

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:

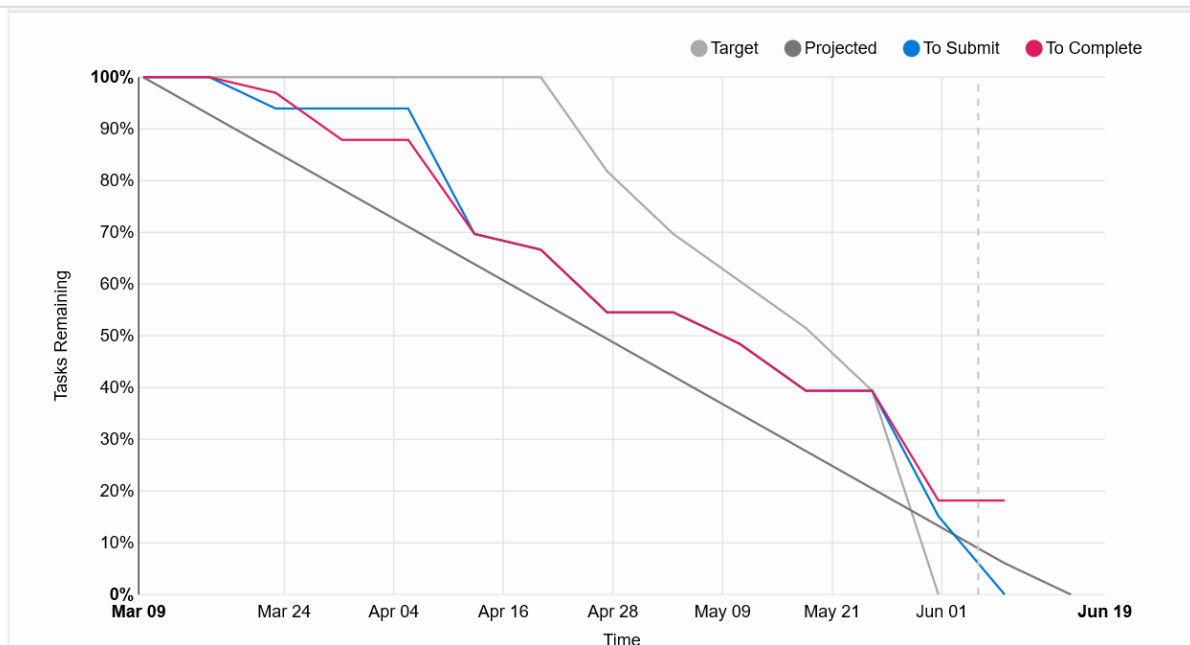
Whilst creating the Bank Program, I came to appreciate the importance of encapsulation and why we need to protect information within the system and ensure that it is not accessible to users by enacting a constructor and a read-only property.

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

1	Learning Summary Report	1
2	Overall Task Status	2
3	Learning Outcomes	3
3.1	Evaluate Code	3
3.2	Principles	3
3.3	Build Programs	3
3.4	Design	5
3.5	Justify	6
4	C# Essentials: Selection and Casting	7
5	C# Essentials: Repetition	15
6	C# Essentials: Classes and Objects	21
7	The Account Class	32
8	The MyTime Class	37
9	C# Essentials: Arrays and Lists	46
10	Validating Accounts	56
11	The MyPolynomial class	63
12	Bucket Sort	69
13	Exceptions and Error Handling	81
14	BuggySoft: Program Design and Class Composition	97
15	C# Essentials: Inheritance	114
16	Bank Transactions	134
17	C# Essentials: Polymorphism	150
18	Multiple Bank Accounts	160
19	A Simple Reaction-Timer Controller	178
20	An Enhanced Reaction-Timer Controller	185
21	Abstract Transactions	198
22	Documenting the Banking System	217
23	Helping Your Peers	220

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 normally use 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 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 can 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_38gij2nd
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



```
1  using System;
2
3  namespace IfStatement
4  {
5      class IfStatement
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Enter the number (as an integer): ");
10             int number=Convert.ToInt32(Console.ReadLine());
11             if(number==1)
12             {
13                 Console.WriteLine("One");
14             }
15             else if (number==2)
16             {
17                 Console.WriteLine("Two");
18             }
19             else if (number==3)
20             {
21                 Console.WriteLine("Three");
22             }
23
24             else if (number==4)
25             {
26                 Console.WriteLine("Four");
27             }
28             else if (number==5)
29             {
30                 Console.WriteLine("Five");
31             }
32             else if (number==6)
33             {
34                 Console.WriteLine("Six");
35             }
36             else if (number==7)
37             {
38                 Console.WriteLine("Seven");
39             }
40             else if (number==8)
41             {
42                 Console.WriteLine("Eight");
43             }
44             else if (number==9)
45             {
46                 Console.WriteLine("Nine");
47             }
48             else
49             {
50                 Console.WriteLine("Please enter a number between 0 and 9");
51             }
52         }
53     }
```



```
54     }  
55 }
```

```
1  using System;
2
3  namespace task_1_2
4  {
5      class SwitchStatement
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Enter a number (as an integer): ");
10             int number=Convert.ToInt32(Console.ReadLine());
11
12             switch(number)
13             {
14                 case 1: Console.WriteLine("One"); break;
15                 case 2: Console.WriteLine("Two"); break;
16                 case 3: Console.WriteLine("Three"); break;
17                 case 4: Console.WriteLine("Four"); break;
18                 case 5: Console.WriteLine("Five"); break;
19                 case 6: Console.WriteLine("Six"); break;
20                 case 7: Console.WriteLine("Seven"); break;
21                 case 8: Console.WriteLine("Eight"); break;
22                 case 9: Console.WriteLine("Nine"); break;
23
24                 default: Console.WriteLine("Error: you must enter an integer
25                     ↪ between 1 and 9"); break;
26             }
27             Console.ReadLine();
28         }
29     }
```

```
1  using System;
2
3  namespace Microwave
4  {
5      class Microwave
6      {
7          public static void Main(string[] args)
8          {
9              int heat=0;
10
11              Console.WriteLine("Enter how many items you wish to heat: ");
12              int number=Convert.ToInt32(Console.ReadLine());
13              for (int i=0; i<number;i++)
14              {
15                  Console.WriteLine("What is the heating time of each individual
16                  ↵ item: ");
17                  heat+=Convert.ToInt32(Console.ReadLine());
18              }
19
20              if(number==2)
21              {
22                  Console.WriteLine("Recommended heating time = " + heat*1.5);
23              }
24              else if(number==3)
25              {
26                  Console.WriteLine("Recommended heating time = " + heat*2);
27              }
28              else if(number>3)
29              {
30                  Console.WriteLine("Heating three items is not recommended");
31              }
32          }
33      }
34  }
```

```
1  using System;
2
3  namespace ConsoleApp1
4  {
5      class DoCasting
6      {
7          static void Main(string[] args)
8          {
9              int sum = 17;
10             int count = 5;
11             int intAverage = sum / count;
12             double doubleAverage = 2 * intAverage;
13             Console.WriteLine(intAverage);
14             Console.WriteLine(doubleAverage);
15             Console.WriteLine(Convert.ToDouble(sum/count));
16         }
17     }
18 }
```

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



```
1  using System;
2
3  namespace repetition
4  {
5      class program
6      {
7
8          public static void Main(string[] args)
9          {
10
11              /* for loop will loop through a given interval
12
13              int sum = 0;
14              int upperbound = 100;
15
16              for(int number=1; number<= upperbound; number++)
17              {
18                  sum += number;
19                  Console.WriteLine("Current number: " + number + " the sum is " +
↪      sum);
20
21              }
22              double average = (sum / upperbound);
23              Console.WriteLine(average);*/
24
25
26              /* While loop will check conditions before it executes block
27              int sum = 0;
28              int upperbound = 100;
29
30              int number = 1;
31              while(number<=upperbound)
32              {
33                  sum += number;
34                  Console.WriteLine("Current number: " + number + " the sum is " +
↪      sum);
35                  number++;
36              }
37
38              double average = (sum / upperbound);
39              Console.WriteLine(average);*/
40
41
42
43
44              //do while loop will loop through at least once
45
46              int sum = 0;
47              int upperbound = 100;
48
49              int number = 1;
50              do
51              {
```

```
52         sum += number;
53         Console.WriteLine("Current number: " + number + " the sum is " +
54             ↳ sum);
54         number++;
55     }while(number<=100);
56
57     double average = (sum / upperbound);
58     Console.WriteLine(average);
59
60     }
61 }
62 }
```



```
1  using System;
2
3  namespace GuessingNumber
4  {
5      class Program
6      {
7          public static void Main(string[] args)
8          {
9
10             //do while statement to prompt user 2 until they guess correctly
11
12             try
13             {
14                 int guess;
15                 Console.WriteLine("User 1, enter a number: ");
16                 int number = Convert.ToInt32(Console.ReadLine());
17
18                 do
19                 {
20
21                     Console.WriteLine("User 2, guess the number set by user 1: ");
22                     guess = Convert.ToInt32(Console.ReadLine());
23
24                     if (guess > number)
25                     {
26                         Console.WriteLine("Your guess is too high");
27                     }
28
29                     else if (guess < number)
30                     {
31                         Console.WriteLine("Your guess is too low");
32                     }
33
34                     } while (guess != number);
35
36                     Console.WriteLine("You have guessed the number! Well done!");
37             }
38
39             catch (Exception e)
40             {
41                 Console.WriteLine("Error, you entered in an unacceptable value");
42             }
43         }
44     }
45 }
46
47 }
```

```
1  using System;
2
3  namespace task1_2DF
4  {
5      class DivisibleFour
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Enter a number: ");
10             int n = Convert.ToInt32(Console.ReadLine());
11
12             for (int i = 1; i < n; i++)
13             {
14                 if (i % 4 == 0 && i % 5 != 0)
15                 {
16                     Console.WriteLine(i);
17                 }
18             }
19         }
20     }
21 }
```

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



```
1  using System;
2
3  namespace task2_1
4  {
5      class MobileProgram
6      {
7          static void Main(string[] args)
8          {
9              Mobile jimMobile = new Mobile("Monthly", "Samsung Galaxy S6",
10                 ↪ "07712223344");
11              Mobile DaleMobile = new Mobile("Weekly", "iPhone", "03922449383");
12
13              //Jim's Mobile Account
14              Console.WriteLine("Account Type: " + jimMobile.getAccType() + "\nMobile
15                 ↪ Number: " +
16              jimMobile.getNumber() + "\nDevice: " + jimMobile.getDevice() +
17                 ↪ "\nBalance: " + jimMobile.getBalance());
18
19              jimMobile.addCredit(10.0);
20              jimMobile.makeCall(5);
21              jimMobile.sendText(2);
22
23              //Dale's Mobile Account
24              Console.WriteLine("\n\nAccount Type: " + DaleMobile.getAccType() +
25                 ↪ "\nMobile Number: " +
26              DaleMobile.getNumber() + "\nDevice: " + DaleMobile.getDevice() +
27                 ↪ "\nBalance: " + DaleMobile.getBalance());
28
29              DaleMobile.addCredit(15.0);
30              DaleMobile.makeCall(10);
31              DaleMobile.sendText(4);
32
33              Console.WriteLine();
34              Console.ReadLine();
35          }
36      }
37  }
```

```
1  using System;
2  namespace task2_1
3  {
4      public class Mobile
5      {
6          //instance variables
7          private String accType, device, number;
8          private double balance;
9
10         private const double CALL_COST = 0.245;
11         private const double TEXT_COST = 0.078;
12
13         //constructor to access instance variables.
14         public Mobile(String accType, String device, String number)
15         {
16             this.accType = accType;
17             this.device = device;
18             this.number = number;
19             this.balance = 0.0;
20         }
21
22         public String getAccType()
23         {
24             return this.accType;
25         }
26
27
28         public String getDevice()
29         {
30             return this.device;
31         }
32
33         public String getNumber()
34         {
35             return this.number;
36         }
37
38         public String getBalance()
39         {
40             return this.balance.ToString("C");
41         }
42
43         public void setAccType(String accType)
44         {
45             this.accType = accType;
46         }
47
48         public void setDevice(String device)
49         {
50             this.device = device;
51         }
52
53         public void setNumber(String number)
```

```
54     {
55         this.number = number;
56     }
57
58     public void setBalance(double balance)
59     {
60         this.balance = balance;
61     }
62
63     //add double amount to balance
64     public void addCredit(double amount)
65     {
66         this.balance += amount;
67         Console.WriteLine("Credit added successfully. New balance " +
68             ↪ getBalance());
69     }
70
71     //deduct cost of making call by minutes
72     public void makeCall(int minutes)
73     {
74         double cost = minutes * CALL_COST;
75         this.balance -= cost;
76         Console.WriteLine("Call made. New balance " + getBalance());
77     }
78
79     //deduct cost of sending a text
80     public void sendText(int numTexts)
81     {
82         double cost = numTexts * TEXT_COST;
83         this.balance -= cost;
84         Console.WriteLine("Text sent. New balance " + getBalance());
85     }
86 }
87 }
```

```
1  using System;
2
3  namespace Task2_1E
4  {
5      public class EmployeeProgram
6      {
7          static void Main(string[] args)
8          {
9              Employee Sarah=new Employee("Sarah", 140000);
10             Employee Frank=new Employee("Frank", 84000);
11
12             //Sarah's salary
13             Console.WriteLine("Employee's name: " + Sarah.getName() + "\nSarah's
14             ↪ salary: " + Sarah.getSalary());
15
16             //Frank's Salary
17             Console.WriteLine("\nEmployee's name: " + Frank.getName() + "\nFrank's
18             ↪ salary: " + Frank.getSalary());
19
20             //raises salary
21             Console.WriteLine("\n");
22             Sarah.RaiseSalary(50);
23             Frank.RaiseSalary(50);
24
25             //deducts tax according to pay scale
26             Console.WriteLine("\n");
27             Sarah.Tax(140000);
28             Frank.Tax(84000);
29
30             Console.WriteLine();
31         }
32     }
```



```
1  using System;
2
3  namespace Task2_1E
4  {
5      class Employee
6      {   //instance variables
7          private string employeeName;
8          private double currentSalary;
9
10         //Constructor
11         public Employee(string employeeName, double currentSalary)
12         {
13             this.employeeName= employeeName;
14             this.currentSalary= currentSalary;
15         }
16
17         public String getName()
18         {
19             return this.employeeName;
20         }
21
22         public String getSalary()
23         {
24             return this.currentSalary.ToString("C");
25         }
26
27         public void RaiseSalary(double percentage)
28         {
29             double raise = ((percentage/100) * currentSalary);
30             this.currentSalary += raise;
31             Console.WriteLine("The salary after the raise is: " + getSalary());
32         }
33
34
35         public void Tax(double currentSalary)
36         {
37
38             if(currentSalary<=18200)
39             {
40                 this.currentSalary = currentSalary;
41                 Console.WriteLine("The salary after tax is " + getSalary());
42             }
43
44             else if (currentSalary>=18201 && currentSalary <= 37000)
45             {
46                 this.currentSalary = (currentSalary-18200)*0.19;
47                 Console.WriteLine("The salary after tax is " + getSalary());
48             }
49
50             else if (currentSalary >= 37001 && currentSalary <= 90000)
51             {
52                 this.currentSalary -= (3572 + (0.325 * (currentSalary - 37000)));
53                 Console.WriteLine("The salary after tax is " + getSalary());
```

```
54         }
55
56         else if (currentSalary >= 90001 && currentSalary <= 180000)
57         {
58             this.currentSalary -= (20797 + (0.37 * (currentSalary - 90000)));
59             Console.WriteLine("The salary after tax is " + getSalary());
60         }
61
62         else if (currentSalary >= 180001)
63         {
64             this.currentSalary -= (54096 + (0.45 * (currentSalary - 180000)));
65             Console.WriteLine("The salary after tax is " + getSalary());
66         }
67     }
68 }
69
70 }
```

```
1  using System;
2
3  namespace _2_1fuel
4  {
5      class CarProgram
6      {
7          static void Main(string[] args)
8          {
9              //create objects
10             Car ford = new Car(0.1, 40, 100);
11             Car toyota = new Car(2, 40, 100);
12
13             //refill car
14             ford.addFuel(10.00);
15             toyota.addFuel(5.00);
16
17
18             //drive method is called as total miles <= gallons
19             ford.drive(1000);
20
21             //drive method produces error message as total miles >= gallons
22             toyota.drive(100000);
23         }
24     }
25 }
```

```
1  using System;
2  namespace _2_1fuel
3  {
4      public class Car
5      {
6          //instance variables
7          private double efficiency;
8          private double fuel;
9          private double totalmiles;
10
11         //constant variable
12         public const double PetrolCost = 1.385;
13
14         public Car(double efficiency, double fuel, double totalmiles)
15         {
16             this.efficiency = efficiency;
17             this.fuel = 0;
18             this.totalmiles = 0;
19
20         }
21
22         public double getFuel()
23         {
24             return this.fuel;
25         }
26
27         public double getTotalMiles()
28         {
29             return this.totalmiles;
30         }
31
32
33         public void setTotalMiles(double totalmiles)
34         {
35             this.totalmiles = totalmiles;
36         }
37
38         public string printFuelCost(double cost)
39         {
40             return (cost.ToString("C"));
41         }
42
43         //add refill
44         public void addFuel(double refill)
45         {
46             this.fuel += refill;
47             Console.WriteLine("The amount of fuel after refill: " + getFuel());
48
49         }
50
51         //calculate cost by the cost of petrol
52         public void calcCost(double fuel)
53         {
```

```
54         double cost = fuel * PetrolCost;
55         Console.WriteLine("The cost of the fuel is: " + printFuelCost(cost));
56
57     }
58
59     public void convertToLitres(double gallons)
60     {
61         this.fuel = gallons * 4.546;
62         Console.WriteLine("The amount of fuel in litres is: " + getFuel());
63     }
64
65     //set miles and calculate gallons needed.
66     public void drive(double miles)
67     {
68
69         setTotalMiles(miles);
70         double gallons = totalmiles / efficiency;
71
72         if(totalmiles<=gallons)
73         {
74             convertToLitres(gallons);
75             calcCost(this.fuel);
76         }
77
78         else
79         {
80             Console.WriteLine("You do not have enough fuel to drive this
81                 ↪ distance");
82         }
83     }
84
85 }
86
87 }
```

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



```
1  using System;
2
3  namespace task2_2
4  {
5      class TestAccount
6      {
7          static void Main(string[] args)
8          {
9              //objects created for two bank users.
10             Account Mary = new Account(89078, "Mary");
11             Account John = new Account(78078, "John");
12
13             //examples of deposit and withdrawals from account
14             Mary.Deposit(1000);
15             John.Withdraw(1000);
16
17             //print bank details
18             Mary.Print();
19             John.Print();
20         }
21     }
22 }
```



```
1  using System;
2
3  namespace task2_2
4  {
5      public class Account
6      {
7          //instance variables declared
8          private decimal _balance;
9          private string _name;
10
11         //constructor
12         public Account(decimal _balance, string _name)
13         {
14             this._balance = _balance;
15             this._name = _name;
16         }
17
18         //prints name and balance
19         public void Print()
20         {
21             Console.WriteLine("The name of the account is " + getName() +
22                 ↳ "\nStarting Account Balance is " + getBalance());
23         }
24
25         //returns name
26         public String getName()
27         {
28             return this._name;
29         }
30
31         //returns balance
32         public decimal getBalance()
33         {
34             return this._balance;
35         }
36
37         //increases balance by adding deposit
38         public void Deposit(decimal amount)
39         {
40             this._balance += amount;
41             Console.WriteLine("Credit added successfully. New balance for " +
42                 ↳ this._name + " is " + this._balance);
43         }
44
45         //decreases balance by withdrawing the giving input amount
46         //does not guard against overdrawing from the account
47         public void Withdraw(decimal amount)
48         {
49             this._balance -= amount;
50             Console.WriteLine("Withdrawn undertaken successfully. New balance for "
51                 ↳ + this._name + " is " + this._balance);
52         }
53     }
54 }
```

