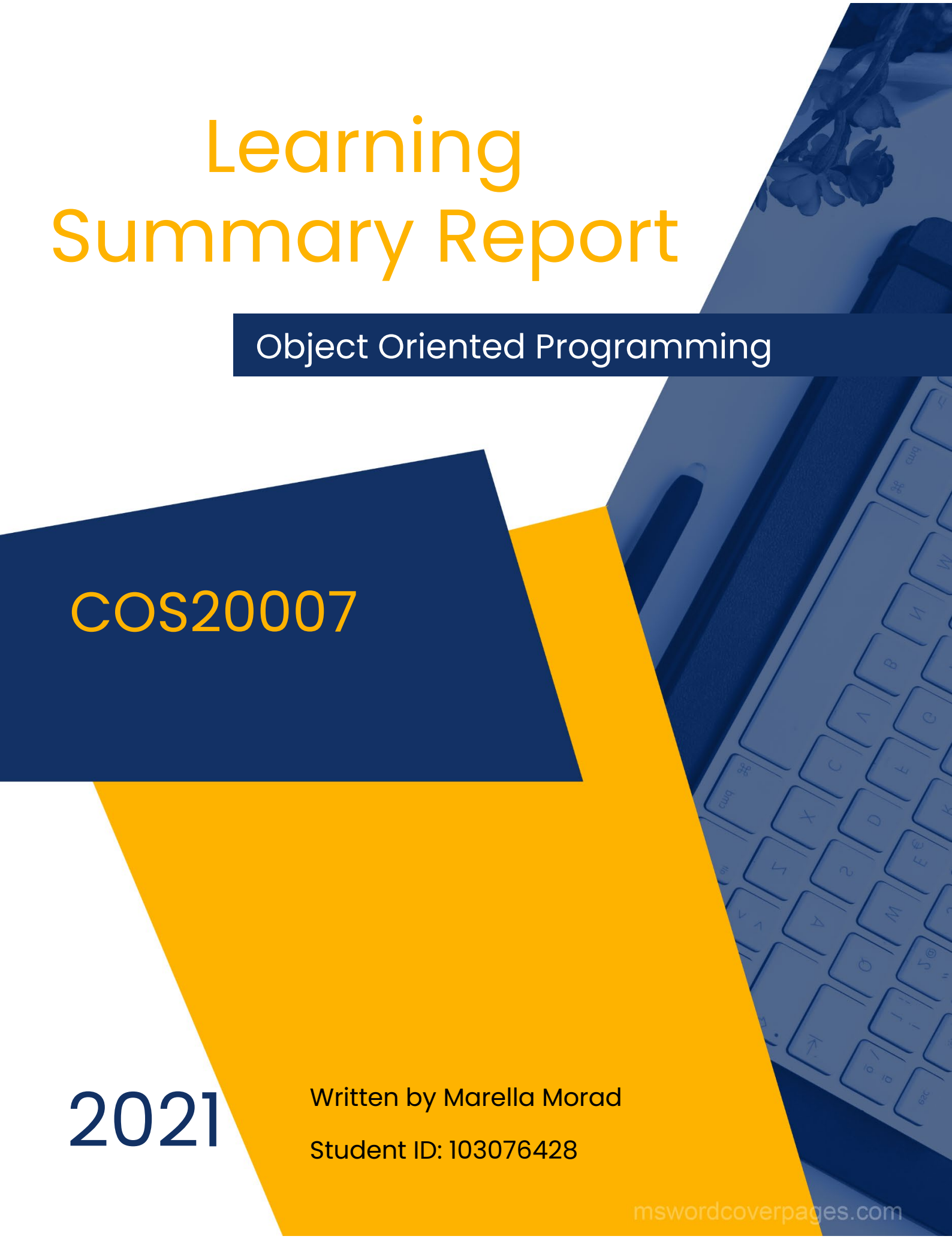
Learning Summary Report

Object Oriented Programming



COS20007

2021

Written by Marella Morad

Student ID: 103076428

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	✓
Test is Complete in Doubtfire	✓
C# programs that demonstrate coverage of core concepts	✓
Explanation of OO principles	✓
All Pass Tasks are Complete on Doubtfire	✓

Minimum Pass Checklist

	Included
All Credit Tasks are Complete on Doubtfire	✓

Minimum Credit Checklist (in addition to Pass Checklist)

	Included
Distinction tasks (other than Custom Program) are Complete	✓
Custom program meets Distinction criteria & Interview booked	✓
Design report has UML diagrams and screenshots of program	✓

Minimum Distinction Checklist (in addition to Credit Checklist)

	Included
HD Project included	
Custom project meets HD requirements	✓

Minimum High Distinction Checklist (in addition to Distinction Checklist)

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: **Marella Morad**

Portfolio Overview

This portfolio includes work that demonstrates that I have achieved all Unit Learning Outcomes for COS20007 Object Oriented Programming to an **HD** level.

Summary of the Tasks Included:

This semester, I completed the following tasks:

- Pass Task 1.1 - Object Oriented Hello World
- Pass Task 1.2: C# Reference Sheet
- Pass Task 2.1: Counter Object
- Pass Task 2.2: Drawing Program – Shapes
- Pass Task 2.3: Case Study Iteration 1
- Pass Task 3.1: Drawing Program – Drawing
- Pass Task 3.2: Clock
- Pass Task 4.1: Drawing Program - Different Shapes
- Pass Task 4.2: Case Study Iteration 2
- Pass Task 5.1: Case Study Iterations 3 and 4
- Credit Task 5.2: Drawing Program – Saving
- Distinction Task 5.3: Drawing Program - Flexible Saving
- Credit Task 6.1: Case Study - Iterations 5 and 6
- Credit Task 6.2: Custom Program Plan and Design
- High Distinction Task 6.4: HD Custom Program – **Super Mario 29.11**
- Pass Task 7.1: Object-Oriented Principles
- Credit Task 7.2: Object-Oriented Programming Concept Map
- Distinction Task 7.3: Case Study - Iterations 7 and 8
- Semester Test
- Distinction Task 9.1: Case Study - Take, GUI and DLL
- Pass Task 10.1: Clock in Another Language
- Distinction Task 10.2: Other Language Report

Evidence of Covering the Learning Outcomes:

My purpose in this section is to showcase how my pieces illustrate my understanding of each of the unit's learning outcomes.

Learning Outcome 1

Explain the principles of the object-oriented programming paradigm specifically including abstraction, encapsulation, inheritance, and polymorphism.

I have demonstrated a deep understanding of the object-oriented programming paradigm which is evident in tasks 7.1P, 7.2C, 8.1P, 5.3D, 9.1D and 6.4HD. In task 7.1P, I included a description of each of the principles along with an example showcasing how to apply them. I created a concept map in 7.2C that explains the relationships, correlations, and descriptions of Object-Oriented Programming in general and the four OOP concepts in particular. Furthermore, in task 8.1P, the semester test, I covered abstraction and polymorphism in relation to the Library solution I created for the task. Moreover, in task 5.3D, I applied inheritance in the child classes that derive from the abstract Shape class (MyRectangle, MyCircle, and MyLine). Other than explaining each of the principles, I practised applying them in the programs I created, beginning with the 1.1P task, which demonstrates encapsulation and progressing to SwinAdventure and Super Mario 29.11 (9.1D and 6.4HD), in which I applied the four principles with increasing complexity.

Learning Outcome 2

Use an object-oriented programming language, and associated class libraries, to develop object-oriented programs.

For this unit, we used C# as the primary programming language to investigate OO programming. Coming from a C# background, it was fascinating to see the other approach, the OOP approach, to developing applications. With the language syntax being the same, it was easier for me to implement the programs in each of the given portfolio tasks, given that I did not need to learn the language from scratch. I started with a simple program, task 1.1P, where I used C#'s System library to print messages on the application Console. As we progressed through the unit, we started using other libraries such as System.Collections.Generic to be able to use Lists in C#. Furthermore, we used an external library called SplashKit to draw on the screen. I used this library in all the Drawing tasks (2.2P, 3.1P, 4.1P, 5.2C and 5.3D) as well as my custom program, 6.4HD, where I learnt and applied new commands from the library. Furthermore, I developed a C# reference sheet that contains the majority of C# language syntax used in this portfolio (Task 1.2P). I also explored OOP in C# further when I researched the similarities and differences between C# and Python as OOP languages in task 10.2D as well as the concept map in 7.2C where I explored the use of classes to represent objects in OOP and how to encapsulate methods and fields inside a class which is accomplished using variable and method scope in C# (public, private, static, protected, etc....). In addition to exploring Python as an OOP language, I implemented OOP in Java for task 10.1P.

Learning Outcome 3

Design, develop, test, and debug programs using object-oriented principles in conjuncture with an integrated development environment.

I have designed, developed, tested, and debugged programs using OO principles in combination with Visual Studio as my IDE. I explored some different types of solutions that can be created using Visual Studio, such as Console and Windows Form applications. I mainly used a console application to develop SwinAdventure classes and associations, which I then built as a dll and included as a part of my 9.1D task, which is a WinForm application demonstrating the game, SwinAdventrue. SwinAdventure is also where I applied testing skills using NUnit testing (using C#'s NUnit.Framework) to ensure each of the classes is performing as expected. I have also done a testing phase for my custom program, Super Mario 29.11, where I tested all the functionalities and fixed the ones that had bugs. During this phase, I used the debug option to be able to spot the bug and follow the stack trace to resolve it. Lastly, I used an online compiler to write the Clock code in Java for task 10.1P.

Learning Outcome 4

Construct appropriate diagrams and textual descriptions to communicate the static structure and dynamic behaviour of an object-oriented solution.

Throughout the unit, I was able to develop and demonstrate advanced design skills in the design and documentation of my programs. This is mainly evident in SwinAdventrue iterations, which is where I applied design principles to design the required classes starting from iteration 5 and onwards (including tasks 6.1C, 7.3D and 9.1D). I also applied design skills in re-designing the library program for task 8.1P, to include an abstract class that other classes can inherit from (applying polymorphism). Moreover, I demonstrated advanced design skills in my custom program, where I included new design patterns such as singleton and strategy to make my classes more coherent and specific. To do all of that, I relied on UML class diagrams, which represent the encapsulation of classes and their relationships with other classes. Moreover, I used UML Sequence diagrams to illustrate certain functionalities of classes and how they interact with each other to achieve the requested functionality. These diagrams can be seen as part of tasks 3.2P, 6.1C, 7.1P, 7.3D, 8.1P and 6.4HD). In terms of documentation, I demonstrated XML documentation skills in task 5.3D, where I stored objects in a text file, which can then be read and re-drawn on the screen using SplashKit.

Learning Outcome 5

Describe and explain the factors that contribute to a good object-oriented solution, reflecting on your own experiences and drawing upon accepted good practices.

After undergoing this unit, I have learnt that a good object-oriented solution applies all four principles correctly. Object-oriented programs are based on creating specialised classes and defining the associations between them to develop the final product. Other than having one class responsible for all the functionalities in the program, roles and responsibilities are broken down into smaller roles that single classes are responsible for. I have applied this observation in my custom program, as I designed and implemented a class called Resize, and another class called Direction which are responsible for enlarging, shrinking and identifying the direction the player is facing (see Super Mario 29.11 for further details). Initially, I planned to include all these roles into the Player class, however, I then decided to refactor these roles into smaller classes that can work in association with the player class. Another important feature of OOP that I realized is that using the concept of inheritance, for example, makes the program highly modular, allowing other classes with similar roles to be added and inherit from the parent class without the need to rewrite large areas of the program.

Reflection

The most important things I learnt:

The most important thing that I learned from this unit is how applying the four object-oriented principles can significantly decrease, if not remove, the need to duplicate code. In the past, I applied procedural programming using C# in which was challenging to avoid code duplication. However, in object-oriented programming, we can apply the inheritance concept to allow multiple classes to have the same property or even override some properties, if necessary, without the need to duplicate code. Another main thing I learned from this unit is the interpretation and design of UML class and sequence diagrams, and how extremely important it is to thoroughly think about a program before beginning to write its code. Even if the plan is incomplete, it is better to have a starting point, which is what I had when I started developing my custom program. Even though my UML class and sequence diagrams look a lot different from my initial plan, having that plan is what allowed me to get to this stage. Lastly, I really found the test-driven programming approach helpful as I could define multiple test cases that revolve around the main functionality of a class and write the class from there using the assistance from the test, NUnit tests in this unit.

The things that helped me most were:

- Lectures (live and recordings): these are where I gained the most knowledge about OOP from. Lectures and lecture notes provided me with a great reference to OO principles which I carried out and applied throughout the semester.
- Lab demonstrations: throughout the semester, my tutor demonstrated key concepts from the unit, especially NUnit testing and UML diagrams and associations which I found very helpful given that I had no prior experience with object-oriented programming.
- Live Consultations: these define the best experience I had in a unit at university as I have access to tutors almost all the time throughout the week for consultation regarding my progress through the unit, my programs, any errors that I could not figure out. The feedback I received from tutors/unit convenor really helped me develop a better understanding of the OOP application as well as following food practices in designing and developing my programs.
- Google, Stack Overflow, and other online resources: I was able to use these resources to find examples that apply similar ideas to what I needed to develop. One main thing that I research and found really helpful online resources about was applying physics to make my character jump (projectile motions, using speed and gravity), which I was then able to apply in my custom program with the help of my tutor.

I found the following topics particularly challenging:

- At the start of this semester, I found NUnit testing, and UML diagrams interpretation very challenging. Initially, I had to keep looking at existing diagrams, and test cases to be able to write my own, however, as time went by, and I got to practice writing my own tests and drawing my own diagrams more, I have become more confident in these two areas.
- Overall, I found shifting from a procedural programming approach to an object-oriented programming approach really challenging as I usually tend to write large classes with small methods/procedures rather than having separate classes performing more specified functionalities.

I found the following topics particularly interesting:

- I found game development, which I applied in my custom program, very interesting. The fact that I had to think about every single movement the player does was frustrating at times, however, very interesting and very rewarding when I was able to make the functionality work.
- I also found NUnit testing interesting along with the fact that we can compare our expected results to the actual results.

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

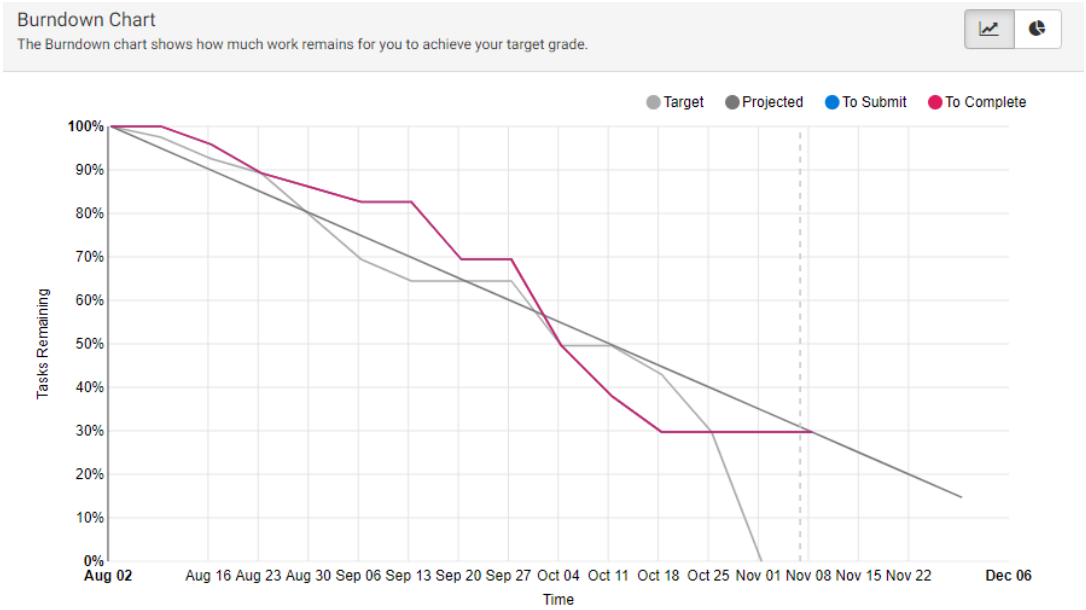
- The four object-oriented principles including encapsulation, abstraction, inheritance, and polymorphism, which I have applied in almost all my tasks as mentioned earlier in Learning Outcome 1.
- Draw.io tool which I extensively used to design my concept map, UML class and sequence diagrams for the tasks mentioned in Learning Outcome 4.
- I also learnt about good OO programming practices in C#, especially how a foreach loop is favoured over a for or a while loop and I experienced the difference in my custom program. Initially, I was trying to implement jumping functionality using a for loop, which did not work as when a for loop is run, other parts of the code cannot run with it simultaneously, resulting in the game stopping for a while to do the jump, then running again for other actions.

