
        //boolean values protect against overdrawing from account
        public Boolean Withdraw(decimal amount)
        {
            if (amount > this._balance || amount < 0)
                return false;


            this._balance -= amount;
            //Console.WriteLine("Withdrawal successfull. New balance for " + this._name + " is " + this._balance);
            return true;

        }

    }
}
```

13 Exceptions and Error Handling

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊

This was a really good way to understand the difference between the different types of exceptions. I know feel more confident in handling exceptions.

Outcome	**Weight**
Principles	♦♦♦◊◊

This was a really good way to understand the difference between the different types of exceptions. I know feel more confident in handling exceptions.

Outcome	**Weight**
Build Programs	♦♦◊◊◊

This was a really good way to understand the difference between the different types of exceptions. I know feel more confident in handling exceptions.

Outcome	**Weight**
Design	♦♦♦◊◊

This was a really good way to understand the difference between the different types of exceptions. I know feel more confident in handling exceptions.

Date	Author	Comment
2020/04/22 21:30	Dale Orders	Ready to Mark
2020/04/25 22:28	Sanjay Segu	Discuss
2020/04/25 22:29	Sanjay Segu	Fantastic work Dale
2020/04/25 22:29	Sanjay Segu	Do you we can have custom ‘Exceptions‘?
2020/04/25 23:23	Dale Orders	Are you asking if it is possible to have custom exceptions? Yes it is possible to throw a custom exception in c#
2020/04/26 22:18	Sanjay Segu	Cool. Can you quickly give me an example of how will you do that ?
2020/05/05 18:36	Dale Orders	There are two ways you can throw a custom exception. The first is that you create an object that derives from a preexisting exception class and then pass a error message as a parameter. This will throw the object and its error message when invoked.
2020/05/05 18:37	Dale Orders	Alternatively you could define a new type of the class exception and throw that object class by creating it.
2020/05/05 18:40	Dale Orders	class Program { static void Main(string[] args) { int x, y; Console.WriteLine(”ENTER TWO INTEGERS”); x = int.Parse(Console.ReadLine()); y = int.Parse(Console.ReadLine()); try { if (y == 0) { //DivideByZeroException ONE = new DivideByZeroException(); //throw ONE; throw new DivideByZeroException(); } } catch (DivideByZeroException one) { Console.WriteLine(one.Message); } Console.WriteLine(”Exiting”); Console.ReadKey(); } }
2020/05/05 18:41	Dale Orders	When I posted it in ontrack, the formatting has been removed. But that is an example of a custom exception
2020/05/05 21:41	Sanjay Segu	Complete
2020/05/05 21:41	Sanjay Segu	That’s ok Dale
2020/05/05 21:41	Sanjay Segu	:)

Deakin University

Object Oriented Development

OnTrack Submission

Exceptions and Error Handling

Submitted By:
Dale Orders
dorders
2020/04/22 21:30

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦♦◊◊
Build Programs	♦♦◊◊◊
Design	♦♦♦◊◊

This was a really good way to understand the difference between the different types of exceptions. I know feel more confident in handling exceptions.

April 22, 2020

Practical Task 4.1

SIT232

NullReferenceException

```
using System;

public class NullReferenceException
{

Public void Person()
{
   String name=”Mary”;
   String Address= null;

if (name != null)
{
     name = “Lucy”;

    if (address != null)
    {
        address = “Melbourne”;
    }
}
}
```

In the example above, a series of if statements are used to verify that the declared variables are not null. Once the method is called, each variable is checked to ensure that it is not null (!=null) before the variable gets the users name and sets the users address according to the data stored in the assigned strings. If the user were to attempt to invoke a variable declared null, the system will identify an unhandled error and the program will crash. In this example, the name will call the GetName() method and assign it to the variable name, however it will not assign a value to the address variable as it has been declared null. This is one method which can be used to effectively avoid instances where there Is null type.

A NullReferenceException is usually caused by the design of the program, and is often the fault of the developer for not identify null values within the code, include methods can could possibly return a null value which may evoke a NullReferenceException. With more attention paid to the design of the programs, such errors can be largely avoided.

IndexOutOfRangeException

```
using System;

public class Number
{
   int[] numbers = { 1, 4, 6, 7, 10, 13, 14, 15, 15, 19 };

   public int GetNumber(int index)
   {
      if (index < 0 || index >= numbers.Length)
      {
         throw new IndexOutOfRangeException();
      }
      return numbers[index];
   }
}
```

In this example, an integer array called numbers is declared and initialised. The GetNumber method accepts the index argument which can then be used to return the element which is at the designated index position. The problem is that a user may input an index number which is greater than the length of the array. In this case, the user may invoke GetNumber(11) which will cause an error as there are only 10 elements in the array. To avoid this situation, the programmer can throw an exception specified as 'IndexOutOfRangeExpeption.' This will catch any errors caused by the user attempting to access elements outside the scope of the array. An IndexOutOfRangeException can also be caused by the developer failing to correctly assign the correct number of elements, or attempting to manipulate arrays using wrongly assigned elements (the index starts at 0). Such errors can be avoided through more careful planning and the structure of the program to catch user errors.

StackOverflowException

```
using System;

class Recursive
{
    static void Recursive(int number)
   {
    While(number<=10)
    {
    Console.WriteLine(number);
    Recursive (++number);
    }
   }

   static void Main()
   {
     Recursive(1)
   }

}
```

In the case of a StackOverFlowException, the memory of the system exceeds its storing capacity and therefore overflows. In the code above, the method Recursive takes a number, prints the number then calls itself with an increment of 1. As written, this code will start at 1, and continue to print a sequence of numbers until the stack overflows and terminates. To avoid this exception you can use a while loop to create a bounded condition and thereby avoid the possibility of the stack exceeding its capacity. This is runtime error, however it is one that can be avoided through ensuring all recursive calls are bounded by an upper limit.

OutOfMemoryException

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            var list = new List<string>();
            while (true)
            {
                list.Add(Guid.NewGuid().ToString());
            }
        }
        catch (OutOfMemoryException e)
        {
         Environment.FailFast(String.Format($"Out of memory: {e.Message}"));
        }
    }
}
```

An error can occur when the memory of the system is exceeded beyond its capacity at which point it terminates. In the example above, the while loops takes a list and adds to it continuously. This will eventually see the system run out of memory and produce an unhandled exception. To avoid this, you could use a try-catch block to confine the possible source of the error to the try block, which will execute the statement until it encounters the exception. The catch block (OutOfMemoryException e) will detect the memory has been exhausted and terminate the program, showing the message “Out of Memory: Insufficient memory to continue the execution of the program.” This runtime error is the fault of the developer and can be avoided if they identify continuous loops and enact try-catch to either end the program or write a conditional statement to limit the size of the string so it doesn’t exceed max capacity. They can also separate a large query into smaller subset queries, which tend to require less data.

InvalidCastException

```
public class Example
{
   public static void Main()
   {
      bool val = true;
      try
       {
         IConvertible convt = val;
         Char ch = Convert.ToChar(val);
         Console.WriteLine("Conversion succeeded.");
       }
      catch (InvalidCastException)
        {
         Console.WriteLine("Cannot convert a Boolean to a Char.");
        }

     }
}
```

An InvalidCastException occurs when the program is unable to convert from one data type to another and therefore terminates unexpectantly. In this example, the val variable is declared as a Boolean value. The code then attempts to convert this value into a character (char) and assign it to ch, which is also a character. It is important that the code be written to detect occurrences, such as this, in which conversion fails and delivers a InvalidCastException. In this case, the program will detect that conversion between these types is not possible in this situation and, as such, it will catch the error when it executes. The user will see the message “Cannot convert a Boolean to a Char.” The programmer is responsible for writing code that minimises the risk of producing an InvalidCastException, by anticipating places where the user may input the wrong data type.

DivideByZeroException

```
public class Example
{
   public static void Main()
   {
      int number_1 = 50;
      int number_2 = 0;
      try {
         Console.WriteLine(number_1 / number_2);
      }
      catch (DivideByZeroException)
      {
         Console.WriteLine("Division of {0} by zero.", number_1);
      }
```

```
    }
}
```

A DivideByZeroException is invoked whenever a number is divided by zero. It is not mathematically possible to divide by zero as the result is undefined. If you were to execute the example above it would produce an error during runtime. To avoid this, you could use a try-catch approach and specify the program to throw a DivideByZeroException if it detects such an occurrence. In this example, number_1 is assigned an integer value of 50 and number_2 is assigned a value of 0. The try block will print the result of number_1 divided by number_2 as long as number_2 is not equal to 0. As this is not the case, the program will run the exception and print the statement “Division of 50 by zero.” The programmer will need to structure the code to identify possible instances wherein a number is divided by 0.

ArgumentException

```
class Account
{
  public Account(string first_Name, string last_Name)
  {
    if (string.IsNullOrEmpty(first_Name))
      throw new ArgumentException("First_Name is invalid");
    if (string.IsNullOrEmpty(last_Name))
      throw new ArgumentException("Last_Name is invalid");
    }

 }
```

In this example, the Account class the arguments, First_Name and last_Name, are put into the Account Method. An if statement is used to assess if each of the arguments are null or empty. For each argument, an Argument is thrown if the argument is invalid (if it is null or valid). In this way, throwing an exception serves to check the validity of the argument that has been called by the method, and prints a statement to the screen to inform them of the error. If the user has put in valid arguments, no message will be produced. To avoid such errors, the programmer will need to consider what may happen if the user were to assign invalid arguments to a method.

ArgumentOutOfRangeException

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      var list = new List<String>();
      Console.WriteLine("Number of items: {0}", list.Count);


      if (list.Count > 0)
     {

        Try
        {
           Console.WriteLine("The first item: '{0}'", list[0]);
        }
        catch (ArgumentOutOfRangeException e)
        {
             Console.WriteLine(e.Message);
         }
     }
   }
}
```

In this example, a new list is declared with a string data type. The code then prints a count of the number of elements in the empty list. This is followed by a try-catch block that attempts to print the first element in the list, however as there are no elements in the list, this will produce an error in the code and it will not execute as intended. To address, this you can use a conditional if statement to ensure that the block of code only execute if count has at least one element in it, which will eliminate the possibility of invoking the exception. This is preferable as throwing too many exceptions can slow down a program, so it is best to use them when only necessary. The use of both measure ensures that the code will run.

ArgumentOutOfRangeException

```
using System;

class ExceptionTest
{
public static void Main()
{
   int x = 0;
      try
      {
         int y = 10/x;
      }
         catch (ArithmeticException e)
         {
       Console.WriteLine("ArithmeticException Handler: {0}",
e.ToString());
         }
            catch (Exception e)
            {
            Console.WriteLine("Generic Exception Handler:
{0}", e.ToString());
            }
      }
}
```

This class serves as the base class for all other exceptions, which detects and throws an exception as the application executes. The exception handling model directs the exception into a try-catch whereby the code which could invoke an exception is placed within a try block and will be handled by the relevant catch block. In this case, int x is initialised as 0. A try block is used to identify possible exceptions that could be thrown if a 0 is in the denominator. A second catch block is used to detect any general exceptions. In this example, the exception class is placed last as the more specific exception should be placed first. The application will consider each exception in order and stop once it finds the first one that applies. As such, the more specific exception should be written first as it will provide greater depth information as to the nature of the exception encountered. This program will produce the following error:
ArithmeticException Handler: System.DivideByZeroException: Attempted to divide by zero. at ExceptionTestClass.Main().

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace ExceptionHandling
{

    class Account
    {
        public string FirstName { get; private set; }
        public string LastName { get; private set; }
        public int Balance { get; private set; }

        public Account(string firstName, string lastName, int balance)
        {
            FirstName = firstName;
            LastName = lastName;
            Balance = balance;
        }

        public void Withdraw(int amount)
        {
            if (amount > Balance)
            {
                throw new InvalidOperationException("Insufficient fund");
            }
            Balance = Balance - amount;
        }



    }



    class Program
    {


        public static void Main()
        {
            try
            {
                Account account = new Account("Sergey", "P", 100);
                account.Withdraw(1000);
            }
            catch (InvalidOperationException exception)
            {
                Console.WriteLine("The following error detected: " +
                ↪ exception.GetType().ToString() + " with message \"" +
                ↪ exception.Message + "\"");
            }

```

```
            Console.ReadKey();

            //Call methods
            Person();
            GetNumber(-1);
            OutofMemory();
            InvalidCast();
            Recursive(5);
            DividebyZero();
            Account("Mary", "Jones");
            ArgumentOutofRange();
            //System_Exception();


            //NullReferenceException
            static void Person()
            {
                string name = "Mary";
                string address = null;
                if (name != null)
                {
                    name = "Lucy";
                    Console.WriteLine(name);

                    if (address != null)
                    {
                        address = "Melbourne";
                        Console.WriteLine(address);
                    }

                }

            }

            //IndexOutofRangeException
            static int GetNumber(int index)
            {
                int[] numbers = { 1, 4, 6, 7, 10, 13, 14, 15, 15, 19 };
                if (index < 0 || index >= numbers.Length)
                {
                    throw new IndexOutOfRangeException();
                }
                return numbers[index];
            }



            //StackoverflowException
            static void Recursive(int number)
            {
                //for loop to avoid exception
                while (number <= 10)
                {
```

```
                Console.WriteLine(number);
                Recursive(++number);
            }
        }



        //OutofmemoryException
        static void OutofMemory()
        {
            try
            {
                var list = new List<string>();
                while (true)
                {
                    list.Add(Guid.NewGuid().ToString());
                }
            }
            catch (OutOfMemoryException e)
            {
                Environment.FailFast(String.Format($"Out of memory: {e.Message}"));
            }


        }

        //InvalidCastException
        static void InvalidCast()
        {
            bool val = true;


            try
            {

                //IConvertible convt = val;
                Char ch =Convert.ToChar(val);
                Console.WriteLine("Conversion succeeded.");
            }
            catch (InvalidCastException)
            {
                Console.WriteLine("Cannot convert a Boolean to a Char.");
            }

        }


        //DividebyZero Exception
        static void DividebyZero()
        {
            int number_1 = 50;
            int number_2 = 0;
```

```
            try
            {
                Console.WriteLine(number_1 / number_2);
            }
            catch (DivideByZeroException)
            {
                Console.WriteLine("Division of {0} by zero.", number_1);
            }

        }


        //ArgumentException
        static void Account(string first_Name, string last_Name)
        {
            if (string.IsNullOrEmpty(first_Name))
                throw new ArgumentException("First_Name is invalid");
            if (string.IsNullOrEmpty(last_Name))
                throw new ArgumentException("Last_Name is invalid");
        }


        //ArgumentOutofRange
        static void ArgumentOutofRange()
        {
            var list = new List<String>();
            Console.WriteLine("Number of items: {0}", list.Count);


            if (list.Count > 0)
            {

                try
                {
                    Console.WriteLine("The first item: '{0}'", list[0]);
                }
                catch (ArgumentOutOfRangeException e)
                {
                    Console.WriteLine(e.Message);
                }

            }


            //SystemException
            static int System_Exception()
            {
                int x = 0;
                try
                {
                    int y = 10 / x;
                }
                catch (ArithmeticException e)
```

```
                    {
                        Console.WriteLine("ArithmeticException Handler: {0}",
                        ↪  e.ToString());
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("Generic Exception Handler: {0}",
                        ↪  e.ToString());
                    }

                    return 0;

                }

                System_Exception();


            }

        }

    }
}
```

14 BuggySoft: Program Design and Class Composition

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Evaluate Code	♦♦♦♦◊

This was a good illustration of how efficient code can be used to improve the design and development of a program.

Outcome	**Weight**
Principles	♦◊◊◊◊

This was a good illustration of how efficient code can be used to improve the design and development of a program.

Outcome	**Weight**
Build Programs	♦◊◊◊◊

This was a good illustration of how efficient code can be used to improve the design and development of a program.

Outcome	**Weight**
Design	♦◊◊◊◊

This was a good illustration of how efficient code can be used to improve the design and development of a program.

Date	**Author**	**Comment**
2020/05/16 17:37	Dale Orders	Ready to Mark
2020/05/17 18:06	Sanjay Segu	Complete
2020/05/17 18:06	Sanjay Segu	Completed

Deakin University

Object Oriented Development

OnTrack Submission

BuggySoft: Program Design and Class Composition

Submitted By:
Dale Orders
dorders
2020/05/16 17:37

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦♦♦♦◊
Principles	♦◊◊◊◊
Build Programs	♦◊◊◊◊
Design	♦◊◊◊◊

This was a good illustration of how efficient code can be used to improve the design and development of a program.