51 }

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



```
1  using System;
2
3  namespace task2_3
4  {
5      class TestMyTime
6      {
7          static void Main(string[] args)
8          {
9
10             MyTime test_time1 = new MyTime();
11             MyTime test_time2 = new MyTime(22, 59, 45);
12
13
14             //test invalid time
15             test_time1.SetHour(68);
16             test_time2.SetTime(88, 43, 21);
17
18             Console.WriteLine("Time: " + test_time2.ToString());
19
20             //valid time
21             test_time1.SetTime(4, 4, 4);
22             Console.WriteLine("Time: " + test_time1.ToString());
23
24             test_time1.SetHour(5);
25             Console.WriteLine("New hour: " + test_time1.GetHour().ToString());
26
27             test_time1.SetMinute(20);
28             Console.WriteLine("New minute: " + test_time1.GetMinute().ToString());
29
30             test_time1.SetSecond(49);
31             Console.WriteLine("New Second: " + test_time1.GetSecond().ToString());
32
33
34             //next second, minute, hour for test_time1
35             Console.WriteLine("The next second for test_time1 will be: " +
36                 ↪ test_time1.NextSecond());
37
38             Console.WriteLine("The next minute for test_time1 will be: " +
39                 ↪ test_time1.NextMinute());
40
41             Console.WriteLine("The next hour for test_time1 will be: " +
42                 ↪ test_time1.NextHour());
43
44             //previous second, minute, hour for test_time1
45             Console.WriteLine("The previous for test_time1 second will be: " +
46                 ↪ test_time1.PreviousSecond());
47
48             Console.WriteLine("The previous for test_time1 minute will be: " +
49                 ↪ test_time1.PreviousMinute());
50
51             Console.WriteLine("The previous for test_time1 hour will be: " +
52                 ↪ test_time1.PreviousHour());
```

```
49
50
51 //next second, minute, hour for test_time2
52 Console.WriteLine("The next second for test_time2 will be: " +
53     ↳ test_time2.NextSecond());
54
55 Console.WriteLine("The next minute for test_time2 will be: " +
56     ↳ test_time2.NextMinute());
57
58 Console.WriteLine("The next hour for test_time2 will be: " +
59     ↳ test_time2.NextHour());
60
61 //previous second, minute, hour for test_time2
62 Console.WriteLine("The previous second for test_time2 will be: " +
63     ↳ test_time2.PreviousSecond());
64
65 Console.WriteLine("The previous minute for test_time2 will be: " +
66     ↳ test_time2.PreviousMinute());
67
68 Console.WriteLine("The previous hour for test_time2 will be: " +
69     ↳ test_time2.PreviousHour());
70 }
```

```
1  using System;
2  namespace task2_3
3  {
4      public class MyTime
5      {
6          //instance variables
7          private int hour;
8          private int minute;
9          private int second;
10
11         //default constructor
12         public MyTime()
13         {
14             hour = 0;
15             minute = 0;
16             second = 0;
17         }
18         //constructor with three arguments
19         public MyTime(int hour, int minute, int second)
20         {
21             //conditional statement to assess validity of input
22             if (hour >= 0 && hour <= 24 && minute >= 0 && minute <= 59 && second >=
                ↪ 0 && second <= 59)
23             {
24                 this.hour = hour;
25                 this.minute = minute;
26                 this.second = second;
27             }
28
29             else
30             {
31                 //print error message
32                 Console.WriteLine("Invalid Time");
33             }
34         }
35
36         public void SetTime(int hour, int minute, int second)
37         {
38             //conditional statement to assess validity of input
39             if (hour >= 0 && hour <= 24 && minute >= 0 && minute <= 59 && second >=
                ↪ 0 && second <= 59)
40             {
41                 this.hour = hour;
42                 this.minute = minute;
43                 this.second = second;
44             }
45
46             else
47             {
48                 Console.WriteLine("Invalid Time");
49             }
50         }
51     }
```

```
52
53
54 public void SetHour(int hour)
55 {
56     //set hour parameters
57     if (hour >= 0 && hour <= 23)
58     {
59         this.hour = hour;
60     }
61     else
62     {
63         Console.WriteLine("Invalid hour");
64     }
65 }
66
67
68 public void SetMinute(int minute)
69 {
70     //set minute parameters
71     if (minute >= 0 && minute <= 59)
72     {
73         this.minute = minute;
74     }
75     else
76     {
77         Console.WriteLine("Invalid minute");
78     }
79 }
80
81
82 public void SetSecond(int second)
83 {
84     //set second parameters
85     if (second >= 0 && second <= 59)
86     {
87         this.second = second;
88     }
89     else
90     {
91         Console.WriteLine("Invalid second");
92     }
93 }
94
95
96 public int GetHour()
97 {
98     return this.hour;
99 }
100
101 public int GetMinute()
102 {
103     return this.minute;
104 }
```



```
105
106     public int GetSecond()
107     {
108         return this.second;
109     }
110
111     public override string ToString()
112     {
113         return String.Format("{0:00}:{1:00}:{2:00}", this.hour, this.minute,
114             ↪ this.second);
115     }
116
117     //Next Second Method
118     public MyTime NextSecond()
119     {
120         int next_second = this.second + 1;
121         int next_minute = this.minute;
122         int next_hour = this.hour;
123
124         if (next_second >= 60)
125         {
126             next_second = 0;
127             next_minute += 1;
128         }
129
130         if (next_minute >= 60)
131         {
132             next_minute = 0;
133             next_hour += 1;
134         }
135         if (next_hour >= 24)
136         {
137             next_hour = 0;
138         }
139
140         MyTime nxtsec = new MyTime(next_hour, next_minute, next_second);
141         return nxtsec;
142     }
143
144
145     public MyTime NextMinute()
146     {
147         int next_minute = this.minute + 1;
148         int next_hour = this.hour;
149
150         if (next_minute >= 60)
151         {
152             next_minute = 0;
153             next_hour += 1;
154         }
155         if (next_hour >= 24)
156         {
```

```
157         next_hour = 0;
158     }
159
160     MyTime nxtmin = new MyTime(next_hour, next_minute, this.second);
161     return nxtmin;
162
163 }
164
165 public MyTime NextHour()
166 {
167     int next_hour = this.hour + 1;
168
169     if (next_hour >= 24)
170     {
171         next_hour = 0;
172     }
173
174     MyTime nxthour = new MyTime(next_hour, this.minute, this.second);
175     return nxthour;
176
177 }
178
179 public MyTime PreviousSecond()
180 {
181     int previous_second = this.second - 1;
182     int previous_minute = this.minute;
183     int previous_hour = this.hour;
184
185     if (previous_second < 0)
186     {
187         previous_second = 59;
188         previous_minute -= 1;
189     }
190
191     if (previous_minute < 0)
192     {
193         previous_minute = 59;
194         previous_hour -= 1;
195     }
196
197     if (previous_hour < 0)
198     {
199         previous_hour = 23;
200     }
201
202     MyTime prevsec = new MyTime(previous_hour, previous_minute,
203     ↪ previous_second);
204     return prevsec;
205
206 }
207
208 public MyTime PreviousMinute()
209 {
```

```
209         int previous_minute = this.minute - 1;
210         int previous_hour = this.hour;
211
212         if (previous_minute < 0)
213         {
214             previous_minute = 59;
215             previous_hour -= 1;
216         }
217         if (previous_hour < 0)
218         {
219             previous_hour = 23;
220         }
221
222         MyTime prevmin = new MyTime(previous_hour, previous_minute,
223             ↪ this.second);
224         return prevmin;
225     }
226
227     public MyTime PreviousHour()
228     {
229         int previous_hour = this.hour - 1;
230
231         if (previous_hour < 0)
232         {
233             previous_hour = 23;
234         }
235
236         MyTime prevhour = new MyTime(previous_hour, this.minute, this.second);
237         return prevhour;
238     }
239
240
241 }
242 }
```

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



```
1  using System;
2  using System.Collections.Generic;
3
4  namespace task3_1
5  {
6      class Program
7      {
8
9          static void Main(string[] args)
10         {
11
12             Console.WriteLine("-----
13             ↳ -----");
14             Console.WriteLine("-----Task
15             ↳ One-----");
16             Console.WriteLine("-----
17             ↳ -----");
18             double[] myArray = new double[10];
19
20             myArray[0] = 1.0;
21             myArray[1] = 1.1;
22             myArray[2] = 1.2;
23             myArray[3] = 1.3;
24             myArray[4] = 1.4;
25             myArray[5] = 1.5;
26             myArray[6] = 1.6;
27             myArray[7] = 1.7;
28             myArray[8] = 1.8;
29             myArray[9] = 1.9;
30
31             Console.WriteLine("The first element in myArray is " + myArray[0]);
32             Console.WriteLine("The second element in myArray is " + myArray[1]);
33             Console.WriteLine("The third element in myArray is " + myArray[2]);
34             Console.WriteLine("The fourth element in myArray is " + myArray[3]);
35             Console.WriteLine("The fifth element in myArray is " + myArray[4]);
36             Console.WriteLine("The sixth element in myArray is " + myArray[5]);
37             Console.WriteLine("The seventh element in myArray is " + myArray[6]);
38             Console.WriteLine("The eighth element in myArray is " + myArray[7]);
39             Console.WriteLine("The ninth element in myArray is " + myArray[8]);
40             Console.WriteLine("The tenth element in myArray is " + myArray[9]);
41
42             Console.WriteLine("\n\n-----
43             ↳ -----");
44             Console.WriteLine("-----Task
45             ↳ Two-----");
46             Console.WriteLine("-----
47             ↳ -----");
48             int[] my_Array = new int[10];
49
50             for (int i = 0; i < my_Array.Length; i++)
51             {
52                 my_Array[i] = i;
```

```
48     }
49     for (int i = 0; i < my_Array.Length; i++)
50     {
51         Console.WriteLine("The element at position " + i + " in the array
           ↳ is " + my_Array[i]);
52     }
53     Console.WriteLine("\n\n-----"]
           ↳ -----");
54     Console.WriteLine("-----Task
           ↳ Three-----");
55     Console.WriteLine("-----"]
           ↳ -----");
56
57
58
59     int[] studentArray = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
60     int total = 0;
61
62     for (int i = 0; i < studentArray.Length; i++)
63     {
64         total += studentArray[i];
65     }
66
67     Console.WriteLine("The total marks for the student is " + total);
68     Console.WriteLine("This consists of " + studentArray.Length + " marks");
69     Console.WriteLine("Therefore the average mark is " + (total /
           ↳ studentArray.Length));
70
71     Console.WriteLine("\n\n-----"]
           ↳ -----");
72     Console.WriteLine("-----Task
           ↳ Four-----");
73     Console.WriteLine("-----"]
           ↳ -----");
74
75     //assigns names to the array
76     string[] studentNames = new string[6];
77
78     for (int i = 0; i < studentNames.Length; i++)
79     {
80         try
81         {
82             Console.WriteLine("Enter in the name of a student: ");
83             studentNames[i] = Console.ReadLine();
84         }
85         catch (FormatException)
86         {
87             Console.WriteLine("You need to enter a name.");
88         }
89     }
90
91     //prints names stored within the array
92     for (int i = 0; i < studentNames.Length; i++)
```

```
93     {
94         Console.Write(studentNames[i] + " ");
95     }
96
97     Console.WriteLine("\n\n-----"]
98     ↪ -----");
99     Console.WriteLine("-----Task
100     ↪ Five-----");
101     Console.WriteLine("-----"]
102     ↪ -----");
103     double[] numbers = new double[10];
104     double currentLargest, currentSmallest;
105
106     try
107     {
108         for (int i = 0; i < numbers.Length; i++)
109         {
110             Console.WriteLine("Enter in a double: ");
111             numbers[i] = Convert.ToDouble(Console.ReadLine());
112         }
113     }
114     catch (FormatException)
115     {
116         Console.WriteLine("That is not a number");
117     }
118
119     currentLargest = numbers[0];
120
121     for (int i = 0; i < numbers.Length; i++)
122     {
123         if (currentLargest < numbers[i])
124         {
125             currentLargest = numbers[i];
126             Console.WriteLine();
127         }
128     }
129
130     Console.WriteLine("\nThe largest value in the array is " +
131     ↪ currentLargest);
132
133     currentSmallest = numbers[0];
134
135     for (int i = 0; i < 10; i++)
136     {
137         if (currentSmallest > numbers[i])
138         {
139             currentSmallest = numbers[i];
140         }
141     }
```



```

142 Console.WriteLine("\nThe smallest value in the array is " +
    ↪ currentSmallest);
143
144 Console.WriteLine("\n\n-----"]
    ↪ -----");
145 Console.WriteLine("-----Task
    ↪ Six-----");
146 Console.WriteLine("-----"]
    ↪ -----");
147
148 int[,] myArray1 = new int[3, 4] { { 1, 2, 3, 4 }, { 1, 1, 1, 1 }, { 2,
    ↪ 2, 2, 2 } };
149
150 for (int i = 0; i < myArray1.GetLength(0); i++)
151 {
152     for (int j = 0; j < myArray1.GetLength(1); j++)
153     {
154         Console.Write(myArray1[i, j] + "\t");
155     }
156     Console.WriteLine();
157 }
158
159 Console.WriteLine("\n");
160 Console.WriteLine("-----"]
    ↪ -----");
161
162
163 List<String> myStudentList = new List<String>();
164 Random randomValue = new Random();
165 int randomNumber = randomValue.Next(1, 12);
166 Console.WriteLine("You now need to add " + randomNumber + " students to
    ↪ your class list");
167
168 for (int i = 0; i < randomNumber; i++)
169 {
170     Console.WriteLine("Please enter the name of Student " + (i + 1) +
    ↪ ": ");
171     myStudentList.Add(Console.ReadLine());
172     Console.WriteLine();
173 }
174
175
176 Console.Write("The name of the students in my list are ");
177 for (int i = 0; i < myStudentList.Count; i++)
178 {
179     Console.Write(myStudentList[i] + " , ");
180 }
181 Console.WriteLine("\n\n-----"]
    ↪ -----");
182 Console.WriteLine("-----Task
    ↪ Seven-----");
183 Console.WriteLine("-----"]
    ↪ -----");

```

```
184
185 Console.WriteLine("How many elements would you like to enter");
186 int elements = Int32.Parse(Console.ReadLine());
187 int[] number = new int[elements];
188
189 Console.WriteLine("Enter integers in your array: ");
190 //populate Array
191
192 for (int i = 0; i < number.GetLength(0); i++)
193 {
194
195     number[i] = Convert.ToInt32(Console.ReadLine());
196 }
197
198
199 int FuncOne(int[] number)
200 {
201     int total_even = 0;
202     int total_odd = 1;
203     int number_FuncOne = 1;
204     if (number.Length > 10)
205     {
206         for (int i = 0; i < number.Length; i++)
207         {
208             if (number[i] % 2 == 0)
209                 total_even++;
210         }
211         Console.Write("The total of the even numbers is " + total_even);
212         return total;
213     }
214
215     else if (number.Length < 10)
216     {
217         for (int i = 1; i < number.Length; i++)
218         {
219             if (number[i] % 2 == 1)
220                 total_odd *= number[i];
221         }
222         Console.Write("The product of the odd numbers is " +
223             ↪ number_FuncOne);
224         return number_FuncOne;
225     }
226
227     else
228     {
229         throw new ArgumentException("Invalid entry");
230     }
231 }
232
233 FuncOne(number);
234 Console.WriteLine("\n\n-----
235 ↪ -----");
```

```
235
236
237 List<double> list_one = new List<double> { 1.2, 3.4, 6.5, 7.2 };
238
239 double FuncTwo(List<double> list_one)
240 {
241     double total_list = 0;
242     for (int i = 0; i < list_one.Count; i++)
243     {
244         total_list += list_one[i];
245     }
246     double average = total_list / list_one.Count;
247     Console.WriteLine("The average is " + average);
248     Console.WriteLine("The calcuation of each element minus the average
    ↳ is: ");
249
250     for (int i = 0; i < list_one.Count; i++)
251     {
252         list_one[i] = list_one[i] - average;
253         Console.Write(list_one[i] + "\t");
254     }
255
256     return 1;
257 }
258
259 FuncTwo(list_one);
260
261 Console.WriteLine("\n\n-----"]
    ↳ -----");
262 Console.WriteLine("-----Task
    ↳ Eight-----");
263 Console.WriteLine("-----"]
    ↳ -----");
264
265
266
267 int[,] FuncTree = new int[3, 4];
268 Console.WriteLine("Enter elements for a 3 x 4 2 dimensional array");
269
270 //populate Array
271 for (int i = 0; i < FuncTree.GetLength(0); i++)
272 {
273     for (int j = 0; j < FuncTree.GetLength(1); j++)
274     {
275         FuncTree[i, j] = Convert.ToInt32(Console.ReadLine());
276     }
277 }
278
279 //assign multiples of three to 1 dimensional array
280 for (int i = 0; i < FuncTree.GetLength(0); i++)
281 {
282     for (int j = 0; j < FuncTree.GetLength(1); j++)
283     {
```

```
284         int element = FuncTree[i, j];
285         if (element % 3 == 0)
286         {
287             List<int> list = new List<int>();
288             list.Add(element);
289             int[] array = new int[element];
290             Console.WriteLine("The number {0} in this array is a
                ↳ multiple of 3: ", element);
291         }
292     }
293 }
294 }
295 }
296 }
297
298
299 Console.WriteLine("\n\n-----"]
    ↳ -----");
300 Console.WriteLine("-----Task
    ↳ Nine-----");
301 Console.WriteLine("-----"]
    ↳ -----");
302
303
304
305
306 int[,] FuncFour(int[] input)
307 {
308
309     int[,] range = new int[input.Length, 5];
310     for (int i = 0; i < input.Length; i++)
311     {
312         for (int j = 1; j <= 5; j++)
313         {
314             range[i, j - 1] = input[i] * j;
315         }
316     }
317     return range;
318 }
319 }
320
321 int[] NewArray = { 1, 2, 3, 4, 5 };
322 int[,] range = FuncFour(NewArray);
323
324 Console.WriteLine(" \t1\t2\t3\t4\t5");
325 for (int i = 0; i < range.GetLength(1); i++)
326 {
327
328     Console.Write(NewArray[i] + "\t");
329     for (int j = 0; j < 5; j++)
330     {
331         Console.Write(range[i, j] + "\t");
332     }
```

```
333         Console.WriteLine();
334     }
335     Console.WriteLine();
336 }
337
338 }
339
340 }
```

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



```
1  using System;
2  using task2_2;
3  //using task2_2;
4
5  namespace task3_2
6  {
7
8
9      class BankSystem
10     {
11
12         enum MenuOption
13         {
14             Withdraw,
15             Deposit,
16             Print,
17             Quit
18         }
19
20         static void DoDeposit(Account account)
21         {
22             Boolean result;
23             Console.WriteLine("Enter deposit amount: ");
24             decimal amount = Convert.ToDecimal(Console.ReadLine());
25             result = account.Deposit(amount);
26
27             if(result==true)
28             {
29                 Console.WriteLine("Your transaction was successful");
30             }
31             else
32             {
33                 Console.WriteLine("Your transaction was unsuccessful");
34             }
35         }
36
37         static void DoWithdrawal(Account account)
38         {
39             Boolean result;
40             Console.WriteLine("Enter withdrawal amount: ");
41             decimal amount = Convert.ToDecimal(Console.ReadLine());
42             result = account.Withdraw(amount);
43
44             if (result == true)
45             {
46                 Console.WriteLine("Your withdrawal was successful");
47             }
48             else if (result==false)
49             {
50                 Console.WriteLine("Your withdrawal was unsuccessful");
51             }
52         }
53     }
```



```
54
55     static MenuOption ReadUserOption()
56     {
57         int? option = null;
58         do
59         {
60             Console.WriteLine("Please select from the following options");
61             Console.WriteLine("MENU \n1. Withdraw \n2. Deposit \n3. Print \n4.
        ↪ Quit");
62
63             try
64             {
65                 option = Convert.ToInt32(Console.ReadLine());
66                 if (option > 4 || option < 1)
67                 {
68                     option = null;
69                     Console.WriteLine("Please enter a number from 1 to 4 from
        ↪ the menu");
70                 }
71             }
72             catch (FormatException)
73             {
74                 Console.WriteLine("Invalid input. Enter an integer from 1-4");
75             }
76
77             } while (option == null);
78
79             return (MenuOption)option;
80         }
81
82
83     static void DoPrint(Account account)
84     {
85
86         account.Print();
87
88     }
89
90
91
92
93     static void Main(string[] args)
94     {
95
96
97         Account person_one = new Account(400, "Lucy");
98         Account person_two = new Account(1000, "Michael");
99         MenuOption option;
100
101         do
102         {
103             option = ReadUserOption()-1;
104
```

```
105         switch (option)
106         {
107             case MenuOption.Withdraw:
108                 Console.WriteLine("You have selected withdraw");
109                 Console.WriteLine("---Lucy's Account---");
110                 DoWithdrawal(person_one);
111                 Console.WriteLine("---Michael's Account---");
112                 DoWithdrawal(person_two);
113                 break;
114             case MenuOption.Deposit:
115                 Console.WriteLine("You have selected deposit");
116                 Console.WriteLine("---Lucy's Account---");
117                 DoDeposit(person_one);
118                 Console.WriteLine("---Michael's Account---");
119                 DoDeposit(person_two);
120                 break;
121             case MenuOption.Print:
122                 Console.WriteLine("You have selected print");
123                 Console.WriteLine("---Lucy's Account---");
124                 DoPrint(person_one);
125                 Console.WriteLine("---Michael's Account---");
126                 DoPrint(person_two);
127                 break;
128             case MenuOption.Quit:
129                 Console.WriteLine("Goodbye");
130                 break;
131         }
132     } while (option != MenuOption.Quit);
133 }
134
135
136
137
138 }
139 }
```

```
1  using System;
2
3  namespace task2_2
4  {
5      public class Account
6      {
7          //instance variables declared
8          private decimal _balance;
9          private string _name;
10
11         //constructor
12         public Account(decimal balance, string name)
13         {
14
15             this._name = name;
16             if (balance <= 0)
17                 return;
18             this._balance = balance;
19
20         }
21
22         //prints name and balance
23         public void Print()
24         {
25             Console.WriteLine("The name of the account holder is " + getName() +
26                 ↳ "\nCurrent Account Balance is " + getBalance());
27
28         }
29
30         //returns name
31         public String getName()
32         {
33             return this._name;
34         }
35
36         //returns balance
37         public decimal getBalance()
38         {
39             return this._balance;
40         }
41
42         //increases balance by adding deposit
43         //boolean ensure that no 0 or a negative value can not be input
44         public Boolean Deposit(decimal amount)
45         {
46
47             if (amount <= 0)
48                 return false;
49
50             this._balance += amount;
51             //Console.WriteLine("Credit added successfully. New balance for " +
52                 ↳ this._name + " is " + this._balance);
53             return true;
54         }
55     }
```

```
52     }
53
54
55     //decreases balanace by withdrawing the giving input amount
56     //boolean values protect against overdrawing from account
57     public Boolean Withdraw(decimal amount)
58     {
59         if (amount > this._balance || amount < 0)
60             return false;
61
62
63         this._balance -= amount;
64         //Console.WriteLine("Withdrawal successfull. New balance for " +
65         ↪ this._name + " is " + this._balance);
66         return true;
67     }
68
69 }
70 }
```