I still need to work on the following areas:

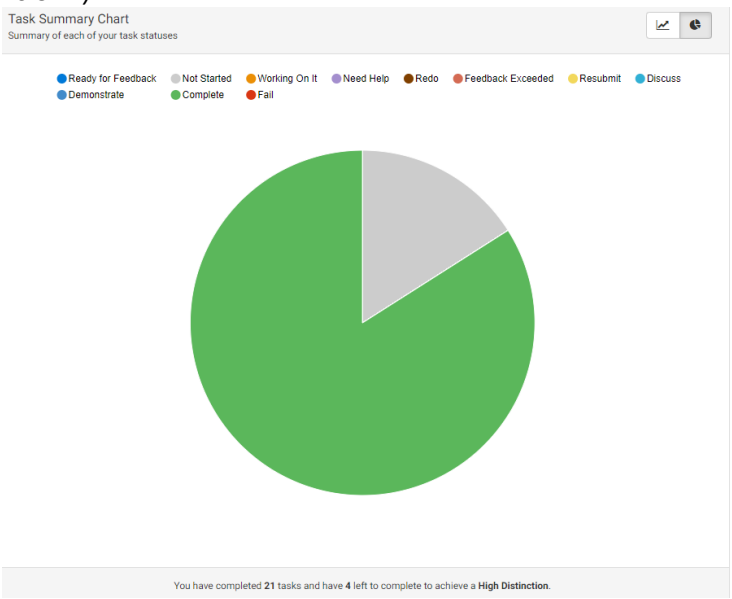
- Writing programs using a fully object-oriented approach: as I mentioned earlier, I have the habit to follow procedural programming which causes my classes to get larger, however, I am working on making them more specific and creating new classes/interfaces that other classes can inherit and implement features from (for example having multiple block classes that inherit from an abstract Block class with each having its specific bitmap and type)
- Drawing accurate UML class and sequence diagrams: I think now I am at a good stage with drawing my diagram, nevertheless, I believe more practice is required to understand the relationships more deeply and be able to apply them.
- Design patterns: I think putting the extra time into learning and applying new design patterns is essential to be able to produce a truly object-oriented program. I have already started learning about new patterns such as singleton and strategy which I applied in my custom program, but I think in the future, some other patterns can also be helpful in this context.

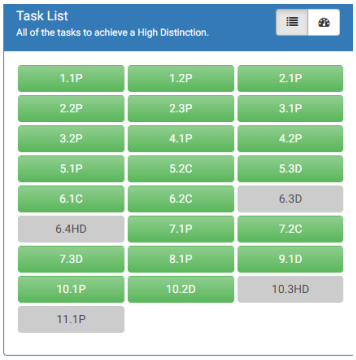
My progress in this unit was:

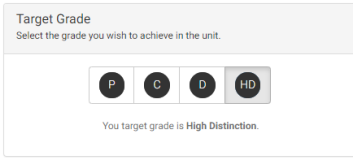
My progress in this unit was consistent, as I was up to date with the tasks, having almost all the tasks done by the target date. After completing SwinAdventure, I shifted my focus to my custom program and started the design stage of it, whilst working on the conceptual tasks such as 7.1P, 7.2C, 8.1P and 10.2D. Overall, I believe I gave myself a good amount of time to complete each of the tasks, however, I could have used some more time on my custom program.



As shown in the pie chart below, I have 84% of my tasks marked as complete with only the custom program and the learning summary report draft missing. *I will not be submitting an HD project (task 10.3HD).*







This unit will help me in the future:

Having learnt and applied the object-oriented programming approach, I believe will really help me in future jobs as well as future units applying similar principles. This unit also showcased that programming does not only differ from a language to another but also there are different styles of programming which I am really keen to explore. One major thing that I think will be really helpful in the future is testing, and the approach of test-driven programming, as the testing phase is one of the most important phases in programming to ensure meeting required functionalities. As a future Software Engineer, having this skill will be very helpful to me.

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

- One thing that I will utilise more if I do the unit again is the help desk. This semester I approach the unit relying mostly on myself and the internet, which was time-consuming, and could have been done a lot quicker if I consulted with a tutor or even a classmate.
- I would also practice taking my programs to the next level of efficiency, applying the cores of object-oriented programming.

Conclusion

Overall, I feel I have demonstrated that my portfolio is adequate to receive an HD.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Marella Morad

Portfolio Submission

Submitted By:
Marella Morad
103076428

Tutor:
Joshua Wright

November 7, 2021

Contents

2 Overall Task Status

Task	Status	Times Assessed
Pass Task 1.1 - Object Oriented Hello World	Complete	1
Pass Task 1.2: C# Reference Sheet	Complete	2
Pass Task 2.1: Counter Object	Complete	1
Pass Task 2.2: Drawing Program - Shapes	Complete	2
Pass Task 2.3: Case Study Iteration 1	Complete	2
Pass Task 3.1: Drawing Program - Drawing	Complete	1
Pass Task 3.2: Clock	Complete	1
Pass Task 4.1: Drawing Program - Different Shapes	Complete	2
Pass Task 4.2: Case Study Iteration 2	Complete	2
Pass Task 5.1: Case Study Iterations 3 and 4	Complete	2
Credit Task 5.2: Drawing Program - Saving	Complete	1
Distinction Task 5.3: Drawing Program - Flexible Saving	Complete	1
Credit Task 6.1: Case Study - Iterations 5 and 6	Complete	1
Credit Task 6.2: Custom Program Plan and Design	Complete	1
Distinction Task 6.3: Custom Program	Not Started	
High Distinction Task 6.4: HD Custom Program	Ready to Mark	0
Pass Task 7.1: Object Oriented Principles	Complete	1
Credit Task 7.2: Object Oriented Programming Concept Map	Complete	1
Distinction Task 7.3: Case Study - Iterations 7 and 8	Complete	3
Semester Test	Complete	1
Distinction Task 9.1: Case Study - Take, GUI and DLL	Complete	2
Pass Task 10.1: Clock in Another Language	Complete	1
Distinction Task 10.2: Other Language Report	Complete	1
High Distinction Task 10.3: HD Project	Not Started	
Pass Task 11.1: Draft Learning Summary Report	Not Started	

3 Learning Outcomes

4 Pass Task 1.1 - Object Oriented Hello World

As always, ”Hello World” is the first program you should write in a new language or with a new set of tools. In this task you will create an object oriented version of this classic program.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 1.1 - Object Oriented Hello World

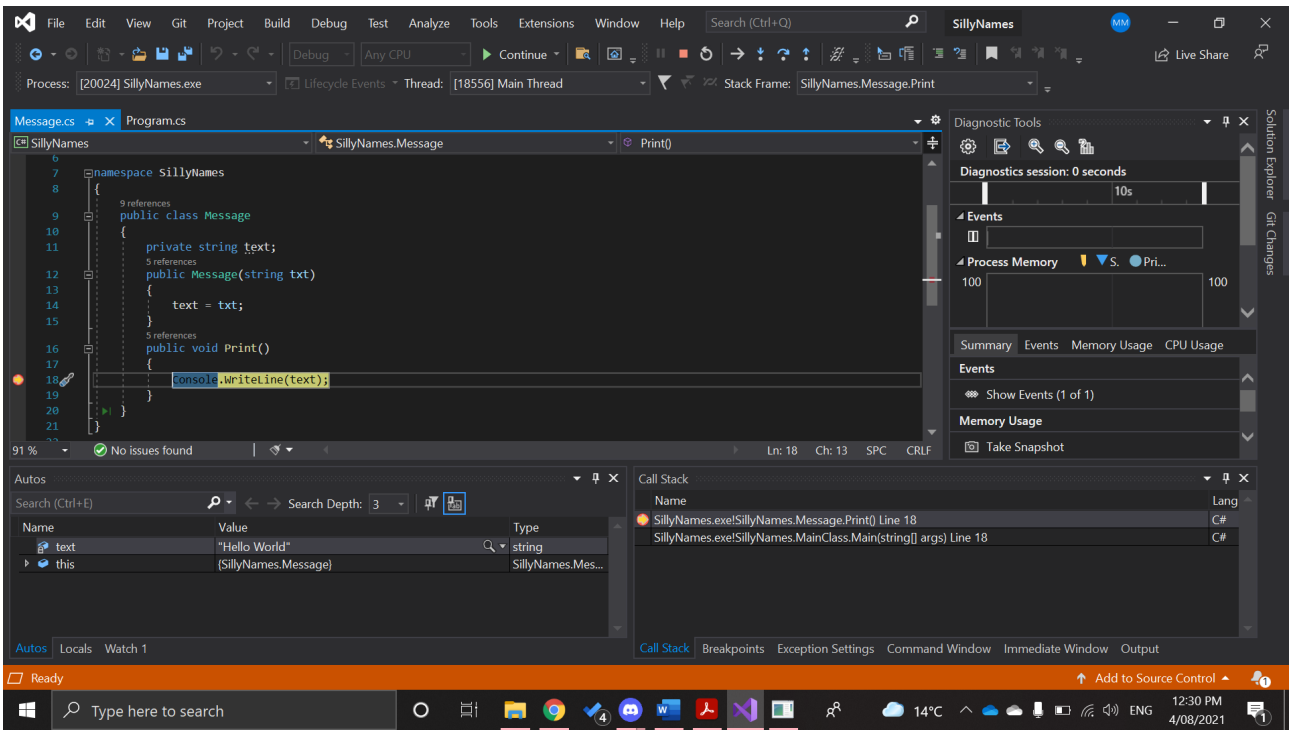
Submitted By:
Marella Morad
103076428
2021/08/04 12:43