May 16, 2020

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace DuplicateCode
{
    class DuplicateCode
    {

        static int MaximumLength(Dictionary<string, List<string>> tasks)
        {
            return tasks.Values.Max(list => list.Count);
        }
        static void PrintTasks(Dictionary<string, List<string>> tasks)
        {
            int max = MaximumLength(tasks);
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine(new string(' ', 12) + "CATEGORIES");
            Console.WriteLine(new string(' ', 10) + new string('-', 94));
            Console.Write("{0,10}", "item #");
            foreach (var category in tasks.Keys)
            {
                Console.WriteLine("{0,10}", category);
            }
            Console.WriteLine();
            Console.WriteLine(new string(' ', 10) + new string('-', 94));
            for (int i = 0; i < max; i++)
            {
                Console.Write("{0,10}|", i + 1);
                foreach (var list in tasks.Values)
                {
                    if (list.Count > i)
                        Console.Write("{0,30}|", list[i]);
                    else
                        Console.Write("{0,30}|", "N/A");
                }
                Console.WriteLine();
            }
            Console.ResetColor();

        }
        static void Main(string[] args)
        {
            var tasks = new Dictionary<string, List<string>>();
            tasks["Personal"] = new List<string>();
            tasks["Family"] = new List<string>();
            tasks["Work"] = new List<string>();

            string task;
            string category;

            while (true)
            {
```

```
                Console.Clear();
                PrintTasks(tasks);

                Console.WriteLine("\nWhich category do you want to place a new task? Type \'Personal\', \'Work\', or \'Family\'");
                category = Console.ReadLine().ToLower();
                if (category.ToLower() == "quit")
                    break;

                Console.WriteLine("Describe your task below (max. 30 symbols).");
                task = Console.ReadLine();
                if (task.Length > 30)
                {
                    task.Substring(0, 30);
                }
                try
                {
                    tasks[category].Add(task);
                }
                catch (ArgumentException)
                {
                    continue;
                }
            }
        }
    }

}
```

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;


namespace buggysoft_program
{
    class Program
    {
        static void Main(string[] args)
        {
            List<string> categories = new List<string> { "Personal", "Work",
            ↪  "Family" };
            TaskPlanner taskPlanner = new TaskPlanner(categories);
            TaskPlannerMenu Menu = new TaskPlannerMenu(taskPlanner);

            Menu.MainMenu();
        }




        class Task
        {
            private string task { get; set; }
            private DateTime date = new DateTime { };
            private ConsoleColor color = ColorSelector.Default;


            public Task(string task) => this.task = task;
            public DateTime GetDate() => date;
            public ConsoleColor GetColor() => color;
            override public string ToString() => task;

            public void SetTask(string task) => this.task = task;
            public void SetDate(DateTime date) => this.date = date;
            public void SetColor(ConsoleColor color) => this.color = color;

        }

        class TasksList
        {
            private List<Task> tasks = new List<Task> { };

            public TaskList() { }

            public TaskList(List<string> tasks)
            {
                foreach (string task in tasks)
                {
                    this.tasks.Add(new Task(task));
                }
            }
```

```

        public TaskList(string task) => tasks.Add(new Task(task));

        public int Count => tasks.Count;

        public bool InDomain(int i) => 0 <= i && i < Count ? true : false;

        public Task GetTask(int i) => InDomain(i) ? tasks[i] : throw new IndexOutOfRangeException("Index out of range");

        //end asklist

        public int GetIndexOfTask(string task)
        {
            for (int i = 0; i < Count; i++)
            {
                if (tasks[i].ToString().ToUpper() == task.ToUpper())
                {
                    return i;
                }
            }
            return -1;
        }

        public void AddTask(Task task) => tasks.Add(task);
        public void DeleteTask(string task) => DeleteTask(GetIndexOfTask(task));

        public bool DeleteTask(int i)
        {
            if (InDomain(i))
            {
                tasks.RemoveAt(i);
                return true;
            }
            return false;
        }

        public void UpdatePriority(string task, int priority) => UpdatePriority(GetIndexOfTask(task), priority);

        public bool UpdatePriority(int i, int priority)
        {
            if(i!=priority&&InDomain(i)&& InDomain(priority))
           {
                tasks.Insert(priority, GetTask(i));
                DeleteTask(i + 1);
                return true;
            }
            return false;
        }

    }

```

```
        class TaskPlanner
        {
            private List<string> Categories { get; set; }
            private List<TasksList> TasksList = new List<TasksList> { };

            public TaskPlanner(List<string> categories)
            {
                Categories = categories;
                for (int i = 0; i < Categories.Count; i++)
                {
                    TasksList.Add(new TasksList());
                }

            }


            public bool InDomain(int i) => 0 <= i && i < Categories.Count ? true : false;

            public bool TaskListDomain(string category, int i) => GetIndexOfCategory(category) != -1 ? TasksList[GetIndexOfCategory(category)].InDomain(i) : false;

            private int GetIndexOfCategory(string category)
            {
                for (int i = 0; i < Categories.Count; i++)
                {
                    if (category.ToUpper() == Categories[i].ToUpper())
                    {
                        return i;
                    }
                }

                return -1;
            }

            public bool CategoryExists(string category)
            {
                if (GetIndexOfCategory(category) != -1) return true;
                return false;
            }

            public bool AddCategory(string category)
            {
                if (CategoryExists(category))
                {
                    Categories.Add(category);
                    TasksList.Add(new TasksList());
                    return true;
                }
                return false;
            }

```

```
        public bool DeleteCategory(string category)
        {
            int index = GetIndexOfCategory(category);
            if (index != -1)
            {
                Categories.RemoveAt(index);
                TasksList.RemoveAt(index);
                return true;
            }
            return false;
        }

        public int GetIndexOfTask(string category, string task)
        {
            int index = GetIndexOfCategory(category);
            if (index != -1)
            {
                for (int i = 0; i < TasksList[index].Count; i++)
                {

                    if (task.ToUpper() ==
                    ↪ TasksList[index].GetTask(i).ToString().ToUpper())
                    {
                        return i;
                    }
                }
            }
            return -1;
        }

        public Task GetTask(string category, string task) => GetTask(category,
        ↪ GetIndexOfTask(category, task));

        public Task GetTask(string category, int i)
        {
            int index = GetIndexOfCategory(category);
            if(index!=-1)
            {
                if(TasksList[index].InDomain(i))
                {
                    return TasksList[index].GetTask(i);
                }
            }
            throw new IndexOutOfRangeException("Error");
        }

        public bool AddTask(string category, string task) => AddTask(category,
        ↪ new Task(task));

        public bool AddTask(string category, Task task)
        {
            int index = GetIndexOfCategory(category);
            if (index != -1)
```

```
                {
                    TasksList[index].AddTask(task);
                    return true;
                }
                return false;
            }

            public bool DeleteTask(string category, int i)
            {
                int index = GetIndexOfCategory(category);
                if(index!=-1)
                {
                    if(TasksList[index].InDomain(i))
                    {
                        TasksList[index].DeleteTask(i);
                        return true;
                    }
                 }
                return false;
            }

            public bool MoveTask(string category1, string category2, string task)
            {
                int index = GetIndexOfCategory(category1);
                if(index!=-1)
                {
                    index = TasksList[index].GetIndexOfTask(task);
                    if(index!=-1)
                    {
                        MoveTask(category1, category2, index);
                        return true;
                    }

                }
                return false;
            }

            public bool updatePriority(string category, int i, int priority)
            {
                if(i!=priority)
                {
                    int index = GetIndexOfCategory(category);
                    if(index!=-1)
                    {
                        if(TasksList[index].InDomain(i)&&
                        ↪ TasksList[index].InDomain(priority))
                        {
                            TasksList[index].UpdatePriority(i, priority);
                            return true;
                        }
                    }

                }
```

```
                return false;
            }

            public bool MoveTask(string category1, string category2, int i)
            {
                int index1 = GetIndexOfCategory(category1);
                int index2 = GetIndexOfCategory(category2);
                if(index1!=-1 && index2!=-1)
                {
                    if(TasksList[index1].InDomain(i))
                    {
                        AddTask(category2, TasksList[index1].GetTask(i));
                        DeleteTask(category1, i);
                        return true;
                    }
                }
                return false;
            }



            public void WriteTaskPlanner()
            {
                int i;
                Console.ForegroundColor = ColorSelector.Default;
                Console.WriteLine(new string(' ', 5)+"CATEGORIES");
                Console.WriteLine(new string(' ', 5) + new string('-',
                ↪  Categories.Count*42+5));
                Console.Write("{0,5}|", "item #");

                for(i=0;i<Categories.Count;i++)
                {
                    Console.WriteLine("{0,30}+{1,10}", Categories[i], "Due Date");
                }
                Console.WriteLine();
                Console.WriteLine(new string(' ', 5) + new string('-',
                ↪  Categories.Count * 42 + 5));

                int max = 0;
                foreach (TasksList tasks in TasksList)
                {
                    max = max > tasks.Count ? max : tasks.Count;
                }

                    foreach(TasksList tasks in TasksList)
                    {
                        if(tasks.Count>i)
                        {
                            Task task = tasks.GetTask(i);
                            Console.ForegroundColor = task.GetColor();
                            Console.Write("{0,30} +", task.ToString());
                            if(task.GetDate().Date.Year!=1)
                            {
```

```
                                Console.Write("{ 0,10}|",
                                    $"{task.GetDate().Date.Day}/{task.GetDate().Date.Month}/{task.GetDate().Date.Year}");
                            }
                            else
                            {
                                Console.Write("{ 0,10}|", "N/S");
                            }
                            Console.ForegroundColor = ColorSelector.Default;
                        }
                        else
                        {
                            Console.Write("{ 0,42}|", "N/A");
                        }
                        Console.WriteLine();
                    }
                    Console.WriteLine("\n");
                    Console.ResetColor();
                }
            }


        enum MENUOPTIONS
        {
            ADD_CATEGORY,
            DELETE_CATEGORY,
            ADD_TASK,
            SELECT_TASK,
            MOVE_TASK,
            UPDATE_TASK,
            DELETE_TASK,
            CHANGE_TASK_PRIORITY,
            CHANGE_TASK_IMPORTANCE,
            SET_DATE,
            CANCEL,
            QUIT
        };

        enum TASKSELECTION
        {
            BY_INDEX,
            BY_NAME,
            CANCEL
        };

        enum TASKPRIORITY
        {
            DEFAULT,
            IMPORTANT
        };

        public class ColorSelector
        {
```

```
        }

        class TaskPlannerMenu
        {
            private TaskPlanner taskPlanner { get; set; }

            public TaskPlannerMenu Menu(TaskPlannerMenu taskplanner) =>
            ↪ this.taskPlanner = taskPlanner;

            private T ReadMenuInput<T>(T[] opts) where T : System.Enum
            {
                int input;
                Console.Clear();
                taskPlanner.WriteTaskPlanner();
                do
                {
                    Console.WriteLine("Select from the following options");
                    for (int i = 0; i < opts.Length; i++)
                    {
                        Console.WriteLine($"(i + 1));(opts[i]");
                    }
                    Console.Write(">>>");

                    try
                    {
                        input = Convert.ToInt32(Console.ReadLine()) - 1;
                        if (0 <= input && input < opts.Length)
                        {
                            Console.WriteLine("Input outside of domain");
                        }
                    }
                    catch (FormatException)
                    {
                        Console.WriteLine("Invalid entry");
                        ReadMenuInput(opts);
                    }
                } while (true);
            }

            public void MainMenu()
            {

                MENUOPTIONS input;
                MENUOPTIONS[] opts = { MENUOPTIONS.ADD_CATEGORY,
                ↪ MENUOPTIONS.DELETE_CATEGORY, MENUOPTIONS.ADD_TASK,
                ↪ MENUOPTIONS.SELECT_TASK, MENUOPTIONS.QUIT }

                do
                {
                    input = ReadMenuInput(opts);
                    switch (input)
                    {
```

```
                    case MENUOPTIONS.ADD_CATEGORY: AddCategory(); break;
                    case MENUOPTIONS.DELETE_CATEGORY: DeleteCategory(); break;
                    case MENUOPTIONS.ADD_TASK: AddTask(); break;
                    case MENUOPTIONS.SELECT_TASK: SelectTaskMenu(); break;
                    case MENUOPTIONS.QUIT: return;
                }

            } while (true);

        }

        private string ReadString(string msg)
        {
            Console.Clear();
            taskPlanner.WriteTaskPlanner();
            Console.WriteLine(msg);
            return Console.ReadLine();
        }

        private int ReadInt(string msg)
        {
            try
            {
                return Convert.ToInt32(ReadString(msg));
            }
            catch (FormatException)
            {
                return ReadInt(msg);
            }
        }

        private void AddCategory()
        {
            string input = ReadString("Enter the category to add: ");
            if (!taskPlanner.AddCategory(input)
            {
                Console.WriteLine("Category not added as it already exists");
                Console.ReadLine();
            }

        }

        private void DeleteCategory()
        {
            string input = ReadString("Enter the category to delete: ");
            if (!taskPlanner.DeleteCategory(input)
            {
                Console.WriteLine("Category not deleted as it already exists");
                Console.ReadLine();
            }

        }

```

```
            private string SelectCategory()
            {
                string input = ReadString("Select Category: ");
                if(taskPlanner.CategoryExists(input))
                {
                    Console.WriteLine();
                    Console.WriteLine("Category does not exist");
                    Console.WriteLine();
                    throw new IndexOutOfRangeException("Category does not exist");
                }
                return input;
            }

           private void AddTask()
            {
                string category;
                try
                {
                    category = SelectCategory();
                }
                catch(IndexOutOfRangeException)
                {
                    return;
                }
                string task = ReadString("What is the name of the task");
                taskPlanner.AddTask(category, task);
            }

            private void SelectTaskMenu()
            {
                string category;
                try
                {
                    category = SelectCategory();
                }
                catch(IndexOutOfRangeException)
                {
                    return;
                }
                TASKSELECTION input;
                TASKSELECTION[] opts = { TASKSELECTION.BY_INDEX, TASKSELECTION.BY_NAME, TASKSELECTION.CANCEL };
                input = ReadMenuInput(opts);
                switch(input)
                {
                    case TASKSELECTION.BY_INDEX: SelectedTaskMenu(category, ReadInt("Enter ndex number: "); break;
                    case TASKSELECTION.BY_NAME: SelectedTaskMenu(category, Convert.ToInt32(ReadString("Enter the name: ")); break;
                    case TASKSELECTION.CANCEL:return;

                }
            }
```

```

        //missing line

        private void SelectedTaskMenu(string category, int i)
        {
            ConsoleColor color = taskPlanner.GetTask(category, i).GetColor();
            taskPlanner.GetTask(category, i).GetColor(ColorSelector.Selected);
            MENUOPTIONS input;
            MENUOPTIONS[] opts = { MENUOPTIONS.UPDATE_TASK, MENUOPTIONS.CANCEL };
            input = ReadMenuInput(opts);
            switch (input)
            {
                case MENUOPTIONS.UPDATE_TASK: color = UpdateTaskMenu(category, i, color); break;
                case MENUOPTIONS.CANCEL: break;

            }
            if (taskPlanner.TaskListDomain(category, i)
            {
                taskPlanner.GetTask(category, i).SetColor(color);
            }

            else
            {
                Console.WriteLine();
                Console.WriteLine("Category does not exist");

            }
        }


        private ConsoleColor UpdateTaskMenu(string category, int i, ConsoleColor color)
        {
            MENUOPTIONS input;
            MENUOPTIONS[] opts = { MENUOPTIONS.MOVE_TASK, MENUOPTIONS.CHANGE_TASK_PRIORITY, MENUOPTIONS.SET_DATE, MENUOPTIONS.DELETE_TASK, MENUOPTIONS.CANCEL };
            input = ReadMenuInput(opts);
            switch(opts)
            {
                case MENUOPTIONS.MOVE_TASK[:return MoveTask(category, i, color);
                case MENUOPTIONS.CHANGE_TASK_PRIORITY:return ChangeTaskPriority(category, i, color);
                case MENUOPTIONS.SET_DATE:SetDate(category, i); return color;
                case MENUOPTIONS.DELETE_TASK:taskPlanner.DeleteTask(category, i); return color;
                case MENUOPTIONS.CANCEL:return color;
                default: return color;
            }
        }

```

```
            private ConsoleColor MoveTask(string category, int i, ConsoleColor color)
            {
                string category2 = ReadString("Enter category to move");
                taskPlanner.GetTask(category, i).SetColor(color);
                if(taskPlanner.MoveTask(category, category2, i))
                {

                    if (taskPlanner.TaskListDomain(category, i))
                    {
                        return taskPlanner.GetTask(category, i).GetColor();
                    }

                }
                return color;
            }

            private ConsoleColor ChangeTaskImportanceMenu()
            {
                TASKPRIORITY input;
                TASKPRIORITY[] opts = { TASKPRIORITY.DEFAULT, TASKPRIORITY.IMPORTANT };

                input = ReadMenuInput(opts);
                switch(input)
                {
                    default: case TASKPRIORITY.DEFAULT: return ColorSelector.Default;
                    case TASKPRIORITY.IMPORTANT:return ColorSelector.Important;

                }
            }

            private ConsoleColor ChangeTaskPriority(string category, int i, ConsoleColor color)
            {
                int j = ReadInt("Enter priority [index]");
                if(i!=j&&taskPlanner.TaskListDomain(category, j))
                {
                    ConsoleColor color2 = taskPlanner.GetTask(category, j).GetColor();
                    taskPlanner.updatePriority(category, i, j);
                    taskPlanner.GetTask(category, j).SetColor(color);
                    return color2;
                }

                return color;
            }


        private void SetDate(string category, int i)
            {
                int day, month, year;
```

```
                year = ReadInt("Enter the year: ");
                if (i > year || year > 9999) return;

                month = ReadInt("Enter the month: ");
                if (i > month ||month > 12) return;

                int[] daysDomain = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 };
                if (i > day || day > daysDomain[month-1]) return;

                taskPlanner.GetTask(category, i).SetDate(new DateTime(year, month, day));

            }
        }

    }

}
```

15 C# Essentials: Inheritance

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Evaluate Code	♦♦♦♦♦

Great task. I wish all of the pass tasks were like this as the learning is clear and it demonstrates the concept through practical application.