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



```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace task3_3
6  {
7      class TestmyPolynomial
8      {
9          static void Main(string[] args)
10         {
11             MyPolynomial polynomial_1 = new MyPolynomial(new double[] { 6, 0, 4, 2,
12                 ↪ 6, 5, 5 });
13             MyPolynomial polynomial_2 = new MyPolynomial(new double[] { 5, 3, 3, 2,
14                 ↪ -1, 0, 2 });
15
16             Console.WriteLine("Polynomial 1 is: " + polynomial_1.ToString());
17             Console.WriteLine("Polynomial 1 is: " + polynomial_2.ToString());
18             Console.WriteLine("The degree of polynomial 1 is: " +
19                 ↪ polynomial_1.GetDegree());
20             Console.WriteLine("The degree of polynomial 2 is: " +
21                 ↪ polynomial_2.GetDegree());
22             Console.WriteLine("The evaluation of polynomial 1 at 4 : " +
23                 ↪ polynomial_1.Evaluate(4));
24             Console.WriteLine("The evaluation of polynomial 1 at 4 : " +
25                 ↪ polynomial_2.Evaluate(4));
26             Console.WriteLine("The multiplication of the two polynomials is: " +
27                 ↪ polynomial_1.Multiply(polynomial_2));
28             Console.WriteLine("The addition of the two polynomials is: " +
29                 ↪ polynomial_1.Add(polynomial_2));
30
31         }
32     }
33 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  namespace task3_3
5  {
6      public class MyPolynomial
7      {
8          private double[] _coeffs;
9
10         public MyPolynomial(double[] array_coeffs)
11         {
12
13             _coeffs = array_coeffs;
14
15         }
16
17         public int GetDegree()
18         {
19             return _coeffs.Length - 1;
20         }
21
22         // Create a string representation
23         public override string ToString()
24         {
25             //null string
26             string poly = null;
27
28             for (int i = _coeffs.Length - 1; i >= 0; i--)
29             {
30                 if (_coeffs[i] == 0) continue;
31
32                 //if _coeffs[i]> greater than zero use "+"
33                 //if _coeffs[i]> less than zero use "-"
34                 poly += $"{(_coeffs[i] > 0 ? "+" : "-")}";
35                 if (i == 0 || _coeffs[i] != 1)
36                 {
37                     poly += Math.Abs(_coeffs[i]);
38                 }
39
40                 if (i == 1)
41                 {
42                     poly += "x";
43                 }
44                 //raise to the value of i
45                 else if (i > 0)
46                 {
47                     poly += $"x^{i}";
48                 }
49             }
50             //remove addition from the start of the polynomial
51             return poly.Trim(' ', '+');
52         }
53     }
```



```
54
55     //evaluate polynomial
56     public double Evaluate(double x)
57     {
58
59         double result=0.0;
60
61         for (int i = 0; i < _coeffs.Length; i++)
62         {
63             result += _coeffs[i] * Math.Pow(x, i);
64         }
65
66         return result;
67     }
68
69
70     public MyPolynomial Multiply(MyPolynomial another)
71     {
72         int degree = GetDegree() + another.GetDegree();
73         var result = new double[degree + 1];
74         for(int i=0; i<_coeffs.Length;i++)
75         {
76             if (_coeffs[i] == 0) continue;
77             for(int j=0;j<another._coeffs.Length;j++)
78             {
79                 if (another._coeffs[j] == 0) continue;
80                 result[i + j] += _coeffs[i] + another._coeffs[j];
81             }
82         }
83         _coeffs = result;
84         return this;
85     }
86
87     public MyPolynomial Add(MyPolynomial another)
88     {
89
90         double[] result = _coeffs;
91         double[] array = another._coeffs;
92         if (another.GetDegree() > GetDegree())
93         {
94             result = another._coeffs;
95             array = _coeffs;
96         }
97
98         for (var i = 0; i < array.Length; i++)
99         {
100             result[i] += array[i];
101         }
102
103         _coeffs = result;
104         return this;
105
106
```

```
107         }
108
109
110     }
111
112
113
114 }
```

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



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5
6  namespace task3_4
7  {
8
9
10     class Program
11     {
12
13         enum MenuOption
14         {
15             Withdraw,
16             Deposit,
17             Print,
18             Quit
19         }
20
21         static void DoDeposit(Account account)
22         {
23             Boolean result;
24             Console.WriteLine("Enter deposit amount: ");
25             decimal amount = Convert.ToDecimal(Console.ReadLine());
26             result = account.Deposit(amount);
27
28             if (result == true)
29             {
30                 Console.WriteLine("Your transaction was successful");
31             }
32             else
33             {
34                 Console.WriteLine("Your transaction was unsuccessful");
35             }
36         }
37
38         static void DoWithdrawal(Account account)
39         {
40             Boolean result;
41             Console.WriteLine("Enter withdrawal amount: ");
42             decimal amount = Convert.ToDecimal(Console.ReadLine());
43             result = account.Withdraw(amount);
44
45             if (result == true)
46             {
47                 Console.WriteLine("Your withdrawal was successful");
48             }
49             else if (result == false)
50             {
51                 Console.WriteLine("Your withdrawal was unsuccessful");
52             }
53         }
54     }
55 }
```

```
54
55
56 static MenuOption ReadUserOption()
57 {
58     int? option = null;
59     do
60     {
61         Console.WriteLine("Please select from the following options");
62         Console.WriteLine("MENU \n1. Withdraw \n2. Deposit \n3. Print \n4.
        ↪ Quit");
63
64         try
65         {
66             option = Convert.ToInt32(Console.ReadLine());
67             if (option > 4 || option < 1)
68             {
69                 option = null;
70                 Console.WriteLine("Please enter a number from 1 to 4 from
        ↪ the menu");
71             }
72         }
73         catch (FormatException)
74         {
75             Console.WriteLine("Invalid input. Enter an integer from 1-4");
76         }
77     } while (option == null);
78
79     return (MenuOption)option;
80 }
81
82
83
84 static void DoPrint(Account account)
85 {
86
87     account.Print();
88
89 }
90
91 static void PrintAccountArray(Account[] accounts)
92 {
93     foreach (Account account in accounts)
94         account.Print();
95 }
96
97
98
99
100 static void Main(string[] args)
101 {
102
103
104     Account[] accounts_Array = new Account[3];
```

```
105
106     accounts_Array[0] = new Account(Convert.ToDecimal(8394.43), "Mary
    ↪ Stevens");
107     accounts_Array[1] = new Account(Convert.ToDecimal(8492.20), "David
    ↪ Marks");
108     accounts_Array[2] = new Account(Convert.ToDecimal(4843.90), "Jack
    ↪ Todd");
109     Console.WriteLine("Accounts before sorting");
110     PrintAccountArray(accounts_Array);
111     Console.WriteLine("Accounts after sorting");
112     AccountsSorter.Sort(accounts_Array, 3);
113     PrintAccountArray(accounts_Array);
114
115     Console.WriteLine("-----");
116     List<Account> accounts_List = new List<Account>();
117     accounts_List.Add(new Account(Convert.ToDecimal(37493.50), "Sam
    ↪ Newton"));
118     accounts_List.Add(new Account(Convert.ToDecimal(84084.70), "James
    ↪ Evans"));
119     accounts_List.Add(new Account(Convert.ToDecimal(484.70), "Kylie
    ↪ Denis"));
120
121     Console.WriteLine("Accounts before sorting");
122     PrintAccountArray(accounts_List.ToArray());
123
124     Console.WriteLine("Accounts after sorting");
125     AccountsSorter.Sort(accounts_List, 3);
126     PrintAccountArray(accounts_List.ToArray());
127
128     Account person_one = new Account(400, "Lucy");
129     Account person_two = new Account(1000, "Michael");
130     MenuOption option;
131
132     do
133     {
134         option = ReadUserOption() - 1;
135
136         switch (option)
137         {
138             case MenuOption.Withdraw:
139                 Console.WriteLine("You have selected withdraw");
140                 Console.WriteLine("---Lucy's Account---");
141                 DoWithdrawal(person_one);
142                 Console.WriteLine("---Michael's Account---");
143                 DoWithdrawal(person_two);
144                 break;
145             case MenuOption.Deposit:
146                 Console.WriteLine("You have selected deposit");
147                 Console.WriteLine("---Lucy's Account---");
148                 DoDeposit(person_one);
149                 Console.WriteLine("---Michael's Account---");
150                 DoDeposit(person_two);
151                 break;
```

```
152         case MenuOption.Print:
153             Console.WriteLine("You have selected print");
154             Console.WriteLine("---Lucy's Account---");
155             DoPrint(person_one);
156             Console.WriteLine("---Michael's Account---");
157             DoPrint(person_two);
158             break;
159         case MenuOption.Quit:
160             Console.WriteLine("Goodbye");
161             break;
162     }
163 } while (option != MenuOption.Quit);
164 }
165 }
166
167
168
169 }
170 }
```



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5
6  namespace task3_4
7  {
8      static class AccountsSorter
9      {
10
11         private static decimal MaximumBalance(Account[] accounts)
12         {
13             return accounts.Max(a => a.Balance);
14         }
15
16         private static List<Account>[] CreateBuckets(int b)
17         {
18             List<Account>[] buckets = new List<Account>[b];
19             for(int i=0;i<buckets.Length;i++)
20             {
21                 buckets[i] = new List<Account>();
22             }
23             return buckets;
24         }
25
26         private static void DistributeAccounts(Account[] accounts, List<Account>[]
27         ↪ buckets)
28         {
29             decimal maximum = MaximumBalance(accounts);
30             foreach (Account account in accounts)
31             {
32                 int bucket = (int)(Math.Floor(buckets.Length * account.Balance /
33                 ↪ maximum));
34                 if (bucket == buckets.Length)
35                     bucket -= 1;
36                 buckets[bucket].Add(account);
37             }
38         }
39
40         private static void SortBuckets(List<Account>[] buckets)
41         {
42             for (int i = 0; i < buckets.Length; i++)
43             {
44                 buckets[i] = buckets[i].OrderBy(a => a.Balance).ToList();
45             }
46         }
47
48         public static void Sort(Account[] accounts, int b)
49         {
50             if(accounts==null)
51             {
```

```
52         throw new NullReferenceException("Accounts can not be null");
53     }
54     if(b<=1)
55     {
56         throw new ArgumentOutOfRangeException("buckets can not be less than
57             ↪ 2");
58     }
59
60     List<Account>[] buckets = CreateBuckets(b);
61     DistributeAccounts(accounts, buckets);
62     SortBuckets(buckets);
63
64     int idx = 0;
65     for (int i = 0; i < buckets.Length; i++)
66     {
67         foreach (Account account in buckets[i])
68         {
69             accounts[idx] = account;
70             idx++;
71         }
72     }
73
74     //Console.WriteLine();
75
76 }
77
78
79 }
80
81 public static void Sort(List<Account> accounts, int b)
82 {
83     if(accounts==null)
84     {
85         throw new NullReferenceException("Accounts cannot be null");
86     }
87     Account[] accountsArray = accounts.ToArray();
88     Sort(accountsArray, b);
89
90     for(int i=0; i<accounts.Count;i++)
91     {
92         accounts[i] = accountsArray[i];
93     }
94
95     // Console.WriteLine();
96 }
97
98 }
99 }
100
101
102
103 //class BucketSort
```

```
104  //{
105  //    public static List<int> Sort(params int[] x)
106  //    {
107  //        List<int> sortedArray = new List<int>();
108
109  //        int numOfBuckets = 10;
110
111  //        //Create buckets
112  //        List<int>[] buckets = new List<int>[numOfBuckets];
113  //        for (int i = 0; i < numOfBuckets; i++)
114  //        {
115  //            buckets[i] = new List<int>();
116  //        }
117
118  //        //Iterate through the passed array and add each integer to the
119  //        ↪ appropriate bucket
120  //        for (int i = 0; i < x.Length; i++)
121  //        {
122  //            int bucket = (x[i] / numOfBuckets);
123  //            buckets[bucket].Add(x[i]);
124  //        }
125
126  //        //Sort each bucket and add it to the result List
127  //        for (int i = 0; i < numOfBuckets; i++)
128  //        {
129  //            List<int> temp = InsertionSort(buckets[i]);
130  //            sortedArray.AddRange(temp);
131  //        }
132  //        return sortedArray;
133  //    }
134
135  //    //Insertion Sort
136  //    public static List<int> InsertionSort(List<int> input)
137  //    {
138  //        for (int i = 1; i < input.Count; i++)
139  //        {
140  //            //2. Store the current value in a variable
141  //            int currentValue = input[i];
142  //            int pointer = i - 1;
143
144  //            //3. As long as we are pointing to a valid value in the array...
145  //            while (pointer >= 0)
146  //            {
147  //                //4. If the current value is less than the value we are pointing
148  //                ↪ at...
149  //                if (currentValue < input[pointer])
150  //                {
151  //                    //5. Move the pointed-at value up one space, and insert the
152  //                    ↪ current value at the pointed-at position.
153  //                    input[pointer + 1] = input[pointer];
154  //                    input[pointer] = currentValue;
155  //                }
156  //            }
157  //            else break;
```

```
154 //      }
155 //      }
156
157 //      return input;
158 //  }
159 //  static void Main(string[] args)
160 //  {
161 //      int[] array = new int[] { 43, 17, 87, 92, 31, 6, 96, 13, 66, 62, 4 };
162
163 //      Console.WriteLine("Bucket Sort");
164
165 //      CommonFunctions.PrintInitial(array);
166
167 //      List<int> sorted = Sort(array);
168
169 //      CommonFunctions.PrintFinal(sorted);
170 //      Console.ReadLine();
171 //  }
172 //}
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace task3_4
6  {
7      public class Account
8      {
9          //instance variables declared
10         private decimal _balance;
11         private string _name;
12
13         //read only properties
14         public decimal Balance { get => _balance; }
15         public string Name { get => _name; }
16
17         //constructor
18         public Account(decimal balance, string name)
19         {
20
21             _name = name;
22             if (balance <= 0)
23                 return;
24             _balance = balance;
25
26         }
27
28         //prints name and balance
29         public void Print()
30         {
31             Console.WriteLine("The name of the account holder is " + getName() +
32                 ↪ "\nCurrent Account Balance is " + getBalance());
33         }
34
35         //returns name
36         public String getName()
37         {
38             return this._name;
39         }
40
41         //returns balance
42         public decimal getBalance()
43         {
44             return this._balance;
45         }
46
47         //increases balance by adding deposit
48         //boolean ensure that no 0 or a negative value can not be input
49         public Boolean Deposit(decimal amount)
50         {
51             if (amount <= 0)
52                 return false;
```

```
53
54     this._balance += amount;
55     //Console.WriteLine("Credit added successfully. New balance for " +
56     → this._name + " is " + this._balance);
57     return true;
58 }
59
60
61 //decreases balance by withdrawing the giving input amount
62 //boolean values protect against overdrawing from account
63 public Boolean Withdraw(decimal amount)
64 {
65     if (amount > this._balance || amount < 0)
66         return false;
67
68
69     this._balance -= amount;
70     //Console.WriteLine("Withdrawal successfull. New balance for " +
71     → this._name + " is " + this._balance);
72     return true;
73 }
74
75 }
76 }
```

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	<pre> 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(); } }</pre>
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 identifying 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 'IndexOutOfRangeException.' 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 unexpectedly. 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().

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace ExceptionHandling
6  {
7
8      class Account
9      {
10         public string FirstName { get; private set; }
11         public string LastName { get; private set; }
12         public int Balance { get; private set; }
13
14         public Account(string firstName, string lastName, int balance)
15         {
16             FirstName = firstName;
17             LastName = lastName;
18             Balance = balance;
19         }
20
21         public void Withdraw(int amount)
22         {
23             if (amount > Balance)
24             {
25                 throw new InvalidOperationException("Insufficient fund");
26             }
27             Balance = Balance - amount;
28         }
29
30
31     }
32
33
34
35
36     class Program
37     {
38
39
40         public static void Main()
41         {
42             try
43             {
44                 Account account = new Account("Sergey", "P", 100);
45                 account.Withdraw(1000);
46             }
47             catch (InvalidOperationException exception)
48             {
49                 Console.WriteLine("The following error detected: " +
50                     ↵ exception.GetType().ToString() + " with message \"" +
51                     ↵ exception.Message + "\"");
52             }
53         }
54     }
55 }
```

```
52         Console.ReadKey();
53
54         //Call methods
55         Person();
56         GetNumber(-1);
57         OutofMemory();
58         InvalidCast();
59         Recursive(5);
60         DividebyZero();
61         Account("Mary", "Jones");
62         ArgumentOutOfRangeException();
63         //System_Exception();
64
65
66         //NullReferenceException
67         static void Person()
68         {
69             string name = "Mary";
70             string address = null;
71             if (name != null)
72             {
73                 name = "Lucy";
74                 Console.WriteLine(name);
75
76                 if (address != null)
77                 {
78                     address = "Melbourne";
79                     Console.WriteLine(address);
80                 }
81             }
82         }
83
84     }
85
86     //IndexOutOfRangeException
87     static int GetNumber(int index)
88     {
89         int[] numbers = { 1, 4, 6, 7, 10, 13, 14, 15, 15, 19 };
90         if (index < 0 || index >= numbers.Length)
91         {
92             throw new IndexOutOfRangeException();
93         }
94         return numbers[index];
95     }
96
97
98
99     //StackoverflowException
100    static void Recursive(int number)
101    {
102        //for loop to avoid exception
103        while (number <= 10)
104        {
```

```
105         Console.WriteLine(number);
106         Recursive(++number);
107     }
108 }
109
110
111
112 //OutOfMemoryException
113 static void OutOfMemory()
114 {
115     try
116     {
117         var list = new List<string>();
118         while (true)
119         {
120             list.Add(Guid.NewGuid().ToString());
121         }
122     }
123     catch (OutOfMemoryException e)
124     {
125         Environment.FailFast(String.Format($"Out of memory:
126             ↳ {e.Message}"));
127     }
128 }
129
130
131 //InvalidCastException
132 static void InvalidCast()
133 {
134     bool val = true;
135
136
137     try
138     {
139
140         //IConvertible convt = val;
141         Char ch = Convert.ToChar(val);
142         Console.WriteLine("Conversion succeeded.");
143     }
144     catch (InvalidCastException)
145     {
146         Console.WriteLine("Cannot convert a Boolean to a Char.");
147     }
148 }
149
150
151
152 //DividebyZero Exception
153 static void DividebyZero()
154 {
155     int number_1 = 50;
156     int number_2 = 0;
```

```
157         try
158         {
159             Console.WriteLine(number_1 / number_2);
160         }
161         catch (DivideByZeroException)
162         {
163             Console.WriteLine("Division of {0} by zero.", number_1);
164         }
165     }
166 }
167
168
169 //ArgumentException
170 static void Account(string first_Name, string last_Name)
171 {
172     if (string.IsNullOrEmpty(first_Name))
173         throw new ArgumentException("First_Name is invalid");
174     if (string.IsNullOrEmpty(last_Name))
175         throw new ArgumentException("Last_Name is invalid");
176 }
177
178
179 //ArgumentOutOfRangeException
180 static void ArgumentOutOfRangeException()
181 {
182     var list = new List<String>();
183     Console.WriteLine("Number of items: {0}", list.Count);
184
185     if (list.Count > 0)
186     {
187
188         try
189         {
190             Console.WriteLine("The first item: '{0}'", list[0]);
191         }
192         catch (ArgumentOutOfRangeException e)
193         {
194             Console.WriteLine(e.Message);
195         }
196     }
197 }
198
199
200
201 //SystemException
202 static int System_Exception()
203 {
204     int x = 0;
205     try
206     {
207         int y = 10 / x;
208     }
209     catch (ArithmeticException e)
```

```
210         {
211             Console.WriteLine("ArithmeticException Handler: {0}",
                ↪ e.ToString());
212         }
213         catch (Exception e)
214         {
215             Console.WriteLine("Generic Exception Handler: {0}",
                ↪ e.ToString());
216         }
217
218         return 0;
219     }
220
221     System_Exception();
222
223
224
225     }
226
227     }
228
229     }
230 }
231
232
233
234
235
236
237
238
239
```

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




```
1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4
5  namespace DuplicateCode
6  {
7      class DuplicateCode
8      {
9
10         static int MaximumLength(Dictionary<string, List<string>> tasks)
11         {
12             return tasks.Values.Max(list => list.Count);
13         }
14         static void PrintTasks(Dictionary<string, List<string>> tasks)
15         {
16             int max = MaximumLength(tasks);
17             Console.ForegroundColor = ConsoleColor.Blue;
18             Console.WriteLine(new string(' ', 12) + "CATEGORIES");
19             Console.WriteLine(new string(' ', 10) + new string('-', 94));
20             Console.Write("{0,10}", "item #");
21             foreach (var category in tasks.Keys)
22             {
23                 Console.WriteLine("{0,10}", category);
24             }
25             Console.WriteLine();
26             Console.WriteLine(new string(' ', 10) + new string('-', 94));
27             for (int i = 0; i < max; i++)
28             {
29                 Console.Write("{0,10}|", i + 1);
30                 foreach (var list in tasks.Values)
31                 {
32                     if (list.Count > i)
33                         Console.Write("{0,30}|", list[i]);
34                     else
35                         Console.Write("{0,30}|", "N/A");
36                 }
37                 Console.WriteLine();
38             }
39             Console.ResetColor();
40
41         }
42         static void Main(string[] args)
43         {
44             var tasks = new Dictionary<string, List<string>>();
45             tasks["Personal"] = new List<string>();
46             tasks["Family"] = new List<string>();
47             tasks["Work"] = new List<string>();
48
49             string task;
50             string category;
51
52             while (true)
53             {
```