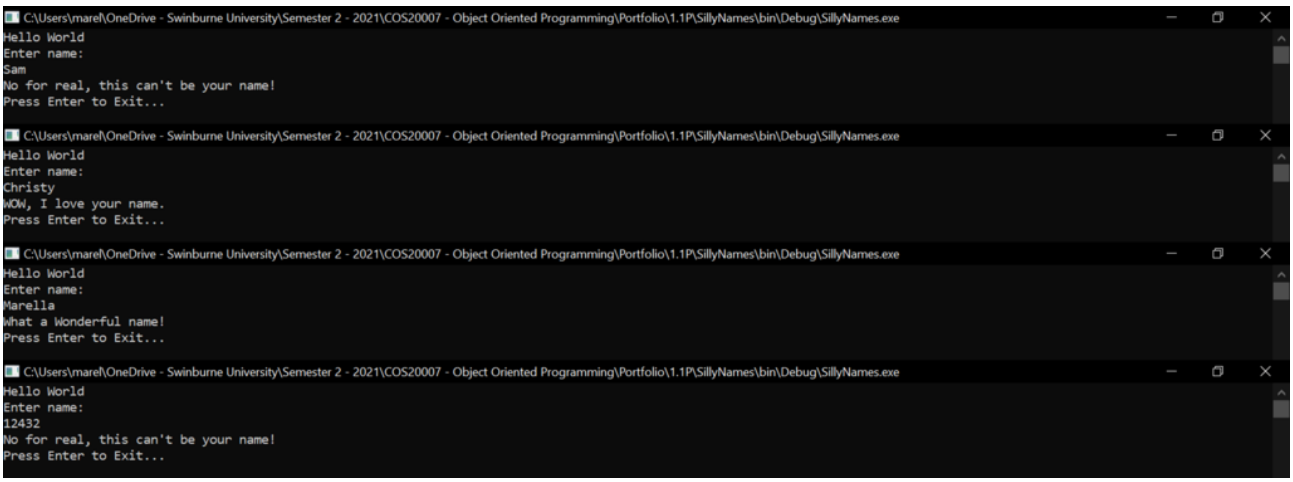
Tutor:
Joshua Wright

August 4, 2021

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

namespace SillyNames
{
    public class Message
    {
        private string text;
        public Message(string txt)
        {
            text = txt;
        }
        public void Print()
        {
            Console.WriteLine(text);
        }
    }
}
```

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

namespace SillyNames
{
    class MainClass
    {
        static void Main(string[] args)
        {
            Message myMessage;
            Message[] messages = new Message[4];
            string name;

            myMessage = new Message("Hello World");
            myMessage.Print();

            messages[0] = new Message("What a Wonderful name!");
            messages[1] = new Message("WOW, I love your name.");
            messages[2] = new Message("Such an interesting name, what does it
             ↪  mean?");
            messages[3] = new Message("No for real, this can't be your name!");

            Console.WriteLine("Enter name: ");
            name = Console.ReadLine();

            if (name.ToLower() == "marella")
            {
                messages[0].Print();
            }
            else if (name.ToLower() == "christy")
            {
                messages[1].Print();
            }
            else if (name.ToLower() == "sama")
            {
                messages[2].Print();
            }
            else
            {
                messages[3].Print();
            }

            //added these two lines because the program would perform everything
             ↪  and close before being able to read the output
            Console.WriteLine("Press Enter to Exit...");
            Console.ReadLine();
        }
    }
}
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
SillyNames
Live Share
Process: [20024] SillyNames.exe
Thread: [18556] Main Thread
Stack Frame: SillyNames.Message.Print
Message.cs
Program.cs
SillyNames
SillyNames.Message
Print()
namespace SillyNames
{
9 references
public class Message
{
private string text;
5 references
public Message(string txt)
{
text = txt;
}
5 references
public void Print()
{
Console.WriteLine(text);
}
}
}
Diagnostic Tools
Diagnostics session: 0 seconds
10s
Events
Process Memory
Summary
Events
Memory Usage
CPU Usage
Show Events (1 of 1)
Take Snapshot
No issues found
Ln: 18 Ch: 13 SPC CRLF
Autos
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
text
"Hello World"
string
this
{SillyNames.Message}
SillyNames.Mes...
Call Stack
Name
Lang
SillyNames.exe!SillyNames.Message.Print() Line 18
C#
SillyNames.exe!SillyNames.MainClass.Main(string[] args) Line 18
C#
Autos
Locals
Watch 1
Call Stack
Breakpoints
Exception Settings
Command Window
Immediate Window
Output
Ready
Add to Source Control
Type here to search
14°C
ENG
12:30 PM
4/08/2021


C:\Users\mareh\OneDrive - Swinburne University\Semester 2 - 2021\COS20007 - Object Oriented Programming\Portfolio\1.1P\SillyNames\bin\Debug\SillyNames.exe
Hello World
Enter name:
Sam
No for real, this can't be your name!
Press Enter to Exit...
C:\Users\mareh\OneDrive - Swinburne University\Semester 2 - 2021\COS20007 - Object Oriented Programming\Portfolio\1.1P\SillyNames\bin\Debug\SillyNames.exe
Hello World
Enter name:
Christy
WOW, I love your name.
Press Enter to Exit...
C:\Users\mareh\OneDrive - Swinburne University\Semester 2 - 2021\COS20007 - Object Oriented Programming\Portfolio\1.1P\SillyNames\bin\Debug\SillyNames.exe
Hello World
Enter name:
Marella
What a Wonderful name!
Press Enter to Exit...
C:\Users\mareh\OneDrive - Swinburne University\Semester 2 - 2021\COS20007 - Object Oriented Programming\Portfolio\1.1P\SillyNames\bin\Debug\SillyNames.exe
Hello World
Enter name:
12432
No for real, this can't be your name!
Press Enter to Exit...

5 Pass Task 1.2: C# Reference Sheet

Object oriented languages like C# build on top of structured programming principles. This means that these languages share many features with languages like Pascal and C. To help you get started you need to learn the new syntax for the C# language.

Date	Author	Comment
2021/08/11 11:13	Joshua Wright	String Construction from other types is incorrect/incomplete. You need to demonstrate the use of the in-built c# method ToString().

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 1.2: C# Reference Sheet

Submitted By:
Marella Morad
103076428
2021/08/11 11:19

Tutor:
Joshua Wright

August 11, 2021

C# Programming Reference Sheet

Built In Data Types & Literals

Integers
```
Byte, Short, Int, Long
(eg: 1, 2, 13, 156)
```
Floating Point Numbers
```
Float, Double, Decimal
(eg: 1.7, 7.9, 3.4)
```
Strings and Characters
```
String, Char (eg: "Okay", 'K')
```
Boolean
```
Bool (eg: True, False)
```

Working with Strings

Assignment (giving a string a value)
```
name = "Marella";
```
Concatenation (joining strings)
```
fullName = name + " Morad";
```
Comparison
```
if(name == "Marella"){
}
```
Construction from other types:
```
int age = 21
nickname = name + age.ToString();
```

Simple Programming Statements

Constant declaration
```
const double pi = 3.14
```
Variable declaration
```
Int age; string name;
```
Assignment
```
Age = 21; name = "Marella";
```
Method call
```
Console.WriteLine(name + " " + age)
```
Sequence of statements – grouped
```
{…}
```

Structured Programming Statements

If statement
```
if (member == true) {…} else{…}
```
Case statement
```
switch(){case 1:…; break; case 2:…}
```
While loop
```
while(condition){…}
```
Do While loop
```
do{…} while(condition);
```
For loop
```
for(int i = 0; i <= 4; i++) {…}
```

Declaring Methods

Declare a method with parameters:
```
static void Print(string name){
        Console.WriteLine(name);}
```
Declare a method that returns data:
```
static int Sum(int num1, int num2){
        return num1 + num2;}
```
Pass by reference:
```
Sum(ref num1, ref num2);
```

Boolean Operators and Other Statements

Comparison: equal, less, larger, not equal, less eq
```
==, <, >, !=, <=
```
Boolean: And, Or and Not
```
&&, ||, !
```
Skip an iteration of a loop
```
continue;
```
End a loop early
```
break;
```
End a method:
```
return;
```

Custom Types

Classes
```
public class Custom {}
```
Enumerations
```
enum day {
        Sunday,
        Monday}
```
Structs
```
struct point{
        public int x;
        public int y;}
```

Arrays

Declaration
```
int[] nums = new int[2]{12, 16}
```
Access
```
nums[0] = 5; nums[6] = 9;
```
Loop with index i
```
for(int i = 0; i < nums.Length; i++){
nums[i] = nums[i] * nums[i];}
```
For each loop
```
foreach(int n in index){
Console.WriteLine(nums[n];}
```

Programs and Modules

Creating a program
```
namespace Test
{
        class Program
        {
        //what the program does
        }
}
```
Using a class from a library
```
using System;
using SplashKitSDK;
```

Other Things

Reading from Terminal
```
Console.ReadLine();
```
Writing to Terminal
```
Console.WriteLine("Hello World");
```
Comments
```
//single line comment
/*
        Multiline comment
*/
```

6 Pass Task 2.1: Counter Object

In this task you will create a Counter class and use it to create and work with Counter objects.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 2.1: Counter Object

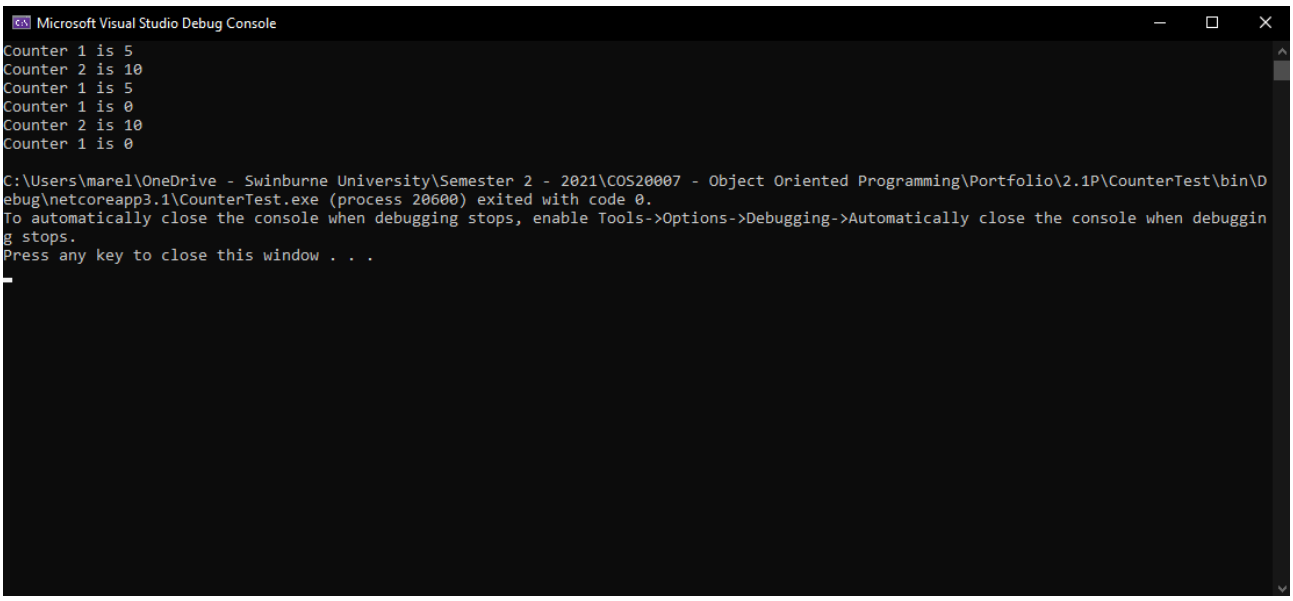
Submitted By:
Marella Morad
103076428
2021/08/18 12:27

Tutor:
Joshua Wright

August 18, 2021

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

namespace CounterTest
{
    public class Counter
    {
        private int _count;
        private string _name;

        public Counter(string name)
        {
            _name = name;
            _count = 0;
        }

        //Name property
        public string Name
        {
            //get is read only
            //get and set are both read and write
            get  //should always return a value
            {
                return _name;
            }
            set //should always change a value
            {
                _name = value;
            }
        }

        //read only property
        public int Ticks
        {
            get
            {
                return _count;
            }
        }

        public void Increment()
        {
            _count = _count + 1;
        }

        public void Reset()
        {
            _count = 0;
        }
    }
}
```

```
using System;

namespace CounterTest
{
    class MainClass
    {
        // PrintCounter static method
        private static void PrintCounters(Counter[] counters)
        {
            foreach (Counter c in counters)
            {
                Console.WriteLine("{0} is {1}", c.Name, c.Ticks);
            }
        }

        public static void Main(string[] args)
        {
            Counter[] myCounters = new Counter[3];
            int i;
            myCounters[0] = new Counter("Counter 1");
            myCounters[1] = new Counter("Counter 2");
            myCounters[2] = myCounters[0];

            for(i = 0; i <= 4; i++)
            {
                myCounters[0].Increment();
            }

            for(i = 0; i <= 9; i++)
            {
                myCounters[1].Increment();
            }

            PrintCounters(myCounters);

            myCounters[2].Reset();

            PrintCounters(myCounters);
        }


    }
}
```


Microsoft Visual Studio Debug Console
Counter 1 is 5
Counter 2 is 10
Counter 1 is 5
Counter 1 is 0
Counter 2 is 10
Counter 1 is 0
C:\Users\marel\OneDrive - Swinburne University\Semester 2 - 2021\COS20007 - Object Oriented Programming\Portfolio\2.1P\CounterTest\bin\Debug\netcoreapp3.1\CounterTest.exe (process 20600) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

Answers to Questions from P2.1

Name: Marella Morad
Student ID: 103076428

How many Counter objects were created?

A total of 2 Counter objects

Variables declared in main() are different to the objects created when we call new. What is the relationship between the declared variables in main and the objects created?

Variables point to objects.

Resetting the counter in myCounters[2] also changes the value of the counter in myCounters[0]. Why does this happen?

myCounter[2] and myCounter[0] are both pointing at the same object since myCounter[2] = myCounter[0], therefore, whatever happens to the object myCounter[2] is pointing at, the same will happen to myCounter[0].

The key difference between memory on the heap compared to the stack and the heap is that the heap holds dynamically allocated memory. What does this mean ?

Dynamic memory allocation means that the amount of storage allocated on heap is determined during runtime of the application by using the operator **new**. Using this method reduces overloading the memory.

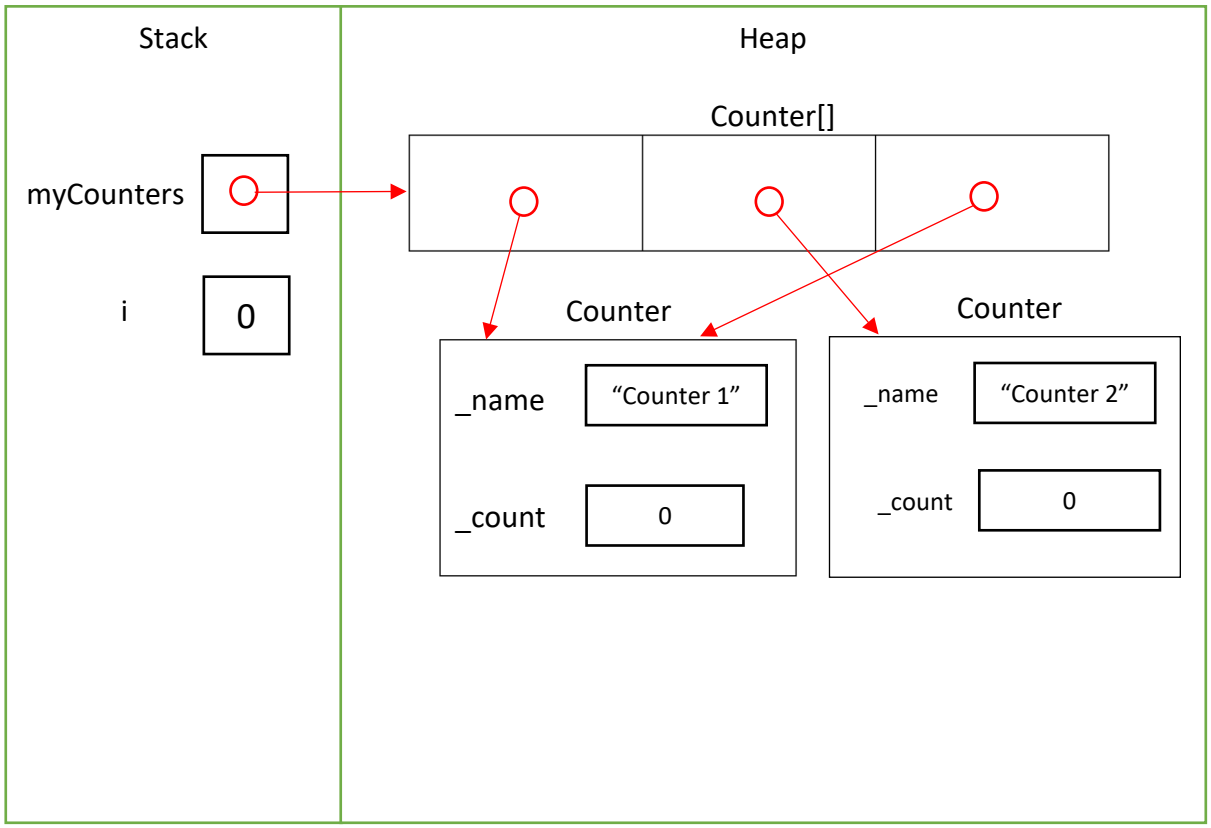
On which are objects allocated (heap or stack) ? On which are local variables allocated (heap or stack) ?

Objects are allocated on the **heap**
Local variables are allocated on the **stack**

What does the new() method do when called for a particular class What does it do and what does it return?

When new is called on a class it *allocates memory for the new object*, *initialises the object by calling the constructor* and then it returns *memory address of the object created*

Draw a diagram showing the locations of the variables and objects in main.



7 Pass Task 2.2: Drawing Program - Shapes

Drawing programs have a natural affinity with object oriented design and programming, with easy to see roles and functionality. In this task you will start to create an object oriented drawing program.

Date	Author	Comment
2021/08/11 11:14	Joshua Wright	Please make sure you follow the UML class diagram correctly and implement all properties as read **and** write properties (get and set) as per the prescribed design.
2021/08/18 15:23	Marella Morad	Demonstration video link
2021/08/18 15:23	Marella Morad	https://www.youtube.com/watch?v=Ntb-sAvTvQeM&ab_channel=MarellaMourad

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 2.2: Drawing Program - Shapes

Submitted By:
Marella Morad
103076428
2021/08/11 11:23

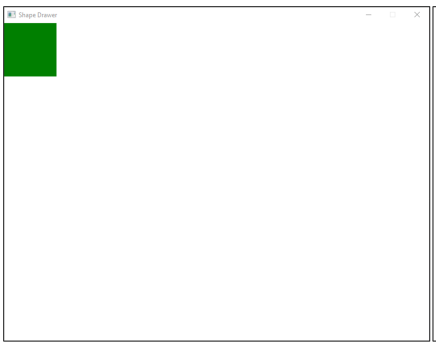
Tutor:
Joshua Wright

August 11, 2021

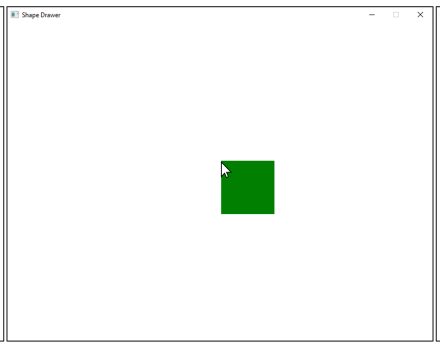
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;

namespace ShapeDrawer
{
    public class Shape
    {
        private Color _color;
        private float _x, _y;
        private int _height, _width;