Outcome	Weight
Principles	♦♦♦♦♦

Great task. I wish all of the pass tasks were like this as the learning is clear and it demonstrates the concept through practical application.

Outcome	Weight
Build Programs	♦♦♦♦♦

Great task. I wish all of the pass tasks were like this as the learning is clear and it demonstrates the concept through practical application.

Outcome	Weight
Design	♦♦♦♦♦

Great task. I wish all of the pass tasks were like this as the learning is clear and it demonstrates the concept through practical application.

Outcome	Weight
Justify	♦♦♦♦♦

Great task. I wish all of the pass tasks were like this as the learning is clear and it demonstrates the concept through practical application.

Date	Author	Comment
2020/04/17 15:32	Dale Orders	Ready to Mark
2020/04/19 21:31	Sanjay Segu	Demonstrate
2020/04/19 21:31	Sanjay Segu	Hi Dale
2020/04/19 21:31	Sanjay Segu	Can I please request you for a demonstration video ?
2020/05/05 20:33	Dale Orders	https://youtu.be/yiN1LhyQ-pw
2020/05/05 21:41	Sanjay Segu	discussion comment
2020/05/05 21:41	Sanjay Segu	Discuss
2020/06/03 23:23	Sanjay Segu	Complete

Deakin University

Object Oriented Development

OnTrack Submission

C# Essentials: Inheritance

Submitted By:
Dale Orders
dorders
2020/04/17 15:32

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦♦♦♦♦
Principles	♦♦♦♦♦
Build Programs	♦♦♦♦♦
Design	♦♦♦♦♦
Justify	♦♦♦♦♦

Great task. I wish all of the pass tasks were like this as the learning is clear and it demonstrates the concept through practical application.

April 17, 2020

```
using System;

namespace task5_1
{
    class ZooPark
    {
        static void Main(string[] args)
        {
            Animal williamWolf = new Animal("William ths Wolf", "Meat", "Dog Villiage", 50.6, 9, "Grey");
            Animal tonyTiger = new Animal("Tony the Tiger", "Meat", "Cat Land", 110, 6, "Orange and White");
            Animal edgarEadle = new Animal("Edgar the Eagle", "Fish", "Bird Mania", 20, 15, "Black");

        }
    }
}
```

```
using System;
namespace task5_1
{
    public class Animal
    {
        class animal
        {
            private String name;
            private string diet;
            private String location;
            private double weight;
            private int age;
            private String colour;
        }


        public Animal(String name, String diet, String location, double weight, int
        ↪ age, String colour)
        {
            this.name = name;
            this.diet = diet;
            this.location = location;
            this.weight = weight;
            this.age = age;
            this.colour = colour;
        }

        public void eat()
        {
            //
        }

        public void sleep()
        {
            //
        }

        public void MakeNoise()
        {
            //
        }

        public void makeLionNoise()
        {
            //
        }

        public void makeEagleNoise()

        {
            //
        }

```

```
        public void makeWolfNoise()
        {
            //
        }

    }
}
```

```
using System;
namespace Task02
{
        public class Animal
        {

                private String name;
                private string diet;
                private String location;
                private double weight;
                private int age;
                private String colour;


            public Animal(String name, String diet, String location, double weight,
             ↪  int age, String colour)
            {
                this.name = name;
                this.diet = diet;
                this.location = location;
                this.weight = weight;
                this.age = age;
                this.colour = colour;
            }

            public virtual void eat()
            {
                 Console.WriteLine("An animal eats");
            }

            public virtual void lives()
            {
                Console.WriteLine("An animal lives");
            }

             public virtual void sleep()
            {
                Console.WriteLine("an animal sleeps");
            }

            public virtual void MakeNoise()
            {
                Console.WriteLine("An animal makes a noise");
            }

            public void makeLionNoise()
            {
                //code for Lion noise
            }

            public void makeEagleNoise()

            {
```

```
                //code for Eagle noise
            }

        public void makeWolfNoise()
            {
                //code for Wolf noise
            }

        }
}
```

```
using System;

namespace Task02
{
    class ZooPark
    {
        static void Main(string[] args)
        {
            //Animal williamWolf = new Animal("William ths Wolf", "Meat", "Dog Villiage", 50.6, 9, "Grey");
            //Animal tonyTiger = new Animal("Tony the Tiger", "Meat", "Cat Land", 110, 6, "Orange and White");
            //Animal edgarEadle = new Animal("Edgar the Eagle", "Fish", "Bird Mania", 20, 15, "Black");

            //Tiger tonytiger = new Tiger();

            Tiger tonyTiger = new Tiger("Tony the Tigger", "Meat", "Cat Land", 110, 6, "Orange and White", "Siberian", "White");
            Wolf williamWolf = new Wolf("William the Wolf", "Meat", "Dog Village", 50.6, 9, "Grey");
            Eagle edgarEagle = new Eagle("Edgar the Eagle", "Fish", "Bird Mania", 20, 15, "Black", "Harpy", 98.5);
            Penguin Penny = new Penguin("Penny the Penguin", "Fish", "Snow Mountain", 10, 5, "White", "King", 44.5);
            Lion Leo = new Lion("Leo the Lion", "Gazelle", "Serengeti", 70, 65, "White", "P.I");
            //tonyTiger.MakeNoise();

            Animal baseAnimal = new Animal("Animal Name", "Animal Diet", "Animal Location", 0.0, 0, "Animal Colour");
            //baseAnimal.MakeNoise();
            //Console.ReadLine();

            Bird baseBird = new Bird("Bird Name", "Bird Diet", "Bird Location", 0.0, 0, "Bird Colour", "Bird Species", 0.0);

            //baseAnimal.sleep();

            Console.WriteLine("---------------------------------TonyTiger----------------------");
            Console.Write("TonyTiger says ");
            tonyTiger.sleep();
            tonyTiger.eat();

            Console.WriteLine("---------------------------------WilliamWolf-------------------------");
            Console.Write("williamWolf says ");
            williamWolf.sleep();
            williamWolf.eat();

            Console.WriteLine("--------------------------------Leo the Lion-----------------------------");
```

```
            Console.Write("Leo the Lion says ");
            Leo.sleep();
            Leo.eat();

            Console.WriteLine("---------------------------------edgarEagle---------
            ----------------------");
            Console.Write("edgarEagle says ");
            edgarEagle.sleep();
            edgarEagle.eat();
            edgarEagle.fur();
            edgarEagle.layEgg();
            edgarEagle.swim();
            edgarEagle.size();
            edgarEagle.feet();
            edgarEagle.weather();


            Console.WriteLine("---------------------------------Penny the
            Penguin-----------------------------");
            Penny.sleep();
            Penny.eat();
            Penny.fur();
            Penny.layEgg();
            Penny.swim();
            Penny.size();
            Penny.feet();
            Penny.weather();




        }

    }
}
```

```
using System;
namespace Task02

{
    class Tiger : Feline
    {
        private String colourStripes;

        public Tiger(String name, String diet, String location, double weight, int age, String colour, String species,
            String colourStripes) : base(name, diet, location, weight, age, colour, species)
        {
            this.colourStripes = colourStripes;
        }

        public override void MakeNoise()
        {
            Console.WriteLine("ROARRRRRRRRRRR");
        }

        public override void eat()
        {
            Console.WriteLine("I can eat 20lbs of meat");
        }

        public override void lives()
        {
            Console.WriteLine("I can live 10 years");
        }

    }
}
```

```
using System;
namespace Task02
{
    class Eagle : Bird
    {


        public Eagle(String name, String diet, String location, double weight, int age, String colour, String species, double wingSpan)
        : base(name, diet, location, weight, age, colour, species, wingSpan)
        {

        }

        public override void MakeNoise()
        {
            Console.WriteLine("WHISTLEEEESSSS");
        }
        public override void lives()
        {
            Console.WriteLine("I can live 20 years");
        }

        public override void layEgg()
        {
            Console.WriteLine("I lay eggs");
        }

        public override void fly()
        {
            Console.WriteLine("I can fly");
        }

        public override void weather()
        {
            Console.WriteLine("I do not like the cold");
        }

        public override void fur()
        {
            Console.WriteLine("I have rough feathers");
        }

        public override void beak()
        {
            Console.WriteLine("I have a large beak");
        }

        public override void swim()
        {
            Console.WriteLine("I can't swim");
        }

```

```
        public override void size()
        {
            Console.WriteLine("I am large");
        }

        public override void feet()
        {
            Console.WriteLine("I have claws");
        }
    }
}
```

```
using System;
namespace Task02
{

    class Wolf : Animal
    {

        public Wolf(String name, String diet, String location, double weight, int age, String colour) : base(name, diet, location, weight, age, colour)
        {

        }

        public override void MakeNoise()
        {
            Console.WriteLine("HOWLLLLLLLS");
        }

        public override void lives()
        {
            Console.WriteLine("I can live 8 years");
        }

    }


}
```

```
using System;
namespace Task02
{
    class Feline:Animal
    {
        private String species;

        public Feline(String name, String diet, String location, double weight, int
        ↪ age, String colour, String species):
        base(name, diet, location, weight, age, colour)
        {
            this.species = species;
        }

        public override void sleep()
        {
            Console.WriteLine("I can sleep 9 hours a night");
        }
    }
}
```

```
using System;
namespace Task02
{
    class Lion : Feline
    {
        public Lion(String name, String diet, String location, double weight, int
        ↪ age, String colour, String species)
            : base(name, diet, location, weight, age, colour, species)
        {

        }
    }
}
```

```
using System;
namespace Task02
{
    class Bird:Animal
    {
        private String species;
        private double wingSpan;

        public Bird(String name, String diet, String location, double weight, int
        ↪  age, String colour, String species, double wingSpan)
        : base(name, diet, location, weight, age, colour)
        {
            this.species = species;
            this.wingSpan = wingSpan;
        }


        public virtual void layEgg()
        {
            Console.WriteLine("A bird may lay eggs");
        }

        public virtual void fly()
        {
            Console.WriteLine("A bird flies");
        }

        public virtual void weather()
        {
            Console.WriteLine("A bird likes cold weather");
        }

        public virtual void fur()
        {
            Console.WriteLine("A bird has fur");
        }

        public virtual void beak()
        {
            Console.WriteLine("A bird has a beak");
        }

        public virtual void swim()
        {
            Console.WriteLine("A bird might swim");
        }

        public virtual void size()
        {
            Console.WriteLine("A bird has a size");
        }

        public virtual void feet()
```

```
        {
            Console.WriteLine("A bird may have webbed feet or claws");
        }



    }

}
```

```
using System;
namespace Task02
{
        class Penguin : Bird
        {


            public Penguin(String name, String diet, String location, double weight, int age, String colour, String species, double wingSpan)
            : base(name, diet, location, weight, age, colour, species, wingSpan)
            {

            }

            public override void layEgg()
            {
                Console.WriteLine("I lay eggs");
            }

            public override void fly()
            {
                Console.WriteLine("I can not fly");
            }

            public override void weather()
            {
                Console.WriteLine("I like the cold");
            }

            public override void fur()
            {
                Console.WriteLine("I have soft fur");
            }

            public override void beak()
            {
                Console.WriteLine("I have a small beak");
            }

            public override void swim()
            {
                Console.WriteLine("I can swim");
            }

            public override void size()
            {
                Console.WriteLine("I am small");
            }

            public override void feet()
            {
                Console.WriteLine("I have webbed feet");
            }
```

```

    }
}
```

```
using System;

namespace Overloading
{
    class Program
    {

        public static void methodToBeOverloaded(string name)
        {
            Console.WriteLine("Name: " + name);
        }

        public static void methodToBeOverloaded(String name, int age)
        {
            Console.WriteLine("Name: " + name + "\nAge: " + age);
        }
        static void Main(string[] args)
        {

            methodToBeOverloaded("Lucy");
            Console.WriteLine("-----------------------------");
            methodToBeOverloaded("Lucy", 20);
        }
    }
}
```

16 Bank Transactions

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊

I really enjoyed the creativity involved in designed and creating this banking system. It is good to be able to see how we can use c# to customise our programs.

Outcome	**Weight**
Principles	♦♦◊◊◊

I really enjoyed the creativity involved in designed and creating this banking system. It is good to be able to see how we can use c# to customise our programs.

Outcome	**Weight**
Build Programs	♦♦♦♦◊

I really enjoyed the creativity involved in designed and creating this banking system. It is good to be able to see how we can use c# to customise our programs.

Outcome	**Weight**
Design	♦♦♦◊◊

I really enjoyed the creativity involved in designed and creating this banking system. It is good to be able to see how we can use c# to customise our programs.

Date	**Author**	**Comment**
2020/05/05 22:04	Dale Orders	https://youtu.be/o4kfbPrVUqs
2020/05/05 22:06	Dale Orders	Ready to Mark
2020/05/07 23:40	Sanjay Segu	Complete
2020/05/07 23:40	Sanjay Segu	Good work dale
2020/05/07 23:40	Sanjay Segu	Dale*

Deakin University

Object Oriented Development

OnTrack Submission

Bank Transactions

Submitted By:
Dale Orders
dorders
2020/05/05 22:06

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦◊◊◊
Build Programs	♦♦♦♦◊
Design	♦♦♦◊◊

I really enjoyed the creativity involved in designed and creating this banking system. It is good to be able to see how we can use c# to customise our programs.

May 5, 2020

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task5_2
{
    public class WithdrawTransaction
    {

        private Account _account;
        private decimal _amount;
        private bool _executed;
        private bool _success;
        private bool _reversed;

        public bool Executed { get { return _executed; } }
        public bool Success { get { return _success; } }
        public bool Reversed { get { return _reversed; } }

        public WithdrawTransaction(Account account, decimal amount)
        {
            this._account = account;
            this._amount = amount;
            _reversed = false;
            _executed = false;
        }


        public void Print()
        {
            Console.WriteLine($"Successfully withdrew ${this._amount} from the account: {this._account.getName()}.");
            Console.WriteLine($"Current balance is {this._account.getBalance()}");
        }

        public void Execute()
        {
            if (Executed==false)
            {
                if(_account.getBalance()>0 && _amount<=_account.getBalance())
                {
                    _account.Withdraw(_amount);
                    _executed = true;
                }
                else
                {
                    Console.WriteLine("Insufficent funds");
                }
            }

            else { throw new InvalidOperationException("Withdrawal has already been performed"); }
        }
```

```

        public void Rollback()
        {
            if (Reversed==false && Executed==true)
            {
                _account.Deposit(_amount);
                _executed = false;
                _reversed = true;
            }
            else
            {
                Console.WriteLine("Transaction has already been reversed");
            }
        }



    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task5_2
{
    public class DepositTransaction
    {

        private Account _account;
        private decimal _amount;
        private bool _executed;
        private bool _success;
        private bool _reversed;

        public bool Executed { get { return _executed; } }
        public bool Success { get { return _success; } }
        public bool Reversed { get { return _reversed; } }

        public DepositTransaction(Account account, decimal amount)
        {
            this._account = account;
            this._amount = amount;
            _reversed = false;
            _executed = false;
        }

        public void print()
        {
            Console.WriteLine($"Successfully deposited ${this._amount} from the account: {this._account.getName()}.");
            Console.WriteLine($"Current balance is {this._account.getBalance()}");
        }

        public void Execute()
        {
            if (Executed == false)
            {
                _account.Deposit(_amount);
                _executed = true;
                _reversed = false;
            }
            else
            {
                throw new InvalidOperationException("You have already performed this transaction");
            }
        }

        public void Rollback()
        {
            if (Executed == true && Reversed == false)
            {
```

```
                _account.Withdraw(_amount);
                _executed = false;
                _reversed = true;

            }
            else
            {
                Console.WriteLine("There is no deposit to reverse");
            }


        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;

namespace task5_2
{
    public class TransferTransaction
    {

        private Account _fromAccount;
        private Account _toAccount;
        private decimal _amount;
        private DepositTransaction _deposit;
        private WithdrawTransaction _withdraw;
        private bool _executed;
        private bool _reversed;
        private bool _success;

        public bool Executed { get { return _executed; } }
        public bool Success { get { return _success; } }
        public bool Reversed { get { return _reversed; } }

        public TransferTransaction(Account fromAccount, Account toAccount, decimal
        ↪ amount)
        {
            _deposit = new DepositTransaction(toAccount, amount);
            _withdraw = new WithdrawTransaction(fromAccount, amount);
            _executed = false;
            this._fromAccount = fromAccount;
            this._toAccount = toAccount;
            this._amount = amount;
        }

        public bool success
        {
            get
            {
                if(_deposit.Success==true && _withdraw.Success==true)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }


        public void print()
        {
            if (_executed == true)
```

```
            {
                Console.WriteLine("Transaction has been performed successfully");
                Console.WriteLine($"Transferred ${_amount} from {_fromAccount.getName()} to {_toAccount.getName()}");
            }

            else
            {
                Console.WriteLine("Transaction has not been successful");
            }