```
54         Console.Clear();
55         PrintTasks(tasks);
56
57         Console.WriteLine("\nWhich category do you want to place a new
    ↪ task? Type \'Personal\', \'Work\', or \'Family\');
58         category = Console.ReadLine().ToLower();
59         if (category.ToLower() == "quit")
60             break;
61
62         Console.WriteLine("Describe your task below (max. 30 symbols).");
63         task = Console.ReadLine();
64         if (task.Length > 30)
65         {
66             task.Substring(0, 30);
67         }
68         try
69         {
70             tasks[category].Add(task);
71         }
72         catch (ArgumentException)
73         {
74             continue;
75         }
76     }
77 }
78 }
79
80 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Threading.Tasks;
4
5
6  namespace buggysoft_program
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12             List<string> categories = new List<string> { "Personal", "Work",
13                 ↪ "Family" };
14             TaskPlanner taskPlanner = new TaskPlanner(categories);
15             TaskPlannerMenu Menu = new TaskPlannerMenu(taskPlanner);
16
17             Menu.MainMenu();
18         }
19
20
21         class Task
22         {
23             private string task { get; set; }
24             private DateTime date = new DateTime { };
25             private ConsoleColor color = ColorSelector.Default;
26
27
28             public Task(string task) => this.task = task;
29             public DateTime GetDate() => date;
30             public ConsoleColor GetColor() => color;
31             override public string ToString() => task;
32
33
34             public void SetTask(string task) => this.task = task;
35             public void SetDate(DateTime date) => this.date = date;
36             public void SetColor(ConsoleColor color) => this.color = color;
37
38         }
39
40         class TasksList
41         {
42             private List<Task> tasks = new List<Task> { };
43
44             public TaskList() { }
45
46             public TaskList(List<string> tasks)
47             {
48                 foreach (string task in tasks)
49                 {
50                     this.tasks.Add(new Task(task));
51                 }
52             }
53         }
54     }
55 }
```

```
53
54     public TaskList(string task) => tasks.Add(new Task(task));
55
56     public int Count => tasks.Count;
57
58     public bool InDomain(int i) => 0 <= i && i < Count ? true : false;
59
60     public Task GetTask(int i) => InDomain(i) ? tasks[i] : throw new
        ↳ IndexOutOfRangeException("Index out of range");
61
62     //end asklist
63
64     public int GetIndexOfTask(string task)
65     {
66         for (int i = 0; i < Count; i++)
67         {
68             if (tasks[i].ToString().ToUpper() == task.ToUpper())
69             {
70                 return i;
71             }
72         }
73         return -1;
74     }
75
76     public void AddTask(Task task) => tasks.Add(task);
77     public void DeleteTask(string task) => DeleteTask(GetIndexOfTask(task));
78
79     public bool DeleteTask(int i)
80     {
81         if (InDomain(i))
82         {
83             tasks.RemoveAt(i);
84             return true;
85         }
86         return false;
87     }
88
89     public void UpdatePriority(string task, int priority) =>
        ↳ UpdatePriority(GetIndexOfTask(task), priority);
90
91     public bool UpdatePriority(int i, int priority)
92     {
93         if(i!=priority&&InDomain(i)&& InDomain(priority))
94         {
95             tasks.Insert(priority, GetTask(i));
96             DeleteTask(i + 1);
97             return true;
98         }
99         return false;
100     }
101
102 }
103
```

```
104     class TaskPlanner
105     {
106         private List<string> Categories { get; set; }
107         private List<TasksList> TasksList = new List<TasksList> { };
108
109         public TaskPlanner(List<string> categories)
110         {
111             Categories = categories;
112             for (int i = 0; i < Categories.Count; i++)
113             {
114                 TasksList.Add(new TasksList());
115             }
116         }
117
118
119
120         public bool InDomain(int i) => 0 <= i && i < Categories.Count ? true :
121             ↪ false;
122
123         public bool TaskListDomain(string category, int i) =>
124             ↪ GetIndexOfCategory(category) != -1 ?
125             ↪ TasksList[GetIndexOfCategory(category)].InDomain(i) : false;
126
127         private int GetIndexOfCategory(string category)
128         {
129             for (int i = 0; i < Categories.Count; i++)
130             {
131                 if (category.ToUpper() == Categories[i].ToUpper())
132                 {
133                     return i;
134                 }
135             }
136
137             return -1;
138         }
139
140         public bool CategoryExists(string category)
141         {
142             if (GetIndexOfCategory(category) != -1) return true;
143             return false;
144         }
145
146         public bool AddCategory(string category)
147         {
148             if (CategoryExists(category))
149             {
150                 Categories.Add(category);
151                 TasksList.Add(new TasksList());
152                 return true;
153             }
154             return false;
155         }
156     }
```

```
154     public bool DeleteCategory(string category)
155     {
156         int index = GetIndexOfCategory(category);
157         if (index != -1)
158         {
159             Categories.RemoveAt(index);
160             TasksList.RemoveAt(index);
161             return true;
162         }
163         return false;
164     }
165
166     public int GetIndexOfTask(string category, string task)
167     {
168         int index = GetIndexOfCategory(category);
169         if (index != -1)
170         {
171             for (int i = 0; i < TasksList[index].Count; i++)
172             {
173
174                 if (task.ToUpper() ==
175                     ↪ TasksList[index].GetTask(i).ToString().ToUpper())
176                 {
177                     return i;
178                 }
179             }
180             return -1;
181         }
182
183         public Task GetTask(string category, string task) => GetTask(category,
184             ↪ GetIndexOfTask(category, task));
185
186         public Task GetTask(string category, int i)
187         {
188             int index = GetIndexOfCategory(category);
189             if(index!=-1)
190             {
191                 if(TasksList[index].InDomain(i))
192                 {
193                     return TasksList[index].GetTask(i);
194                 }
195             }
196             throw new IndexOutOfRangeException("Error");
197         }
198
199         public bool AddTask(string category, string task) => AddTask(category,
200             ↪ new Task(task));
201
202         public bool AddTask(string category, Task task)
203         {
204             int index = GetIndexOfCategory(category);
205             if (index != -1)
```

```
204         {
205             TasksList[index].AddTask(task);
206             return true;
207         }
208         return false;
209     }
210
211     public bool DeleteTask(string category, int i)
212     {
213         int index = GetIndexOfCategory(category);
214         if(index!=-1)
215         {
216             if(TasksList[index].InDomain(i))
217             {
218                 TasksList[index].DeleteTask(i);
219                 return true;
220             }
221         }
222         return false;
223     }
224
225     public bool MoveTask(string category1, string category2, string task)
226     {
227         int index = GetIndexOfCategory(category1);
228         if(index!=-1)
229         {
230             index = TasksList[index].GetIndexOfTask(task);
231             if(index!=-1)
232             {
233                 MoveTask(category1, category2, index);
234                 return true;
235             }
236         }
237         return false;
238     }
239
240     public bool updatePriority(string category, int i, int priority)
241     {
242         if(i!=priority)
243         {
244             int index = GetIndexOfCategory(category);
245             if(index!=-1)
246             {
247                 if(TasksList[index].InDomain(i)&&
248                    ↪ TasksList[index].InDomain(priority))
249                 {
250                     TasksList[index].UpdatePriority(i, priority);
251                     return true;
252                 }
253             }
254         }
255     }
```

```
256         return false;
257     }
258
259     public bool MoveTask(string category1, string category2, int i)
260     {
261         int index1 = GetIndexOfCategory(category1);
262         int index2 = GetIndexOfCategory(category2);
263         if(index1!=-1 && index2!=-1)
264         {
265             if(TasksList[index1].InDomain(i))
266             {
267                 AddTask(category2, TasksList[index1].GetTask(i));
268                 DeleteTask(category1, i);
269                 return true;
270             }
271         }
272         return false;
273     }
274
275
276
277     public void WriteTaskPlanner()
278     {
279         int i;
280         Console.ForegroundColor = ColorSelector.Default;
281         Console.WriteLine(new string(' ', 5)+"CATEGORIES");
282         Console.WriteLine(new string(' ', 5) + new string('-',
283             ↳ Categories.Count*42+5));
284         Console.Write("{0,5}|" , "item #");
285
286         for(i=0;i<Categories.Count;i++)
287         {
288             Console.WriteLine("{0,30}+{1,10}", Categories[i], "Due Date");
289         }
290         Console.WriteLine();
291         Console.WriteLine(new string(' ', 5) + new string('-',
292             ↳ Categories.Count * 42 + 5));
293
294         int max = 0;
295         foreach (TasksList tasks in TasksList)
296         {
297             max = max > tasks.Count ? max : tasks.Count;
298         }
299
300         foreach(TasksList tasks in TasksList)
301         {
302             if(tasks.Count>i)
303             {
304                 Task task = tasks.GetTask(i);
305                 Console.ForegroundColor = task.GetColor();
306                 Console.Write("{0,30} +", task.ToString());
307                 if(task.GetDate().Date.Year!=1)
308                 {
```



```
307         Console.Write("{ 0,10}|",
    ↪     $"{{task.GetDate().Date.Day}}/{{task.GetDate().Date.
    ↪     ate.Month}}/{{task.GetDate().Date.Year}}");
308     }
309     else
310     {
311         Console.Write("{ 0,10}|", "N/S");
312     }
313     Console.ForegroundColor = ColorSelector.Default;
314 }
315 else
316 {
317     Console.Write("{ 0,42}|", "N/A");
318 }
319 Console.WriteLine();
320 }
321 Console.WriteLine("\n");
322 Console.ResetColor();
323 }
324 }
325
326
327 enum MENUOPTIONS
328 {
329     ADD_CATEGORY,
330     DELETE_CATEGORY,
331     ADD_TASK,
332     SELECT_TASK,
333     MOVE_TASK,
334     UPDATE_TASK,
335     DELETE_TASK,
336     CHANGE_TASK_PRIORITY,
337     CHANGE_TASK_IMPORTANCE,
338     SET_DATE,
339     CANCEL,
340     QUIT
341 };
342
343 enum TASKSELECTION
344 {
345     BY_INDEX,
346     BY_NAME,
347     CANCEL
348 };
349
350 enum TASKPRIORITY
351 {
352     DEFAULT,
353     IMPORTANT
354 };
355
356 public class ColorSelector
357 {
```

```
358
359     }
360
361     class TaskPlannerMenu
362     {
363         private TaskPlanner taskPlanner { get; set; }
364
365         public TaskPlannerMenu Menu(TaskPlannerMenu taskplanner) =>
366             ↪ this.taskPlanner = taskPlanner;
367
368         private T ReadMenuInput<T>(T[] opts) where T : System.Enum
369         {
370             int input;
371             Console.Clear();
372             taskPlanner.WriteTaskPlanner();
373             do
374             {
375                 Console.WriteLine("Select from the following options");
376                 for (int i = 0; i < opts.Length; i++)
377                 {
378                     Console.WriteLine($"(i + 1));(opts[i]");
379                 }
380                 Console.Write(">>>");
381
382                 try
383                 {
384                     input = Convert.ToInt32(Console.ReadLine()) - 1;
385                     if (0 <= input && input < opts.Length)
386                     {
387                         Console.WriteLine("Input outside of domain");
388                     }
389                 }
390                 catch (FormatException)
391                 {
392                     Console.WriteLine("Invalid entry");
393                     ReadMenuInput(opts);
394                 }
395             } while (true);
396         }
397
398         public void MainMenu()
399         {
400             MENUOPTIONS input;
401             MENUOPTIONS[] opts = { MENUOPTIONS.ADD_CATEGORY,
402                 ↪ MENUOPTIONS.DELETE_CATEGORY, MENUOPTIONS.ADD_TASK,
403                 ↪ MENUOPTIONS.SELECT_TASK, MENUOPTIONS.QUIT }
404
405             do
406             {
407                 input = ReadMenuInput(opts);
408                 switch (input)
409                 {
```

```
408         case MENUOPTIONS.ADD_CATEGORY: AddCategory(); break;
409         case MENUOPTIONS.DELETE_CATEGORY: DeleteCategory(); break;
410         case MENUOPTIONS.ADD_TASK: AddTask(); break;
411         case MENUOPTIONS.SELECT_TASK: SelectTaskMenu(); break;
412         case MENUOPTIONS.QUIT: return;
413     }
414
415     } while (true);
416
417 }
418
419 private string ReadString(string msg)
420 {
421     Console.Clear();
422     taskPlanner.WriteTaskPlanner();
423     Console.WriteLine(msg);
424     return Console.ReadLine();
425 }
426
427 private int ReadInt(string msg)
428 {
429     try
430     {
431         return Convert.ToInt32(ReadString(msg));
432     }
433     catch (FormatException)
434     {
435         return ReadInt(msg);
436     }
437 }
438
439 private void AddCategory()
440 {
441     string input = ReadString("Enter the category to add: ");
442     if (!taskPlanner.AddCategory(input))
443     {
444         Console.WriteLine("Category not added as it already exists");
445         Console.ReadLine();
446     }
447 }
448
449
450 private void DeleteCategory()
451 {
452     string input = ReadString("Enter the category to delete: ");
453     if (!taskPlanner.DeleteCategory(input))
454     {
455         Console.WriteLine("Category not deleted as it already exists");
456         Console.ReadLine();
457     }
458 }
459
460 }
```

```
461     private string SelectCategory()
462     {
463         string input = ReadString("Select Category: ");
464         if(taskPlanner.CategoryExists(input))
465         {
466             Console.WriteLine();
467             Console.WriteLine("Category does not exist");
468             Console.WriteLine();
469             throw new IndexOutOfRangeException("Category does not exist");
470         }
471         return input;
472     }
473
474     private void AddTask()
475     {
476         string category;
477         try
478         {
479             category = SelectCategory();
480         }
481         catch(IndexOutOfRangeException)
482         {
483             return;
484         }
485         string task = ReadString("What is the name of the task");
486         taskPlanner.AddTask(category, task);
487     }
488
489     private void SelectTaskMenu()
490     {
491         string category;
492         try
493         {
494             category = SelectCategory();
495         }
496         catch(IndexOutOfRangeException)
497         {
498             return;
499         }
500         TASKSELECTION input;
501         TASKSELECTION[] opts = { TASKSELECTION.BY_INDEX,
502             ↳ TASKSELECTION.BY_NAME, TASKSELECTION.CANCEL };
503         input = ReadMenuInput(opts);
504         switch(input)
505         {
506             case TASKSELECTION.BY_INDEX: SelectedTaskMenu(category,
507                 ↳ ReadInt("Enter ndex number: "); break;
508             case TASKSELECTION.BY_NAME: SelectedTaskMenu(category,
509                 ↳ Convert.ToInt32(ReadString("Enter the name: ")); break;
510             case TASKSELECTION.CANCEL: return;
511         }
512     }
```

```
//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;
```

```
    }
```

```
}
```

```
557     private ConsoleColor MoveTask(string category, int i, ConsoleColor
        ↳ color)
558     {
559         string category2 = ReadString("Enter category to move");
560         taskPlanner.GetTask(category, i).SetColor(color);
561         if(taskPlanner.MoveTask(category, category2, i))
562         {
563
564             if (taskPlanner.TaskListDomain(category, i))
565             {
566                 return taskPlanner.GetTask(category, i).GetColor();
567             }
568
569         }
570         return color;
571     }
572
573     private ConsoleColor ChangeTaskImportanceMenu()
574     {
575         TASKPRIORITY input;
576         TASKPRIORITY[] opts = { TASKPRIORITY.DEFAULT,
        ↳ TASKPRIORITY.IMPORTANT };
577
578         input = ReadMenuInput(opts);
579         switch(input)
580         {
581             default: case TASKPRIORITY.DEFAULT: return
        ↳ ColorSelector.Default;
582             case TASKPRIORITY.IMPORTANT: return ColorSelector.Important;
583
584         }
585     }
586
587     private ConsoleColor ChangeTaskPriority(string category, int i,
        ↳ ConsoleColor color)
588     {
589         int j = ReadInt("Enter priority [index]");
590         if(i!=j&&taskPlanner.TaskListDomain(category, j))
591         {
592             ConsoleColor color2 = taskPlanner.GetTask(category,
        ↳ j).GetColor();
593             taskPlanner.updatePriority(category, i, j);
594             taskPlanner.GetTask(category, j).SetColor(color);
595             return color2;
596         }
597
598         return color;
599     }
600
601
602     private void SetDate(string category, int i)
603     {
604         int day, month, year;
```

```
605         year = ReadInt("Enter the year: ");
606         if (i > year || year > 9999) return;
607
608         month = ReadInt("Enter the month: ");
609         if (i > month || month > 12) return;
610
611         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 };
612         if (i > day || day > daysDomain[month-1]) return;
613
614         taskPlanner.GetTask(category, i).SetDate(new DateTime(year, month,
        ↪ day));
615
616     }
617 }
618
619 }
620
621 }
```

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



```
1  using System;
2
3  namespace task5_1
4  {
5      class ZooPark
6      {
7          static void Main(string[] args)
8          {
9              Animal williamWolf = new Animal("William ths Wolf", "Meat", "Dog
10                 ↪ Villiage", 50.6, 9, "Grey");
11              Animal tonyTiger = new Animal("Tony the Tiger", "Meat", "Cat Land",
12                 ↪ 110, 6, "Orange and White");
13              Animal edgarEadle = new Animal("Edgar the Eagle", "Fish", "Bird Mania",
14                 ↪ 20, 15, "Black");
15          }
16      }
17  }
```

```
1  using System;
2  namespace task5_1
3  {
4      public class Animal
5      {
6          class animal
7          {
8              private String name;
9              private string diet;
10             private String location;
11             private double weight;
12             private int age;
13             private String colour;
14         }
15
16
17         public Animal(String name, String diet, String location, double weight, int
            ↪ age, String colour)
18         {
19             this.name = name;
20             this.diet = diet;
21             this.location = location;
22             this.weight = weight;
23             this.age = age;
24             this.colour = colour;
25         }
26
27         public void eat()
28         {
29             //
30         }
31
32         public void sleep()
33         {
34             //
35         }
36
37         public void MakeNoise()
38         {
39             //
40         }
41
42         public void makeLionNoise()
43         {
44             //
45         }
46
47         public void makeEagleNoise()
48
49         {
50             //
51         }
52
```

```
53     public void makeWolfNoise()  
54     {  
55         //  
56     }  
57  
58 }  
59 }
```

```
1  using System;
2  namespace Task02
3  {
4      public class Animal
5      {
6
7          private String name;
8          private string diet;
9          private String location;
10         private double weight;
11         private int age;
12         private String colour;
13
14
15         public Animal(String name, String diet, String location, double weight,
16             ↪ int age, String colour)
17         {
18             this.name = name;
19             this.diet = diet;
20             this.location = location;
21             this.weight = weight;
22             this.age = age;
23             this.colour = colour;
24         }
25
26         public virtual void eat()
27         {
28             Console.WriteLine("An animal eats");
29         }
30
31         public virtual void lives()
32         {
33             Console.WriteLine("An animal lives");
34         }
35
36         public virtual void sleep()
37         {
38             Console.WriteLine("an animal sleeps");
39         }
40
41         public virtual void MakeNoise()
42         {
43             Console.WriteLine("An animal makes a noise");
44         }
45
46         public void makeLionNoise()
47         {
48             //code for Lion noise
49         }
50
51         public void makeEagleNoise()
52         {
```

```
53         //code for Eagle noise
54     }
55
56     public void makeWolfNoise()
57     {
58         //code for Wolf noise
59     }
60
61 }
62 }
```

```

1  using System;
2
3  namespace Task02
4  {
5      class ZooPark
6      {
7          static void Main(string[] args)
8          {
9              //Animal williamWolf = new Animal("William ths Wolf", "Meat", "Dog
              ↳ Villiage", 50.6, 9, "Grey");
10             //Animal tonyTiger = new Animal("Tony the Tiger", "Meat", "Cat Land",
              ↳ 110, 6, "Orange and White");
11             //Animal edgarEagle = new Animal("Edgar the Eagle", "Fish", "Bird
              ↳ Mania", 20, 15, "Black");
12
13             //Tiger tonytiger = new Tiger();
14
15             Tiger tonyTiger = new Tiger("Tony the Tigger", "Meat", "Cat Land", 110,
              ↳ 6, "Orange and White", "Siberian", "White");
16             Wolf williamWolf = new Wolf("William the Wolf", "Meat", "Dog Village",
              ↳ 50.6, 9, "Grey");
17             Eagle edgarEagle = new Eagle("Edgar the Eagle", "Fish", "Bird Mania",
              ↳ 20, 15, "Black", "Harpy", 98.5);
18             Penguin Penny = new Penguin("Penny the Penguin", "Fish", "Snow
              ↳ Mountain", 10, 5, "White", "King", 44.5);
19             Lion Leo = new Lion("Leo the Lion", "Gazelle", "Serengeti", 70, 65,
              ↳ "White", "P.I");
20             //tonyTiger.MakeNoise();
21
22             Animal baseAnimal = new Animal("Animal Name", "Animal Diet", "Animal
              ↳ Location", 0.0, 0, "Animal Colour");
23             //baseAnimal.MakeNoise();
24             //Console.ReadLine();
25
26             Bird baseBird = new Bird("Bird Name", "Bird Diet", "Bird Location",
              ↳ 0.0, 0, "Bird Colour", "Bird Species", 0.0);
27
28             //baseAnimal.sleep();
29
30             Console.WriteLine("-----TonyTiger-----");
              ↳ -----");
31             Console.Write("TonyTiger says ");
32             tonyTiger.sleep();
33             tonyTiger.eat();
34
35             Console.WriteLine("-----WilliamWolf-----");
              ↳ -----");
36             Console.Write("williamWolf says ");
37             williamWolf.sleep();
38             williamWolf.eat();
39
40             Console.WriteLine("-----Leo the
              ↳ Lion-----");

```

```
41         Console.WriteLine("Leo the Lion says ");
42         Leo.sleep();
43         Leo.eat();
44
45         Console.WriteLine("-----edgarEagle-----
↳ -----");
46         Console.WriteLine("edgarEagle says ");
47         edgarEagle.sleep();
48         edgarEagle.eat();
49         edgarEagle.fur();
50         edgarEagle.layEgg();
51         edgarEagle.swim();
52         edgarEagle.size();
53         edgarEagle.feet();
54         edgarEagle.weather();
55
56
57         Console.WriteLine("-----Penny the
↳ Penguin-----");
58         Console.WriteLine("Penny the Penguin says ");
59         Penny.sleep();
60         Penny.eat();
61         Penny.fur();
62         Penny.layEgg();
63         Penny.swim();
64         Penny.size();
65         Penny.feet();
66         Penny.weather();
67
68
69
70
71
72
73
74     }
75
76 }
77 }
```