        //Shape constructor
        public Shape()
        {
            _color = Color.Green;
            _x = 0;
            _y = 0;
            _height = 100;
            _width = 100;
        }

        public Color Color
        {
            get
            {
                return _color;
            }
            set
            {
                _color = value;
            }
        }

        public float X
        {
            get
            {
                return _x;
            }
            set
            {
                _x = value;
            }
        }

        public float Y
        {
            get
            {
```

```
54                  return _y;
55              }
56              set
57              {
58                  _y = value;
59              }
60          }
61
62          public int Width
63          {
64              get
65              {
66                  return _width;
67              }
68              set
69              {
70                  _width = value;
71              }
72          }
73
74          public int Height
75          {
76              get
77              {
78                  return _height;
79              }
80              set
81              {
82                  _height = value;
83              }
84          }
85
86          public void Draw()
87          {
88              SplashKit.FillRectangle(_color, _x, _y, _width, _height);
89          }
90
91          public bool IsAt(Point2D pt)
92          {
93              if((pt.X >= _x && pt.X <= (_x + _width)) && (pt.Y >= _y && pt.Y <= (_y
                ↪  + _height)))
94              {
95                  return true;
96              }
97              else
98              {
99                  return false;
100             }
101         }
102     }
103 }
```

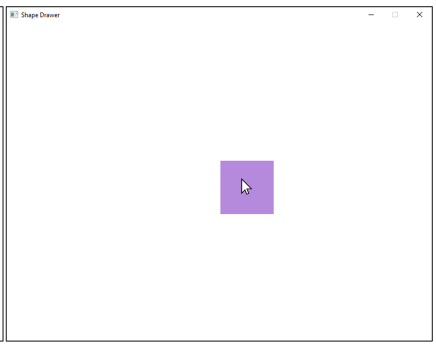
```
using System;
using SplashKitSDK;

namespace ShapeDrawer
{
    public class Program
    {
        public static void Main()
        {
            //Opens a new window called Shape Drawer with dimensions 800 x 600
            new Window("Shape Drawer", 800, 600);

            //create a new Shape variable
            Shape myShape = new Shape();

            //leaves the window open unless requested to be closed
            do
            {
                SplashKit.ProcessEvents(); //allows Splashkit to react to user
                ↪  interactions
                SplashKit.ClearScreen();

                //If the user clicks the LeftButton on their mouse
                if (SplashKit.MouseClicked(MouseButton.LeftButton) == true)
                {
                    //set the shapes x, y to be at the mouse's position
                    myShape.X = SplashKit.MouseX();
                    myShape.Y = SplashKit.MouseY();
                }

                //If the mouse is over the shape (pass the mouse position to the
                ↪  IsAt() method as a 2D point)
                if (myShape.IsAt(SplashKit.MousePosition()))
                {
                    //also check if the user presses the spacebar
                    if (SplashKit.KeyTyped(KeyCode.SpaceKey) == true)
                    {
                        //change the Color of the shape to a random color
                        myShape.Color = SplashKit.RandomRGBColor(255); //alpha =
                        ↪  255 for the color to be non-transparent
                    }
                }

                //draw the rectangle after checking and processing user input
                myShape.Draw();

                 //refresh
                SplashKit.RefreshScreen();
            } while (!SplashKit.WindowCloseRequested("Shape Drawer"));
        }
    }
}
```



Initial position



Changing position according to the mouse's position



Pressing the space bar while IsAt = true (the mouse over the shape) to change its color

8 Pass Task 2.3: Case Study Iteration 1

Object oriented programming makes best sense with larger programs. The case study will be your opportunity to create a larger program and better see how these abstractions make it easier to create software solutions.

Date	Author	Comment
2021/08/18 10:35	Joshua Wright	- Strings in constructor are not correctly converted to lowercase before being added to the list. - FirstID should return an empty string if there are no identifiers in the list.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 2.3: Case Study Iteration 1

Submitted By:
Marella Morad
103076428
2021/08/18 10:58

Tutor:
Joshua Wright

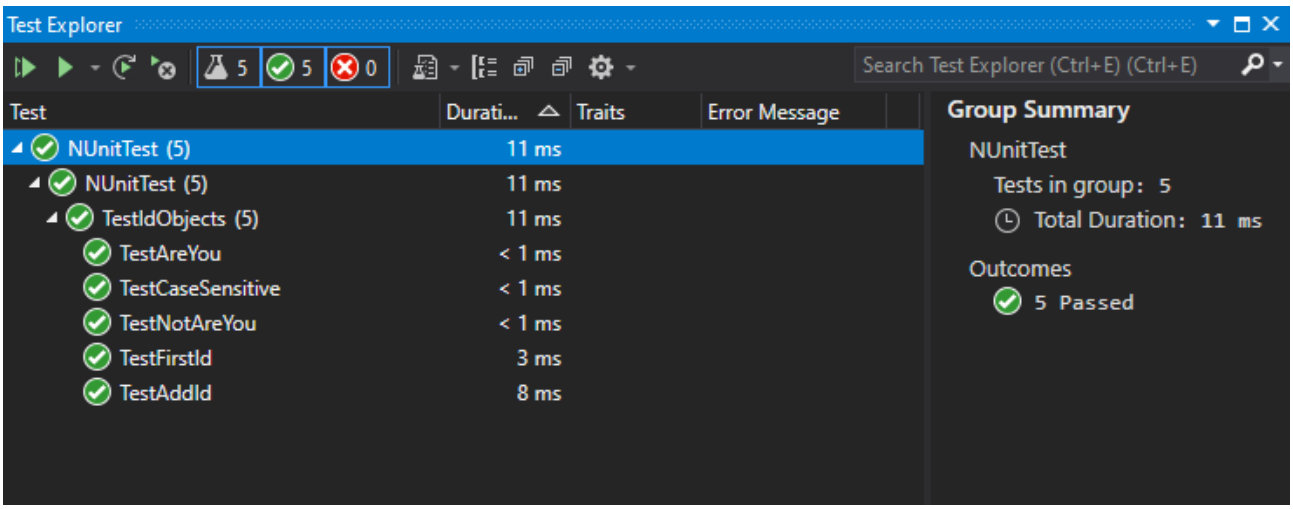
August 18, 2021

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

namespace IterationOne
{
    public class IdentifiableObject
    {
        List<string> _identifiers = new List<string>();

        public IdentifiableObject(string[] idents)
        {
            for(int i = 0; i < idents.Length; i++)
            {
                _identifiers.Add(idents[i].ToLower());
            }
        }

        public bool AreYou(string id)
        {
            foreach(string i in _identifiers)
            {
                if(_identifiers.Contains(id.ToLower()))
                {
                    return true;
                }
            }

            return false;
        }

        //read only, no set property
        public string FirstId
        {
            get
            {
                //returns the first identifier from _identifiers (or an empty
                ↪  string)
                if(_identifiers.Count > 0)
                {
                    return _identifiers[0];
                }
                else
                {
                    return " ";
                }
            }
        }

        public void AddIdentifier(string id)
        {
            //converts the identifier to lower case and stores it in _identifiers
            _identifiers.Add(id.ToLower());
```

```
53            }
54        }
55    }
```

```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using IterationOne;

namespace NUnitTest
{
    [TestFixture]
    public class TestIdObjects
    {
        private IdentifiableObject _testableObject;

        [SetUp]
        public void Setup()
        {
            _testableObject = new IdentifiableObject(new string[] { "fred", "bob" });

        }

        [Test]
        public void TestAreYou()
        {
            Assert.IsTrue(_testableObject.AreYou("fred"), "testing fred");
            Assert.IsTrue(_testableObject.AreYou("bob"), "testing bob");
        }

        [Test]
        public void TestNotAreYou()
        {
            Assert.IsFalse(_testableObject.AreYou("wilma"));
            Assert.IsFalse(_testableObject.AreYou("boby"));
        }

        [Test]
        public void TestCaseSensitive()
        {
            Assert.IsTrue(_testableObject.AreYou("FRED"));
            Assert.IsTrue(_testableObject.AreYou("bOB"));
        }

        [Test]
        public void TestFirstId()
        {
            Assert.AreEqual("fred", _testableObject.FirstId);
        }

        [Test]
        public void TestAddId()
        {
            Assert.IsFalse(_testableObject.AreYou("wilma"));
```

```
53              _testableObject.AddIdentifier("wilma");
54              Assert.IsTrue(_testableObject.AreYou("wilma"));
55          }
56      }
57  }
```


Test Explorer
5
5
0
Search Test Explorer (Ctrl+E) (Ctrl+E)
Test
Durati...
Traits
Error Message
NUnitTest (5) 11 ms
NUnitTest (5) 11 ms
TestIdObjects (5) 11 ms
TestAreYou < 1 ms
TestCaseSensitive < 1 ms
TestNotAreYou < 1 ms
TestFirstId 3 ms
TestAddId 8 ms
Group Summary
NUnitTest
Tests in group: 5
Total Duration: 11 ms
Outcomes
5 Passed

9 Pass Task 3.1: Drawing Program - Drawing

Drawing programs have a natural affinity with object oriented design and programming, with easy to see roles and functionality. In this task you will start to create an object oriented drawing program.

Date	Author	Comment
2021/08/18 15:49	Marella Morad	Demostration video link
2021/08/18 15:49	Marella Morad	https://youtu.be/y_emPyx3w04

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 3.1: Drawing Program - Drawing

Submitted By:
Marella Morad
103076428
2021/08/16 14:10

Tutor:
Joshua Wright

August 16, 2021

```
using SplashKitSDK;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ShapeDrawer
{
    public class Drawing
    {
        //Add a private, read only, field to store the list of _shapes. Use
        ↪  List<Shape> as the type
        private readonly List<Shape> _shapes;

        //_background private field
        private Color _background;

        //public Background property for the background color.
        public Color Background
        {
            get
            {
                return _background;
            }
            set
            {
                _background = value;
            }
        }

        //constructor that takes in the background color as a parameter
        public Drawing(Color background)
        {
            _shapes = new List<Shape>();
            _background = background;
        }

        //default constructor - without parameters
        public Drawing() : this(Color.White)
        {
            _shapes = new List<Shape>();
            _background = Color.White;
        }

        public int ShapeCount
        {
            get
            {
                return _shapes.Count;
            }
        }

```

```
53          public void AddShape(Shape shape)
54          {
55              _shapes.Add(shape);
56          }
57
58          public void Draw()
59          {
60              SplashKit.ClearScreen(Background);
61              foreach (Shape shape in _shapes)
62              {
63                  shape.Draw();
64              }
65          }
66
67          public void SelectShapeAt(Point2D pt)
68          {
69              foreach(Shape s in _shapes)
70              {
71                  if(s.IsAt(pt))
72                  {
73                      s.Selected = true;
74                  }
75                  else
76                  {
77                      s.Selected = false;
78                  }
79              }
80          }
81
82          public List<Shape> SelectedShapes
83          {
84              get
85              {
86                  List<Shape> result = new List<Shape>();
87                  foreach (Shape s in _shapes)
88                  {
89                      if (s.Selected == true)
90                      {
91                          result.Add(s);
92                      }
93                  }
94
95                  return result;
96              }
97          }
98
99          public void RemoveShape(Shape shape)
100         {
101             _shapes.Remove(shape);
102         }
103     }
104 }
```

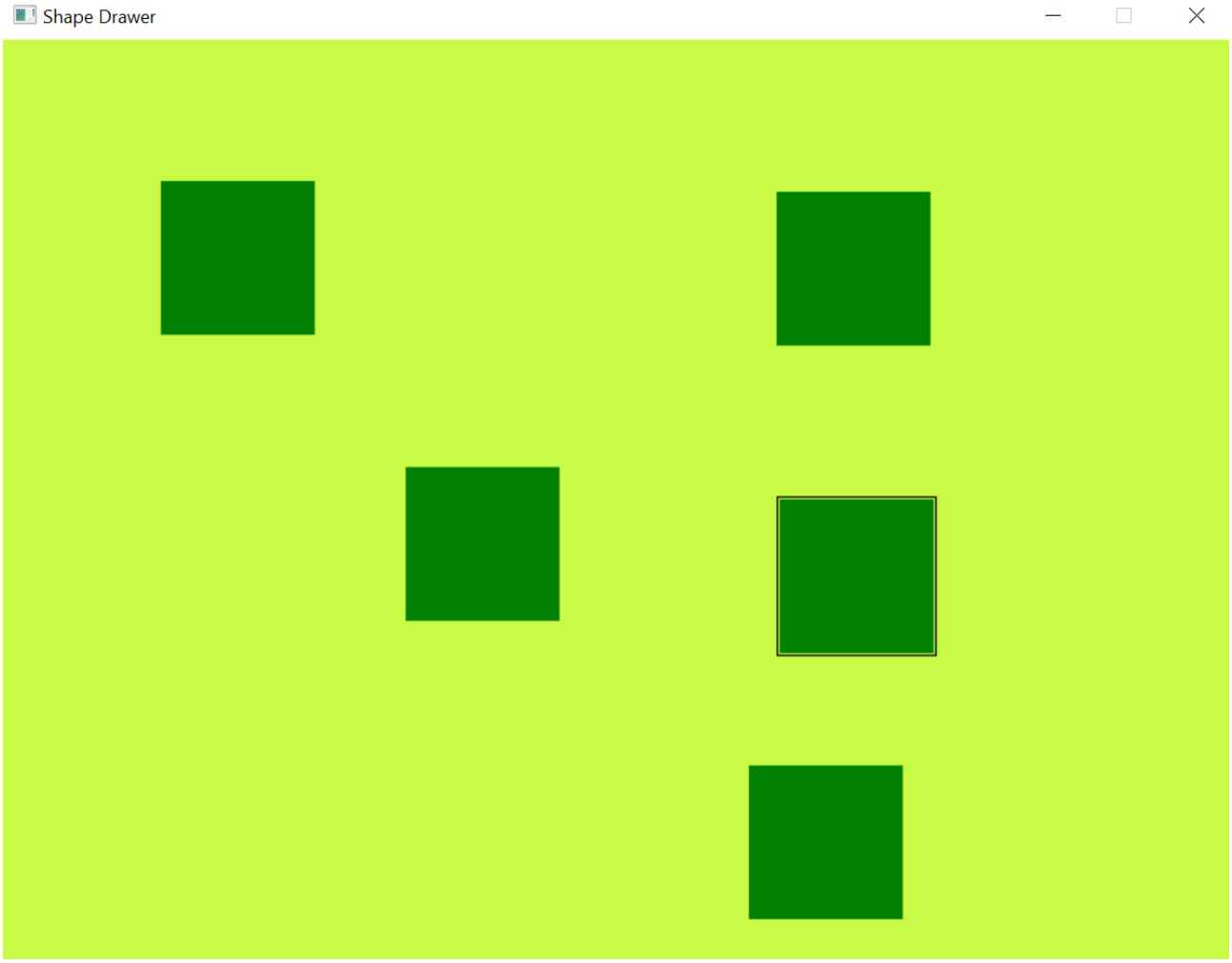
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;

namespace ShapeDrawer
{
    public class Shape
    {
        private Color _color;
        private float _x, _y;
        private int _height, _width;
        private bool _selected;

        //Shape constructor
        public Shape()
        {
            _color = Color.Green;
            _x = 0;
            _y = 0;
            _height = 100;
            _width = 100;
        }

        public Color Color
        {
            get
            {
                return _color;
            }
            set
            {
                _color = value;
            }
        }

        public float X
        {
            get
            {
                return _x;
            }
            set
            {
                _x = value;
            }
        }

        public float Y
        {
            get
```

```
54              {
55                  return _y;
56              }
57              set
58              {
59                  _y = value;
60              }
61          }
62
63          public int Width
64          {
65              get
66              {
67                  return _width;
68              }
69              set
70              {
71                  _width = value;
72              }
73          }
74
75          public int Height
76          {
77              get
78              {
79                  return _height;
80              }
81              set
82              {
83                  _height = value;
84              }
85          }
86
87          public void Draw()
88          {
89              SplashKit.FillRectangle(_color, _x, _y, _width, _height);
90              if(Selected == true)
91              {
92                  DrawOutline();
93              }
94          }
95
96          public bool IsAt(Point2D pt)
97          {
98              if((pt.X >= _x && pt.X <= (_x + _width)) && (pt.Y >= _y && pt.Y <= (_y
                ↪  + _height)))
99              {
100                 return true;
101             }
102             else
103             {
104                 return false;
105             }
```

```
106         }
107
108         public bool Selected
109         {
110             get
111             {
112                 return _selected;
113             }
114             set
115             {
116                 _selected = value;
117             }
118         }
119
120         public void DrawOutline()
121         {
122             SplashKit.DrawRectangle(Color.Black, X - 2, Y - 2, Width + 4, Height +
                ↪ 4);
123         }
124     }
125 }
```

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

namespace ShapeDrawer
{
    public class Program
    {
        public static void Main()
        {
            //Opens a new window called Shape Drawer with dimensions 800 x 600
            new Window("Shape Drawer", 800, 600);

            Drawing drawing = new Drawing();


            //leaves the window open unless requested to be closed
            do
            {
                SplashKit.ProcessEvents(); //allows Splashkit to react to user
                ↪  interactions
                SplashKit.ClearScreen();

                if(SplashKit.MouseClicked(MouseButton.LeftButton) == true)
                {
                    //add a new Shape to your Drawing object based on the mouse's
                    ↪  location.
                    Shape myShape = new Shape();
                    myShape.X = SplashKit.MouseX();
                    myShape.Y = SplashKit.MouseY();
                    drawing.AddShape(myShape);
                }

                if(SplashKit.KeyTyped(KeyCode.SpaceKey) == true)
                {
                    drawing.Background = SplashKit.RandomRGBColor(255);
                }

                if (SplashKit.MouseClicked(MouseButton.RightButton) == true)
                {
                    drawing.SelectShapeAt(SplashKit.MousePosition());
                }

                if(SplashKit.KeyTyped(KeyCode.BackspaceKey) == true ||
                ↪  SplashKit.KeyTyped(KeyCode.DeleteKey) == true)
                {
                    List<Shape> selectedShapes = drawing.SelectedShapes;
                    foreach(Shape s in selectedShapes)
                    {
                        drawing.RemoveShape(s);
                    }
                }

```

```
51                  drawing.Draw();
52
53                  //refresh
54                  SplashKit.RefreshScreen();
55
56              } while (!SplashKit.WindowCloseRequested("Shape Drawer"));
57          }
58      }
59  }
```


Shape Drawer

10 Pass Task 3.2: Clock

Collaboration involves objects working together to achieve tasks. In this task you will reuse the Counter class to develop a clock.