        }

        public void Execute()
        {
            if(Executed==false)
            {
                if(_fromAccount.getBalance()<_amount)
                {
                    Console.WriteLine("Insufficient funds");

                }
                else
                {
                    _withdraw.Execute();
                    _deposit.Execute();
                    _executed = true;

                }
            }
            else { throw new InvalidOperationException("Transacation has been already performed"); }
        }

        public void Rollback()
        {
            if(Reversed==true)
            {
                throw new InvalidOperationException("Reverse transaction has already been performed");
            }
            if(_toAccount.getBalance()<_amount)
            {
                throw new InvalidOperationException("Insufficient funds");
            }
            else if(Executed==true)
            {
                _deposit.Rollback();
                _withdraw.Rollback();
                _reversed = true;
                _executed = false;

            }
```

```
            }

        }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

namespace task5_2
{


    class BankSystem
    {

        enum MenuOption
        {
            Withdraw,
            Deposit,
            Print,
            TransferTransaction,
            Quit
        }

        static void DoDeposit(Account account)
        {
            Console.WriteLine("Enter deposit amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            DepositTransaction deposit = new DepositTransaction(account, amount);

            while(account != null)
            {
                string execute;
                string rollback;

                Console.WriteLine($"Deposit the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Rollback the transaction. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
                    deposit.Execute();
                }
                if (rollback == "Y".ToLower())
                {
                    deposit.Rollback();
                    break;
                }
                else
                {
                    deposit.print();
                    break;
                }

            }
        }
```

```

        static void DoWithdrawal(Account account)
        {

            Console.WriteLine("Enter withdrawal amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            WithdrawTransaction deposit = new WithdrawTransaction(account, amount);

            while (account != null)
            {
                string execute;
                string rollback;

                Console.WriteLine($"Withdraw the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Rollback the transaction. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
                    deposit.Execute();
                }
                if (rollback == "Y".ToLower())
                {
                    deposit.Rollback();
                    break;
                }
                else
                {
                    deposit.Print();
                    break;
                }

            }
        }

        static void DoTransfer(Account fromaccount, Account toaccount, decimal amount)
        {
;
            TransferTransaction transferobj = new TransferTransaction(fromaccount, toaccount, amount);

            while (fromaccount!= null)
            {
                string execute;
                string rollback;

                Console.WriteLine($"Transfer the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Would you like to rollback the trandfer. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
```

```
                    transferobj.Execute();
                }
                if (rollback == "Y".ToLower())
                {
                    transferobj.Rollback();
                    break;
                }
                else
                {
                    transferobj.print();
                    break;
                }
            }
        }


        static MenuOption ReadUserOption()
        {
            int? option = null;
            do
            {
                Console.ForegroundColor = ConsoleColor.Blue;
                Console.WriteLine("\n   Welcome to the Banking System    ");
                Console.WriteLine("|-------------------------------------|");
                Console.WriteLine("\n        Bank System Menu           ");
                Console.WriteLine("|-------------------------------------|");
                Console.WriteLine("\n  1. Withdraw                       ");
                Console.WriteLine("|-------------------------------------|");
                Console.WriteLine("\n  2. Deposit                        ");
                Console.WriteLine("|-------------------------------------|");
                Console.WriteLine("\n  3. Print Account                  ");
                Console.WriteLine("|-------------------------------------|");
                Console.WriteLine("\n  4. Transfer Transaction           ");
                Console.WriteLine("|-------------------------------------|");
                Console.WriteLine("\n  5. Quit                           ");
                Console.WriteLine("|-------------------------------------|");

                try
                {
                    option = Convert.ToInt32(Console.ReadLine());
                    if (option > 4 || option < 1)
                    {
                        option = null;
                        Console.WriteLine("Please enter a number from 1 to 4 from the menut");
                    }
                }
                catch (FormatException)
                {
                    Console.WriteLine("Invalid input. Enter an integer from 1-4");
                }

            } while (option == null);
```

```

            return (MenuOption)option;
        }


        static void DoPrint(Account account)
        {

            account.Print();

        }


        static void Main(string[] args)
        {
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine(new string(' ', 10) + "BANKING SYSTEM");
            Console.WriteLine(new string(' ', 10) + new string('-', 82));
            Console.Write("{0,19}{1,20}{2,20}{3,30} ", "|Account|", "|Action|",
             ↪  "|Status|", "|Current Balance|");


            Console.WriteLine("\n\nEnter name of user one: ");
            string name1 = Console.ReadLine();
            Console.WriteLine("Enter starting balance of user one: ");
            decimal balance1 = Convert.ToDecimal(Console.ReadLine());
            Console.WriteLine("Enter name of user two: ");
            string name2 = Console.ReadLine();
            Console.WriteLine("Enter starting balance of user two: ");
            decimal balance2 = Convert.ToDecimal(Console.ReadLine());
            Account person_one = new Account(balance1, name1);
            Account person_two = new Account(balance2, name2);
            MenuOption option;

            do
            {
                option = ReadUserOption() - 1;

                switch (option)
                {
                    case MenuOption.Withdraw:
                        Console.WriteLine("You have selected withdraw");
                        Console.WriteLine("|-----------------------------------|");
                        Console.WriteLine("      Account holder: " + name1        );
                        Console.WriteLine("|-----------------------------------|");
                        DoWithdrawal(person_one);
                        Console.WriteLine("|-----------------------------------|");
                        Console.WriteLine("      Account holder: " + name2);
                        Console.WriteLine("|-----------------------------------|");
                        DoWithdrawal(person_two);
                        break;
                    case MenuOption.Deposit:
                        Console.WriteLine("You have selected deposit");
```

```
                        Console.WriteLine("|-----------------------------------|");
                        Console.WriteLine("      Account holder: " + name1        );
                        Console.WriteLine("|-----------------------------------|");
                        DoDeposit(person_one);
                        Console.WriteLine("|-----------------------------------|");
                        Console.WriteLine("      Account holder: " + name2        );
                        Console.WriteLine("|-----------------------------------|");
                        DoDeposit(person_two);
                        break;
                    case MenuOption.Print:
                        Console.WriteLine("You have selected print");
                        Console.WriteLine("|-----------------------------------|");
                        Console.WriteLine("      Account holder: " + name1        );
                        Console.WriteLine("|-----------------------------------|");
                        DoPrint(person_one);
                        Console.WriteLine("|-----------------------------------|");
                        Console.WriteLine("      Account holder: " + name2        );
                        Console.WriteLine("|-----------------------------------|");
                        DoPrint(person_two);
                        break;
                    case MenuOption.TransferTransaction:
                        Console.WriteLine("You have selected transfer");
                        Console.WriteLine("Enter amount to transfer: ");
                        decimal amount = Convert.ToDecimal(Console.ReadLine());
                        Console.WriteLine("Would you like to transfer from User One's account to User Two's account? Y/N");
                        string input = Console.ReadLine();
                        if (input.ToLower() == "Y".ToLower())
                        {
                            DoTransfer(person_one, person_two, amount);
                            break;
                        }
                        Console.WriteLine("Would you like to transfer from User Two's account to User One's account? Y/N");
                        string select = Console.ReadLine();
                        if (select.ToLower() == "Y".ToLower())
                        {
                            DoTransfer(person_two, person_one, amount);
                            break;
                        }
                        else
                        break;
                    case MenuOption.Quit:
                        Console.WriteLine("Goodbye");
                        break;

                }
            } while (option != MenuOption.Quit);
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task5_2
{
    public class Account
    {
        //instance variables declared
        private decimal _balance;
        private string _name;

        //constructor
        public Account(decimal balance, string name)
        {

            this._name = name;
            if (balance <= 0)
                return;
            this._balance = balance;

        }

        //prints name and balance
        public void Print()
        {
            Console.WriteLine("The name of the account holder is " + getName() +
            ↪  "\nCurrent Account Balance is $" + getBalance());
        }

        //returns name
        public String getName()
        {
            return this._name;
        }

        //returns balance
        public decimal getBalance()
        {
            return this._balance;
        }

        //increases balance by adding deposit
        //boolean ensure that no 0 or a negative value can not be input
        public Boolean Deposit(decimal amount)
        {

            if (amount <= 0)
                return false;

            this._balance += amount;
            return true;

```

```
        }


        //decreases balanace by withdrawing the giving input amount
        //boolean values protect against overdrawing from account
        public Boolean Withdraw(decimal amount)
        {
            if (amount > this._balance || amount < 0)
                return false;


            this._balance -= amount;
            return true;

        }

    }
}
```

17 C# Essentials: Polymorphism

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Evaluate Code	♦♦♦◊◊

This was an interesting task, that allowed us to see how in inheritance works within the design of the program.

Outcome	Weight
Principles	♦♦◊◊◊

This was an interesting task, that allowed us to see how in inheritance works within the design of the program.

Outcome	Weight
Build Programs	♦◊◊◊◊

This was an interesting task, that allowed us to see how in inheritance works within the design of the program.

Outcome	Weight
Design	♦◊◊◊◊

This was an interesting task, that allowed us to see how in inheritance works within the design of the program.

Date	Author	Comment
2020/05/05 23:47	Dale Orders	https://youtu.be/KQJFDimPOY8
2020/05/06 00:52	Dale Orders	Ready to Mark
2020/05/07 23:42	Sanjay Segu	Discuss
2020/05/07 23:42	Sanjay Segu	Anyway, can you please tell me what are the differences between ‘Overloading‘ and ‘Overriding‘ methods ?
2020/05/14 19:11	Dale Orders	Overriding a method occurs when a method in a derived class is invoked and and overrides the virtual method of the base class
2020/05/14 19:12	Dale Orders	This is fundamental principle of inheritance
2020/05/14 19:13	Dale Orders	Overloading a method, on the other hand, is when there are multiple methods with the same name, but with different parameters. The method which is called depends on the parameter.
2020/05/14 21:23	Sanjay Segu	Good answer.
2020/05/14 21:24	Sanjay Segu	Complete

Deakin University

Object Oriented Development

OnTrack Submission

C# Essentials: Polymorphism

Submitted By:
Dale Orders
dorders
2020/05/06 00:52

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦♦♦◊◊
Principles	♦♦◊◊◊
Build Programs	♦◊◊◊◊
Design	♦◊◊◊◊

This was an interesting task, that allowed us to see how in inheritance works within the design of the program.

May 7, 2020

```
using System;
namespace task6_1
{
    public class Bird
    {
        public string name { get; set; }

        public Bird()
        {

        }

        public virtual void fly()
        {
            Console.WriteLine("Flap, Flap, Flap");
        }

        public override string ToString()
        {
            return "A bird called " + name;
        }
    }
}
```

```
using System;
namespace task6_1
{
    class Penguin:Bird
    {
        public override void fly()
        {
            Console.WriteLine("Penguins cannot fly");
        }

        public override string ToString()
        {
            return "A penguin named " + base.name;
        }
    }
}
```

```
using System;
namespace task6_1
{
    class Duck:Bird
    {
        public double size { get; set; }
        public String kind { get; set; }

        public override string ToString()
        {
            return "A duck named " + base.name + " is a " + size + " inch " + kind;
        }
    }
}
```

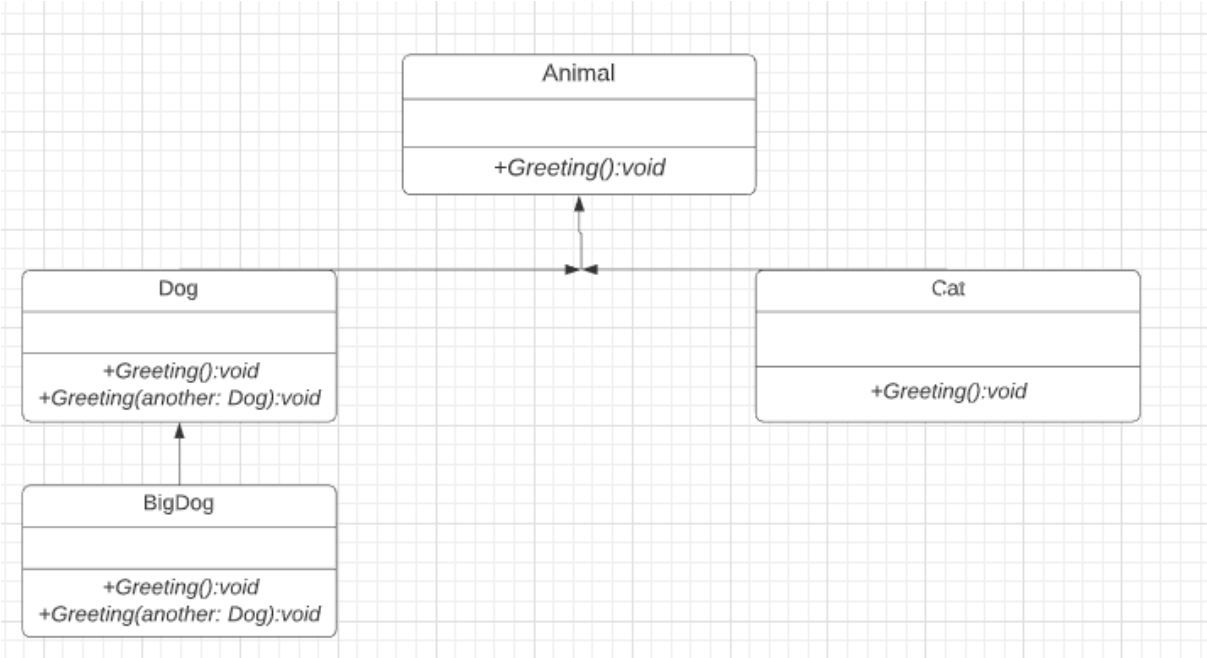
```
using System;
using System.Collections.Generic;
using System.Linq;

namespace task6_1
{
    class Program
    {
        static void Main(string[] args)
        {
            //Step 1
            Bird bird1 = new Bird();
            Bird bird2 = new Bird();

            bird1.name = "Feathers";
            bird2.name = "Polly";

            Console.WriteLine(bird1.ToString());
            bird1.fly();

            Console.WriteLine(bird2.ToString());
            bird2.fly();

            Console.WriteLine("-----------------");

            //Step 2

            Penguin penguin1 = new Penguin();
            Penguin penguin2 = new Penguin();
            penguin1.name = "Happy Feet";
            penguin2.name = "Gloria";

            Console.WriteLine(penguin1.ToString());
            penguin1.fly();

            Console.WriteLine(penguin2.ToString());
            penguin2.fly();

            Duck duck1 = new Duck();
            duck1.name = "Daffy";
            duck1.size = 15;
            duck1.kind = "Maillard";

            Duck duck2 = new Duck();
            duck2.name = "Donald";
            duck2.size = 20;
            duck2.kind = "Decoy";

            Console.WriteLine(duck1.ToString());
            Console.WriteLine(duck2.ToString());

            Console.WriteLine("-----------------");

```

```
            //Step3

            List<Bird> birds = new List<Bird>();
            Bird bird3 = new Bird();
            bird3.name = "Feathers";
            Bird bird4 = new Bird();
            bird3.name = "Polly";

            Penguin penguin3 = new Penguin();
            Penguin penguin4 = new Penguin();
            penguin3.name = "Happy Feet";
            penguin4.name = "Gloria";

            Duck duck3 = new Duck();
            duck3.name = "Daffy";
            duck3.size = 15;
            duck3.kind = "Maillard";

            Duck duck4 = new Duck();
            duck4.name = "Donald";
            duck4.size = 20;
            duck4.kind = "Decoy";

            birds.Add(bird3);
            birds.Add(bird4);
            birds.Add(penguin3);
            birds.Add(penguin4);
            birds.Add(duck3);
            birds.Add(duck4);

            birds.Add(new Bird { name = "Tweety" });

            foreach (Bird bird in birds)
            {
                Console.WriteLine(bird);
            }

            //Step 3


            Duck duck5 = new Duck();
            duck5.name = "Daffy";
            duck5.size = 15;
            duck5.kind = "Maillard";

            Duck duck6 = new Duck();
            duck6.name = "Donald";
            duck6.size = 20;
            duck6.kind = "Decoy";


            List<Duck> ducksToAdd = new List<Duck>()
            {
```

```
                duck5,
                duck6
            };

            IEnumerable<Bird> upcastDucks = ducksToAdd;

            List <Bird> birds2 = new List<Bird>();
            birds2.Add(new Bird(){ name="Feather"});

            birds2.AddRange(upcastDucks);

            foreach(Bird bird in birds)
            {
                Console.WriteLine(bird);
            }
        }
    }
}
```

SIT232: Object Orientated Programming

Task 6.1 Report




```
// Using the subclasses
Cat cat1 = new Cat();
cat1.greeting();
Dog dog1 = new Dog();
dog1.greeting();
BigDog bigDog1 = new BigDog();
bigDog1.greeting();
```

This will produce an error in the code there is case sensitivity in c#, which means that greeting() will be treated as a different method compared to Greeting().

```
// Using Polymorphism
Animal animal1 = new Cat();
animal1.greeting();
Animal animal2 = new Dog();
animal2.greeting();
Animal animal3 = new BigDog();
animal3.greeting();
Animal animal4 = new Animal();
```

This will fail as it attempts to instantiate, or create an object, from an abstract class, which is not possible in c#.