```
1  using System;
2  namespace Task02
3
4  {
5      class Tiger : Feline
6      {
7          private String colourStripes;
8
9          public Tiger(String name, String diet, String location, double weight, int
            ↪ age, String colour, String species,
10             String colourStripes) : base(name, diet, location, weight, age, colour,
            ↪ species)
11      {
12          this.colourStripes = colourStripes;
13      }
14
15      public override void MakeNoise()
16      {
17          Console.WriteLine("ROARRRRRRRRRRRR");
18      }
19
20      public override void eat()
21      {
22          Console.WriteLine("I can eat 20lbs of meat");
23      }
24
25      public override void lives()
26      {
27          Console.WriteLine("I can live 10 years");
28      }
29
30  }
31 }
```

```
1  using System;
2  namespace Task02
3  {
4      class Eagle : Bird
5      {
6
7
8          public Eagle(String name, String diet, String location, double weight, int
             ↪ age, String colour, String species, double wingSpan)
9              : base(name, diet, location, weight, age, colour, species, wingSpan)
10             {
11
12             }
13
14             public override void MakeNoise()
15             {
16                 Console.WriteLine("WHISTLEEEESSSS");
17             }
18             public override void lives()
19             {
20                 Console.WriteLine("I can live 20 years");
21             }
22
23             public override void layEgg()
24             {
25                 Console.WriteLine("I lay eggs");
26             }
27
28             public override void fly()
29             {
30                 Console.WriteLine("I can fly");
31             }
32
33             public override void weather()
34             {
35                 Console.WriteLine("I do not like the cold");
36             }
37
38             public override void fur()
39             {
40                 Console.WriteLine("I have rough feathers");
41             }
42
43             public override void beak()
44             {
45                 Console.WriteLine("I have a large beak");
46             }
47
48             public override void swim()
49             {
50                 Console.WriteLine("I can't swim");
51             }
52
```

```
53     public override void size()
54     {
55         Console.WriteLine("I am large");
56     }
57
58     public override void feet()
59     {
60         Console.WriteLine("I have claws");
61     }
62 }
63 }
```

```
1  using System;
2  namespace Task02
3  {
4
5      class Wolf : Animal
6      {
7
8          public Wolf(String name, String diet, String location, double weight, int
            ↪ age, String colour) : base(name, diet, location, weight, age, colour)
9          {
10
11          }
12
13          public override void MakeNoise()
14          {
15              Console.WriteLine("HOWLLLLLLS");
16          }
17
18          public override void lives()
19          {
20              Console.WriteLine("I can live 8 years");
21          }
22      }
23  }
24
25
26 }
```

```
1  using System;
2  namespace Task02
3  {
4      class Feline:Animal
5      {
6          private String species;
7
8          public Feline(String name, String diet, String location, double weight, int
            ↪ age, String colour, String species):
9              base(name, diet, location, weight, age, colour)
10         {
11             this.species = species;
12         }
13
14         public override void sleep()
15         {
16             Console.WriteLine("I can sleep 9 hours a night");
17         }
18     }
19 }
```

```
1  using System;
2  namespace Task02
3  {
4      class Lion : Feline
5      {
6          public Lion(String name, String diet, String location, double weight, int
            ↳ age, String colour, String species)
7              : base(name, diet, location, weight, age, colour, species)
8          {
9
10         }
11     }
12 }
```

```
1  using System;
2  namespace Task02
3  {
4      class Bird:Animal
5      {
6          private String species;
7          private double wingSpan;
8
9          public Bird(String name, String diet, String location, double weight, int
            ↪ age, String colour, String species, double wingSpan)
10         : base(name, diet, location, weight, age, colour)
11         {
12             this.species = species;
13             this.wingSpan = wingSpan;
14         }
15
16
17         public virtual void layEgg()
18         {
19             Console.WriteLine("A bird may lay eggs");
20         }
21
22         public virtual void fly()
23         {
24             Console.WriteLine("A bird flies");
25         }
26
27         public virtual void weather()
28         {
29             Console.WriteLine("A bird likes cold weather");
30         }
31
32         public virtual void fur()
33         {
34             Console.WriteLine("A bird has fur");
35         }
36
37         public virtual void beak()
38         {
39             Console.WriteLine("A bird has a beak");
40         }
41
42         public virtual void swim()
43         {
44             Console.WriteLine("A bird might swim");
45         }
46
47         public virtual void size()
48         {
49             Console.WriteLine("A bird has a size");
50         }
51
52         public virtual void feet()
```

```
53     {  
54         Console.WriteLine("A bird may have webbed feet or claws");  
55     }  
56  
57  
58  
59 }  
60  
61 }
```



```
1  using System;
2  namespace Task02
3  {
4      class Penguin : Bird
5      {
6
7
8          public Penguin(String name, String diet, String location, double
              ↳ weight, int age, String colour, String species, double wingSpan)
9              : base(name, diet, location, weight, age, colour, species, wingSpan)
10         {
11
12         }
13
14         public override void layEgg()
15         {
16             Console.WriteLine("I lay eggs");
17         }
18
19         public override void fly()
20         {
21             Console.WriteLine("I can not fly");
22         }
23
24         public override void weather()
25         {
26             Console.WriteLine("I like the cold");
27         }
28
29         public override void fur()
30         {
31             Console.WriteLine("I have soft fur");
32         }
33
34         public override void beak()
35         {
36             Console.WriteLine("I have a small beak");
37         }
38
39         public override void swim()
40         {
41             Console.WriteLine("I can swim");
42         }
43
44         public override void size()
45         {
46             Console.WriteLine("I am small");
47         }
48
49         public override void feet()
50         {
51             Console.WriteLine("I have webbed feet");
52         }
53     }
```

```
53
54     }
55 }
```

```
1  using System;
2
3  namespace Overloading
4  {
5      class Program
6      {
7
8          public static void methodToBeOverloaded(string name)
9          {
10             Console.WriteLine("Name: " + name);
11         }
12
13         public static void methodToBeOverloaded(String name, int age)
14         {
15             Console.WriteLine("Name: " + name + "\nAge: " + age);
16         }
17         static void Main(string[] args)
18         {
19
20             methodToBeOverloaded("Lucy");
21             Console.WriteLine("-----");
22             methodToBeOverloaded("Lucy", 20);
23         }
24     }
25 }
```

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



```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task5_2
6  {
7      public class WithdrawTransaction
8      {
9
10         private Account _account;
11         private decimal _amount;
12         private bool _executed;
13         private bool _success;
14         private bool _reversed;
15
16         public bool Executed { get { return _executed; } }
17         public bool Success { get { return _success; } }
18         public bool Reversed { get { return _reversed; } }
19
20         public WithdrawTransaction(Account account, decimal amount)
21         {
22             this._account = account;
23             this._amount = amount;
24             _reversed = false;
25             _executed = false;
26         }
27
28
29         public void Print()
30         {
31             Console.WriteLine($"Successfully withdrew ${this._amount} from the
32             ↪ account: {this._account.getName()}.");
33             Console.WriteLine($"Current balance is {this._account.getBalance()}");
34         }
35
36         public void Execute()
37         {
38             if (Executed==false)
39             {
40                 if(_account.getBalance()>0 && _amount<=_account.getBalance())
41                 {
42                     _account.Withdraw(_amount);
43                     _executed = true;
44                 }
45                 else
46                 {
47                     Console.WriteLine("Insufficient funds");
48                 }
49             }
50
51             else { throw new InvalidOperationException("Withdrawal has already been
52             ↪ performed"); }
```

```
52
53     public void Rollback()
54     {
55         if (Reversed==false && Executed==true)
56         {
57             _account.Deposit(_amount);
58             _executed = false;
59             _reversed = true;
60         }
61         else
62         {
63             Console.WriteLine("Transaction has already been reversed");
64         }
65     }
66
67
68
69 }
70 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task5_2
6  {
7      public class DepositTransaction
8      {
9
10         private Account _account;
11         private decimal _amount;
12         private bool _executed;
13         private bool _success;
14         private bool _reversed;
15
16         public bool Executed { get { return _executed; } }
17         public bool Success { get { return _success; } }
18         public bool Reversed { get { return _reversed; } }
19
20         public DepositTransaction(Account account, decimal amount)
21         {
22             this._account = account;
23             this._amount = amount;
24             _reversed = false;
25             _executed = false;
26         }
27
28         public void print()
29         {
30             Console.WriteLine($"Successfully deposited ${this._amount} from the
31             ↪ account: {this._account.getName()}.");
32             Console.WriteLine($"Current balance is {this._account.getBalance()}");
33         }
34
35         public void Execute()
36         {
37             if (Executed == false)
38             {
39                 _account.Deposit(_amount);
40                 _executed = true;
41                 _reversed = false;
42             }
43             else
44             {
45                 throw new InvalidOperationException("You have already performed
46                 ↪ this transaction");
47             }
48         }
49
50         public void Rollback()
51         {
52             if (Executed == true && Reversed == false)
53             {
54                 _account.Withdraw(_amount);
55                 _executed = false;
56                 _reversed = true;
57             }
58         }
59     }
60 }
```



```
52         _account.Withdraw(_amount);
53         _executed = false;
54         _reversed = true;
55
56     }
57     else
58     {
59         Console.WriteLine("There is no deposit to reverse");
60     }
61
62 }
63
64 }
65 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5
6  namespace task5_2
7  {
8      public class TransferTransaction
9      {
10
11          private Account _fromAccount;
12          private Account _toAccount;
13          private decimal _amount;
14          private DepositTransaction _deposit;
15          private WithdrawTransaction _withdraw;
16          private bool _executed;
17          private bool _reversed;
18          private bool _success;
19
20          public bool Executed { get { return _executed; } }
21          public bool Success { get { return _success; } }
22          public bool Reversed { get { return _reversed; } }
23
24          public TransferTransaction(Account fromAccount, Account toAccount, decimal
25              ↪ amount)
26          {
27              _deposit = new DepositTransaction(toAccount, amount);
28              _withdraw = new WithdrawTransaction(fromAccount, amount);
29              _executed = false;
30              this._fromAccount = fromAccount;
31              this._toAccount = toAccount;
32              this._amount = amount;
33          }
34
35          public bool success
36          {
37              get
38              {
39                  if(_deposit.Success==true && _withdraw.Success==true)
40                  {
41                      return true;
42                  }
43                  else
44                  {
45                      return false;
46                  }
47              }
48          }
49
50          public void print()
51          {
52              if (_executed == true)
```

```
53     {
54         Console.WriteLine("Transaction has been performed successfully");
55         Console.WriteLine($"Transferred ${_amount} from
56             ↳ {_fromAccount.getName()} to {_toAccount.getName()}");
57     }
58     else
59     {
60         Console.WriteLine("Transaction has not been successful");
61     }
62 }
63
64
65 public void Execute()
66 {
67     if(Executed==false)
68     {
69         if(_fromAccount.getBalance()<_amount)
70         {
71             Console.WriteLine("Insufficient funds");
72
73         }
74         else
75         {
76             _withdraw.Execute();
77             _deposit.Execute();
78             _executed = true;
79
80         }
81     }
82     else { throw new InvalidOperationException("Transacation has been
83         ↳ already performed"); }
84
85 public void Rollback()
86 {
87     if(Reversed==true)
88     {
89         throw new InvalidOperationException("Reverse transaction has
90             ↳ already been performed");
91     }
92     if(_toAccount.getBalance()<_amount)
93     {
94         throw new InvalidOperationException("Insufficient funds");
95     }
96     else if(Executed==true)
97     {
98         _deposit.Rollback();
99         _withdraw.Rollback();
100         _reversed = true;
101         _executed = false;
102     }
```

```
103
104     }
105
106     }
107 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading;
5
6  namespace task5_2
7  {
8
9
10     class BankSystem
11     {
12
13         enum MenuOption
14         {
15             Withdraw,
16             Deposit,
17             Print,
18             TransferTransaction,
19             Quit
20         }
21
22         static void DoDeposit(Account account)
23         {
24             Console.WriteLine("Enter deposit amount: ");
25             decimal amount = Convert.ToDecimal(Console.ReadLine());
26             DepositTransaction deposit = new DepositTransaction(account, amount);
27
28             while(account != null)
29             {
30                 string execute;
31                 string rollback;
32
33                 Console.WriteLine($"Deposit the given amount: ${amount}. Y/N? ");
34                 execute = Console.ReadLine();
35                 Console.WriteLine("Rollback the transaction. Y/N? ");
36                 rollback = Console.ReadLine();
37                 if (execute == "Y".ToLower())
38                 {
39                     deposit.Execute();
40                 }
41                 if (rollback == "Y".ToLower())
42                 {
43                     deposit.Rollback();
44                     break;
45                 }
46                 else
47                 {
48                     deposit.print();
49                     break;
50                 }
51             }
52         }
53     }
```

```
54
55     static void DoWithdrawal(Account account)
56     {
57
58         Console.WriteLine("Enter withdrawal amount: ");
59         decimal amount = Convert.ToDecimal(Console.ReadLine());
60         WithdrawTransaction deposit = new WithdrawTransaction(account, amount);
61
62         while (account != null)
63         {
64             string execute;
65             string rollback;
66
67             Console.WriteLine($"Withdraw the given amount: ${amount}. Y/N? ");
68             execute = Console.ReadLine();
69             Console.WriteLine("Rollback the transaction. Y/N? ");
70             rollback = Console.ReadLine();
71             if (execute == "Y".ToLower())
72             {
73                 deposit.Execute();
74             }
75             if (rollback == "Y".ToLower())
76             {
77                 deposit.Rollback();
78                 break;
79             }
80             else
81             {
82                 deposit.Print();
83                 break;
84             }
85         }
86     }
87
88
89     static void DoTransfer(Account fromaccount, Account toaccount, decimal
90     ↪ amount)
91     {
92         ;
93
94         TransferTransaction transferobj = new TransferTransaction(fromaccount,
95         ↪ toaccount, amount);
96
97         while (fromaccount != null)
98         {
99             string execute;
100             string rollback;
101
102             Console.WriteLine($"Transfer the given amount: ${amount}. Y/N? ");
103             execute = Console.ReadLine();
104             Console.WriteLine("Would you like to rollback the transfer. Y/N? ");
105             rollback = Console.ReadLine();
106             if (execute == "Y".ToLower())
107             {
```

```
105         transferobj.Execute();
106     }
107     if (rollback == "Y".ToLower())
108     {
109         transferobj.Rollback();
110         break;
111     }
112     else
113     {
114         transferobj.print();
115         break;
116     }
117 }
118 }
119
120
121 static MenuOption ReadUserOption()
122 {
123     int? option = null;
124     do
125     {
126         Console.ForegroundColor = ConsoleColor.Blue;
127         Console.WriteLine("\n Welcome to the Banking System ");
128         Console.WriteLine("|-----|");
129         Console.WriteLine("\n Bank System Menu ");
130         Console.WriteLine("|-----|");
131         Console.WriteLine("\n 1. Withdraw ");
132         Console.WriteLine("|-----|");
133         Console.WriteLine("\n 2. Deposit ");
134         Console.WriteLine("|-----|");
135         Console.WriteLine("\n 3. Print Account ");
136         Console.WriteLine("|-----|");
137         Console.WriteLine("\n 4. Transfer Transaction ");
138         Console.WriteLine("|-----|");
139         Console.WriteLine("\n 5. Quit ");
140         Console.WriteLine("|-----|");
141
142         try
143         {
144             option = Convert.ToInt32(Console.ReadLine());
145             if (option > 4 || option < 1)
146             {
147                 option = null;
148                 Console.WriteLine("Please enter a number from 1 to 4 from
149                 ↪ the menu");
150             }
151         } catch (FormatException)
152         {
153             Console.WriteLine("Invalid input. Enter an integer from 1-4");
154         }
155     } while (option == null);
156 }
```

```
157
158     return (MenuOption)option;
159 }
160
161
162 static void DoPrint(Account account)
163 {
164
165     account.Print();
166
167 }
168
169
170 static void Main(string[] args)
171 {
172     Console.ForegroundColor = ConsoleColor.Blue;
173     Console.WriteLine(new string(' ', 10) + "BANKING SYSTEM");
174     Console.WriteLine(new string(' ', 10) + new string('-', 82));
175     Console.Write("{0,19}{1,20}{2,20}{3,30} ", "|Account|", "|Action|",
176         ↪ "|Status|", "|Current Balance|");
177
178     Console.WriteLine("\n\nEnter name of user one: ");
179     string name1 = Console.ReadLine();
180     Console.WriteLine("Enter starting balance of user one: ");
181     decimal balance1 = Convert.ToDecimal(Console.ReadLine());
182     Console.WriteLine("Enter name of user two: ");
183     string name2 = Console.ReadLine();
184     Console.WriteLine("Enter starting balance of user two: ");
185     decimal balance2 = Convert.ToDecimal(Console.ReadLine());
186     Account person_one = new Account(balance1, name1);
187     Account person_two = new Account(balance2, name2);
188     MenuOption option;
189
190     do
191     {
192         option = ReadUserOption() - 1;
193
194         switch (option)
195         {
196             case MenuOption.Withdraw:
197                 Console.WriteLine("You have selected withdraw");
198                 Console.WriteLine("|-----|");
199                 Console.WriteLine("        Account holder: " + name1        );
200                 Console.WriteLine("|-----|");
201                 DoWithdrawal(person_one);
202                 Console.WriteLine("|-----|");
203                 Console.WriteLine("        Account holder: " + name2);
204                 Console.WriteLine("|-----|");
205                 DoWithdrawal(person_two);
206                 break;
207             case MenuOption.Deposit:
208                 Console.WriteLine("You have selected deposit");
```



```
209         Console.WriteLine("|-----|");
210         Console.WriteLine("        Account holder: " + name1      );
211         Console.WriteLine("|-----|");
212         DoDeposit(person_one);
213         Console.WriteLine("|-----|");
214         Console.WriteLine("        Account holder: " + name2      );
215         Console.WriteLine("|-----|");
216         DoDeposit(person_two);
217         break;
218     case MenuOption.Print:
219         Console.WriteLine("You have selected print");
220         Console.WriteLine("|-----|");
221         Console.WriteLine("        Account holder: " + name1      );
222         Console.WriteLine("|-----|");
223         DoPrint(person_one);
224         Console.WriteLine("|-----|");
225         Console.WriteLine("        Account holder: " + name2      );
226         Console.WriteLine("|-----|");
227         DoPrint(person_two);
228         break;
229     case MenuOption.TransferTransaction:
230         Console.WriteLine("You have selected transfer");
231         Console.WriteLine("Enter amount to transfer: ");
232         decimal amount = Convert.ToDecimal(Console.ReadLine());
233         Console.WriteLine("Would you like to transfer from User
234             ↪ One's account to User Two's account? Y/N");
235         string input = Console.ReadLine();
236         if (input.ToLower() == "Y".ToLower())
237         {
238             DoTransfer(person_one, person_two, amount);
239             break;
240         }
241         Console.WriteLine("Would you like to transfer from User
242             ↪ Two's account to User One's account? Y/N");
243         string select = Console.ReadLine();
244         if (select.ToLower() == "Y".ToLower())
245         {
246             DoTransfer(person_two, person_one, amount);
247             break;
248         }
249         else
250             break;
251     case MenuOption.Quit:
252         Console.WriteLine("Goodbye");
253         break;
254     }
255     } while (option != MenuOption.Quit);
256 }
257 }
258 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task5_2
6  {
7      public class Account
8      {
9          //instance variables declared
10         private decimal _balance;
11         private string _name;
12
13         //constructor
14         public Account(decimal balance, string name)
15         {
16
17             this._name = name;
18             if (balance <= 0)
19                 return;
20             this._balance = balance;
21
22         }
23
24         //prints name and balance
25         public void Print()
26         {
27             Console.WriteLine("The name of the account holder is " + getName() +
28                 ↪ "\nCurrent Account Balance is $" + getBalance());
29         }
30
31         //returns name
32         public String getName()
33         {
34             return this._name;
35         }
36
37         //returns balance
38         public decimal getBalance()
39         {
40             return this._balance;
41         }
42
43         //increases balance by adding deposit
44         //boolean ensure that no 0 or a negative value can not be input
45         public Boolean Deposit(decimal amount)
46         {
47             if (amount <= 0)
48                 return false;
49
50             this._balance += amount;
51             return true;
52         }
53     }
```

```
53     }
54
55
56     //decreases balanace by withdrawing the giving input amount
57     //boolean values protect against overdrawing from account
58     public Boolean Withdraw(decimal amount)
59     {
60         if (amount > this._balance || amount < 0)
61             return false;
62
63
64         this._balance -= amount;
65         return true;
66     }
67
68
69 }
70 }
```

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 inheritance works within the design of the program.

Outcome	Weight
Principles	◆◆◆◆◆

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

Outcome	Weight
Build Programs	◆◆◆◆◆

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

Outcome	Weight
Design	◆◆◆◆◆

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

May 7, 2020



```
1  using System;
2  namespace task6_1
3  {
4      public class Bird
5      {
6          public string name { get; set; }
7
8          public Bird()
9          {
10
11          }
12
13          public virtual void fly()
14          {
15              Console.WriteLine("Flap, Flap, Flap");
16          }
17
18          public override string ToString()
19          {
20              return "A bird called " + name;
21          }
22      }
23 }
```

```
1  using System;
2  namespace task6_1
3  {
4      class Penguin:Bird
5      {
6          public override void fly()
7          {
8              Console.WriteLine("Penguins cannot fly");
9          }
10
11         public override string ToString()
12         {
13             return "A penguin named " + base.name;
14         }
15     }
16 }
```