Date	Author	Comment
2021/08/18 10:41	Joshua Wright	Good work, some minor feedback to discuss though :)

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 3.2: Clock

Submitted By:
Marella Morad
103076428
2021/08/16 19:37

Tutor:
Joshua Wright

August 16, 2021

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

namespace ClockApp
{
    public class Clock
    {
        private Counter _hours;
        private Counter _minutes;
        private Counter _seconds;

        public Clock()
        {
            //creating new Counter objects for hours, minutes and seconds
            _hours = new Counter("hours");
            _minutes = new Counter("minutes");
            _seconds = new Counter("seconds");
        }

        public void Tick()
        {
            //resetting seconds when reaching 59, otherwise keep incrementing
            if(_seconds.Ticks == 59)
            {
                _seconds.Reset();
                //resetting minutes when reaching 59, otherwise keep incrementing
                if (_minutes.Ticks == 59)
                {
                    _minutes.Reset();
                    //resetting hours when reaching 23, otherwise keep incrementing
                    if (_hours.Ticks == 23)
                    {
                        _hours.Reset();
                    }
                    else
                    {
                        _hours.Increment();
                    }
                }
                else
                {
                    _minutes.Increment();
                }
            }
            else
            {
                _seconds.Increment();
            }

        }

        public void Reset()
```

```
54          {
55              //resetting all the Counter objects back to 00
56              _hours.Reset();
57              _minutes.Reset();
58              _seconds.Reset();
59          }
60
61          public string Time
62          {
63              get
64              {
65                  // returning time in the required format hh:mm:ss
66                  return $"{_hours.Ticks:00}:{_minutes.Ticks:00}:{_seconds.Ticks:00}
    ";
67                  // using the $ sign, makes it easier to format the time, all in one
                    ↪  line
68              }
69          }
70      }
71  }
```

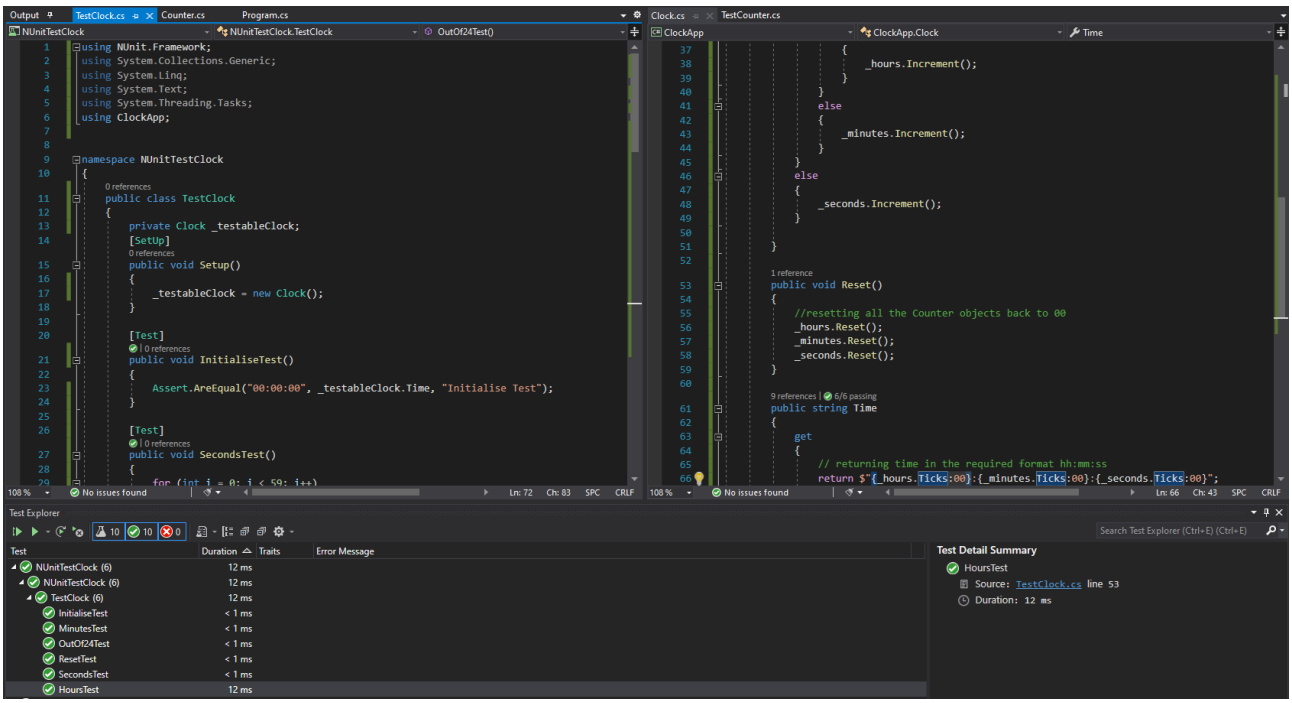
```
using System;
using System.Collections.Generic;
using System.Text;

namespace ClockApp
{
    public class Counter
    {
        private int _count;
        private string _name;

        public Counter(string name)
        {
            _name = name;
            _count = 0;
        }

        //Name property
        public string Name
        {
            //get is read only
            //get and set are both read and write
            get  //should always return a value
            {
                return _name;
            }
            set //should always change a value
            {
                _name = value;
            }
        }

        //read only property
        public int Ticks
        {
            get
            {
                return _count;
            }
        }

        public void Increment()
        {
            _count = _count + 1;
        }

        public void Reset()
        {
            _count = 0;
        }
    }
}
```

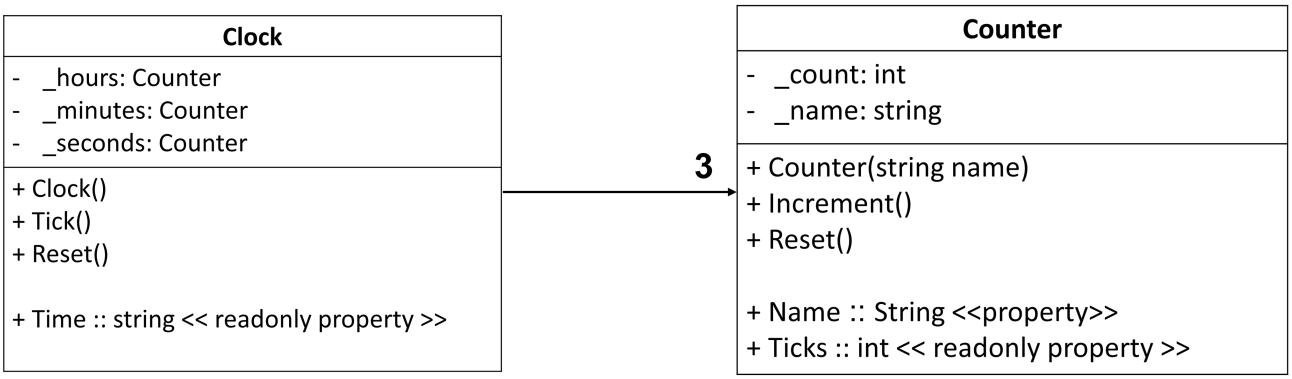
```
using System;

namespace ClockApp
{
    class Program
    {
        static void Main()
        {
            Clock clock = new Clock();
            /*
             * 46824/3600 = 13.00667                --> hours = 13
             * 13.00667 - 13 = 0.00667 * 60 = 0.4   --> minutes = 00
             * 0.4 * 60 = 24                        --> seconds = 24
             */
            for(int i = 0; i < 46824; i++)
            {
                clock.Tick();
            }

            Console.WriteLine("First Try...");
            Console.WriteLine("Expected Output: 13:00:24");
            Console.WriteLine("Actual Output: {0}", clock.Time);
            Console.WriteLine();

            clock.Reset();

            for (int i = 0; i < 23104; i++)
            {
                clock.Tick();
            }

            Console.WriteLine("Second Try...");
            Console.WriteLine("Expected Output: 06:25:04");
            Console.WriteLine("Actual Output: {0}", clock.Time);
        }
    }
}
```

```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ClockApp;


namespace NUnitTestClock
{
    public class TestClock
    {
        private Clock _testableClock;
        [SetUp]
        public void Setup()
        {
            _testableClock = new Clock();
        }

        [Test]
        public void InitialiseTest()
        {
            Assert.AreEqual("00:00:00", _testableClock.Time, "Initialise Test");
        }

        [Test]
        public void SecondsTest()
        {
            for (int i = 0; i < 59; i++)
            {
                _testableClock.Tick();
            }
            Assert.AreEqual("00:00:59", _testableClock.Time, "Testing seconds =
            ↪  59");

            _testableClock.Tick();
            Assert.AreEqual("00:01:00", _testableClock.Time, "Testing seconds don't
            ↪  go over 59");
        }

        [Test]
        public void MinutesTest()
        {
            for (int i = 0; i < 3599; i++)
            {
                _testableClock.Tick();
            }
            Assert.AreEqual("00:59:59", _testableClock.Time, "Testing minutes =
            ↪  59");

            _testableClock.Tick();
            Assert.AreEqual("01:00:00", _testableClock.Time, "Testing minutes don't
            ↪  go over 59");
```