```
// Downcast
Dog dog2 = (Dog)animal2;
BigDog bigDog2 = (BigDog)animal3;
Dog dog3 = (Dog)animal3;
Cat cat2 = (Cat)animal2;
dog2.greeting(dog3);
dog3.greeting(dog2);
dog2.greeting(bigDog2);
bigDog2.greeting(dog2);
bigDog2.greeting(bigDog1);
```

This will fail as it attempts to animal2 as a cat type, but this is not possible as animal was previously cast as a dog, which exists along a difference branch in the family of inheritance (see UML above). As such, code will not produce an output, due to a fundamental misunderstanding of inheritance.

18 Multiple Bank Accounts

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Principles	♦♦♦◊◊

This task presented a great opportunity to expand upon my original design and impart greater functionality to my system.

Outcome	Weight
Build Programs	♦♦♦♦◊

This task presented a great opportunity to expand upon my original design and impart greater functionality to my system.

Outcome	Weight
Design	♦♦♦♦◊

This task presented a great opportunity to expand upon my original design and impart greater functionality to my system.

Outcome	Weight
Justify	♦◊◊◊◊

This task presented a great opportunity to expand upon my original design and impart greater functionality to my system.

Date	Author	Comment
2020/05/13 21:48	Dale Orders	Ready to Mark
2020/05/13 22:01	Dale Orders	https://www.youtube.com/watch?v=nIKgs2EyrTw&feature=youtu.be
2020/05/17 17:54	Sanjay Segu	Complete
2020/05/17 17:54	Sanjay Segu	Well done Dale

Deakin University

Object Oriented Development

OnTrack Submission

Multiple Bank Accounts

Submitted By:
Dale Orders
dorders
2020/05/13 21:48

Tutor:
Sanjay Segu

Outcome	**Weight**
Principles	♦♦♦◊◊
Build Programs	♦♦♦♦◊
Design	♦♦♦♦◊
Justify	♦◊◊◊◊

This task presented a great opportunity to expand upon my original design and impart greater functionality to my system.

May 13, 2020

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

namespace task6_2
{


    class BankSystem
    {

        enum MenuOption
        {
            AddnewAccount,
            Withdraw,
            Deposit,
            Print,
            TransferTransaction,
            Quit
        }

        static void AddAccount(Bank bank)
        {
            Console.WriteLine("Enter the name of the account holder");
            string name = Console.ReadLine();
            Console.WriteLine("Enter their starting accoung balance");
            decimal balance = Convert.ToDecimal(Console.ReadLine());
            bank.AddAccount(new Account(balance, name));
        }

        private static Account FindAccount(Bank bank)
        {
            Account account = null;
            Console.WriteLine("Enter the account name");
            string name = Console.ReadLine();
            account = bank.GetAccount(name);
            if(account==null)
            {
                Console.WriteLine("Account does not exist");
            }
            return account;
        }
        static void DoDeposit(Bank bank)
        {
            Account account = FindAccount(bank);
            Console.WriteLine("Enter deposit amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            DepositTransaction deposit = new DepositTransaction(account, amount);

            if(account!=null)
            {
                string execute;
```

```
                string rollback;

                Console.WriteLine($"Deposit the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Rollback the transaction. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
                    bank.ExecuteTransaction(deposit);
                }
                if (rollback == "Y".ToLower())
                {
                    deposit.Rollback();
                }

            }
        }

        static void DoWithdrawal(Bank bank)
        {
            Account account = FindAccount(bank);
            Console.WriteLine("Enter withdrawal amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            WithdrawTransaction withdrawal = new WithdrawTransaction(account,
            ↪ amount);

            if (account != null)
            {
                string execute;
                string rollback;

                Console.WriteLine($"Withdraw the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Rollback the transaction. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
                    bank.ExecuteTransaction(withdrawal);
                }
                if (rollback == "Y".ToLower())
                {
                    withdrawal.Rollback();
                }
            }

        }

        static void DoTransfer(Bank bank1)
        {
            Account fromaccount = FindAccount(bank1);
            Account toaccount = FindAccount(bank1);
            Console.WriteLine("Transfer the following amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
```

```
            TransferTransaction transfer = new TransferTransaction(fromaccount, toaccount, amount);

            if(bank1!=null)
            {
                string execute;
                string rollback;

                Console.WriteLine($"Transfer the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Would you like to rollback the trandfer. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
                    bank1.ExecuteTransaction(transfer);
                }
                if (rollback == "Y".ToLower())
                {
                    transfer.Rollback();
                }
            }
        }


        static MenuOption ReadUserOption()
        {
            int? option = null;
            do
            {
                Console.WriteLine("-------------------------------------------------------");
                Console.WriteLine("Please select from the following options");
                Console.WriteLine("MENU \n1. Add new Account \n2. Withdraw \n3. Deposit \n4. Print \n5. Transfer Transaction \n6. Quit");
                Console.WriteLine("-------------------------------------------------------");

                try
                {
                    option = Convert.ToInt32(Console.ReadLine());
                    if (option > 5 || option < 1)
                    {
                        option = null;
                        Console.WriteLine("Please enter a number from 1 to 5 from the menut");
                    }
                }
                catch (FormatException)
                {
                    Console.WriteLine("Invalid input. Enter an integer from 1-5");
                }

            } while (option == null);
```

```

            return (MenuOption)option;
        }


        static void DoPrint(Bank bank)
        {
            Account account = FindAccount(bank);
            if(account!=null)
            {
                account.Print();
            }

        }


        static void Main(string[] args)
        {
            Bank newbank=new Bank();

            Account person_one = new Account(100, "Jon");
            Account person_two = new Account(150, "Amy");
            MenuOption option;

            do
            {
                option = ReadUserOption() - 1;

                switch (option)
                {
                    case MenuOption.AddnewAccount:
                        AddAccount(newbank);
                        break;
                    case MenuOption.Withdraw:
                        Console.WriteLine("You have selected withdraw");
                        DoWithdrawal(newbank);
                        break;
                    case MenuOption.Deposit:
                        Console.WriteLine("You have selected deposit");
                        DoDeposit(newbank);
                        break;
                    case MenuOption.Print:
                        Console.WriteLine("You have selected print");
                        DoPrint(newbank);
                        break;
                    case MenuOption.TransferTransaction:
                        DoTransfer(newbank);
                        break;
                    case MenuOption.Quit:
                        Console.WriteLine("Goodbye");
                        break;

                }
```

```
            } while (option != MenuOption.Quit);
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task6_2
{
    public class Bank
    {
        private static List<Account> _accounts=new List<Account>();


        public void AddAccount(Account account)
        {
            _accounts.Add(account);
        }

        public Account GetAccount(String name)
        {
            foreach(Account account in _accounts)
            {
                if(account.getName().ToLower()==name.ToLower())
                {
                    return account;
                }
            }
            { return null; }
        }

        public void ExecuteTransaction(DepositTransaction transaction)
        {
            try
            {
                transaction.Execute();
            }

            catch(InvalidOperationException)
            {
                Console.WriteLine("Error in performing the transaction");
            }
        }

        public void ExecuteTransaction(WithdrawTransaction transaction)
        {
            try
            {
                transaction.Execute();
            }

            catch (InvalidOperationException)
            {
                Console.WriteLine("Error in performing the transaction");
            }

```

```
        }

        public void ExecuteTransaction(TransferTransaction transaction)
        {
            try
            {
                transaction.Execute();
            }

            catch (InvalidOperationException)
            {
                Console.WriteLine("Error in performing the transaction");
            }

        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task6_2
{
    public class Account
    {
        //instance variables declared
        private decimal _balance;
        private string _name;

        //constructor
        public Account(decimal balance, string name)
        {

            this._name = name;
            if (balance <= 0)
                return;
            this._balance = balance;

        }

        //prints name and balance
        public void Print()
        {
            Console.WriteLine("The name of the account holder is " + getName() +
            ↪  "\nCurrent Account Balance is $" + getBalance());
        }

        //returns name
        public String getName()
        {
            return this._name;
        }

        //returns balance
        public decimal getBalance()
        {
            return this._balance;
        }

        //increases balance by adding deposit
        //boolean ensure that no 0 or a negative value can not be input
        public Boolean Deposit(decimal amount)
        {

            if (amount <= 0)
                return false;

            this._balance += amount;
            return true;

```

```
        }


        //decreases balanace by withdrawing the giving input amount
        //boolean values protect against overdrawing from account
        public Boolean Withdraw(decimal amount)
        {
            if (amount > this._balance || amount < 0)
                return false;


            this._balance -= amount;
            return true;

        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task6_2
{
    public class WithdrawTransaction
    {

        private Account _account;
        private decimal _amount;
        private bool _executed;
        private bool _success;
        private bool _reversed;

        public bool Executed { get { return _executed; } }
        public bool Success { get { return _success; } }
        public bool Reversed { get { return _reversed; } }

        public WithdrawTransaction(Account account, decimal amount)
        {
            this._account = account;
            this._amount = amount;
            _reversed = false;
            _executed = false;
        }


        public void Print()
        {
            Console.WriteLine($"Successfully withdrew ${this._amount} from the account: {this._account.getName()}");
        }

        public void Execute()
        {
            if (Executed == false)
            {
                if (_account.getBalance() > 0 && _amount <= _account.getBalance())
                {
                    _account.Withdraw(_amount);
                    _executed = true;
                }
                else
                {
                    Console.WriteLine("Insufficent funds");
                }
            }

            else { throw new InvalidOperationException("Withdrawal has already been performed"); }
        }

```

```
        public void Rollback()
        {
            if (Reversed == false && Executed == true)
            {
                _account.Deposit(_amount);
                _executed = false;
                _reversed = true;
            }
            else
            {
                Console.WriteLine("Transaction has already been reversed");
            }
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task6_2
{
    public class DepositTransaction
    {

        private Account _account;
        private decimal _amount;
        private bool _executed;
        private bool _success;
        private bool _reversed;

        public bool Executed { get { return _executed; } }
        public bool Success { get { return _success; } }
        public bool Reversed { get { return _reversed; } }

        public DepositTransaction(Account account, decimal amount)
        {
            this._account = account;
            this._amount = amount;
            _reversed = false;
            _executed = false;
        }

        public void print()
        {
            Console.WriteLine($"Successfully deposited ${this._amount} into the account: {this._account.getName()}");
        }

        public void Execute()
        {
            if (Executed == false)
            {
                _account.Deposit(_amount);
                _executed = true;
                _reversed = false;
            }
            else
            {
                throw new InvalidOperationException("You have already performed this transaction");
            }
        }

        public void Rollback()
        {
            if (Executed == true && Reversed == false)
            {
                _account.Withdraw(_amount);
```

```
                _executed = false;
                _reversed = true;

            }
            else
            {
                Console.WriteLine("There is no deposit to reverse");
            }


        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;

namespace task6_2
{
    public class TransferTransaction
    {

        private Account _fromAccount;
        private Account _toAccount;
        private decimal _amount;
        private DepositTransaction _deposit;
        private WithdrawTransaction _withdraw;
        private bool _executed;
        private bool _reversed;
        private bool _success;

        public bool Executed { get { return _executed; } }
        public bool Success { get { return _success; } }
        public bool Reversed { get { return _reversed; } }

        public TransferTransaction(Account fromAccount, Account toAccount, decimal
        ↪ amount)
        {
            _deposit = new DepositTransaction(toAccount, amount);
            _withdraw = new WithdrawTransaction(fromAccount, amount);
            _executed = false;
            this._fromAccount = fromAccount;
            this._toAccount = toAccount;
            this._amount = amount;
        }

        public bool success
        {
            get
            {
                if (_deposit.Success == true && _withdraw.Success == true)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }


        public void print()
        {
            if (_executed == true)
```

```
            {
                Console.WriteLine("Transaction has been performed successfully");
                Console.WriteLine($"Transferred ${_amount} from {_fromAccount.getName()} to {_toAccount.getName()}");
            }

            else
            {
                Console.WriteLine("Transaction has not been successful");
            }

        }

        public void Execute()
        {
            if (Executed == false)
            {
                if (_fromAccount.getBalance() < _amount)
                {
                    Console.WriteLine("Insufficient funds");

                }
                else
                {
                    _withdraw.Execute();
                    _deposit.Execute();
                    _executed = true;

                }
            }
            else { throw new InvalidOperationException("Transacation has been already performed"); }
        }

        public void Rollback()
        {
            if (Reversed == true)
            {
                throw new InvalidOperationException("Reverse transaction has already been performed");
            }
            if (_toAccount.getBalance() < _amount)
            {
                throw new InvalidOperationException("Insufficient funds");
            }
            else if (Executed == true)
            {
                _deposit.Rollback();
                _withdraw.Rollback();
                _reversed = true;
                _executed = false;

            }
```

```
            }

        }

}
```

19 A Simple Reaction-Timer Controller

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Evaluate Code	♦♦♦◊◊

This was a great task as it gave me a board introduction to the concept of finite state machines and how this logic can be used to enact certain conditions.

Outcome	**Weight**
Principles	♦◊◊◊◊

This was a great task as it gave me a board introduction to the concept of finite state machines and how this logic can be used to enact certain conditions.

Outcome	**Weight**
Build Programs	♦♦♦◊◊

This was a great task as it gave me a board introduction to the concept of finite state machines and how this logic can be used to enact certain conditions.

Outcome	**Weight**
Design	♦♦♦◊◊

This was a great task as it gave me a board introduction to the concept of finite state machines and how this logic can be used to enact certain conditions.

Outcome	**Weight**
Justify	♦◊◊◊◊

This was a great task as it gave me a board introduction to the concept of finite state machines and how this logic can be used to enact certain conditions.

Date	**Author**	**Comment**
2020/05/27 13:21	Dale Orders	Ready to Mark
2020/05/27 13:33	Dale Orders	https://www.youtube.com/watch?v=7vSPdhYGzfw&feature=youtu.be
2020/05/30 14:06	Sanjay Segu	Complete

Deakin University

Object Oriented Development

OnTrack Submission

A Simple Reaction-Timer Controller

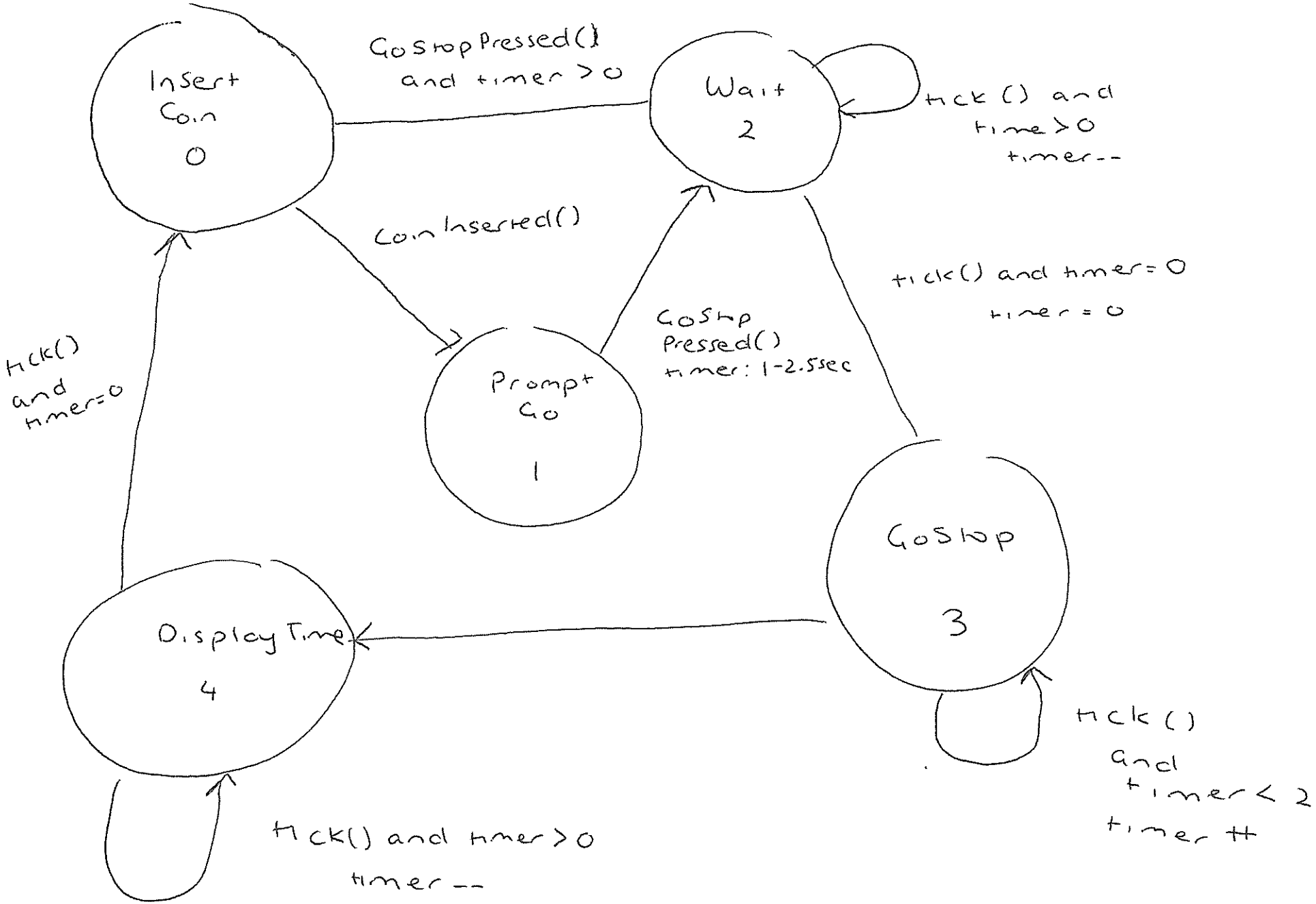
Submitted By:
Dale Orders
dorders
2020/05/27 13:21

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦♦♦◊◊
Principles	♦◊◊◊◊
Build Programs	♦♦♦◊◊
Design	♦♦♦◊◊
Justify	♦◊◊◊◊

This was a great task as it gave me a board introduction to the concept of finite state machines and how this logic can be used to enact certain conditions.