```
1  using System;
2  namespace task6_1
3  {
4      class Duck:Bird
5      {
6          public double size { get; set; }
7          public String kind { get; set; }
8
9          public override string ToString()
10         {
11             return "A duck named " + base.name + " is a " + size + " inch " + kind;
12         }
13     }
14 }
```



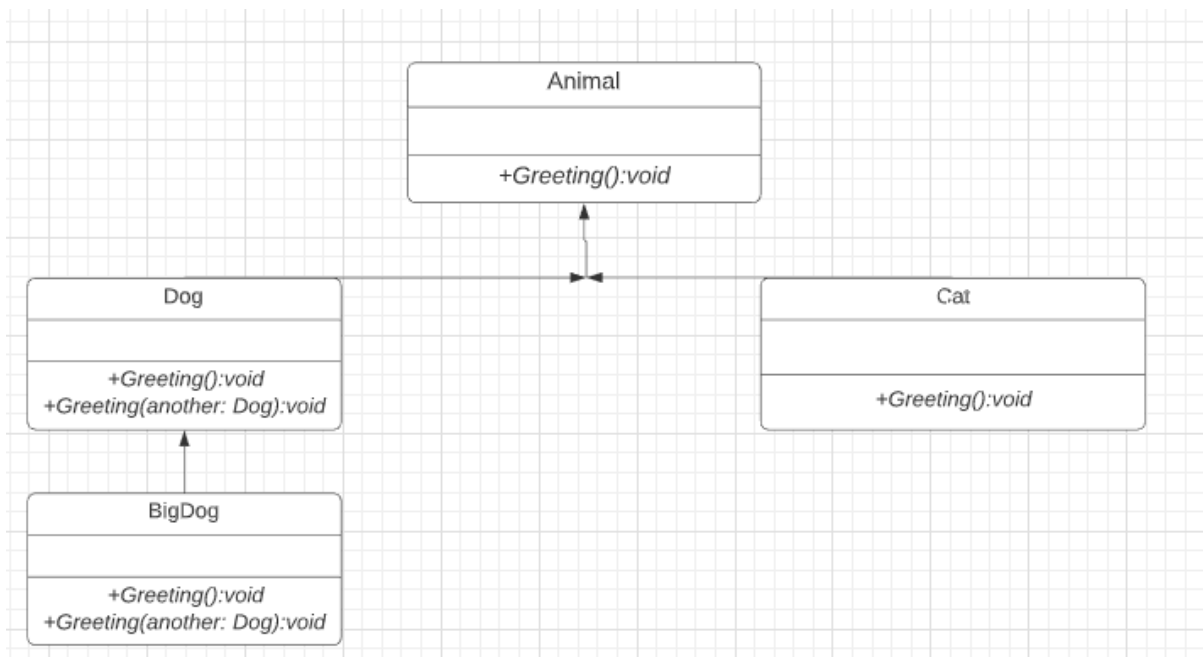
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace task6_1
6  {
7      class Program
8      {
9          static void Main(string[] args)
10         {
11             //Step 1
12             Bird bird1 = new Bird();
13             Bird bird2 = new Bird();
14
15             bird1.name = "Feathers";
16             bird2.name = "Polly";
17
18             Console.WriteLine(bird1.ToString());
19             bird1.fly();
20
21             Console.WriteLine(bird2.ToString());
22             bird2.fly();
23
24             Console.WriteLine("-----");
25
26             //Step 2
27
28             Penguin penguin1 = new Penguin();
29             Penguin penguin2 = new Penguin();
30             penguin1.name = "Happy Feet";
31             penguin2.name = "Gloria";
32
33             Console.WriteLine(penguin1.ToString());
34             penguin1.fly();
35
36             Console.WriteLine(penguin2.ToString());
37             penguin2.fly();
38
39             Duck duck1 = new Duck();
40             duck1.name = "Daffy";
41             duck1.size = 15;
42             duck1.kind = "Maillard";
43
44             Duck duck2 = new Duck();
45             duck2.name = "Donald";
46             duck2.size = 20;
47             duck2.kind = "Decoy";
48
49             Console.WriteLine(duck1.ToString());
50             Console.WriteLine(duck2.ToString());
51
52             Console.WriteLine("-----");
53
```

```
54      //Step3
55
56      List<Bird> birds = new List<Bird>();
57      Bird bird3 = new Bird();
58      bird3.name = "Feathers";
59      Bird bird4 = new Bird();
60      bird3.name = "Polly";
61
62      Penguin penguin3 = new Penguin();
63      Penguin penguin4 = new Penguin();
64      penguin3.name = "Happy Feet";
65      penguin4.name = "Gloria";
66
67      Duck duck3 = new Duck();
68      duck3.name = "Daffy";
69      duck3.size = 15;
70      duck3.kind = "Maillard";
71
72      Duck duck4 = new Duck();
73      duck4.name = "Donald";
74      duck4.size = 20;
75      duck4.kind = "Decoy";
76
77      birds.Add(bird3);
78      birds.Add(bird4);
79      birds.Add(penguin3);
80      birds.Add(penguin4);
81      birds.Add(duck3);
82      birds.Add(duck4);
83
84      birds.Add(new Bird { name = "Tweety" });
85
86      foreach (Bird bird in birds)
87      {
88          Console.WriteLine(bird);
89      }
90
91      //Step 3
92
93
94      Duck duck5 = new Duck();
95      duck5.name = "Daffy";
96      duck5.size = 15;
97      duck5.kind = "Maillard";
98
99      Duck duck6 = new Duck();
100     duck6.name = "Donald";
101     duck6.size = 20;
102     duck6.kind = "Decoy";
103
104
105     List<Duck> ducksToAdd = new List<Duck>()
106     {
```

```
107         duck5,
108         duck6
109     };
110
111     IEnumerable<Bird> upcastDucks = ducksToAdd;
112
113     List<Bird> birds2 = new List<Bird>();
114     birds2.Add(new Bird(){ name="Feather"});
115
116     birds2.AddRange(upcastDucks);
117
118     foreach(Bird bird in birds)
119     {
120         Console.WriteLine(bird);
121     }
122 }
123 }
124 }
```

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



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading;
5
6  namespace task6_2
7  {
8
9
10     class BankSystem
11     {
12
13         enum MenuOption
14         {
15             AddnewAccount,
16             Withdraw,
17             Deposit,
18             Print,
19             TransferTransaction,
20             Quit
21         }
22
23         static void AddAccount(Bank bank)
24         {
25             Console.WriteLine("Enter the name of the account holder");
26             string name = Console.ReadLine();
27             Console.WriteLine("Enter their starting account balance");
28             decimal balance = Convert.ToDecimal(Console.ReadLine());
29             bank.AddAccount(new Account(balance, name));
30         }
31
32         private static Account FindAccount(Bank bank)
33         {
34             Account account = null;
35             Console.WriteLine("Enter the account name");
36             string name = Console.ReadLine();
37             account = bank.GetAccount(name);
38             if(account==null)
39             {
40                 Console.WriteLine("Account does not exist");
41             }
42             return account;
43         }
44         static void DoDeposit(Bank bank)
45         {
46             Account account = FindAccount(bank);
47             Console.WriteLine("Enter deposit amount: ");
48             decimal amount = Convert.ToDecimal(Console.ReadLine());
49             DepositTransaction deposit = new DepositTransaction(account, amount);
50
51             if(account!=null)
52             {
53                 string execute;
```



```
54         string rollback;
55
56         Console.WriteLine($"Deposit the given amount: ${amount}. Y/N? ");
57         execute = Console.ReadLine();
58         Console.WriteLine("Rollback the transaction. Y/N? ");
59         rollback = Console.ReadLine();
60         if (execute == "Y".ToLower())
61         {
62             bank.ExecuteTransaction(deposit);
63         }
64         if (rollback == "Y".ToLower())
65         {
66             deposit.Rollback();
67         }
68     }
69 }
70
71
72 static void DoWithdrawal(Bank bank)
73 {
74     Account account = FindAccount(bank);
75     Console.WriteLine("Enter withdrawal amount: ");
76     decimal amount = Convert.ToDecimal(Console.ReadLine());
77     WithdrawTransaction withdrawal = new WithdrawTransaction(account,
78         ↪ amount);
79
80     if (account != null)
81     {
82         string execute;
83         string rollback;
84
85         Console.WriteLine($"Withdraw the given amount: ${amount}. Y/N? ");
86         execute = Console.ReadLine();
87         Console.WriteLine("Rollback the transaction. Y/N? ");
88         rollback = Console.ReadLine();
89         if (execute == "Y".ToLower())
90         {
91             bank.ExecuteTransaction(withdrawal);
92         }
93         if (rollback == "Y".ToLower())
94         {
95             withdrawal.Rollback();
96         }
97     }
98 }
99
100 static void DoTransfer(Bank bank1)
101 {
102     Account fromaccount = FindAccount(bank1);
103     Account toaccount = FindAccount(bank1);
104     Console.WriteLine("Transfer the following amount: ");
105     decimal amount = Convert.ToDecimal(Console.ReadLine());
```

```
106         TransferTransaction transfer = new TransferTransaction(fromaccount,
107             ↳ toaccount, amount);
108
109         if(bank1!=null)
110         {
111             string execute;
112             string rollback;
113
114             Console.WriteLine($"Transfer the given amount: ${amount}. Y/N? ");
115             execute = Console.ReadLine();
116             Console.WriteLine("Would you like to rollback the transfer. Y/N? ");
117             rollback = Console.ReadLine();
118             if (execute == "Y".ToLower())
119             {
120                 bank1.ExecuteTransaction(transfer);
121             }
122             if (rollback == "Y".ToLower())
123             {
124                 transfer.Rollback();
125             }
126         }
127
128
129         static MenuOption ReadUserOption()
130         {
131             int? option = null;
132             do
133             {
134                 Console.WriteLine("-----")
135                 ↳ --");
136                 Console.WriteLine("Please select from the following options");
137                 Console.WriteLine("MENU \n1. Add new Account \n2. Withdraw \n3.
138                 ↳ Deposit \n4. Print \n5. Transfer Transaction \n6. Quit");
139                 Console.WriteLine("-----")
140                 ↳ --");
141
142                 try
143                 {
144                     option = Convert.ToInt32(Console.ReadLine());
145                     if (option > 5 || option < 1)
146                     {
147                         option = null;
148                         Console.WriteLine("Please enter a number from 1 to 5 from
149                         ↳ the menu");
150                     }
151                 }
152                 catch (FormatException)
153                 {
154                     Console.WriteLine("Invalid input. Enter an integer from 1-5");
155                 }
156             } while (option == null);
```

```
154
155     return (MenuOption)option;
156 }
157
158
159 static void DoPrint(Bank bank)
160 {
161     Account account = FindAccount(bank);
162     if(account!=null)
163     {
164         account.Print();
165     }
166 }
167
168
169
170 static void Main(string[] args)
171 {
172     Bank newbank=new Bank();
173
174     Account person_one = new Account(100, "Jon");
175     Account person_two = new Account(150, "Amy");
176     MenuOption option;
177
178     do
179     {
180         option = ReadUserOption() - 1;
181
182         switch (option)
183         {
184             case MenuOption.AddnewAccount:
185                 AddAccount(newbank);
186                 break;
187             case MenuOption.Withdraw:
188                 Console.WriteLine("You have selected withdraw");
189                 DoWithdrawal(newbank);
190                 break;
191             case MenuOption.Deposit:
192                 Console.WriteLine("You have selected deposit");
193                 DoDeposit(newbank);
194                 break;
195             case MenuOption.Print:
196                 Console.WriteLine("You have selected print");
197                 DoPrint(newbank);
198                 break;
199             case MenuOption.TransferTransaction:
200                 DoTransfer(newbank);
201                 break;
202             case MenuOption.Quit:
203                 Console.WriteLine("Goodbye");
204                 break;
205
206         }
```

```
207         } while (option != MenuOption.Quit);
208     }
209
210 }
211 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task6_2
6  {
7      public class Bank
8      {
9          private static List<Account> _accounts=new List<Account>();
10
11
12         public void AddAccount(Account account)
13         {
14             _accounts.Add(account);
15         }
16
17         public Account GetAccount(String name)
18         {
19             foreach(Account account in _accounts)
20             {
21                 if(account.getName().ToLower()==name.ToLower())
22                 {
23                     return account;
24                 }
25             }
26             { return null; }
27         }
28
29         public void ExecuteTransaction(DepositTransaction transaction)
30         {
31             try
32             {
33                 transaction.Execute();
34             }
35
36             catch(InvalidOperationException)
37             {
38                 Console.WriteLine("Error in performing the transaction");
39             }
40         }
41
42         public void ExecuteTransaction(WithdrawTransaction transaction)
43         {
44             try
45             {
46                 transaction.Execute();
47             }
48
49             catch (InvalidOperationException)
50             {
51                 Console.WriteLine("Error in performing the transaction");
52             }
53         }
```

```
54     }
55
56     public void ExecuteTransaction(TransferTransaction transaction)
57     {
58         try
59         {
60             transaction.Execute();
61         }
62
63         catch (InvalidOperationException)
64         {
65             Console.WriteLine("Error in performing the transaction");
66         }
67     }
68
69 }
70
71 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task6_2
6  {
7      public class Account
8      {
9          //instance variables declared
10         private decimal _balance;
11         private string _name;
12
13         //constructor
14         public Account(decimal balance, string name)
15         {
16
17             this._name = name;
18             if (balance <= 0)
19                 return;
20             this._balance = balance;
21
22         }
23
24         //prints name and balance
25         public void Print()
26         {
27             Console.WriteLine("The name of the account holder is " + getName() +
28                 ↳ "\nCurrent Account Balance is $" + getBalance());
29         }
30
31         //returns name
32         public String getName()
33         {
34             return this._name;
35         }
36
37         //returns balance
38         public decimal getBalance()
39         {
40             return this._balance;
41         }
42
43         //increases balance by adding deposit
44         //boolean ensure that no 0 or a negative value can not be input
45         public Boolean Deposit(decimal amount)
46         {
47             if (amount <= 0)
48                 return false;
49
50             this._balance += amount;
51             return true;
52
53         }
```

```
53     }
54
55
56     //decreases balanace by withdrawing the giving input amount
57     //boolean values protect against overdrawing from account
58     public Boolean Withdraw(decimal amount)
59     {
60         if (amount > this._balance || amount < 0)
61             return false;
62
63
64         this._balance -= amount;
65         return true;
66     }
67
68
69 }
70 }
```



```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task6_2
6  {
7      public class WithdrawTransaction
8      {
9
10         private Account _account;
11         private decimal _amount;
12         private bool _executed;
13         private bool _success;
14         private bool _reversed;
15
16         public bool Executed { get { return _executed; } }
17         public bool Success { get { return _success; } }
18         public bool Reversed { get { return _reversed; } }
19
20         public WithdrawTransaction(Account account, decimal amount)
21         {
22             this._account = account;
23             this._amount = amount;
24             _reversed = false;
25             _executed = false;
26         }
27
28
29         public void Print()
30         {
31             Console.WriteLine($"Successfully withdrew ${this._amount} from the
32             ↪ account: {this._account.getName()}");
33         }
34
35         public void Execute()
36         {
37             if (Executed == false)
38             {
39                 if (_account.getBalance() > 0 && _amount <= _account.getBalance())
40                 {
41                     _account.Withdraw(_amount);
42                     _executed = true;
43                 }
44                 else
45                 {
46                     Console.WriteLine("Insufficient funds");
47                 }
48             }
49             else { throw new InvalidOperationException("Withdrawal has already been
50             ↪ performed"); }
51         }
52     }
```

```
52     public void Rollback()
53     {
54         if (Reversed == false && Executed == true)
55         {
56             _account.Deposit(_amount);
57             _executed = false;
58             _reversed = true;
59         }
60         else
61         {
62             Console.WriteLine("Transaction has already been reversed");
63         }
64     }
65 }
66 }
67 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task6_2
6  {
7      public class DepositTransaction
8      {
9
10         private Account _account;
11         private decimal _amount;
12         private bool _executed;
13         private bool _success;
14         private bool _reversed;
15
16         public bool Executed { get { return _executed; } }
17         public bool Success { get { return _success; } }
18         public bool Reversed { get { return _reversed; } }
19
20         public DepositTransaction(Account account, decimal amount)
21         {
22             this._account = account;
23             this._amount = amount;
24             _reversed = false;
25             _executed = false;
26         }
27
28         public void print()
29         {
30             Console.WriteLine($"Successfully deposited ${this._amount} into the
31             ↪ account: {this._account.getName()}");
32         }
33
34         public void Execute()
35         {
36             if (Executed == false)
37             {
38                 _account.Deposit(_amount);
39                 _executed = true;
40                 _reversed = false;
41             }
42             else
43             {
44                 throw new InvalidOperationException("You have already performed
45                 ↪ this transaction");
46             }
47         }
48
49         public void Rollback()
50         {
51             if (Executed == true && Reversed == false)
52             {
53                 _account.Withdraw(_amount);
54             }
55         }
56     }
57 }
```

```
52         _executed = false;
53         _reversed = true;
54
55     }
56     else
57     {
58         Console.WriteLine("There is no deposit to reverse");
59     }
60
61
62     }
63 }
64 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5
6  namespace task6_2
7  {
8      public class TransferTransaction
9      {
10
11          private Account _fromAccount;
12          private Account _toAccount;
13          private decimal _amount;
14          private DepositTransaction _deposit;
15          private WithdrawTransaction _withdraw;
16          private bool _executed;
17          private bool _reversed;
18          private bool _success;
19
20          public bool Executed { get { return _executed; } }
21          public bool Success { get { return _success; } }
22          public bool Reversed { get { return _reversed; } }
23
24          public TransferTransaction(Account fromAccount, Account toAccount, decimal
25              ↪ amount)
26          {
27              _deposit = new DepositTransaction(toAccount, amount);
28              _withdraw = new WithdrawTransaction(fromAccount, amount);
29              _executed = false;
30              this._fromAccount = fromAccount;
31              this._toAccount = toAccount;
32              this._amount = amount;
33          }
34
35          public bool success
36          {
37              get
38              {
39                  if (_deposit.Success == true && _withdraw.Success == true)
40                  {
41                      return true;
42                  }
43                  else
44                  {
45                      return false;
46                  }
47              }
48          }
49
50          public void print()
51          {
52              if (_executed == true)
```

```
53     {
54         Console.WriteLine("Transaction has been performed successfully");
55         Console.WriteLine($"Transferred ${_amount} from
56             ↳ {_fromAccount.getName()} to {_toAccount.getName()}");
57     }
58     else
59     {
60         Console.WriteLine("Transaction has not been successful");
61     }
62 }
63
64
65 public void Execute()
66 {
67     if (Executed == false)
68     {
69         if (_fromAccount.getBalance() < _amount)
70         {
71             Console.WriteLine("Insufficient funds");
72         }
73         else
74         {
75             {
76                 _withdraw.Execute();
77                 _deposit.Execute();
78                 _executed = true;
79             }
80         }
81     }
82     else { throw new InvalidOperationException("Transacation has been
83         ↳ already performed"); }
84 }
85
86 public void Rollback()
87 {
88     if (Reversed == true)
89     {
90         throw new InvalidOperationException("Reverse transaction has
91             ↳ already been performed");
92     }
93     if (_toAccount.getBalance() < _amount)
94     {
95         throw new InvalidOperationException("Insufficient funds");
96     }
97     else if (Executed == true)
98     {
99         {
100             _deposit.Rollback();
101             _withdraw.Rollback();
102             _reversed = true;
103             _executed = false;
104         }
105     }
```

103
104 }
105
106 }
107
108 }

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:

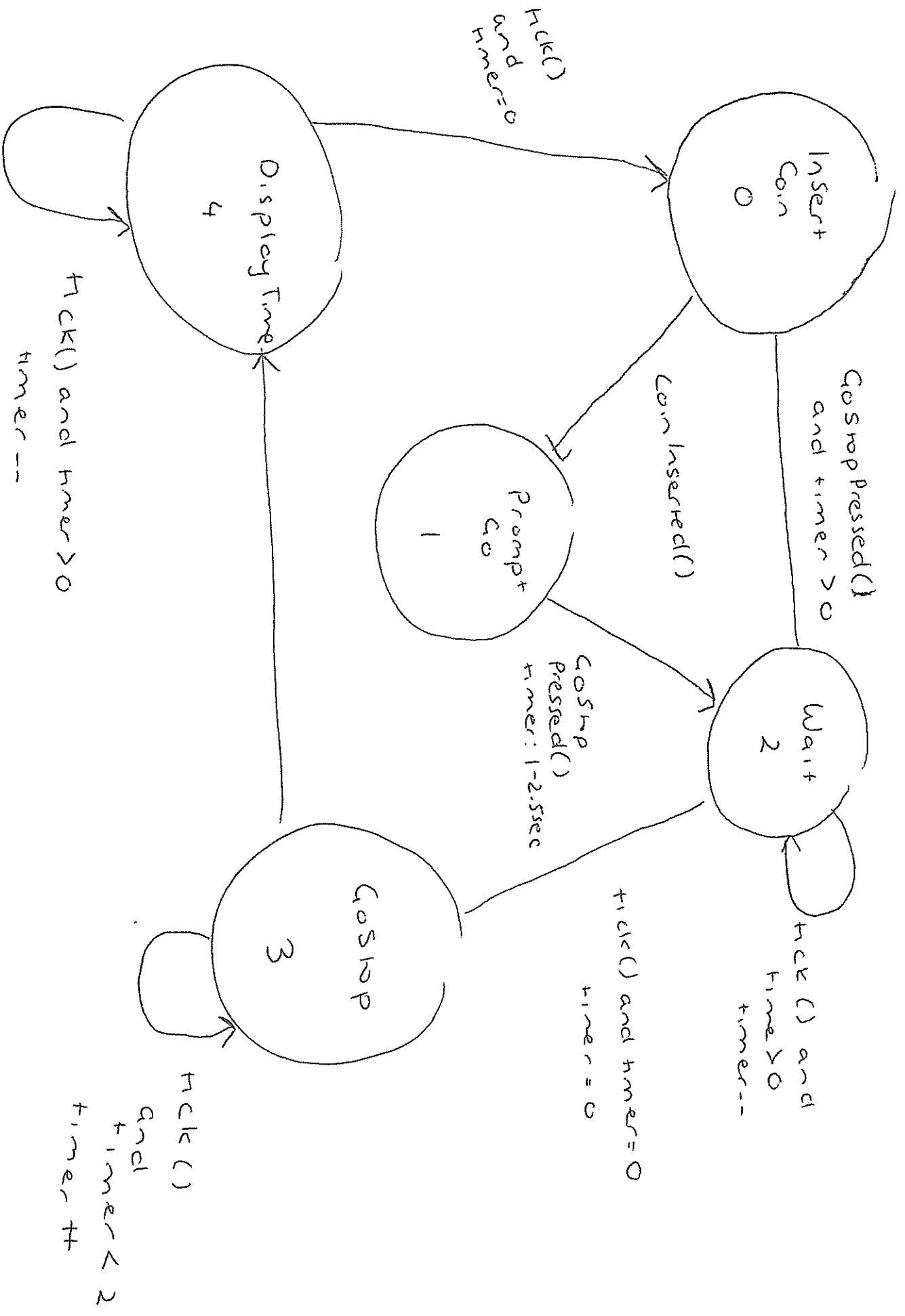
Sanjay SEGU

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