```
50          }
51
52          [Test]
53          public void HoursTest()
54          {
55              for (int i = 0; i < 86399; i++)
56              {
57                  _testableClock.Tick();
58              }
59              Assert.AreEqual("23:59:59", _testableClock.Time, "Testing hours = 23");
60
61              _testableClock.Tick();
62              Assert.AreEqual("00:00:00", _testableClock.Time, "Testing hours don't
                ↪  go over 23");
63          }
64
65          [Test]
66          public void OutOf24Test()
67          {
68              for (int i = 0; i < 86401; i++)
69              {
70                  _testableClock.Tick();
71              }
72              Assert.AreEqual("00:00:01", _testableClock.Time, "Out of 24 hours
                ↪  Test");
73          }
74
75          [Test]
76          public void ResetTest()
77          {
78              for (int i = 0; i < 70142; i++)
79              {
80                  _testableClock.Tick();
81              }
82              Assert.AreEqual("19:29:02", _testableClock.Time, "Testing Time is not
                ↪  0");
83
84              _testableClock.Reset();
85              Assert.AreEqual("00:00:00", _testableClock.Time, "Reset Test");
86          }
87      }
88  }
```


Microsoft Visual Studio Debug Console
First Try...
Expected Output: 13:00:24
Actual Output: 13:00:24
Second Try...
Expected Output: 06:25:04
Actual Output: 06:25:04
C:\Users\marel\OneDrive - Swinburne University\Semester 2 - 2021\COS20007 - Object Oriented Programming\Portfolio\3.2P\ClockApp\bin\Debug\netcoreapp3.1\ClockApp.exe (process 15080) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .


Output
TestClock.cs
Counter.cs
Program.cs
NUnitTestClock
NUnitTestClock.TestClock
OutOf24Test()
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ClockApp;
namespace NUnitTestClock
{
0 references
public class TestClock
{
private Clock _testableClock;
[SetUp]
0 references
public void Setup()
{
_testableClock = new Clock();
}
[Test]
0 references
public void InitialiseTest()
{
Assert.AreEqual("00:00:00", _testableClock.Time, "Initialise Test");
}
[Test]
0 references
public void SecondsTest()
{
for (int i = 0; i < 59; i++)
Clock.cs
TestCounter.cs
ClockApp
ClockApp.Clock
Time
_hours.Increment();
}
}
else
{
_minutes.Increment();
}
}
else
{
_seconds.Increment();
}
}
1 reference
public void Reset()
{
//resetting all the Counter objects back to 00
_hours.Reset();
_minutes.Reset();
_seconds.Reset();
}
9 references | 6/6 passing
public string Time
{
get
{
// returning time in the required format hh:mm:ss
return $"{_hours.Ticks:00}:{_minutes.Ticks:00}:{_seconds.Ticks:00}";
No issues found
Ln: 72 Ch: 83 SPC CRLF
No issues found
Ln: 66 Ch: 43 SPC CRLF
Test Explorer
Search Test Explorer (Ctrl+E) (Ctrl+E)
Test
Duration
Traits
Error Message
NUnitTestClock (6) 12 ms
NUnitTestClock (6) 12 ms
TestClock (6) 12 ms
InitialiseTest < 1 ms
MinutesTest < 1 ms
OutOf24Test < 1 ms
ResetTest < 1 ms
SecondsTest < 1 ms
HoursTest 12 ms
Test Detail Summary
HoursTest
Source: TestClock.cs line 53
Duration: 12 ms


Clock
- _hours: Counter
- _minutes: Counter
- _seconds: Counter
+ Clock()
+ Tick()
+ Reset()
+ Time :: string << readonly property >>
3
Counter
- _count: int
- _name: string
+ Counter(string name)
+ Increment()
+ Reset()
+ Name :: String <<property>>
+ Ticks :: int << readonly property >>

11 Pass Task 4.1: Drawing Program - Different Shapes

See how to use inheritance to create families of related objects and interact with them as a group (polymorphism)

Date	Author	Comment
2021/08/23 23:26	Marella Morad	Demo Video Link
2021/08/23 23:26	Marella Morad	https://youtu.be/kLAYHU5jCLQ
2021/08/25 12:13	Joshua Wright	For IsAt in circle your code lacks precision. Instead please make use of the SplashKit methods PointInCircle, CircleAt, to do the heavy lifting for you here. (You can find information on how to use those methods in the documentation on the splashkit website (splashkit.io).

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 4.1: Drawing Program - Different Shapes

Submitted By:
Marella Morad
103076428
2021/08/25 12:27

Tutor:
Joshua Wright

August 25, 2021

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;

namespace DifferentShapes
{
    public abstract class Shape
    {
        private Color _color;
        private float _x, _y;
        private bool _selected;

        //Shape constructor
        public Shape() : this(Color.Yellow)
        {

        }

        public Shape(Color color)
        {
            _color = color;
        }

        public Color Color
        {
            get
            {
                return _color;
            }
            set
            {
                _color = value;
            }
        }

        public float X
        {
            get
            {
                return _x;
            }
            set
            {
                _x = value;
            }
        }

        public float Y
        {
            get
```

```
54              {
55                  return _y;
56              }
57              set
58              {
59                  _y = value;
60              }
61          }
62
63
64          public abstract void Draw();
65
66          public abstract bool IsAt(Point2D pt);
67
68          public bool Selected
69          {
70              get
71              {
72                  return _selected;
73              }
74              set
75              {
76                  _selected = value;
77              }
78          }
79
80          public abstract void DrawOutline();
81      }
82  }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;

namespace DifferentShapes
{
    public class MyRectangle : Shape //MyRectangle class now inherits all the features from the Shape class
    {
        private int _height, _width;

        //default constructor
        public MyRectangle() : this(Color.Green, 0, 0, 100, 100)
        {}

        //constructor that takes in parameters
        public MyRectangle(Color clr, float x, float y, int width, int height) : base (clr)
        {
            X = x;
            Y = y;
            Width = width;
            Height = height;
        }

        //Width property
        public int Width
        {
            get
            {
                return _width;
            }
            set
            {
                _width = value;
            }
        }

        //Height property
        public int Height
        {
            get
            {
                return _height;
            }
            set
            {
                _height = value;
            }
        }
```

```
52
53          //override draw method to draw rectangles only
54          public override void Draw()
55          {
56              if (Selected)
57              {
58                  DrawOutline();
59              }
60              SplashKit.FillRectangle(Color, X, Y, _width, _height);
61          }
62
63          //override draw method to draw outlines for rectangles
64          public override void DrawOutline()
65          {
66              SplashKit.DrawRectangle(Color.Black, X-2, Y-2, Width + 4, Height + 4);
67          }
68
69          //checking if the cursor is at a drawn rectangle
70          public override bool IsAt(Point2D pt)
71          {
72              //the X and Y are the top left corner of the rectangle
73              //therefore, check if mouse x-position is in the range [X, X + Width]
                ↪  and if the mouse y-position is in the range [Y, Y + height]
74              if ((pt.X >= X && pt.X <= (X + Width)) && (pt.Y) >= Y && pt.Y <= (Y +
                ↪  Height))
75              {
76                  return true;
77              }
78              else
79              {
80                  return false;
81              }
82          }
83      }
84  }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;

namespace DifferentShapes
{
    public class MyCircle : Shape
    {
        private int _radius; //radius field to store the radius of the circle

        //radius property, allows the user to get and set the radius
        public int Radius
        {
            get
            {
                return _radius;
            }
            set
            {
                _radius = value;
            }
        }

        //default constructor calling the other constructor and passing the color
        ↪  blue and radius 50
        public MyCircle() : this(Color.Blue, 50)
        {}

        //constructor with parameters
        public MyCircle(Color clr, int radius) : base (clr)
        {
            _radius = radius;
        }

        //override method to draw circles only
        public override void Draw()
        {
            if (Selected)
            {
                DrawOutline();
            }
            SplashKit.FillCircle(Color, X, Y, _radius);
        }

        //override method to draw the outline for the circle
        public override void DrawOutline()
        {
            //drawing outline with a radius bigger than the circle by 2 pixels
            SplashKit.DrawCircle(Color.Black, X, Y, _radius + 2);
        }
```

```
53
54          public override bool IsAt(Point2D pt)
55          {
56              //checking if the point is at the circle
57              //since for the circle, X and Y are the coordinates of the circle, then
                ↪  we need to cover the entier circle
58              //checking if the mouse x-position is in the range [X - radius, X +
                ↪  radius] and if the mouse y-position is in the range [Y - radius, Y
                ↪  + radius]
59              Circle circle = SplashKit.CircleAt(X, Y, _radius);
60              if (SplashKit.PointInCircle(pt, circle))
61              {
62                  return true;
63              }
64              else
65              {
66                  return false;
67              }
68          }
69      }
70  }
```

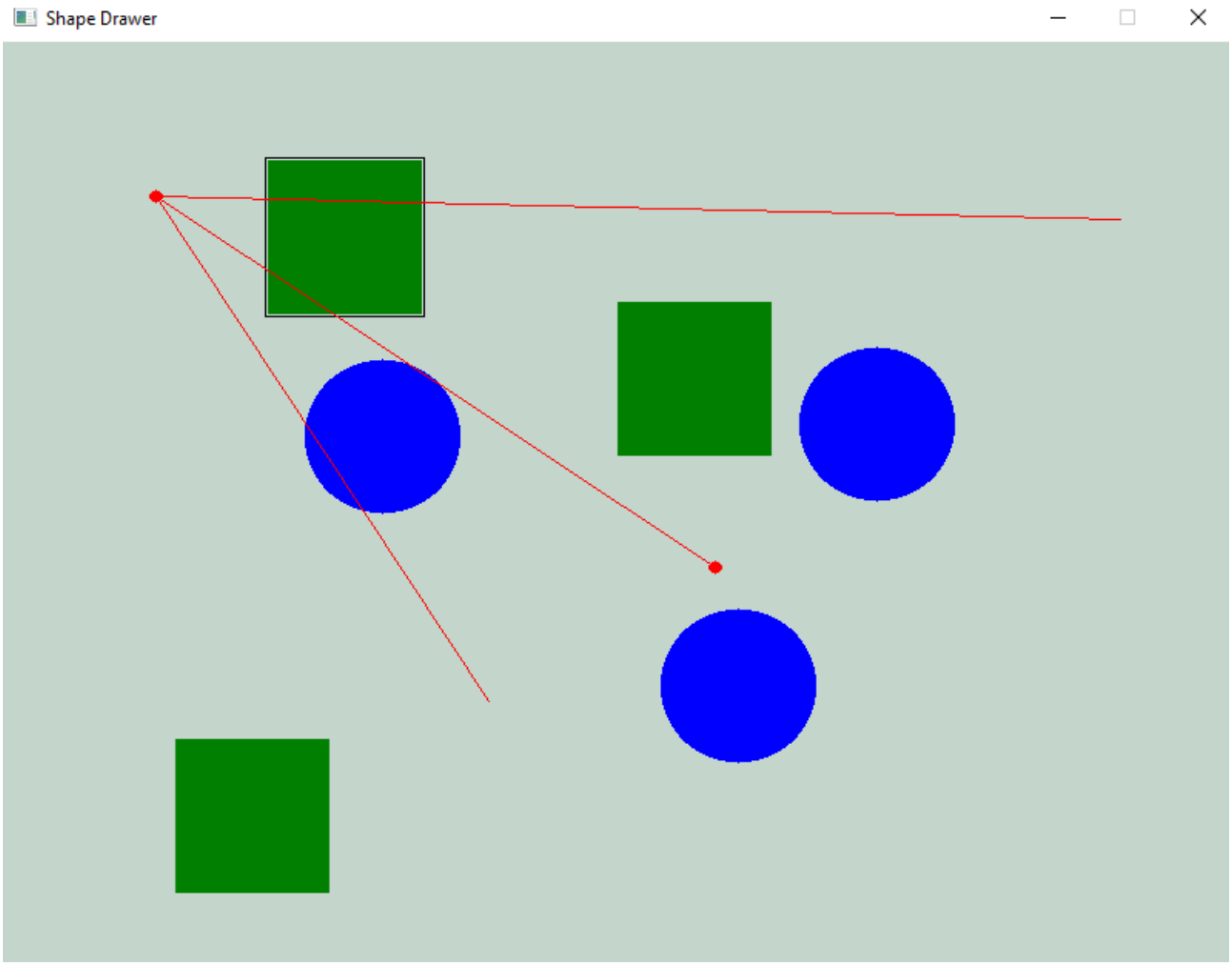
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;

namespace DifferentShapes
{
    public class MyLine : Shape
    {
        private float _endX;
        private float _endY;

        //default constructor
        public MyLine() : this(Color.Red, 0, 0, 100, 100)
        {}

        //constructor that takes in parameters
        public MyLine(Color clr, float starX, float starY, float endX, float endY)
        ↪  : base (clr)
        {
            X = starX;
            Y = starY;
            EndX = endX;
            EndY = endY;
        }

        //EndX property
        public float EndX
        {
            get
            {
                return _endX;
            }
            set
            {
                _endX = value;
            }
        }

        //EndY property
        public float EndY
        {
            get
            {
                return _endY;
            }
            set
            {
                _endY = value;
            }
        }
```

```
53
54          //override draw method to draw lines
55          public override void Draw()
56          {
57              if(Selected)
58              {
59                  DrawOutline();
60              }
61              SplashKit.DrawLine(Color, X, Y, EndX, EndY);
62          }
63
64          //override draw outline method to draw two filled circles at both ends of
            ↪  the line using the coordinates (X,Y) and (EndX, EndY)
65          public override void DrawOutline()
66          {
67              SplashKit.FillCircle(Color, X, Y, 3);
68              SplashKit.FillCircle(Color, EndX, EndY, 3);
69          }
70
71          public override bool IsAt(Point2D pt)
72          {
73              //defining the line with the given coordinates (X,Y) and (EndX, EndY)
                ↪  using SplashKit's LineFrom
74              Line line = SplashKit.LineFrom(X, Y, EndX, EndY);
75              //checking if the point is close to the line, less than 10 pixels
76              if(SplashKit.PointLineDistance(pt, line) < 10)
77              {
78                  return true;
79              }
80              else
81              {
82                  return false;
83              }
84          }
85      }
86  }
```

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

namespace DifferentShapes
{
    public class Program
    {
        //creating ShaeKind that will determine what type of shape the user
        ↪  requested to draw
        private enum ShapeKind
        {
            Rectangle,
            Circle,
            Line
        }
        public static void Main()
        {
            //Opens a new window called Shape Drawer with dimensions 800 x 600
            new Window("Shape Drawer", 800, 600);

            Drawing myDrawing = new Drawing();
            //setting the default shapeKind to circle
            ShapeKind kindToAdd = ShapeKind.Circle;


            //leaves the window open unless requested to be closed
            do
            {
                SplashKit.ProcessEvents(); //allows Splashkit to react to user
                ↪  interactions
                SplashKit.ClearScreen();
                //if the user presses the R Key, then they've chosen to draw a
                ↪  rectangle
                if(SplashKit.KeyTyped(KeyCode.RKey))
                {
                    kindToAdd = ShapeKind.Rectangle;
                }
                //if the user presses the C Key, then they've chosen to draw a
                ↪  circle
                else if (SplashKit.KeyTyped(KeyCode.CKey))
                {
                    kindToAdd = ShapeKind.Circle;
                }
                //if the user presses the L Key, then they've chosen to draw a line
                else if (SplashKit.KeyTyped(KeyCode.LKey))
                {
                    kindToAdd = ShapeKind.Line;
                }

                //drawing the shapes when the mouse's left button is pressed, after
                ↪  taking in the user selection
                if (SplashKit.MouseClicked(MouseButton.LeftButton))
```

```
49                  {
50                      Shape newShape;
51
52                      if(kindToAdd == ShapeKind.Circle)
53                      {
54                          //add a new Shape to your Drawing object based on the
                            ↪  mouse's location.
55                          MyCircle newCircle = new MyCircle();
56
57                          newShape = newCircle;
58                      }
59                      else if(kindToAdd == ShapeKind.Rectangle)
60                      {
61                          //add a new Shape to your Drawing object based on the
                            ↪  mouse's location.
62                          MyRectangle newRect = new MyRectangle();
63
64                          newShape = newRect;
65                      }
66                      else
67                      {
68                          MyLine newLine = new MyLine();
69                          newShape = newLine;
70                      }
71
72                      newShape.X = SplashKit.MouseX();
73                      newShape.Y = SplashKit.MouseY();
74                      myDrawing.AddShape(newShape);
75
76                  }
77
78                  //setting the background color to a random color when the user
                    ↪  presses space
79                  if (SplashKit.KeyTyped(KeyCode.SpaceKey) == true)
80                  {
81                      myDrawing.Background = SplashKit.RandomRGBColor(255);
82                  }
83
84                  //if the mouse's right button is clicked, select the shape (if the
                    ↪  mouse is at a shape)
85                  if (SplashKit.MouseClicked(MouseButton.RightButton) == true)
86                  {
87                      myDrawing.SelectShapeAt(SplashKit.MousePosition());
88                  }
89
90                  //pressing backspace or delete, allows the user to delete the
                    ↪  selected shapes
91                  if (SplashKit.KeyTyped(KeyCode.BackspaceKey) == true ||
                    ↪  SplashKit.KeyTyped(KeyCode.DeleteKey) == true)
92                  {
93                      List<Shape> selectedShapes = myDrawing.SelectedShapes;
94                      foreach (Shape s in selectedShapes)
95                      {
```

```
96                          myDrawing.RemoveShape(s);
97                      }
98                  }
99
100                 myDrawing.Draw();
101
102                 //refresh
103                 SplashKit.RefreshScreen();
104
105             } while (!SplashKit.WindowCloseRequested("Shape Drawer"));
106         }
107     }
108 }
```


Shape Drawer

12 Pass Task 4.2: Case Study Iteration 2

Object oriented programming makes best sense with larger programs. The case study will be your opportunity to create a larger program and better see how these abstractions make it easier to create software solutions.