May 27, 2020


Insert Coin
0
GoStopPressed()
and timer > 0
Wait
2
tick () and
time > 0
timer--
CoinInserted()
tick() and timer = 0
timer = 0
GoStop
Pressed()
timer: 1-2.5sec
Prompt
Go
1
tick()
and
timer=0
GoStop
3
Display Time
4
tick()
and
timer < 2
timer ++
tick() and timer > 0
timer --

```
using System;


namespace task5_3
{
    public class SimpleReactionController:IController
    {
        private const int MIN_WAIT_TIME = 100;
        private const int MAX_WAIT_TIME = 250;
        private const int MAX_GAME_TIME = 200;
        private const int GAMEOVER_TIME = 300;
        private const double TICKS_PER_SECOND =100.0;

        private State _state;
        private IGui Gui { get; set; }
        private IRandom Rng { get; set; }
        private int Ticks { get; set; }

        public void Connect(IGui gui, IRandom rng)
        {
            Gui = gui;
            Rng = rng;
            Init();
        }
        public void Init()
        {
            _state = new OnState(this);
        }

        public void GoStopPressed()
        {
            _state.GoStopPressed();
        }
        public void CoinInserted()
        {
            _state.CoinInserted();
        }
        public void Tick()
        {
            _state.Tick();
        }


        private void SetState(State state)
        {
            _state = state;
        }

        private abstract class State
        {
            protected SimpleReactionController _controller;

            public State(SimpleReactionController controller)
```

```
            {
                _controller = controller;
            }

            public abstract void CoinInserted();
            public abstract void GoStopPressed();
            public abstract void Tick();

        }

        private class OnState : State
        {
            public OnState(SimpleReactionController controller):base(controller)
            {
                _controller.Gui.SetDisplay("Insert Coin");
            }

            public override void CoinInserted()
            {
                _controller.SetState(new ReadyState(_controller));
            }
            public override void GoStopPressed() { }
            public override void Tick() { }

        }

        private class WaitState : State
        {
            private int _waitTime;
            public WaitState(SimpleReactionController controller):base(controller)
            {
                _controller.Gui.SetDisplay("Wait...");
                _controller.Ticks = 0;
                _waitTime = _controller.Rng.GetRandom(MIN_WAIT_TIME, MAX_WAIT_TIME);
            }

            public override void CoinInserted() { }

            public override void GoStopPressed()
            {
                _controller.SetState(new OnState(_controller));
            }

            public override void Tick()
            {
                _controller.Ticks++;
                if(_controller.Ticks==_waitTime)
                {
                    _controller.SetState(new RunningState(_controller));
                }
            }
        }

```

```
        private class RunningState : State
        {
            public RunningState(SimpleReactionController controller) : base(controller)
            {
                _controller.Gui.SetDisplay("0.00");
                _controller.Ticks = 0;
            }
            public override void CoinInserted() { }
            public override void GoStopPressed()
            {
                _controller.SetState(new GameOverState(_controller));
            }

            public override void Tick()
            {
                _controller.Ticks++;
                _controller.Gui.SetDisplay((_controller.Ticks / TICKS_PER_SECOND).ToString("0.00"));
                if (_controller.Ticks == MAX_GAME_TIME)
                {
                    _controller.SetState(new GameOverState(_controller));
                }
            }

        }



        private class ReadyState : State
        {
            public ReadyState(SimpleReactionController controller):base(controller)
            {
                _controller.Gui.SetDisplay("Press Go");
            }

            public override void CoinInserted() { }
            public override void GoStopPressed()
            {
                _controller.SetState(new WaitState(_controller));
            }
            public override void Tick() { }
        }

        private class GameOverState : State
        {
            public GameOverState(SimpleReactionController controller) : base(controller)
            {
                _controller.Ticks = 0;
            }
            public override void CoinInserted() { }
            public override void GoStopPressed()
```

```
            {
                _controller.SetState(new OnState(_controller));
            }
            public override void Tick()
            {
                _controller.Ticks++;
                if (_controller.Ticks == GAMEOVER_TIME)
                {
                    _controller.SetState(new OnState(_controller));
                }
            }
        }

    }
}
```

20 An Enhanced Reaction-Timer Controller

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Evaluate Code	♦◊◊◊◊

This task presented a great opportunity to extend task 5.3 and develop some of our own test methods. It was something that forced me to think more broadly about the certain outcomes I wanted to test for and how I could implement them into my program. As such I thought it was very beneficial to my learning.

Outcome	Weight
Principles	♦♦♦◊◊

This task presented a great opportunity to extend task 5.3 and develop some of our own test methods. It was something that forced me to think more broadly about the certain outcomes I wanted to test for and how I could implement them into my program. As such I thought it was very beneficial to my learning.

Outcome	Weight
Build Programs	♦♦♦◊◊

This task presented a great opportunity to extend task 5.3 and develop some of our own test methods. It was something that forced me to think more broadly about the certain outcomes I wanted to test for and how I could implement them into my program. As such I thought it was very beneficial to my learning.

Outcome	Weight
Design	♦♦♦♦◊

This task presented a great opportunity to extend task 5.3 and develop some of our own test methods. It was something that forced me to think more broadly about the certain outcomes I wanted to test for and how I could implement them into my program. As such I thought it was very beneficial to my learning.

Date	Author	Comment
2020/05/29 16:18	Dale Orders	Ready to Mark
2020/05/30 14:20	Sanjay Segu	Demonstrate
2020/05/30 14:20	Sanjay Segu	Code seems ok Dale
2020/05/30 14:20	Sanjay Segu	Demo video
2020/05/30 14:21	Sanjay Segu	Please stress on the tester (Which you are expected to develop for this one) and the testing the GUI with different scenarios
2020/06/03 22:10	Dale Orders	https://www.youtube.com/watch?v=i-XCDht2kGQ&feature=youtu.be

Deakin University

Object Oriented Development

OnTrack Submission

An Enhanced Reaction-Timer Controller

Submitted By:
Dale Orders
dorders
2020/05/29 16:18

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦♦◊◊
Build Programs	♦♦♦◊◊
Design	♦♦♦♦◊

This task presented a great opportunity to extend task 5.3 and develop some of our own test methods. It was something that forced me to think more broadly about the certain outcomes I wanted to test for and how I could implement them into my program. As such I thought it was very beneficial to my learning.

May 29, 2020

```
using System;


namespace task5_3
{
    public class EnhancedReactionController : IController
    {
        private const int MAX_READY_TIME = 1000;
        private const int MIN_WAIT_TIME = 100;
        private const int MAX_WAIT_TIME = 250;
        private const int MAX_GAME_TIME = 200;
        private const int GAMEOVER_TIME = 300;
        private const int RESULTS_TIME = 500;
        private const double TICKS_PER_SECOND = 100.0;

        private State _state;
        private IGui Gui { get; set; }
        private IRandom Rng { get; set; }
        private int Ticks { get; set; }
        private int Games{ get; set; }
        private int TotalReactionTime { get; set; }

        public void Connect(IGui gui, IRandom rng)
        {
            Gui = gui;
            Rng = rng;
            Init();
        }
        public void Init()
        {
            _state = new OnState(this);
        }

        public void GoStopPressed()
        {
            _state.GoStopPressed();
        }
        public void CoinInserted()
        {
            _state.CoinInserted();
        }
        public void Tick()
        {
            _state.Tick();
        }


        private void SetState(State state)
        {
            _state = state;
        }

        private abstract class State
```

```
        {
            protected EnhancedReactionController controller;

            public State(EnhancedReactionController cont)
            {
                controller=cont;
            }

            public abstract void CoinInserted();
            public abstract void GoStopPressed();
            public abstract void Tick();

        }

        private class OnState : State
        {
            public OnState(EnhancedReactionController cont):base(cont)
            {
                controller.Games = 0;
                controller.TotalReactionTime = 0;
                controller.Gui.SetDisplay("Insert Coin");
            }

            public override void CoinInserted()
            {
                controller.SetState(new ReadyState(controller));
            }
            public override void GoStopPressed() { }
            public override void Tick() { }

        }

        private class WaitState : State
        {
            private int _waitTime;
            public WaitState(EnhancedReactionController cont):base(cont)
            {
                controller.Gui.SetDisplay("Wait...");
                controller.Ticks = 0;
                _waitTime = controller.Rng.GetRandom(MIN_WAIT_TIME, MAX_WAIT_TIME);
            }

            public override void CoinInserted() { }

            public override void GoStopPressed()
            {
                controller.SetState(new OnState(controller));
            }

            public override void Tick()
            {
                controller.Ticks++;
                if(controller.Ticks==_waitTime)
```

```
                {
                    controller.SetState(new RunningState(controller));
                }
            }
        }

        private class RunningState : State
        {
            public RunningState(EnhancedReactionController cont) : base(cont)
            {
                controller.Gui.SetDisplay("0.00");
                controller.Ticks = 0;
            }
            public override void CoinInserted() { }
            public override void GoStopPressed()
            {
                controller.SetState(new GameOverState(controller));
           }

            public override void Tick()
            {
                controller.Ticks++;
                controller.Gui.SetDisplay((controller.Ticks /
                ↪  TICKS_PER_SECOND).ToString("0.00"));
                if (controller.Ticks == MAX_GAME_TIME)
                {
                    controller.SetState(new GameOverState(controller));
                }
            }

        }



        private class ReadyState : State
        {
            public ReadyState(EnhancedReactionController cont):base(cont)
            {
                controller.Gui.SetDisplay("Press Go");
                controller.Ticks = 0;
            }

            public override void CoinInserted() { }
            public override void GoStopPressed()
            {
                controller.SetState(new WaitState(controller));
            }

            public override void Tick()
            {
                controller.Ticks++;
                if (controller.Ticks == MAX_READY_TIME)
                    controller.SetState(new OnState(controller));
```

```
            }



        }

        private class GameOverState : State
        {
            public GameOverState(EnhancedReactionController cont) : base(cont)
            {
                controller.Ticks = 0;
            }
            public override void CoinInserted() { }
            public override void GoStopPressed() => CheckGames();

            public override void Tick()
            {
                controller.Ticks++;
                if (controller.Ticks == GAMEOVER_TIME)
                    CheckGames();
            }
            private void CheckGames()
            {
                if(controller.Games==3)
                {
                    controller.SetState(new ResultsState(controller));
                    return;
                }
                controller.SetState(new WaitState(controller));
            }
        }

        class ResultsState:State
        {
            public ResultsState(EnhancedReactionController cont):base(cont)
            {
                controller.Gui.SetDisplay("Average: " +
                ↪  (((double)controller.TotalReactionTime / controller.Games) /
                ↪  TICKS_PER_SECOND).ToString("0.00"));
                controller.Ticks = 0;
            }
            public override void CoinInserted() { }
            public override void GoStopPressed() => controller.SetState(new
            ↪  OnState(controller));

            public override void Tick()
            {
                controller.Ticks++;
                if (controller.Ticks == RESULTS_TIME)
                    controller.SetState(new OnState(controller));

            }
        }
```

```

    }
}
```

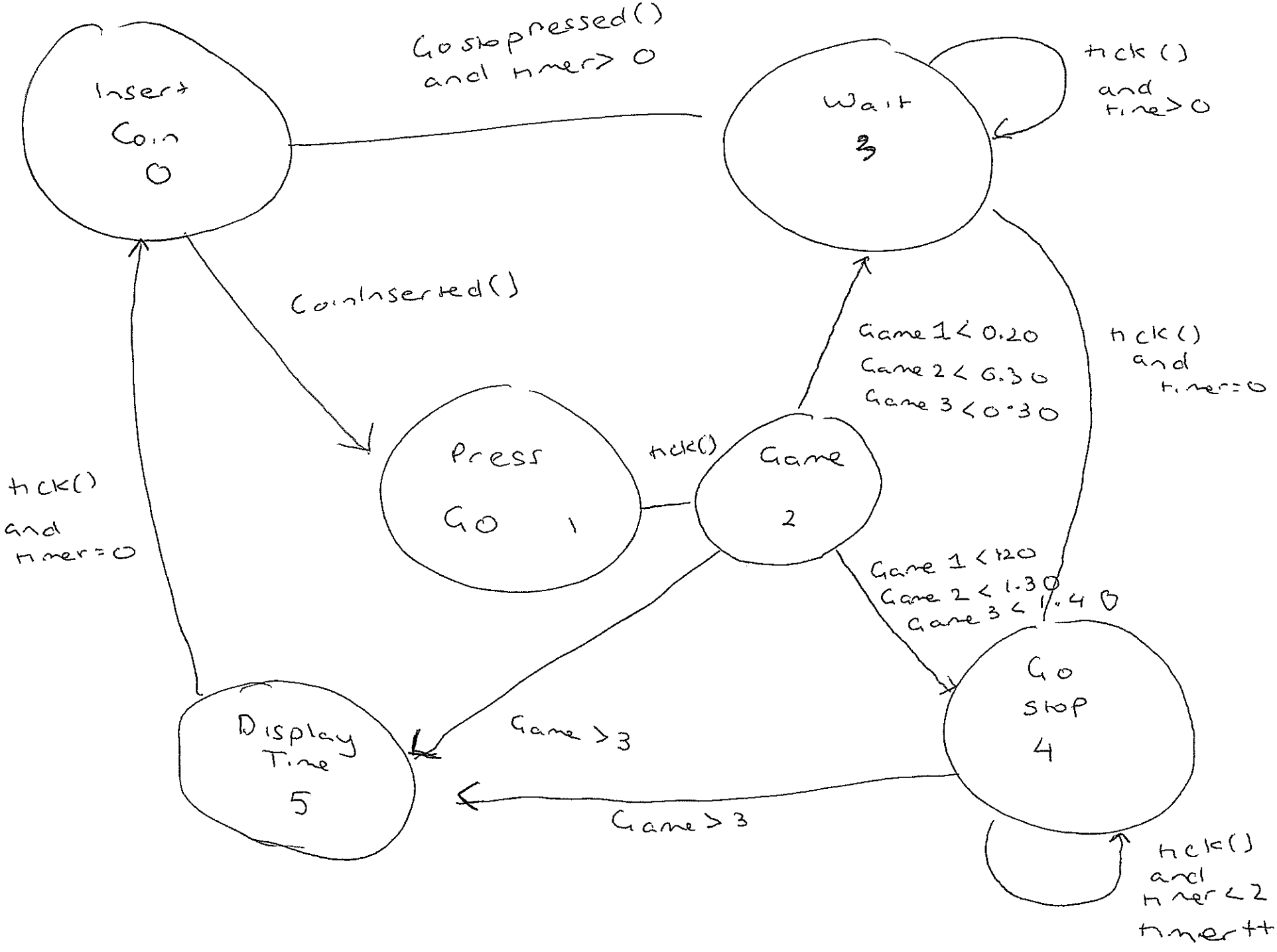
```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
//using EnhancedReactionController;

namespace task5_3
{
    [TestClass]
    public class UnitTest1
    {
        //Construct a ReactionController
        private static IController controller;
        private static IGui gui;
        private static IRandom rng;
        private static string displayText;

        private static int RandomNumber { get; set; }


        [TestMethod]

        public void Test_Connect_Controller()
        {
            //Connect them to each other
            controller = new EnhancedReactionController();
            gui = new DummyGui();
            gui.Connect(controller);
            controller.Connect(gui, new RndGenerator());
            controller.Init();
            Assert.AreEqual("Insert Coin", displayText);
        }

        //Test Waiting Time after coin inserted
        [TestMethod]
        public void Test_Waiting_CoinInserted()
        {
            controller = new EnhancedReactionController();
            gui = new DummyGui();
            rng = new RndGenerator();
            WaitingState(controller, gui, rng);

            Assert.AreEqual("Wait", displayText);
            controller.CoinInserted();
            Assert.AreEqual("Wait", displayText);

        }

        //user presses go, wait ensues
       [TestMethod]
        public void Test_Start_Wait()
        {
            controller = new EnhancedReactionController();
            gui = new DummyGui();
            rng = new RndGenerator();
```

```
            StartingState(controller, gui, rng);


            Assert.AreEqual("Press Go", displayText);
            controller.GoStopPressed();
            Assert.AreEqual("Wait", displayText);

        }

        //user presses go, insert coin response
        [TestMethod]
        public void Test_GameOver()
        {
            controller = new EnhancedReactionController();
            gui = new DummyGui();
            rng = new RndGenerator();
            StartingState(controller, gui, rng);

            //random time up until 100 miliseconds
            for (int t = 0; t < 100; t++) controller.Tick();
            Assert.AreEqual("Press Go", displayText);
            controller.Tick();
            Assert.AreEqual("Insert Coin", displayText);

        }



        //Tests running time, less than 2.04 then GoStopPress()
        [TestMethod]
        public void Test_GoStopPressed()
        {
            controller = new EnhancedReactionController();
            gui = new DummyGui();
            rng = new RndGenerator();
            RunningState(controller, gui, rng);

            for (int t = 0; t < 204; t++) controller.Tick();
            Assert.AreEqual("2.04", displayText);
            controller.GoStopPressed();
            Assert.AreEqual("2.04", displayText);

        }


        //Tests running time, less than 0.99 then GoStopPress()
        [TestMethod]
        public void Test_GameOverState()
        {
            controller = new EnhancedReactionController();
            gui = new DummyGui();
            rng = new RndGenerator();
            RunningState(controller, gui, rng);
```

```
            for (int t = 0; t < 99; t++) controller.Tick();
            Assert.AreEqual("0.99", displayText);
            controller.CoinInserted();
            Assert.AreEqual("0.99", displayText);

        }

        //test three games with waiting time
        [TestMethod]
        public void Test_Playing_Three_Games()
        {
            controller = new EnhancedReactionController();
            gui = new DummyGui();
            rng = new RndGenerator();
            RunningState(controller, gui, rng);

            //game 1, user presses before 20 miliseconds, Wait
            for (int t = 0; t < 20; t++) controller.Tick();
            controller.GoStopPressed();
            Assert.AreEqual("0.20", displayText);
            for (int t = 0; t < 199; t++) controller.Tick();
            Assert.AreEqual("0.20", displayText);
            controller.Tick();
            Assert.AreEqual("Wait", displayText);

            //game 2, user presses before 30 miliseconds, Wait
            for (int t = 0; t < RandomNumber+30; t++) controller.Tick();
            controller.GoStopPressed();
            Assert.AreEqual("0.30", displayText);
            for (int t = 0; t < 299; t++) controller.Tick();
            Assert.AreEqual("0.30", displayText);
            controller.Tick();
            Assert.AreEqual("Wait", displayText);

            //game 3, user presses before 40 miliseconds, Wait
            for (int t = 0; t < RandomNumber + 40; t++) controller.Tick();
            controller.GoStopPressed();
            Assert.AreEqual("0.40", displayText);
            for (int t = 0; t < 299; t++) controller.Tick();
            controller.Tick();
            Assert.AreEqual("Average: 30", displayText);
        }

        //User presses GoStopPressed if time meets conditions or GoPressed()
        [TestMethod]
        public void Test_Play_Three_Games()
        {
            controller = new EnhancedReactionController();
            gui = new DummyGui();
            rng = new RndGenerator();
            RunningState(controller, gui, rng);