```
1  using System;
2
3
4  namespace task5_3
5  {
6      public class SimpleReactionController: IController
7      {
8          private const int MIN_WAIT_TIME = 100;
9          private const int MAX_WAIT_TIME = 250;
10         private const int MAX_GAME_TIME = 200;
11         private const int GAMEOVER_TIME = 300;
12         private const double TICKS_PER_SECOND = 100.0;
13
14         private State _state;
15         private IGui Gui { get; set; }
16         private IRandom Rng { get; set; }
17         private int Ticks { get; set; }
18
19         public void Connect(IGui gui, IRandom rng)
20         {
21             Gui = gui;
22             Rng = rng;
23             Init();
24         }
25         public void Init()
26         {
27             _state = new OnState(this);
28         }
29
30         public void GoStopPressed()
31         {
32             _state.GoStopPressed();
33         }
34         public void CoinInserted()
35         {
36             _state.CoinInserted();
37         }
38         public void Tick()
39         {
40             _state.Tick();
41         }
42
43
44         private void SetState(State state)
45         {
46             _state = state;
47         }
48
49         private abstract class State
50         {
51             protected SimpleReactionController _controller;
52
53             public State(SimpleReactionController controller)
```

```
54         {
55             _controller = controller;
56         }
57
58         public abstract void CoinInserted();
59         public abstract void GoStopPressed();
60         public abstract void Tick();
61
62     }
63
64     private class OnState : State
65     {
66         public OnState(SimpleReactionController controller):base(controller)
67         {
68             _controller.Gui.SetDisplay("Insert Coin");
69         }
70
71         public override void CoinInserted()
72         {
73             _controller.SetState(new ReadyState(_controller));
74         }
75         public override void GoStopPressed() { }
76         public override void Tick() { }
77
78     }
79
80     private class WaitState : State
81     {
82         private int _waitTime;
83         public WaitState(SimpleReactionController controller):base(controller)
84         {
85             _controller.Gui.SetDisplay("Wait...");
86             _controller.Ticks = 0;
87             _waitTime = _controller.Rng.GetRandom(MIN_WAIT_TIME, MAX_WAIT_TIME);
88         }
89
90         public override void CoinInserted() { }
91
92         public override void GoStopPressed()
93         {
94             _controller.SetState(new OnState(_controller));
95         }
96
97         public override void Tick()
98         {
99             _controller.Ticks++;
100             if(_controller.Ticks==_waitTime)
101             {
102                 _controller.SetState(new RunningState(_controller));
103             }
104         }
105     }
106 }
```

```
107     private class RunningState : State
108     {
109         public RunningState(SimpleReactionController controller) :
110             ↪ base(controller)
111         {
112             _controller.Gui.SetDisplay("0.00");
113             _controller.Ticks = 0;
114         }
115         public override void CoinInserted() { }
116         public override void GoStopPressed()
117         {
118             _controller.SetState(new GameOverState(_controller));
119         }
120
121         public override void Tick()
122         {
123             _controller.Ticks++;
124             _controller.Gui.SetDisplay((_controller.Ticks /
125             ↪ TICKS_PER_SECOND).ToString("0.00"));
126             if (_controller.Ticks == MAX_GAME_TIME)
127             {
128                 _controller.SetState(new GameOverState(_controller));
129             }
130         }
131     }
132
133     private class ReadyState : State
134     {
135         public ReadyState(SimpleReactionController controller):base(controller)
136         {
137             _controller.Gui.SetDisplay("Press Go");
138         }
139
140         public override void CoinInserted() { }
141         public override void GoStopPressed()
142         {
143             _controller.SetState(new WaitState(_controller));
144         }
145         public override void Tick() { }
146     }
147
148     private class GameOverState : State
149     {
150         public GameOverState(SimpleReactionController controller) :
151             ↪ base(controller)
152         {
153             _controller.Ticks = 0;
154         }
155         public override void CoinInserted() { }
156         public override void GoStopPressed()
```

```
157         {
158             _controller.SetState(new OnState(_controller));
159         }
160         public override void Tick()
161         {
162             _controller.Ticks++;
163             if (_controller.Ticks == GAMEOVER_TIME)
164             {
165                 _controller.SetState(new OnState(_controller));
166             }
167         }
168     }
169 }
170 }
171 }
```

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




```
1  using System;
2
3
4  namespace task5_3
5  {
6      public class EnhancedReactionController : IController
7      {
8          private const int MAX_READY_TIME = 1000;
9          private const int MIN_WAIT_TIME = 100;
10         private const int MAX_WAIT_TIME = 250;
11         private const int MAX_GAME_TIME = 200;
12         private const int GAMEOVER_TIME = 300;
13         private const int RESULTS_TIME = 500;
14         private const double TICKS_PER_SECOND = 100.0;
15
16         private State _state;
17         private IGui Gui { get; set; }
18         private IRandom Rng { get; set; }
19         private int Ticks { get; set; }
20         private int Games{ get; set; }
21         private int TotalReactionTime { get; set; }
22
23         public void Connect(IGui gui, IRandom rng)
24         {
25             Gui = gui;
26             Rng = rng;
27             Init();
28         }
29         public void Init()
30         {
31             _state = new OnState(this);
32         }
33
34         public void GoStopPressed()
35         {
36             _state.GoStopPressed();
37         }
38         public void CoinInserted()
39         {
40             _state.CoinInserted();
41         }
42         public void Tick()
43         {
44             _state.Tick();
45         }
46
47
48         private void SetState(State state)
49         {
50             _state = state;
51         }
52
53         private abstract class State
```

```
54     {
55         protected EnhancedReactionController controller;
56
57         public State(EnhancedReactionController cont)
58         {
59             controller=cont;
60         }
61
62         public abstract void CoinInserted();
63         public abstract void GoStopPressed();
64         public abstract void Tick();
65
66     }
67
68     private class OnState : State
69     {
70         public OnState(EnhancedReactionController cont):base(cont)
71         {
72             controller.Games = 0;
73             controller.TotalReactionTime = 0;
74             controller.Gui.SetDisplay("Insert Coin");
75         }
76
77         public override void CoinInserted()
78         {
79             controller.SetState(new ReadyState(controller));
80         }
81         public override void GoStopPressed() { }
82         public override void Tick() { }
83
84     }
85
86     private class WaitState : State
87     {
88         private int _waitTime;
89         public WaitState(EnhancedReactionController cont):base(cont)
90         {
91             controller.Gui.SetDisplay("Wait...");
92             controller.Ticks = 0;
93             _waitTime = controller.Rng.GetRandom(MIN_WAIT_TIME, MAX_WAIT_TIME);
94         }
95
96         public override void CoinInserted() { }
97
98         public override void GoStopPressed()
99         {
100             controller.SetState(new OnState(controller));
101         }
102
103         public override void Tick()
104         {
105             controller.Ticks++;
106             if(controller.Ticks==_waitTime)
```

```
107         {
108             controller.SetState(new RunningState(controller));
109         }
110     }
111 }
112
113 private class RunningState : State
114 {
115     public RunningState(EnhancedReactionController cont) : base(cont)
116     {
117         controller.Gui.SetDisplay("0.00");
118         controller.Ticks = 0;
119     }
120     public override void CoinInserted() { }
121     public override void GoStopPressed()
122     {
123         controller.SetState(new GameOverState(controller));
124     }
125
126     public override void Tick()
127     {
128         controller.Ticks++;
129         controller.Gui.SetDisplay((controller.Ticks /
130             ↪ TICKS_PER_SECOND).ToString("0.00"));
131         if (controller.Ticks == MAX_GAME_TIME)
132         {
133             controller.SetState(new GameOverState(controller));
134         }
135     }
136 }
137
138
139
140 private class ReadyState : State
141 {
142     public ReadyState(EnhancedReactionController cont):base(cont)
143     {
144         controller.Gui.SetDisplay("Press Go");
145         controller.Ticks = 0;
146     }
147
148     public override void CoinInserted() { }
149     public override void GoStopPressed()
150     {
151         controller.SetState(new WaitState(controller));
152     }
153
154     public override void Tick()
155     {
156         controller.Ticks++;
157         if (controller.Ticks == MAX_READY_TIME)
158             controller.SetState(new OnState(controller));
```

```
159     }
160
161
162
163 }
164
165 private class GameOverState : State
166 {
167     public GameOverState(EnhancedReactionController cont) : base(cont)
168     {
169         controller.Ticks = 0;
170     }
171     public override void CoinInserted() { }
172     public override void GoStopPressed() => CheckGames();
173
174     public override void Tick()
175     {
176         controller.Ticks++;
177         if (controller.Ticks == GAMEOVER_TIME)
178             CheckGames();
179     }
180     private void CheckGames()
181     {
182         if(controller.Games==3)
183         {
184             controller.SetState(new ResultsState(controller));
185             return;
186         }
187         controller.SetState(new WaitState(controller));
188     }
189 }
190
191 class ResultsState:State
192 {
193     public ResultsState(EnhancedReactionController cont):base(cont)
194     {
195         controller.Gui.SetDisplay("Average: " +
196             ↪ (((double)controller.TotalReactionTime / controller.Games) /
197             ↪ TICKS_PER_SECOND).ToString("0.00"));
198         controller.Ticks = 0;
199     }
200
201     public override void CoinInserted() { }
202     public override void GoStopPressed() => controller.SetState(new
203         ↪ OnState(controller));
204
205     public override void Tick()
206     {
207         controller.Ticks++;
208         if (controller.Ticks == RESULTS_TIME)
209             controller.SetState(new OnState(controller));
210     }
211 }
```

```
209  
210     }  
211 }
```

```
1  using Microsoft.VisualStudio.TestTools.UnitTesting;
2  using System;
3  //using EnhancedReactionController;
4
5  namespace task5_3
6  {
7      [TestClass]
8      public class UnitTest1
9      {
10         //Construct a ReactionController
11         private static IController controller;
12         private static IGui gui;
13         private static IRandom rng;
14         private static string displayText;
15
16         private static int RandomNumber { get; set; }
17
18
19         [TestMethod]
20
21         public void Test_Connect_Controller()
22         {
23             //Connect them to each other
24             controller = new EnhancedReactionController();
25             gui = new DummyGui();
26             gui.Connect(controller);
27             controller.Connect(gui, new RndGenerator());
28             controller.Init();
29             Assert.AreEqual("Insert Coin", displayText);
30         }
31
32         //Test Waiting Time after coin inserted
33         [TestMethod]
34         public void Test_Waiting_CoinInserted()
35         {
36             controller = new EnhancedReactionController();
37             gui = new DummyGui();
38             rng = new RndGenerator();
39             WaitingState(controller, gui, rng);
40
41             Assert.AreEqual("Wait", displayText);
42             controller.CoinInserted();
43             Assert.AreEqual("Wait", displayText);
44         }
45
46
47         //user presses go, wait ensues
48         [TestMethod]
49         public void Test_Start_Wait()
50         {
51             controller = new EnhancedReactionController();
52             gui = new DummyGui();
53             rng = new RndGenerator();
```

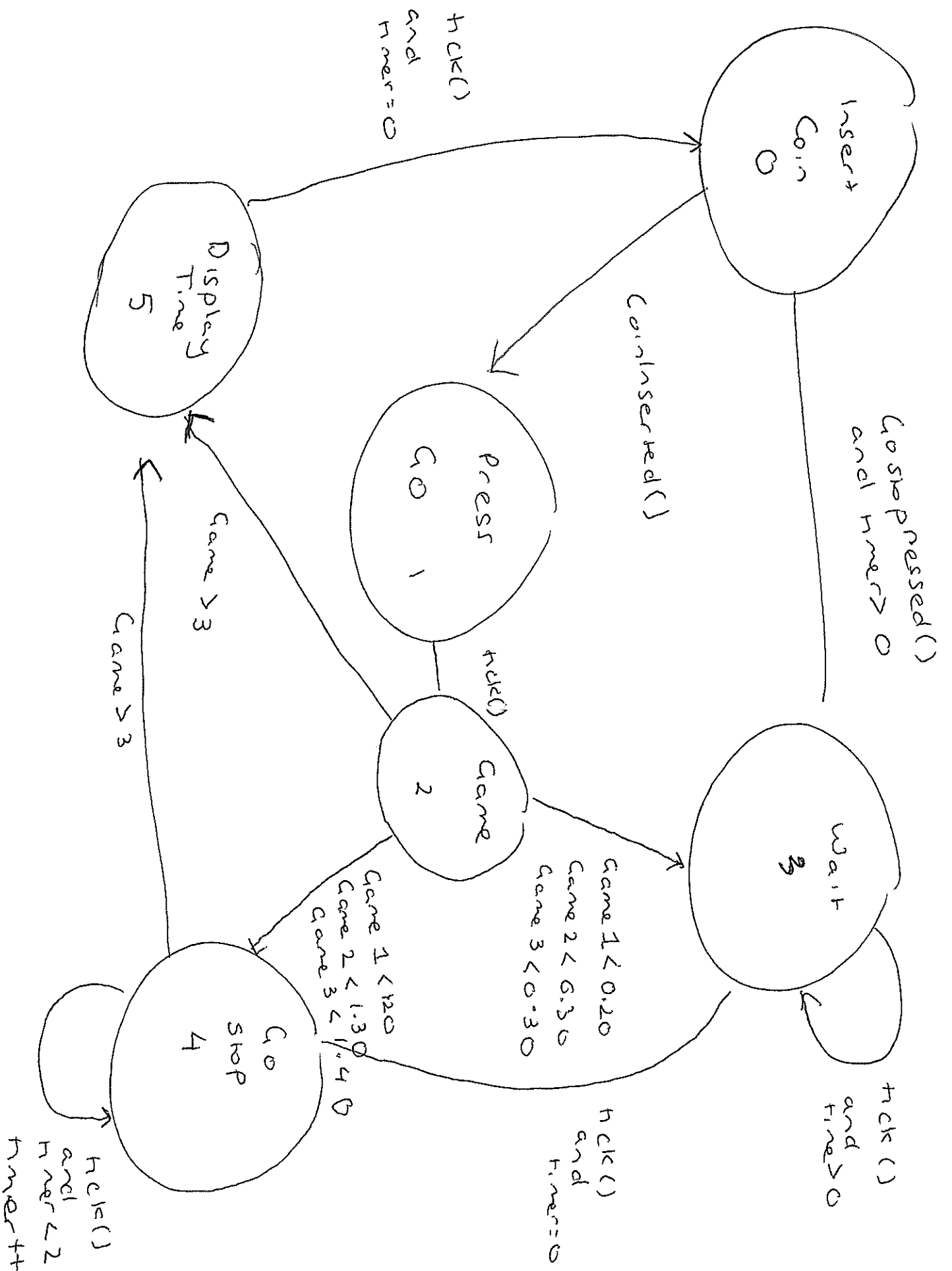
```
54         StartingState(controller, gui, rng);
55
56
57         Assert.AreEqual("Press Go", displayText);
58         controller.GoStopPressed();
59         Assert.AreEqual("Wait", displayText);
60
61     }
62
63     //user presses go, insert coin response
64     [TestMethod]
65     public void Test_GameOver()
66     {
67         controller = new EnhancedReactionController();
68         gui = new DummyGui();
69         rng = new RndGenerator();
70         StartingState(controller, gui, rng);
71
72         //random time up until 100 miliseconds
73         for (int t = 0; t < 100; t++) controller.Tick();
74         Assert.AreEqual("Press Go", displayText);
75         controller.Tick();
76         Assert.AreEqual("Insert Coin", displayText);
77
78     }
79
80
81
82     //Tests running time, less than 2.04 then GoStopPress()
83     [TestMethod]
84     public void Test_GoStopPressed()
85     {
86         controller = new EnhancedReactionController();
87         gui = new DummyGui();
88         rng = new RndGenerator();
89         RunningState(controller, gui, rng);
90
91         for (int t = 0; t < 204; t++) controller.Tick();
92         Assert.AreEqual("2.04", displayText);
93         controller.GoStopPressed();
94         Assert.AreEqual("2.04", displayText);
95
96     }
97
98
99     //Tests running time, less than 0.99 then GoStopPress()
100    [TestMethod]
101    public void Test_GameOverState()
102    {
103        controller = new EnhancedReactionController();
104        gui = new DummyGui();
105        rng = new RndGenerator();
106        RunningState(controller, gui, rng);
```

```
107
108     for (int t = 0; t < 99; t++) controller.Tick();
109     Assert.AreEqual("0.99", displayText);
110     controller.CoinInserted();
111     Assert.AreEqual("0.99", displayText);
112
113 }
114
115 //test three games with waiting time
116 [TestMethod]
117 public void Test_Playing_Three_Games()
118 {
119     controller = new EnhancedReactionController();
120     gui = new DummyGui();
121     rng = new RndGenerator();
122     RunningState(controller, gui, rng);
123
124     //game 1, user presses before 20 miliseconds, Wait
125     for (int t = 0; t < 20; t++) controller.Tick();
126     controller.GoStopPressed();
127     Assert.AreEqual("0.20", displayText);
128     for (int t = 0; t < 199; t++) controller.Tick();
129     Assert.AreEqual("0.20", displayText);
130     controller.Tick();
131     Assert.AreEqual("Wait", displayText);
132
133     //game 2, user presses before 30 miliseconds, Wait
134     for (int t = 0; t < RandomNumber+30; t++) controller.Tick();
135     controller.GoStopPressed();
136     Assert.AreEqual("0.30", displayText);
137     for (int t = 0; t < 299; t++) controller.Tick();
138     Assert.AreEqual("0.30", displayText);
139     controller.Tick();
140     Assert.AreEqual("Wait", displayText);
141
142     //game 3, user presses before 40 miliseconds, Wait
143     for (int t = 0; t < RandomNumber + 40; t++) controller.Tick();
144     controller.GoStopPressed();
145     Assert.AreEqual("0.40", displayText);
146     for (int t = 0; t < 299; t++) controller.Tick();
147     controller.Tick();
148     Assert.AreEqual("Average: 30", displayText);
149 }
150
151 //User presses GoStopPressed if time meets conditions or GoPressed()
152 [TestMethod]
153 public void Test_Play_Three_Games()
154 {
155     controller = new EnhancedReactionController();
156     gui = new DummyGui();
157     rng = new RndGenerator();
158     RunningState(controller, gui, rng);
159
```



```
160         //game 1, user presses before 1.20 miliseconds, Press GoStop
161         for (int t = 0; t < 120; t++) controller.Tick();
162         controller.GoStopPressed();
163         Assert.AreEqual("1.20", displayText);
164         controller.GoStopPressed();
165         Assert.AreEqual("Wait", displayText);
166
167         //game 2, user presses before 1.30 miliseconds, Press GoStop
168         for (int t = 0; t < RandomNumber + 130; t++) controller.Tick();
169         controller.GoStopPressed();
170         Assert.AreEqual("1.30", displayText);
171         controller.GoStopPressed();
172         Assert.AreEqual("Wait", displayText);
173
174         //game 1, user presses before 1.40 miliseconds, Press GoStop
175         for (int t = 0; t < RandomNumber + 140; t++) controller.Tick();
176         controller.GoStopPressed();
177         Assert.AreEqual("1.40", displayText);
178         controller.GoStopPressed();
179         Assert.AreEqual("Average: 1.30", displayText);
180     }
181
182
183     private void StartingState(IController controller, IGui gui, IRandom rng)
184     {
185         gui.Connect(controller);
186         controller.Connect(gui, rng);
187         gui.Init();
188         controller.Init();
189     }
190
191
192     private void WaitingState(IController controller, IGui gui, IRandom rng)
193     {
194         OnState(controller, gui, rng);
195         for (int t = 0; t < RandomNumber; t++)
196             controller.Tick();
197     }
198
199
200
201     public void RunningState(IController controller, IGui gui, IRandom rng)
202     {
203         WaitingState(controller, gui, rng);
204         for (int t = 0; t < RandomNumber; t++)
205             controller.Tick();
206     }
207
208     public class DummyGui : IGui
209     {
210         private IController _controller;
211         public void Connect(IController controller) => _controller = controller;
212         public void Init() => displayText = "?reset?";
```

```
213
214     public void SetDisplay(string msg)
215     {
216         displayText = msg;
217     }
218 }
219
220 public class RndGenerator : IRandom
221 {
222     Random rnd = new Random(42);
223     public int GetRandom(int from, int to)
224     {
225         RandomNumber = rnd.Next(from) + to;
226         return RandomNumber;
227     }
228 }
229 }
230 }
```



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