Date	Author	Comment
2021/09/01 11:26	Joshua Wright	- FullDescription for player is missing information and/or isn't correctly formated, it should be: "You are **Name** **base.FullDescription**\nYou are carring:\n**ItemList**"

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 4.2: Case Study Iteration 2

Submitted By:
Marella Morad
103076428
2021/10/25 14:43

Tutor:
Joshua Wright

October 25, 2021

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

namespace SwinAdventure
{
    public abstract class GameObject : IdentifiableObject
    {
        private string _description;
        private string _name;

        public GameObject(string[] ids, string name, string desc) : base(ids)
        {
            _name = name;
            _description = desc;
        }

        //short textual description of the game object
        public string Name
        {
            get
            {
                return _name;
            }
        }

        //referring to the object
        public string ShortDescription
        {
            get
            {
                //return name + firstId of the game object in the format:
                // a small computer (pc) - where small computer is the _name of the
                ↪  game object and pc is the first id of the game object
                string shortDesc = "A " + _name + " (" + FirstId + ")";
                return shortDesc;
            }
        }

        public virtual string FullDescription
        {
            get
            {
                return _description; //by default, returns description
                //will be overridden by child classes
            }
        }
    }
}
```

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

namespace SwinAdventure
{
    public class Player : GameObject, IHaveInventory
    {
        private Location _location; //to allow player to be in a location
        private Inventory _inventory; //to manage the Player's items
        private Location _lastLocation; //to allow player to leave a location (go back to the previous one)

        //default constructor
        public Player(string name, string desc) : base(new string[] { "me", "inventory" }, name, desc)
        {
            _inventory = new Inventory();
            _location = new Location(new string[] { "hallway" }, "Hallway", "This is a long well lit Hallway");
        }

        public GameObject Locate(string id)
        {
            //three checks
            if (AreYou(id)) //if the player is what is to be located
            {
                return this; //retunring the player as a game object around the player
            }
            else if (_inventory.HasItem(id)) //if the player had what is being located
            {
                return _inventory.Fetch(id); //if the game object is not the player, then fetch it from inventory
            }
            else //if the item can be located where the player is
            {
                return _location.Locate(id);
            }
        }

        public override string FullDescription //overridden with the extra infromation
        {
            get
            {
                //returns the full description
                String fullDesc = "You are " + Name + " " + base.FullDescription + "\nYou are carrying:\n" + _inventory.ItemList;

                return fullDesc;
            }
```

```
46          }
47
48          public Inventory Inventory
49          {
50              get
51              {
52                  return _inventory;
53              }
54          }
55
56          public Location Location
57          {
58              get
59              {
60                  return _location;
61              }
62              set
63              {
64                  //setting the last location to the current location then changing
                    ↪  to the new location
65                  if(_location != null)
66                  {
67                      _lastLocation = _location;
68                  }
69                  else
70                  {
71                      _lastLocation = value;
72                  }
73
74                  _location = value;
75              }
76          }
77
78          public void Leave()
79          {
80              _location = _lastLocation; //returning to the last location
81          }
82      }
83  }
```

```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SwinAdventure;

namespace SwinAdventureTests
{
    [TestFixture]
    public class PlayerUnitTests
    {
        Player player;
        Item shovel;
        Item sword;
        Item pc;
        Item gem;

        Location location;

        [SetUp]
        public void Setup()
        {
            location = new Location(new string[] { "closet" }, "small Closet", "A small dark closet, with an odd smell");
            player = new Player("Marella", "the amazing player");
            player.Location = location;
            shovel = new Item(new string[] { "shovel" }, "shovel", "This is a might fine shovel");
            sword = new Item(new string[] { "sword" }, "bronze sword", "This is a shiny sword");
            pc = new Item(new string[] { "pc" }, "small computer", "This is a computer from the future");
            gem = new Item(new string[] { "gem" }, "red gem", "This is a shiny red gem");

            player.Inventory.Put(shovel);
            player.Inventory.Put(sword);
            player.Inventory.Put(pc);
        }

        [Test]
        public void TestIdentifiablePlayer()
        {
            Assert.IsTrue(player.AreYou("me"), "defauls player not identifiable1");
            Assert.IsTrue(player.AreYou("inventory"), "defauls player not identifiable2");
        }

        [Test]
        public void TestPlayerLocatesItems()
        {
            Assert.AreEqual(player.Locate("pc"), pc, "Player cannot locate pc");
```

```
48          }
49
50          [Test]
51          public void TestPlayerLocatesItself()
52          {
53              Assert.AreEqual(player.Locate("me"), player, "Player cannot locate
                ↪  itself1");
54              Assert.AreEqual(player.Locate("inventory"), player, "Player cannot
                ↪  locate itself2");
55          }
56
57          [Test]
58          public void TestPlayerLocatesNothing()
59          {
60              Assert.IsNull(player.Locate("apple"), "Player can locate apple");
61          }
62
63          [Test]
64          public void TestPlayerFullDescription()
65          {
66              string expected =
67              "You are Marella the amazing player\n" +
68              "You are carrying:\n" +
69              "     a shovel (shovel)\n" +
70              "     a bronze sword (sword)\n" +
71              "     a small computer (pc)\n";
72              Assert.AreEqual(player.FullDescription, expected, "Full description
                ↪  incorrectly formatted");
73          }
74
75          //Add a test for Players to locate items in their location
76          [Test]
77          public void TestPlayerLocateItemInItsLocation()
78          {
79              location.Inventory.Put(gem);
80              Assert.AreEqual(player.Locate("gem"), gem);
81          }
82      }
83  }
```

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

namespace SwinAdventure
{
    public class Item : GameObject
    {
        public Item(string[] idents, string name, string desc) : base(idents, name,
        ↪ desc)
        {
        }
    }
}
```

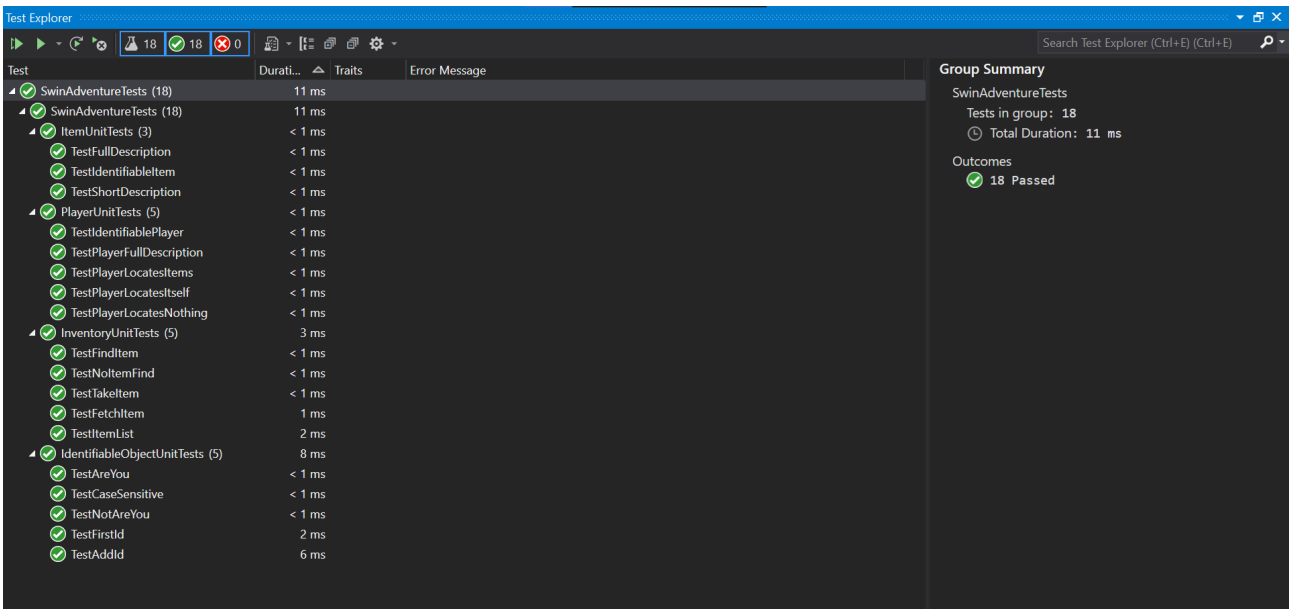
```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SwinAdventure;

namespace SwinAdventureTests
{
    [TestFixture]
    public class ItemUnitTests
    {
        Item shovel;
        Item sowrd;
        Item pc;

        [SetUp]
        public void Setup()
        {
            shovel = new Item(new string[] { "shovel" }, "shovel", "This is a might fine shovel");
            sowrd = new Item(new string[] { "sowrd" }, "bronze sowrd", "This is a shiny sowrd");
            pc = new Item(new string[] { "pc" }, "small computer", "This is a computer from the future");
        }

        [Test]
        public void TestIdentifiableItem()
        {
            Assert.IsTrue(shovel.AreYou("shovel"), "Item does not match id");
        }

        [Test]
        public void TestShortDescription()
        {
            Assert.AreEqual("A shovel (shovel)", shovel.ShortDescription, "Short Description does not match");
        }

        [Test]
        public void TestFullDescription()
        {
            Assert.AreEqual(shovel.FullDescription, "This is a might fine shovel", "Full Description does not match");
        }
    }
}
```

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

namespace SwinAdventure
{
    public class Inventory
    {
        private List<Item> _items;

        public Inventory()
        {
            _items = new List<Item>();
        }

        public bool HasItem(string id)
        {
            foreach (Item i in _items)
            {
                if (i.AreYou(id.ToLower()))
                {
                    return true;
                }
            }
            return false;
        }

        public void Put(Item itm)
        {
            _items.Add(itm);
        }

        public Item Take(string id)
        {
            foreach (Item i in _items)
            {
                if (i.AreYou(id))
                {
                    _items.Remove(i);
                    return i;
                }
            }

            return null;
        }

        public Item Fetch(string id)
        {
            foreach (Item i in _items)
            {
                //locating the item using AreYou and returning it
                if (i.AreYou(id))
                {
```

```
54                      return i;
55                  }
56              }
57
58              return null;
59          }
60
61          public string ItemList
62          {
63              get
64              {
65                  StringBuilder itemList = new StringBuilder();
66
67                  foreach (Item i in _items)
68                  {
69                      //speces are used instead of \t as \t does not work with
                        ↪  winForms when using the GUI
70                      itemList.Append("     " + i.ShortDescription.ToLower() + "\n");
71                  }
72
73                  return itemList.ToString();
74              }
75          }
76      }
77  }
```

```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SwinAdventure;

namespace SwinAdventureTests
{
    [TestFixture]
    public class InventoryUnitTests
    {
        Item shovel;
        Item sword;
        Item pc;

        Inventory inventory;

        [SetUp]
        public void Setup()
        {
            shovel = new Item(new string[] { "shovel" }, "shovel", "This is a might fine shovel");
            sword = new Item(new string[] { "sword" }, "bronze sword", "This is a shiny sword");
            pc = new Item(new string[] { "pc" }, "small computer", "This is a computer from the future");

            inventory = new Inventory();
            inventory.Put(shovel);
            inventory.Put(sword);
            inventory.Put(pc);
        }

        [Test]
        public void TestFindItem()
        {
            Assert.IsTrue(inventory.HasItem("shovel"), "Inventory is empty"); //checking if pc is found in inventory
        }

        [Test]
        public void TestNoItemFind()
        {
            Assert.IsFalse(inventory.HasItem("crown"), "Inventory does not have crown"); //testing if inventory has sword (should return false)
        }

        [Test]
        public void TestFetchItem()
        {
            Assert.AreEqual(inventory.Fetch("pc"), pc, "Inventory does not have pc");
```

```
48              Assert.IsTrue(inventory.HasItem("pc"), "Inventory is empty"); //item
                ↪  has not been removed - only fetched
49          }
50
51          [Test]
52          public void TestTakeItem()
53          {
54              inventory.Take("pc");
55              Assert.IsFalse(inventory.HasItem("pc"), "Item not removed");
56          }
57
58          [Test]
59          public void TestItemList()
60          {
61              string expected =
62                  "     a shovel (shovel)\n" +
63                  "     a bronze sword (sword)\n" +
64                  "     a small computer (pc)\n";
65              Assert.AreEqual(inventory.ItemList, expected, "List incorrectly
                ↪  formatted");
66          }
67      }
68  }
```


Test Explorer
18
18
0
Search Test Explorer (Ctrl+E) (Ctrl+E)
Test
Durati...
Traits
Error Message
SwinAdventureTests (18) 11 ms
SwinAdventureTests (18) 11 ms
ItemUnitTests (3) < 1 ms
TestFullDescription < 1 ms
TestIdentifiableItem < 1 ms
TestShortDescription < 1 ms
PlayerUnitTests (5) < 1 ms
TestIdentifiablePlayer < 1 ms
TestPlayerFullDescription < 1 ms
TestPlayerLocatesItems < 1 ms
TestPlayerLocatesItself < 1 ms
TestPlayerLocatesNothing < 1 ms
InventoryUnitTests (5) 3 ms
TestFindItem < 1 ms
TestNoItemFind < 1 ms
TestTakeItem < 1 ms
TestFetchItem 1 ms
TestItemList 2 ms
IdentifiableObjectUnitTests (5) 8 ms
TestAreYou < 1 ms
TestCaseSensitive < 1 ms
TestNotAreYou < 1 ms
TestFirstId 2 ms
TestAddId 6 ms
Group Summary
SwinAdventureTests
Tests in group: 18
Total Duration: 11 ms
Outcomes
18 Passed

13 Pass Task 5.1: Case Study Iterations 3 and 4

Object oriented programming makes best sense with larger programs. The case study will be your opportunity to create a larger program and better see how these abstractions make it easier to create software solutions.

Date	Author	Comment
2021/09/17 15:38	Charlotte Pierce	Why does LookCommand have a boolean class variable player? You shouldn't need this.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 5.1: Case Study Iterations 3 and 4

Submitted By:
Marella Morad
103076428
2021/10/25 14:44

Tutor:
Joshua Wright

October 25, 2021

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

namespace SwinAdventure
{
    public class Bag : Item, IHaveInventory
    {
        private Inventory _inventory;

        //default constructor
        public Bag(string[] ids, string name, string desc) : base(ids, name, desc)
        {
            _inventory = new Inventory();
        }

        public GameObject Locate(string id)
        {
            if (AreYou(id))
            {
                return this; //retunring the bag as a game object around the player
            }
            else
            {
                return _inventory.Fetch(id); //fetch the bag it from inventory
            }
        }

        public override string FullDescription
        {
            get
            {
                string fullDesc = base.FullDescription + "\nYou look in the " +
                ↪  Name + " and you see:\n" + _inventory.ItemList;
                return fullDesc;
            }
        }

        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }
    }
}
```

```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SwinAdventure;

namespace SwinAdventureTests
{
    [TestFixture]
    public class BagUnitTests
    {
        Bag b1, b2;

        Item shovel;
        Item sword;
        Item pc;

        [SetUp]
        public void Setup()
        {
            shovel = new Item(new string[] { "shovel" }, "shovel", "This is a might
            ↪  fine shovel");
            sword = new Item(new string[] { "sword" }, "bronze sword", "This is a
            ↪  shiny sword");
            pc = new Item(new string[] { "pc" }, "small computer", "This is a
            ↪  computer from the future");

            b1 = new Bag(new string[] {"bag"}, "plastic bag", "This is a clear
            ↪  plastic bag");
            b2 = new Bag(new string[] { "wallet" }, "wallet", "This is a black
            ↪  wallet");

            b1.Inventory.Put(shovel);
            b2.Inventory.Put(pc);
            b1.Inventory.Put(b2);
        }

        [Test]
        public void TestBagLocatesItems()
        {
            Assert.AreEqual(b1.Locate("shovel"), shovel, "Bag cannot locate
            ↪  shovel");
        }

        [Test]
        public void TestBagLocatesIteslef()
        {
            Assert.AreEqual(b1.Locate("bag"), b1, "Bag cannot locate itself1");
        }

        [Test]
        public void TestBagLocatesNothing()
```

```
48          {
49              Assert.IsNull(b1.Locate("pc"), "Bag can locate pc");
50          }
51
52          [Test]
53          public void TestBagFullDescription()
54          {
55              string expected =
56               "This is a clear plastic bag\n"+
57              "You look in the " + b1.Name + " and you see:\n" +
58              "     a shovel (shovel)\n" +
59              "     a wallet (wallet)\n";
60              Assert.AreEqual(expected, b1.FullDescription, "Full description
                ↪  incorrectly formatted");
61          }
62
63          [Test]
64          public void TestBagInBag()
65          {
66              Assert.AreEqual(b1.Locate("wallet"), b2, "Bag 2 is not in Bag 1"); //b1
                ↪  can locate b2
67              Assert.AreEqual(b1.Locate("shovel"), shovel, "shovel cannot be located
                ↪  in bag 1"); //b1 can locate other items in b1
68              Assert.IsNull(b2.Locate("bag"), "Bag 1 is in Bag 2"); //b2 cannot
                ↪  locate b1
69              Assert.IsNull(b1.Locate("pc"), "Bag 1 can locate pc"); //b1 cannot
                ↪  locate items from b2
70          }
71      }
72  }
```

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

namespace SwinAdventure
{
    public abstract class Command : IdentifiableObject
    {
        public Command(string[] ids) : base(ids)
        {

        }

        public abstract string Execute(Player p, string[] text);
    }
}
```

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

namespace SwinAdventure
{
    public class LookCommand : Command
    {
        public LookCommand() : base(new string[] { "look", "examine", "inventory" })
        {