```

```
            //game 1, user presses before 1.20 miliseconds, Press GoStop
            for (int t = 0; t < 120; t++) controller.Tick();
            controller.GoStopPressed();
            Assert.AreEqual("1.20", displayText);
            controller.GoStopPressed();
            Assert.AreEqual("Wait", displayText);

            //game 2, user presses before 1.30 miliseconds, Press GoStop
            for (int t = 0; t < RandomNumber + 130; t++) controller.Tick();
            controller.GoStopPressed();
            Assert.AreEqual("1.30", displayText);
            controller.GoStopPressed();
            Assert.AreEqual("Wait", displayText);

            //game 1, user presses before 1.40 miliseconds, Press GoStop
            for (int t = 0; t < RandomNumber + 140; t++) controller.Tick();
            controller.GoStopPressed();
            Assert.AreEqual("1.40", displayText);
            controller.GoStopPressed();
            Assert.AreEqual("Average: 1.30", displayText);
        }


        private void StartingState(IController controller, IGui gui, IRandom rng)
        {
            gui.Connect(controller);
            controller.Connect(gui, rng);
            gui.Init();
            controller.Init();
        }


        private void WaitingState(IController controller, IGui gui, IRandom rng)
        {
            OnState(controller, gui, rng);
            for (int t = 0; t < RandomNumber; t++)
                controller.Tick();
        }



        public void RunningState(IController controller, IGui gui, IRandom rng)
        {
            WaitingState(controller, gui, rng);
            for (int t = 0; t < RandomNumber; t++)
                controller.Tick();
        }

        public class DummyGui : IGui
        {
            private IController _controller;
            public void Connect(IController controller) => _controller = controller;
            public void Init() => displayText = "?reset?";
```

```

            public void SetDisplay(string msg)
            {
                displayText = msg;
            }
        }

        public class RndGenerator : IRandom
        {
            Random rnd = new Random(42);
            public int GetRandom(int from, int to)
            {
                RandomNumber = rnd.Next(from) + to;
                return RandomNumber;
            }
        }
    }
}
```


Insert Coin 0
Go stop pressed() and timer > 0
Wait 3
tick() and time > 0
CoinInserted()
Press Go 1
tick()
Game 2
Game 1 < 0.20
Game 2 < 0.30
Game 3 < 0.30
tick() and timer = 0
tick() and timer = 0
Game 1 < 120
Game 2 < 1.30
Game 3 < 1.40
Go stop 4
Display Time 5
Game > 3
Game > 3
tick() and timer < 2
timer++

21 Abstract Transactions

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Evaluate Code	♦◊◊◊◊

This task allowed me to see the full development of my design and to apply everything that I have learnt this unit into a functional program. As such, it was a great opportunity to see OOP in action.

Outcome	Weight
Principles	♦♦♦◊◊

This task allowed me to see the full development of my design and to apply everything that I have learnt this unit into a functional program. As such, it was a great opportunity to see OOP in action.

Outcome	Weight
Build Programs	♦♦♦♦◊

This task allowed me to see the full development of my design and to apply everything that I have learnt this unit into a functional program. As such, it was a great opportunity to see OOP in action.

Outcome	Weight
Design	♦♦♦◊◊

This task allowed me to see the full development of my design and to apply everything that I have learnt this unit into a functional program. As such, it was a great opportunity to see OOP in action.

Outcome	Weight
Justify	♦◊◊◊◊

This task allowed me to see the full development of my design and to apply everything that I have learnt this unit into a functional program. As such, it was a great opportunity to see OOP in action.

Date	Author	Comment
2020/05/29 01:45	Dale Orders	Ready to Mark
2020/05/29 01:51	Dale Orders	https://www.youtube.com/watch?v=snx1r8Ql-RzA&feature=youtu.be
2020/05/30 14:18	Sanjay Segu	Discuss
2020/05/30 14:18	Sanjay Segu	discussion comment
2020/06/01 18:23	Dale Orders	Sorry it didn't record.
2020/06/01 18:24	Dale Orders	can I try again
2020/06/01 18:24	Dale Orders	It is not possible to create an object from an abstract class.
2020/06/01 18:25	Dale Orders	The abstract class protects internal details which can be accessed through instantiating an object
2020/06/01 18:25	Dale Orders	can not be accessed
2020/06/01 18:26	Dale Orders	Sorry I said this in the recording, but when I play it back I can't hear anything
2020/06/01 19:44	Sanjay Segu	Complete
2020/06/01 19:44	Sanjay Segu	No problems
2020/06/01 19:44	Sanjay Segu	May be there's some issue with that feature

Deakin University

Object Oriented Development

OnTrack Submission

Abstract Transactions

Submitted By:
Dale Orders
dorders
2020/05/29 01:45

Tutor:
Sanjay Segu

Outcome	**Weight**
Evaluate Code	♦◊◊◊◊
Principles	♦♦♦◊◊
Build Programs	♦♦♦♦◊
Design	♦♦♦◊◊
Justify	♦◊◊◊◊

This task allowed me to see the full development of my design and to apply everything that I have learnt this unit into a functional program. As such, it was a great opportunity to see OOP in action.

May 29, 2020

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task7_1
{
    class DepositTransaction:Transaction
    {


        private Account _account;
        private bool _executed;
        private bool _reversed;
        public Account Account { get => _account; }

        public DepositTransaction(Account account, decimal amount):base(amount)
        {
            _account = account;
        }

        public override void Print()
        {
            Console.WriteLine($"Successfully deposited ${this._amount} into the account: {this._account.getName()}");
        }

        //only executes if the transaction has yet to be performed
        public override void Execute()
        {
            base.Execute();
            if (_executed== false)
            {
                _account.Deposit(_amount);
                _executed = true;
            }
            else
            {
                throw new InvalidOperationException("You have already performed this transaction");
            }
        }

        //rollbaack deposit by withdrawing _amount
        public override void Rollback()
        {
            base.Rollback();
            if (_executed== true && _reversed == false)
            {
                _account.Withdraw(_amount);
                _executed = false;

            }
            else
```

```
            {
                Console.WriteLine("There is no deposit to reverse");
            }


        }

        //returns user's name
        public override string GetAccName()
        {
            return _account.getName();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task7_1
{
    public class WithdrawTransaction : Transaction
    {

        private Account _account;
        private bool _executed;
        private bool _reversed;



        public WithdrawTransaction(Account account, decimal amount) : base(amount)
        {
            this._account = account;
            this._amount = amount;
            _reversed = false;
            _executed = false;
        }

        public override string GetAccName()
        {
            return _account.getName();
        }


        public override void Print()
        {
            Console.WriteLine($"Successfully withdrew ${this._amount} from the account: {this._account.getName()}");
        }

        public override void Execute()
        {
            base.Execute();
            if (_executed == false)
            {
                if (_account.getBalance() > 0 && _amount <= _account.getBalance())
                {
                    _account.Withdraw(_amount);
                    _executed = true;
                }
                else
                {
                    Console.WriteLine("Insufficent funds");
                }
            }

           else { throw new InvalidOperationException("Withdrawal has already been performed"); }
```

```
        }

        public override void Rollback()
        {
            base.Rollback();
            if (_reversed == false && _executed == true)
            {
                _account.Deposit(_amount);
                _executed = false;
                _reversed = true;
            }
            else
            {
                Console.WriteLine("Transaction has already been reversed");
            }


        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;

namespace task7_1
{
    public class TransferTransaction: Transaction
    {

        private Account _fromAccount;
        private Account _toAccount;
        private DepositTransaction _deposit;
        private WithdrawTransaction _withdraw;
        private bool _executed;
        private bool _reversed;




        public TransferTransaction(Account fromAccount, Account toAccount, decimal amount):base(amount)
        {
            _deposit = new DepositTransaction(toAccount, amount);
            _withdraw = new WithdrawTransaction(fromAccount, amount);
            _executed = false;
            this._fromAccount = fromAccount;
            this._toAccount = toAccount;
            this._amount = amount;
        }

        //prints statement
        public override void Print()
        {
            if (_executed == true)
            {
                Console.WriteLine("Transaction has been performed successfully");
                Console.WriteLine($"Transferred ${_amount} from {_fromAccount.getName()} to {_toAccount.getName()}");
            }

            else
            {
                Console.WriteLine("Transaction has not been successful");
            }

        }
        //executes legitimate transactions
        public override void Execute()
        {
            if (_executed == false)
            {
                if (_amount<=0 || _fromAccount.getBalance() < _amount)
```

```
                {
                    Console.WriteLine("Insufficient funds");

                }
                else
                {
                    _withdraw.Execute();
                    _deposit.Execute();
                    _executed = true;

                }
            }
            else { throw new InvalidOperationException("Transacation has been already performed"); }

            _executed = true;
        }

        //check to see if rollback meets onditions
        public override void Rollback()
        {
            if (_executed == false)
            {
                throw new InvalidOperationException("Reverse transaction has already been performed");
            }
            if (_toAccount.getBalance() < _amount)
            {
                throw new InvalidOperationException("Insufficient funds");
            }
            else if (_executed == true)
            {
                _deposit.Rollback();
                _withdraw.Rollback();
                _reversed = true;
                _executed = false;

            }

        }



    }

}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;

namespace task7_1
{
    public abstract class Transaction
    {
        protected decimal _amount;
        protected Boolean _success;
        private Boolean _executed;
        private Boolean _reversed;
        private DateTime _dateStamp;

        public Boolean Executed { get => _executed; }
        public Boolean Success { get => _success; }
        public Boolean Reversed { get => _reversed; set => _reversed = value; }
        public DateTime DateStamp { get => _dateStamp; }
        public decimal Amount { get => _amount; }

        public Transaction(decimal amount)
        {
            if (amount > 0)
            {
                _amount = amount;
            }
            else
            {
                amount = 0;
                throw new ArgumentOutOfRangeException("Amount must be greater than 0");
            }
        }

        public virtual String GetAccName()
        {
            return "Various";
        }

        public virtual void Print()
        {
            Console.WriteLine("Transaction amount: {0} Executed: {1}, Reversed: {3}", _amount.ToString("C"), _executed, _reversed);
        }

        public virtual void Execute()
        {
            if (_executed)
            {
                throw new InvalidOperationException("transaction has previously occurred");
            }
```

```
            _dateStamp = DateTime.Now;
            _executed = true;
        }

        public virtual void Rollback()
        {
            if (_reversed==true)
            {
                throw new InvalidOperationException("Transaction has already been reversed");
            }
            else if (!_executed)
            {
                throw new InvalidOperationException("Transaction can not be rollback as it was never successfully executed");
            }
            _dateStamp = DateTime.Now;
            _executed = false;
        }


    }


}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

namespace task7_1
{

    class BankSystem
    {
        //enum options user can choose from
        enum MenuOption
        {
            AddnewAccount,
            Withdraw,
            Deposit,
            Print,
            TransferTransaction,
            Rollback,
            Quit
        }

        List<Transaction> transactions = new List<Transaction>();

        static void AddAccount(Bank bank)
        {
            Console.WriteLine("Enter the name of the account holder");
            string name = Console.ReadLine();
            Console.WriteLine("Enter their starting account balance");
            decimal balance = Convert.ToDecimal(Console.ReadLine());
            bank.AddAccount(new Account(balance, name));
        }

        //find account
        private static Account FindAccount(Bank bank)
        {
            Account account = null;
            Console.WriteLine("Enter the account name");
            string name = Console.ReadLine();
            account = bank.GetAccount(name);
            if (account == null)
            {
                Console.WriteLine("Account does not exist");
            }
            return account;
        }

        //Takes an amount to deposit
        static void DoDeposit(Bank bank)
        {
            Account account = FindAccount(bank);
            Console.WriteLine("Enter deposit amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
```

```
            DepositTransaction deposit = new DepositTransaction(account, amount);

            if (account != null)
            {
                string execute;
                string rollback;

                Console.WriteLine($"Deposit the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Rollback the transaction. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
                    bank.Execute(deposit);
                }
                if (rollback == "Y".ToLower())
                {
                    deposit.Rollback();
                }

            }
        }

        //takes an amount to withdraw
        static void DoWithdrawal(Bank bank)
        {
            Account account = FindAccount(bank);
            Console.WriteLine("Enter withdrawal amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            WithdrawTransaction withdrawal = new WithdrawTransaction(account,
            ↪ amount);

            if (account != null)
            {
                string execute;
                string rollback;

                Console.WriteLine($"Withdraw the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Rollback the transaction. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
                    bank.Execute(withdrawal);
                }
                if (rollback == "Y".ToLower())
                {
                    withdrawal.Rollback();
                }


            }

```

```
        }

        //takes an amount to transfer
        static void DoTransfer(Bank bank1)
        {
            Account fromaccount = FindAccount(bank1);
            Account toaccount = FindAccount(bank1);
            Console.WriteLine("Transfer the following amount: ");
            decimal amount = Convert.ToDecimal(Console.ReadLine());
            TransferTransaction transfer = new TransferTransaction(fromaccount,
            ↪ toaccount, amount);

            if (bank1 != null)
            {
                string execute;
                string rollback;

                Console.WriteLine($"Transfer the given amount: ${amount}. Y/N? ");
                execute = Console.ReadLine();
                Console.WriteLine("Would you like to rollback the trandfer. Y/N? ");
                rollback = Console.ReadLine();
                if (execute == "Y".ToLower())
                {
                    bank1.Execute(transfer);
                }
                if (rollback == "Y".ToLower())
                {
                    transfer.Rollback();
                }
                bank1.Print();
            }
        }

        //rolls backs a specified transaction
        static void DoRollback(Bank bank)
        {
            bank.Print();
            Console.WriteLine("Enter transaction to rollback or press 0 to exist:
            ↪ ");
            int rollback = Convert.ToInt32(Console.ReadLine());
            if (rollback != 0)
            {
                bank.Rollback(bank.getTheList()[rollback - 1]);
            }
        }


            static MenuOption ReadUserOption()
            {
                int? option = null;
                do
                {
```

```
                    Console.WriteLine("-------------------------------------------------------");
                    Console.WriteLine("Please select from the following options");
                    Console.WriteLine("MENU \n1. Add new Account \n2. Withdraw \n3. Deposit \n4. Print \n5. Transfer Transaction \n6. Rollback \n7. Quit");
                    Console.WriteLine("-------------------------------------------------------");

                    try
                    {
                        option = Convert.ToInt32(Console.ReadLine());
                        if (option > 7 || option < 1)
                        {
                            option = null;
                            Console.WriteLine("Please enter a number from 1 to 7 from the menu");
                        }
                    }
                    catch (FormatException)
                    {
                        Console.WriteLine("Invalid input. Enter an integer from 1-7");
                    }