```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task7_1
6  {
7      class DepositTransaction:Transaction
8      {
9
10
11         private Account _account;
12         private bool _executed;
13         private bool _reversed;
14         public Account Account { get => _account; }
15
16         public DepositTransaction(Account account, decimal amount):base(amount)
17         {
18             _account = account;
19         }
20
21         public override void Print()
22         {
23             Console.WriteLine($"Successfully deposited ${this._amount} into the
24             ↪ account: {this._account.getName()}");
25         }
26
27         //only executes if the transaction has yet to be performed
28         public override void Execute()
29         {
30             base.Execute();
31             if (_executed== false)
32             {
33                 _account.Deposit(_amount);
34                 _executed = true;
35             }
36             else
37             {
38                 throw new InvalidOperationException("You have already performed
39                 ↪ this transaction");
40             }
41         }
42
43         //rollbaack deposit by withdrawing _amount
44         public override void Rollback()
45         {
46             base.Rollback();
47             if (_executed== true && _reversed == false)
48             {
49                 _account.Withdraw(_amount);
50                 _executed = false;
51             }
52             else
```

```
52         {
53             Console.WriteLine("There is no deposit to reverse");
54         }
55
56
57     }
58
59     //returns user's name
60     public override string GetAccName()
61     {
62         return _account.getName();
63     }
64 }
65 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task7_1
6  {
7      public class WithdrawTransaction : Transaction
8      {
9
10         private Account _account;
11         private bool _executed;
12         private bool _reversed;
13
14
15
16         public WithdrawTransaction(Account account, decimal amount) : base(amount)
17         {
18             this._account = account;
19             this._amount = amount;
20             _reversed = false;
21             _executed = false;
22         }
23
24         public override string GetAccName()
25         {
26             return _account.getName();
27         }
28
29
30         public override void Print()
31         {
32             Console.WriteLine($"Successfully withdrew ${this._amount} from the
33             ↪ account: {this._account.getName()}");
34         }
35
36         public override void Execute()
37         {
38             base.Execute();
39             if (_executed == false)
40             {
41                 if (_account.getBalance() > 0 && _amount <= _account.getBalance())
42                 {
43                     _account.Withdraw(_amount);
44                     _executed = true;
45                 }
46                 else
47                 {
48                     Console.WriteLine("Insufficient funds");
49                 }
50             }
51
52             else { throw new InvalidOperationException("Withdrawal has already been
53             ↪ performed"); }
```



```
52     }
53
54     public override void Rollback()
55     {
56         base.Rollback();
57         if (_reversed == false && _executed == true)
58         {
59             _account.Deposit(_amount);
60             _executed = false;
61             _reversed = true;
62         }
63         else
64         {
65             Console.WriteLine("Transaction has already been reversed");
66         }
67
68     }
69 }
70
71 }
72 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5
6  namespace task7_1
7  {
8      public class TransferTransaction: Transaction
9      {
10
11         private Account _fromAccount;
12         private Account _toAccount;
13         private DepositTransaction _deposit;
14         private WithdrawTransaction _withdraw;
15         private bool _executed;
16         private bool _reversed;
17
18
19
20
21         public TransferTransaction(Account fromAccount, Account toAccount, decimal
22         ↪ amount):base(amount)
23         {
24             _deposit = new DepositTransaction(toAccount, amount);
25             _withdraw = new WithdrawTransaction(fromAccount, amount);
26             _executed = false;
27             this._fromAccount = fromAccount;
28             this._toAccount = toAccount;
29             this._amount = amount;
30         }
31
32         //prints statement
33         public override void Print()
34         {
35             if (_executed == true)
36             {
37                 Console.WriteLine("Transaction has been performed successfully");
38                 Console.WriteLine($"Transferred ${_amount} from
39                 ↪ {_fromAccount.getName()} to {_toAccount.getName()}");
40             }
41
42             else
43             {
44                 Console.WriteLine("Transaction has not been successful");
45             }
46         }
47
48         //executes legitimate transactions
49         public override void Execute()
50         {
51             if (_executed == false)
52             {
53                 if (_amount<=0 || _fromAccount.getBalance() < _amount)
```

```
52         {
53             Console.WriteLine("Insufficient funds");
54         }
55     else
56     {
57         _withdraw.Execute();
58         _deposit.Execute();
59         _executed = true;
60     }
61 }
62 }
63 }
64 else { throw new InvalidOperationException("Transacation has been
65     ↳ already performed"); }
66
67 _executed = true;
68 }
69
70 //check to see if rollback meets onditions
71 public override void Rollback()
72 {
73     if (_executed == false)
74     {
75         throw new InvalidOperationException("Reverse transaction has
76             ↳ already been performed");
77     }
78     if (_toAccount.getBalance() < _amount)
79     {
80         throw new InvalidOperationException("Insufficient funds");
81     }
82     else if (_executed == true)
83     {
84         _deposit.Rollback();
85         _withdraw.Rollback();
86         _reversed = true;
87         _executed = false;
88     }
89 }
90
91 }
92
93 }
94
95 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5
6  namespace task7_1
7  {
8      public abstract class Transaction
9      {
10         protected decimal _amount;
11         protected Boolean _success;
12         private Boolean _executed;
13         private Boolean _reversed;
14         private DateTime _dateStamp;
15
16         public Boolean Executed { get => _executed; }
17         public Boolean Success { get => _success; }
18         public Boolean Reversed { get => _reversed; set => _reversed = value; }
19         public DateTime DateStamp { get => _dateStamp; }
20         public decimal Amount { get => _amount; }
21
22         public Transaction(decimal amount)
23         {
24             if (amount > 0)
25             {
26                 _amount = amount;
27             }
28             else
29             {
30                 amount = 0;
31                 throw new ArgumentOutOfRangeException("Amount must be greater than
32                     ↪ 0");
33             }
34         }
35
36         public virtual String GetAccName()
37         {
38             return "Various";
39         }
40
41         public virtual void Print()
42         {
43             Console.WriteLine("Transaction amount: {0} Executed: {1}, Reversed:
44                 ↪ {3}", _amount.ToString("C"), _executed, _reversed);
45         }
46
47         public virtual void Execute()
48         {
49             if (_executed)
50             {
51                 throw new InvalidOperationException("transaction has previously
52                     ↪ occurred");
53             }
54         }
55     }
```

```
51         _dateStamp = DateTime.Now;
52         _executed = true;
53     }
54
55     public virtual void Rollback()
56     {
57         if (_reversed==true)
58         {
59             throw new InvalidOperationException("Transaction has already been
60                 ↪ reversed");
61         }
62         else if (!_executed)
63         {
64             throw new InvalidOperationException("Transaction can not be
65                 ↪ rollback as it was never successfully executed");
66         }
67         _dateStamp = DateTime.Now;
68         _executed = false;
69     }
70 }
71
72
73 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading;
5
6  namespace task7_1
7  {
8
9      class BankSystem
10     {
11         //enum options user can choose from
12         enum MenuOption
13         {
14             AddnewAccount,
15             Withdraw,
16             Deposit,
17             Print,
18             TransferTransaction,
19             Rollback,
20             Quit
21         }
22
23         List<Transaction> transactions = new List<Transaction>();
24
25         static void AddAccount(Bank bank)
26         {
27             Console.WriteLine("Enter the name of the account holder");
28             string name = Console.ReadLine();
29             Console.WriteLine("Enter their starting account balance");
30             decimal balance = Convert.ToDecimal(Console.ReadLine());
31             bank.AddAccount(new Account(balance, name));
32         }
33
34         //find account
35         private static Account FindAccount(Bank bank)
36         {
37             Account account = null;
38             Console.WriteLine("Enter the account name");
39             string name = Console.ReadLine();
40             account = bank.GetAccount(name);
41             if (account == null)
42             {
43                 Console.WriteLine("Account does not exist");
44             }
45             return account;
46         }
47
48         //Takes an amount to deposit
49         static void DoDeposit(Bank bank)
50         {
51             Account account = FindAccount(bank);
52             Console.WriteLine("Enter deposit amount: ");
53             decimal amount = Convert.ToDecimal(Console.ReadLine());
```

```
54         DepositTransaction deposit = new DepositTransaction(account, amount);
55
56         if (account != null)
57         {
58             string execute;
59             string rollback;
60
61             Console.WriteLine($"Deposit the given amount: ${amount}. Y/N? ");
62             execute = Console.ReadLine();
63             Console.WriteLine("Rollback the transaction. Y/N? ");
64             rollback = Console.ReadLine();
65             if (execute == "Y".ToLower())
66             {
67                 bank.Execute(deposit);
68             }
69             if (rollback == "Y".ToLower())
70             {
71                 deposit.Rollback();
72             }
73         }
74     }
75 }
76
77 //takes an amount to withdraw
78 static void DoWithdrawal(Bank bank)
79 {
80     Account account = FindAccount(bank);
81     Console.WriteLine("Enter withdrawal amount: ");
82     decimal amount = Convert.ToDecimal(Console.ReadLine());
83     WithdrawTransaction withdrawal = new WithdrawTransaction(account,
84         ↪ amount);
85
86     if (account != null)
87     {
88         string execute;
89         string rollback;
90
91         Console.WriteLine($"Withdraw the given amount: ${amount}. Y/N? ");
92         execute = Console.ReadLine();
93         Console.WriteLine("Rollback the transaction. Y/N? ");
94         rollback = Console.ReadLine();
95         if (execute == "Y".ToLower())
96         {
97             bank.Execute(withdrawal);
98         }
99         if (rollback == "Y".ToLower())
100         {
101             withdrawal.Rollback();
102         }
103     }
104 }
105
```

```
106     }
107
108     //takes an amount to transfer
109     static void DoTransfer(Bank bank1)
110     {
111         Account fromaccount = FindAccount(bank1);
112         Account toaccount = FindAccount(bank1);
113         Console.WriteLine("Transfer the following amount: ");
114         decimal amount = Convert.ToDecimal(Console.ReadLine());
115         TransferTransaction transfer = new TransferTransaction(fromaccount,
116             ↪ toaccount, amount);
117
118         if (bank1 != null)
119         {
120             string execute;
121             string rollback;
122
123             Console.WriteLine($"Transfer the given amount: ${amount}. Y/N? ");
124             execute = Console.ReadLine();
125             Console.WriteLine("Would you like to rollback the transfer. Y/N? ");
126             rollback = Console.ReadLine();
127             if (execute == "Y".ToLower())
128             {
129                 bank1.Execute(transfer);
130             }
131             if (rollback == "Y".ToLower())
132             {
133                 transfer.Rollback();
134             }
135             bank1.Print();
136         }
137     }
138
139     //rolls backs a specified transaction
140     static void DoRollback(Bank bank)
141     {
142         bank.Print();
143         Console.WriteLine("Enter transaction to rollback or press 0 to exist:
144             ↪ ");
145         int rollback = Convert.ToInt32(Console.ReadLine());
146         if (rollback != 0)
147         {
148             bank.Rollback(bank.getList()[rollback - 1]);
149         }
150     }
151
152     static MenuOption ReadUserOption()
153     {
154         int? option = null;
155         do
156         {
```



```
156         Console.WriteLine("-----");
157         Console.WriteLine("Please select from the following options");
158         Console.WriteLine("MENU \n1. Add new Account \n2. Withdraw \n3.
        ↳ Deposit \n4. Print \n5. Transfer Transaction \n6. Rollback
        ↳ \n7. Quit");
159         Console.WriteLine("-----");
        Console.WriteLine("-----");
160
161     try
162     {
163         option = Convert.ToInt32(Console.ReadLine());
164         if (option > 7 || option < 1)
165         {
166             option = null;
167             Console.WriteLine("Please enter a number from 1 to 7
        ↳ from the menu");
168         }
169     }
170     catch (FormatException)
171     {
172         Console.WriteLine("Invalid input. Enter an integer from
        ↳ 1-7");
173     }
174
175     } while (option == null);
176
177     return (MenuOption)option;
178 }
179
180
181 static void DoPrint(Bank bank)
182 {
183     Account account = FindAccount(bank);
184     if (account != null)
185     {
186         account.Print();
187     }
188 }
189
190
191
192 static void Main(string[] args)
193 {
194
195
196     Bank newbank = new Bank();
197
198     Account person_one = new Account(100, "Jon");
199     Account person_two = new Account(150, "Amy");
200     MenuOption option;
201
202     do
```

```
203     {
204         option = ReadUserOption() - 1;
205
206         switch (option)
207         {
208             case MenuOption.AddnewAccount:
209                 AddAccount(newbank);
210                 break;
211             case MenuOption.Withdraw:
212                 Console.WriteLine("You have selected withdraw");
213                 DoWithdrawal(newbank);
214                 break;
215             case MenuOption.Deposit:
216                 Console.WriteLine("You have selected deposit");
217                 DoDeposit(newbank);
218                 break;
219             case MenuOption.Print:
220                 Console.WriteLine("You have selected print");
221                 DoPrint(newbank);
222                 break;
223             case MenuOption.TransferTransaction:
224                 DoTransfer(newbank);
225                 break;
226             case MenuOption.Rollback:
227                 DoRollback(newbank);
228                 break;
229             case MenuOption.Quit:
230                 Console.WriteLine("Goodbye");
231                 break;
232         }
233     } while (option != MenuOption.Quit);
234
235
236 }
237
238
239 }
240
241
242 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task7_1
6  {
7      public class Bank
8      {
9          private static List<Account> _accounts = new List<Account>();
10         private List<Transaction> _transactions = new List<Transaction>();
11
12
13         public void AddAccount(Account account)
14         {
15             _accounts.Add(account);
16         }
17
18
19         public List<Transaction> getTheList()
20         {
21             return _transactions;
22         }
23
24         public Account GetAccount(String name)
25         {
26             foreach(Account account in _accounts)
27             {
28                 if(account.getName().ToLower()==name.ToLower())
29                 {
30                     return account;
31                 }
32             }
33             { return null; }
34         }
35         //Add and Execute transaction
36         public void Execute(Transaction transaction)
37         {
38             _transactions.Add(transaction);
39
40             transaction.Execute();
41
42         }
43
44         //Rollback
45         public void Rollback(Transaction transaction)
46         {
47             try
48             {
49                 transaction.Rollback();
50             }
51             catch(InvalidOperationException exception)
52             {
53                 Console.WriteLine("Unable to perform the Rollback");
54             }
55         }
56     }
57 }
```

```
54         Console.WriteLine("The error was: " + exception.Message);
55     }
56 }
57
58 //print transaction history
59 public void Print()
60 {
61     int index = 1;
62     Console.ForegroundColor = ConsoleColor.Black;
63     Console.WriteLine(new string(' ', 10) + "Banking System");
64     Console.WriteLine(new string(' ', 10) + new string('-', 104));
65     Console.WriteLine(new string(' ', 10) + new string('-', 104));
66     foreach(var transaction in _transactions)
67     {
68         string type = "test";
69         if (transaction.GetType().Name.Equals("DepositTransaction"))
70         {
71             type = "Deposit";
72         }
73         if (transaction.GetType().Name.Equals("WithdrawTransaction"))
74         {
75             type = "Withdraw";
76         }
77         if (transaction.GetType().Name.Equals("TransferTransaction"))
78         {
79             type = "Transfer";
80         }
81
82         Console.ForegroundColor = ConsoleColor.Black;
83         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"));
84
85
86     }
87 }
88
89
90
91 }
92 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Linq;
5  namespace task7_1
6  {
7      public class Account
8      {
9          //instance variables declared
10         private decimal _balance;
11         private string _name;
12
13         //constructor
14         public Account(decimal balance, string name)
15         {
16
17             this._name = name;
18             if (balance <= 0)
19                 return;
20             this._balance = balance;
21
22         }
23
24         //prints name and balance
25         public void Print()
26         {
27             Console.WriteLine("The name of the account holder is " + getName() +
28                 ↪ "\nCurrent Account Balance is $" + getBalance());
29         }
30
31         //returns name
32         public String getName()
33         {
34             return this._name;
35         }
36
37         //returns balance
38         public decimal getBalance()
39         {
40             return this._balance;
41         }
42
43         //increases balance by adding deposit
44         //boolean ensure that no 0 or a negative value can not be input
45         public Boolean Deposit(decimal amount)
46         {
47             if (amount <= 0)
48                 return false;
49
50             this._balance += amount;
51             return true;
52         }
53     }
```

```
53     }
54
55
56     //decreases balanace by withdrawing the giving input amount
57     //boolean values protect against overdrawing from account
58     public Boolean Withdraw(decimal amount)
59     {
60         if (amount > this._balance || amount < 0)
61             return false;
62
63
64         this._balance -= amount;
65         return true;
66     }
67
68
69 }
70 }
```

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:

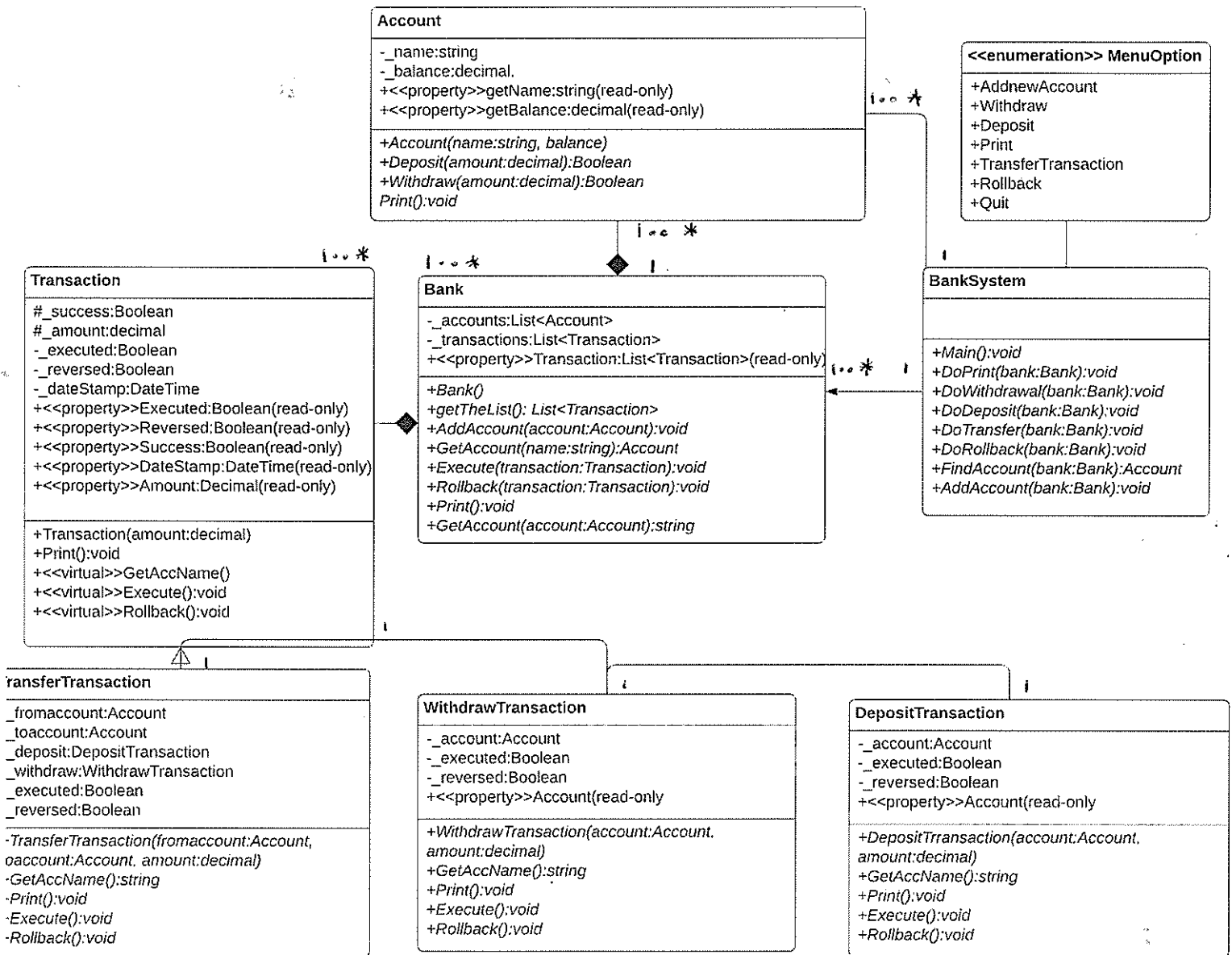
Sanjay SEGU

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](#) [< Previous](#)

 **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](#)

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](#)

Useful resource for debugging

DALE ORDERS 10 April, 2020 4:04 PM

Rating

🗨️★★★★★

Average Rating

★★★★★

(2 ratings)

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=d6lYH8Ro9aI&t=1217s>

<https://www.youtube.com/watch?v=G1S6NZfFvOg&t=850s>

Hopefully they help,

Dale

Reply

Edit Post

More Actions ▼

★ [Subscribe](#) ☆ [Unsubscribe](#)

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](#)
[Next >](#)

Rating

Average Rating

(0 ratings)

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](#)
[Next >](#)

Rating

Average Rating

(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 dividing 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](#)
[Next >](#)

Rating

Average Rating

(0 ratings)

illegal name?

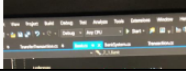
DALE ORDERS 17 March, 2020 3:38 PM

Yes, you can not use a space in the title.

[See previous replies](#)

DO **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'?

AA **ARRIB AHMED** 5/29 3:22 AM



DO **DALE ORDERS** 5/29 3:23 AM ❤️ 1
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?'

It will only roll back if the user enters a 'y'. So also write 'press y to proceed.'

AA **ARRIB AHMED** 5/29 3:27 AM
yeah man just fixed that, lets see how it runs

appreciate the help man

thats all sorted

DO **DALE ORDERS** 5/29 3:29 AM
all good

DO **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

AA **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

DO **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.

AA **ARRIB AHMED** 5/29 3:54 AM
yeah so why does it say "enter an account name" after getting done with a transaction

DO **DALE ORDERS** 5/29 3:55 AM
Is this in your main method
It needs to be in your main method

▼ [Collapse all](#)

DO **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

That looks great. I would also recommend this video <https://www.youtube.com/watch?v=R8Bl5c-Vi4&t=838s>. He provides source code in the decription

1

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

Hey Jesse.

1

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