                } while (option == null);

                return (MenuOption)option;
            }


            static void DoPrint(Bank bank)
            {
                Account account = FindAccount(bank);
                if (account != null)
                {
                    account.Print();
                }

            }


            static void Main(string[] args)
            {


                Bank newbank = new Bank();

                Account person_one = new Account(100, "Jon");
                Account person_two = new Account(150, "Amy");
                MenuOption option;

                do
```

```
                {
                    option = ReadUserOption() - 1;

                    switch (option)
                    {
                        case MenuOption.AddnewAccount:
                            AddAccount(newbank);
                            break;
                        case MenuOption.Withdraw:
                            Console.WriteLine("You have selected withdraw");
                            DoWithdrawal(newbank);
                            break;
                        case MenuOption.Deposit:
                            Console.WriteLine("You have selected deposit");
                            DoDeposit(newbank);
                            break;
                        case MenuOption.Print:
                            Console.WriteLine("You have selected print");
                            DoPrint(newbank);
                            break;
                        case MenuOption.TransferTransaction:
                            DoTransfer(newbank);
                            break;
                        case MenuOption.Rollback:
                            DoRollback(newbank);
                            break;
                        case MenuOption.Quit:
                            Console.WriteLine("Goodbye");
                            break;

                    }
                } while (option != MenuOption.Quit);


            }


        }

    }
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task7_1
{
    public class Bank
    {
        private static List<Account> _accounts = new List<Account>();
        private List<Transaction> _transactions = new List<Transaction>();


        public void AddAccount(Account account)
        {
            _accounts.Add(account);
        }


        public List<Transaction> getTheList()
        {
            return _transactions;
        }

        public Account GetAccount(String name)
        {
            foreach(Account account in _accounts)
            {
                if(account.getName().ToLower()==name.ToLower())
                {
                    return account;
                }
            }
            { return null; }
        }
        //Add and Execute transaction
        public void Execute(Transaction transaction)
        {
            _transactions.Add(transaction);

            transaction.Execute();

        }

        //Rollback
        public void Rollback(Transaction transaction)
        {
            try
            {
                transaction.Rollback();
            }
            catch(InvalidOperationException exception)
            {
                Console.WriteLine("Unable to perform the Rollback");
```

```
                Console.WriteLine("The error was: " + exception.Message);
            }
        }

        //print transaction history
        public void Print()
        {
            int index = 1;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.WriteLine(new string(' ', 10)+ "Banking System");
            Console.WriteLine(new string(' ', 10) + new string('-', 104));
            Console.WriteLine(new string(' ', 10) + new string('-', 104));
            foreach(var transaction in _transactions)
            {
                string type = "test";
                if (transaction.GetType().Name.Equals("DepositTransaction"))
                {
                    type = "Deposit";
                }
                if (transaction.GetType().Name.Equals("WithdrawTransaction"))
                {
                    type = "Withdraw";
                }
                if (transaction.GetType().Name.Equals("TransferTransaction"))
                {
                    type = "Transfer";
                }

                Console.ForegroundColor = ConsoleColor.Black;
                Console.WriteLine("{0,12}{1,5}{2,10}{3,25}{4,20}{5,17}", index,
                ↪ transaction.DateStamp,transaction.GetAccName(),type,
                ↪ transaction.Executed==true&&transaction.Reversed==false?"Execut
                ↪ able":"Executed",
                ↪ transaction.Amount.ToString("C"));



            }
        }


    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace task7_1
{
    public class Account
    {
        //instance variables declared
        private decimal _balance;
        private string _name;

        //constructor
        public Account(decimal balance, string name)
        {

            this._name = name;
            if (balance <= 0)
                return;
            this._balance = balance;

        }

        //prints name and balance
        public void Print()
        {
            Console.WriteLine("The name of the account holder is " + getName() +
            ↪  "\nCurrent Account Balance is $" + getBalance());
        }

        //returns name
        public String getName()
        {
            return this._name;
        }

        //returns balance
        public decimal getBalance()
        {
            return this._balance;
        }

        //increases balance by adding deposit
        //boolean ensure that no 0 or a negative value can not be input
        public Boolean Deposit(decimal amount)
        {

            if (amount <= 0)
                return false;

            this._balance += amount;
            return true;

```

```
        }


        //decreases balanace by withdrawing the giving input amount
        //boolean values protect against overdrawing from account
        public Boolean Withdraw(decimal amount)
        {
            if (amount > this._balance || amount < 0)
                return false;


            this._balance -= amount;
            return true;

        }

    }
}
```

22 Documenting the Banking System

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Design	♦♦♦◊◊

This gave me experience of creating a UML. Being familiar with the principles will help me in designing programs in the future by identifying cardinality and associations between classes.

Outcome	**Weight**
Justify	♦♦◊◊◊

This gave me experience of creating a UML. Being familiar with the principles will help me in designing programs in the future by identifying cardinality and associations between classes.

Date	**Author**	**Comment**
2020/05/29 02:35	Dale Orders	Ready to Mark
2020/05/30 14:18	Sanjay Segu	Fix and Resubmit
2020/05/30 14:19	Sanjay Segu	90% of your work looks correct
2020/05/30 14:19	Sanjay Segu	Apart from missing multiplicities
2020/05/31 14:47	Dale Orders	Sorry which part needs to be fixed? Can you explain that?
2020/05/31 21:35	Sanjay Segu	https://www.uml-diagrams.org/multiplicity.html
2020/05/31 21:35	Sanjay Segu	Please refer the above link for more information on Multiplicities.
2020/06/05 14:48	Dale Orders	Ready to Mark
2020/06/05 14:48	Sanjay Segu	Time Exceeded
2020/06/05 14:50	Dale Orders	pdf document
2020/06/05 14:51	Dale Orders	Hi Sanyjay, I added multiplicities and resubmitted my UML.

Deakin University

Object Oriented Development

OnTrack Submission

Documenting the Banking System

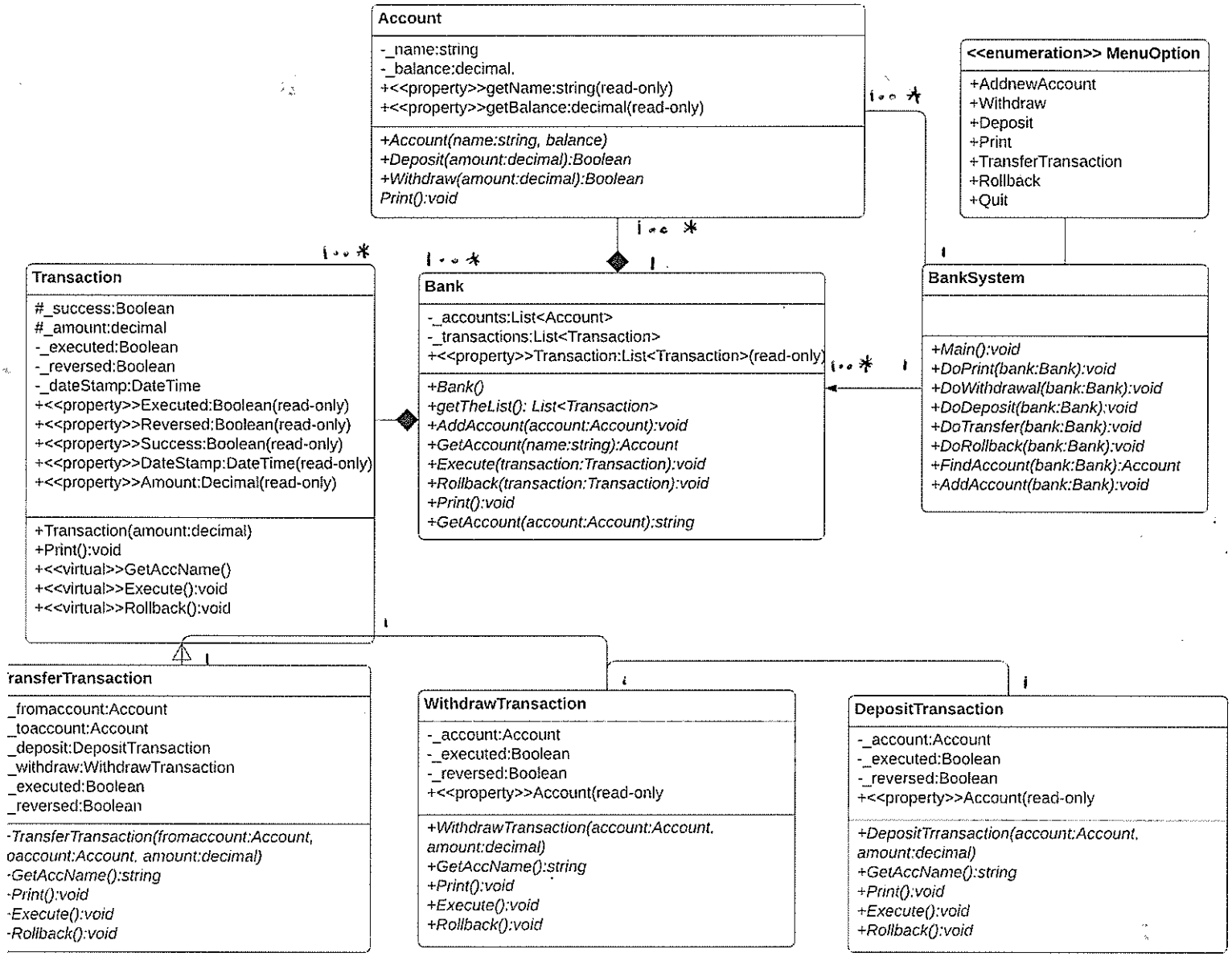
Submitted By:
Dale Orders
dorders
2020/06/05 14:48

Tutor:
Sanjay Segu

Outcome	**Weight**
Design	♦♦♦◊◊
Justify	♦♦◊◊◊

This gave me experience of creating a UML. Being familiar with the principles will help me in designing programs in the future by identifying cardinality and associations between classes.

June 5, 2020



Assumptions

- A transaction can occur across banks.
- A BankSystem (BS) may be used by multiple banks.
- Each bank has only one BS.
- A BS holds one or more accounts

23 Helping Your Peers

Note that we will not accept your report after the submission deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit.

Outcome	Weight
Justify	♦♦♦◊◊

This was a good opportunity to reflect upon my engagement with the learning process and how I have worked with others to succeed in this unit.

Date	Author	Comment
2020/06/01 18:21	Dale Orders	Ready to Mark
2020/06/01 19:42	Sanjay Segu	Complete
2020/06/01 19:43	Sanjay Segu	Thanks for all the help in this trimester Dale.

Deakin University

Object Oriented Development

OnTrack Submission

Helping Your Peers

Submitted By:
Dale Orders
dorders
2020/06/01 18:21

Tutor:
Sanjay Segu

Outcome	**Weight**
Justify	♦♦♦◊◊

This was a good opportunity to reflect upon my engagement with the learning process and how I have worked with others to succeed in this unit.

June 1, 2020

Practical Task 1.3

I have endeavoured to help other students throughout this unit by engaging with discussions on the Deakin forum, attending helphub, participating in the teams chat and being a member of a cloud Deakin study group. In this way, I have continued to work with others to try and foster a collaborative approach. For me, I have enjoyed being actively involved in helping others, something which I hope to continue doing next trimester and beyond.

Subscribe Unsubscribe

Study group

DALE ORDERS 09 March, 2020 4:43 PM

Hi all,

Is there anyone looking to form a study group. For my programming last year, I joined one and found it really helped.

Dale

Subscribe Unsubscribe Previ

Study group

DALE ORDERS 09 March, 2020 9:13 PM

Hi Jesse,

That would be great, I'm cloud as well. Last year we had a online study group. If we get more interest, I can get a poll of a time that will work.

Dale

Subscribe Unsubscribe

Previous Nex

Study group

DALE ORDERS 10 March, 2020 2:05 PM

Online sounds good.

Also if anyone is interested, I am part of a coding group that meets every Saturday in the city. We do all things coding. Happy to give the details if anyone is interested in coming along.

Dale

Subscribe Unsubscribe

Rating

Average Rating

(2 ratings)

Useful resource for debugging

DALE ORDERS 10 April, 2020 4:04 PM

Hi all,

For 4.2 you need to debug a program. On youtube, I found some great videos on how to do this step by step. Thought I would share.

https://www.youtube.com/watch?v=d6IYH8Ro9aI&t=1217s

https://www.youtube.com/watch?v=G1S6NZfFvOg&t=850s

Hopefully they help,

Dale

Reply Edit Post More Actions

Subscribe Unsubscribe

Previous N

Study group - Slack Group

DALE ORDERS 15 March, 2020 8:02 PM

Slack and discord do much the same thing, so I guess it would depend on which one people prefer. I think having a weekly time might be a good way to do it. Alternatively, I am involved in a coding group that meets at Deakin library every Sunday, so people who prefer to meet face to face are welcome to attend that as well.

Subscribe Unsubscribe Previous

Rating

Average R

(0 rating

task 2.1 part 3

DALE ORDERS 22 March, 2020 2:46 PM

Agree. As it doesn't say, we are able to do either. I think the easiest would be to simply create two people and show that their accounts can be manipulated by depositing or withdrawing an amount (your choosing). Once that works, you can then make changes if you so wish, such as adding a message or giving an option. However we are not expected to do so.

Subscribe Unsubscribe Previous Ne

Rating

Average Ratir

(0 ratings)

2.1 question 4 - CarProgram

DALE ORDERS 28 March, 2020 2:12 AM

Hi Jesse,

The question I believe is asking you to set the total miles by calling the function with the parameter miles then calculate gallons by diving total miles by efficiency to give you gallons. If total miles is less than or equal to gallons, you convert it to litres and return the cost. That's my understanding.

Let me know if you need more clarity on this

Dale

Subscribe Unsubscribe Previous Ne

Rating

Average Ratir

(0 ratings)

illegal name?

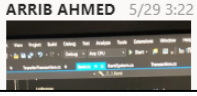
DALE ORDERS 17 March, 2020 3:38 PM

Yes, you can not use a space in the title.

See previous replies

DALE ORDERS 5/29 3:21 AM
actually, looking at this method why you you ask the user which transaction they want to rollback and then accept the answer as 'y'?

ARRIB AHMED 5/29 3:22 AM



DALE ORDERS 5/29 3:23 AM
ok, so in the do rollback, change the question from 'which transaction do you want to reverse' to 'do you want to reverse a transaction?'

1

It will only roll back if the user enters a 'y'. So also write 'press y to proceed.'

ARRIB AHMED 5/29 3:27 AM
yeah man just fixed that, lets see how it runs

appreciate the help man

thats all sorted

DALE ORDERS 5/29 3:29 AM
all good

DALE ORDERS 5/29 3:38 AM Edited
Can I see your main method. Your menu should be in a do while loop so if keeps coming back as long as the user does not press 7

ARRIB AHMED 5/29 3:49 AM

```
public static MenuOption ReadUserOption()
    {
        Console.WriteLine("============Menu================");
        Console.WriteLine("================================");
        Console.WriteLine("---------Choose an option-------");
```

See more

it is in a dowhile

DALE ORDERS 5/29 3:53 AM
the do while needs to be around the whole thing. Otherwise it will only keep bringing up "Select an option" again and again

Only the thing inside the do while brackets repeats.

ARRIB AHMED 5/29 3:54 AM
yeah so why does it say "enter an account name" after getting done with a transaction

DALE ORDERS 5/29 3:55 AM
Is this in your main method

It needs to be in your main method

Collapse all

DALE ORDERS 5/27 4:59 PM
insufficient funds will occur when the amount you enter is greater than the amount stored in the Account class/ ie. _amount>_account.GetBalance().

Messages 71 | Files 0 | Channels 0 | People 0

Most relevant

general – Apr 10th
daleorders7 5:40 PM
Jesse did you need help with 3.2?

general – Apr 7th
daleorders7 8:55 PM
do you need help samantha?

general – Apr 10th
daleorders7 5:41 PM
or did you work it out?

Filter by
Shared by
daleorders7
More...
Shared in
general
More...
Date
Start
End

KURT RASMUSSEN 4/22 7:22 PM Edited
here is a small program i made from the above video that only uses things we have done in the unit so far if anyone wants to check it out. I put explanations of whats going on in the comments hopefully its clear

DelegateProgram.cs
SIT232ObjectOrientedDevelopment20...

4/22 7:35 PM Edited 1
That looks great. I would also recommend this video https://www.youtube.com/watch?v=R8Blt5c-Vi4&t=838s. He provides source code in the decription

---

Online Peer-to-Peer Study Group > **Post by DALE ORDERS May 15, 2020**

5/15 1:27 PM 1
Have you tried commenting out the _amount in the Deposit transaction clas

SE
SEAN EMERY 5/15 1:27 PM 1
Hey Jesse.

I think you need to remove the variable in your child class (the one shown in the picture). I believe having it in the child class will hide the one in the parent class which is where you are passing the variable in the constructor. So when you use _amount in the child class it uses the one here instead
See more

5/15 1:27 PM 1
Green means that you are not using it, I believe