        }

        //FIX to work with the location

        public override string Execute(Player p, string[] text)
        {
            if(text.Length !=1 && text.Length != 2 && text.Length != 3 &&
            ↪  text.Length != 5)
            {
                return "I don't know how to look like that";
            }

            if(text.Length == 1 && text[0] == "look") //if the player types look
            {
                return p.Location.FullDescription; //prints the location's full
                ↪  description
            }

            bool player = false;

            if (text.Length == 1 && text[0] == "inventory") //if the player types
            ↪  inventory, it is same as look at inventory
            {
                player = true;
                return LookAtIn(text[0], p, player);
            }

            //text.Length = 2
            if (text.Length == 2 && text[0] == "examine")
            {
                player = true;
                return LookAtIn(text[1], p, player);
            }

            //text.Length = 3
            if (text.Length == 3)
            {
                if(text[1].ToLower() == "in")
                {
                    player = false;
```

```
                return LookAtIn(text[2], p, player);
            }

            if (text[1].ToLower() != "at")
            {
                return "What do you want to look at?";
            }

            player = true;
            return LookAtIn(text[2], p, player);
        }
        else //if text.Length == 5
        {
            player = false;
            if(text[3] != "in")
            {
                return "What do you want to look in?";
            }

            IHaveInventory _container = FetchContainer(p, text[4]);
            if(_container != null)
            {
                return LookAtIn(text[2], _container, player);
            }
            else
            {
                return "I cannot find the " + text[4];
            }
        }
    }

    public IHaveInventory FetchContainer(Player p, string containerId)
    {
        return p.Locate(containerId) as IHaveInventory;
    }

    public string LookAtIn(string thingId, IHaveInventory container, bool
    ↪ player)
    {
        if (container.Locate(thingId) != null)
        {
            return container.Locate(thingId).FullDescription;
        }
        else
        {
            if(player == true)
            {
                return "I cannot find the " + thingId;
            }
            else
            {
                return "I cannot find the " + thingId + " in the " +
                ↪ container.Name;
```

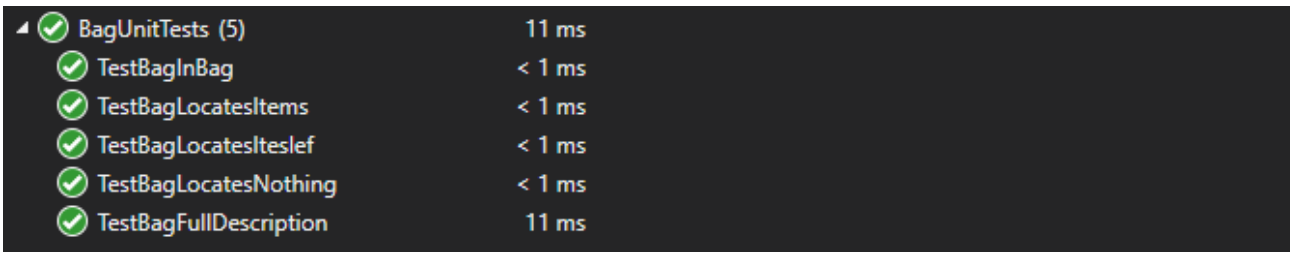
```
102
103                     }
104
105                 }
106             }
107         }
108 }
```

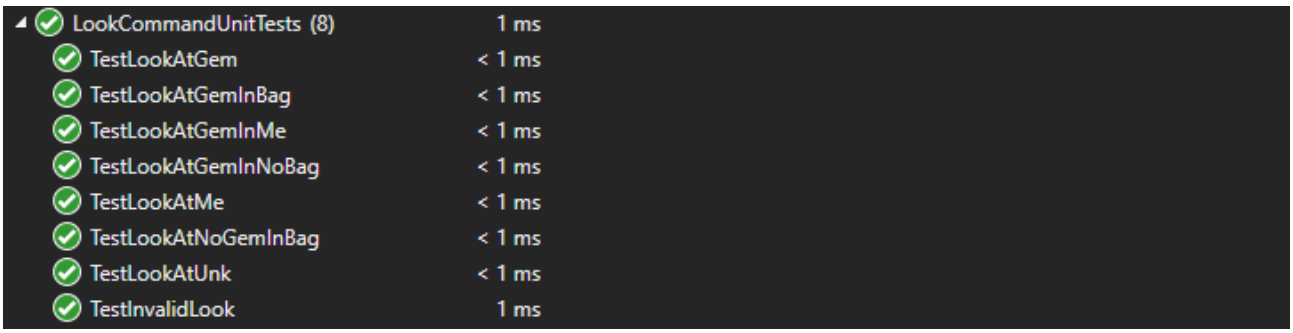
```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SwinAdventure;

namespace SwinAdventureTests
{
    [TestFixture]
    public class LookCommandUnitTests
    {
        Item shovel;
        Item sword;
        Item gem;

        Player player;

        Bag b1;
        Bag b2;

        LookCommand look;

        Location hall;

        [SetUp]
        public void Setup()
        {
            shovel = new Item(new string[] { "shovel" }, "shovel", "This is a might fine shovel");
            sword = new Item(new string[] { "sword" }, "bronze sword", "This is a shiny sword");
            gem = new Item(new string[] { "gem" }, "red gem", "This is a shiny red gem");

            b1 = new Bag(new string[] { "bag" }, "plastic bag", "This is a clear plastic bag");
            b2 = new Bag(new string[] { "wallet" }, "wallet", "This is a black wallet");

            hall = new Location(new string[] { "hallway" }, "Hallway", "This is a long well lit Hallway");

            player = new Player("Marella", "The amazing player");

            player.Location = hall;

            look = new LookCommand();

            b2.Inventory.Put(shovel);
            b1.Inventory.Put(b2);

            player.Inventory.Put(shovel);
```

```
48              player.Inventory.Put(b1);
49              player.Inventory.Put(gem);
50
51              look = new LookCommand();
52
53
54          }
55
56          [Test]
57          public void TestLookAtMe()
58          {
59              string expected =
60              "You are Marella The amazing player\n" +
61              "You are carrying:\n" +
62              "     a shovel (shovel)\n" +
63              "     a plastic bag (bag)\n" +
64              "     a red gem (gem)\n";
65              Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "inventory"}), "TestLookAtMe failed");
66          }
67
68          [Test]
69          public void TestLookAtGem()
70          {
71              player.Inventory.Put(gem);
72              string expected = "This is a shiny red gem";
73              Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "gem"}), "TestLookAtGem Failed");
74          }
75
76          [Test]
77          public void TestLookAtUnk()
78          {
79              player.Inventory.Take("gem");
80              string expected = "I cannot find the gem";
81              Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "gem" }), "TestLookAtUnk Failed");
82          }
83
84          [Test]
85          public void TestLookAtGemInMe()
86          {
87
88              string expected = "This is a shiny red gem";
89              Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "gem", "in", "inventory"}), "TestLookAtGemInMe Failed");
90          }
91
92          [Test]
93          public void TestLookAtGemInBag()
94          {
95              b1.Inventory.Put(gem);
96              string expected = "This is a shiny red gem";
```

```
97              Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "gem", "in", "bag" }), "TestLookAtGemInMe Failed");
98          }
99
100         [Test]
101         public void TestLookAtGemInNoBag()
102         {
103             player.Inventory.Take("bag");
104             string expected = "I cannot find the bag";
105             Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "gem", "in", "bag" }), "TestLookAtGemInMe Failed");
106         }
107
108         [Test]
109         public void TestLookAtNoGemInBag()
110         {
111             string expected = "I cannot find the gem in the plastic bag";
112             Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "gem", "in", "bag" }), "TestLookAtGemInMe Failed");
113         }
114
115         [Test]
116         public void TestInvalidLook()
117         {
118             string expected = "I don't know how to look like that";
119             Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "gem", "in"}), "Test text length = 4 not accepted Failed");
120
121             expected = "What do you want to look at?";
122             Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "int", "gem"}), "Test look at Failed");
123
124             expected = "What do you want to look in?";
125             Assert.AreEqual(expected, look.Execute(player, new string[] { "look",
                ↪  "at", "gem", "on", "bag" }), "Test look at in Failed");
126         }
127     }
128 }
```

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

namespace SwinAdventure
{
    public interface IHaveInventory
    {
        public GameObject Locate(string id); //locate an item using its id (inside
        ↪  the container)

        public string Name //returnig the name of containter
        {
            get;
        }

        public Inventory Inventory
        {
            get;
        }
    }
}
```


BagUnitTests (5) 11 ms
TestBagInBag < 1 ms
TestBagLocatesItems < 1 ms
TestBagLocatesIteslef < 1 ms
TestBagLocatesNothing < 1 ms
TestBagFullDescription 11 ms


LookCommandUnitTests (8) 1 ms
TestLookAtGem < 1 ms
TestLookAtGemInBag < 1 ms
TestLookAtGemInMe < 1 ms
TestLookAtGemInNoBag < 1 ms
TestLookAtMe < 1 ms
TestLookAtNoGemInBag < 1 ms
TestLookAtUnk < 1 ms
TestInvalidLook 1 ms

14 Credit Task 5.2: Drawing Program - Saving

Drawing programs have a natural affinity with object oriented design and programming, with easy to see roles and functionality. In this task you will start to create an object oriented drawing program.

Date	Author	Comment
2021/09/06 23:26	Marella Morad	Demo video link
2021/09/06 23:26	Marella Morad	https://youtu.be/L7mwwwK7i2s

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Credit Task 5.2: Drawing Program - Saving

Submitted By:
Marella Morad
103076428
2021/09/06 23:19

Tutor:
Joshua Wright

September 6, 2021

```
using SplashKitSDK;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace DifferentShapes
{
    public class Drawing
    {
        //Add a private, read only, field to store the list of _shapes. Use
        ↪  List<Shape> as the type
        private readonly List<Shape> _shapes;

        //_background private field
        private Color _background;

        //public Background property for the background color.
        public Color Background
        {
            get
            {
                return _background;
            }
            set
            {
                _background = value;
            }
        }

        //constructor that takes in the background color as a parameter
        public Drawing(Color background)
        {
            _shapes = new List<Shape>();
            _background = background;
        }

        //default constructor - without parameters
        public Drawing() : this(Color.White)
        {
            _shapes = new List<Shape>();
            _background = Color.White;
        }

        public int ShapeCount
        {
            get
            {
                return _shapes.Count;
            }
        }
```

```
53
54          public void AddShape(Shape shape)
55          {
56              _shapes.Add(shape);
57          }
58
59          public void Draw()
60          {
61              SplashKit.ClearScreen(Background);
62              foreach (Shape shape in _shapes)
63              {
64                  shape.Draw();
65              }
66          }
67
68          public void SelectShapeAt(Point2D pt)
69          {
70              foreach (Shape s in _shapes)
71              {
72                  if (s.IsAt(pt))
73                  {
74                      s.Selected = true;
75                  }
76                  else
77                  {
78                      s.Selected = false;
79                  }
80              }
81          }
82
83          public List<Shape> SelectedShapes
84          {
85              get
86              {
87                  List<Shape> result = new List<Shape>();
88                  foreach (Shape s in _shapes)
89                  {
90                      if (s.Selected == true)
91                      {
92                          result.Add(s);
93                      }
94                  }
95
96                  return result;
97              }
98          }
99
100         public void RemoveShape(Shape shape)
101         {
102             _shapes.Remove(shape);
103         }
104
105         public void Save(string filename)
```

```
106         {
107             StreamWriter writer;
108 
109             writer = new StreamWriter(filename);
110             try
111             {
112                 writer.WriteColor(Background); //output the background color as
                    ↪ three floating point numbers (RGB), which can
113                                                //then be used to recreate the color
                                                   ↪ using Color's RGBColor function
                                                   ↪ when the file is loaded.
114                 writer.WriteLine(ShapeCount);  //outputs the number of shapes, so
                    ↪ that when the file is read back in we can use
115                                                //that number to determine how many
                                                   ↪ shapes need to be read in
116 
117                 foreach (Shape s in _shapes)
118                 {
119                     s.SaveTo(writer);
120                 }
121             }
122             finally
123             {
124                 writer.Close();
125             }
126         }
127 
128         public void Load(string filename)
129         {
130             StreamReader reader;
131             int count;
132             Shape s;
133             string kind;
134 
135             reader = new StreamReader(filename);
136             try
137             {
138                 Background = reader.ReadColor();
139                 count = reader.ReadInteger();
140                 _shapes.Clear();
141 
142                 for (int i = 0; i < count; i++)
143                 {
144                     kind = reader.ReadLine();
145 
146                     switch (kind)
147                     {
148                         case "Rectangle":
149                             s = new MyRectangle();
150                             break;
151 
152                         case "Circle":
153                             s = new MyCircle();
```

```
154                             break;
155
156                         case "Line":
157                             s = new MyLine();
158                             break;
159
160                         default:
161                             throw new InvalidDataException("Unknown shape kind: " +
                                ↪  kind);
162
163                     }
164
165                     s.LoadFrom(reader);
166                     _shapes.Add(s);
167                 }
168             }
169             finally
170             {
171                 reader.Close();
172             }
173         }
174     }
175 }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;

namespace DifferentShapes
{
    public abstract class Shape
    {
        private Color _color;
        private float _x, _y;
        private bool _selected;

        //Shape constructor
        public Shape() : this(Color.Yellow)
        {

        }

        public Shape(Color color)
        {
            _color = color;
        }

        public Color Color
        {
            get
            {
                return _color;
            }
            set
            {
                _color = value;
            }
        }

        public float X
        {
            get
            {
                return _x;
            }
            set
            {
                _x = value;
            }
        }

        public float Y
        {
```

```
54              get
55              {
56                  return _y;
57              }
58              set
59              {
60                  _y = value;
61              }
62          }
63
64
65          public abstract void Draw();
66
67          public abstract bool IsAt(Point2D pt);
68
69          public bool Selected
70          {
71              get
72              {
73                  return _selected;
74              }
75              set
76              {
77                  _selected = value;
78              }
79          }
80
81          public abstract void DrawOutline();
82
83          public virtual void SaveTo(StreamWriter writer)
84          {
85              writer.WriteColor(Color);
86              writer.WriteLine(X);
87              writer.WriteLine(Y);
88          }
89
90          public virtual void LoadFrom(StreamReader reader)
91          {
92              Color = reader.ReadColor();
93              X = reader.ReadInteger();
94              Y = reader.ReadInteger();
95          }
96      }
97  }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;

namespace DifferentShapes
{
    public class MyRectangle : Shape //MyRectangle class now inherits all the features from the Shape class
    {
        private int _height, _width;

        //default constructor
        public MyRectangle() : this(Color.Green, 0, 0, 100, 100)
        {}

        //constructor that takes in parameters
        public MyRectangle(Color clr, float x, float y, int width, int height) : base (clr)
        {
            X = x;
            Y = y;
            Width = width;
            Height = height;
        }

        //Width property
        public int Width
        {
            get
            {
                return _width;
            }
            set
            {
                _width = value;
            }
        }

        //Height property
        public int Height
        {
            get
            {
                return _height;
            }
            set
            {
                _height = value;
            }
```

```
52          }
53
54          //override draw method to draw rectangles only
55          public override void Draw()
56          {
57              if (Selected)
58              {
59                  DrawOutline();
60              }
61              SplashKit.FillRectangle(Color, X, Y, _width, _height);
62          }
63
64          //override draw method to draw outlines for rectangles
65          public override void DrawOutline()
66          {
67              SplashKit.DrawRectangle(Color.Black, X-2, Y-2, Width + 4, Height + 4);
68          }
69
70          //checking if the cursor is at a drawn rectangle
71          public override bool IsAt(Point2D pt)
72          {
73              //the X and Y are the top left corner of the rectangle
74              //therefore, check if mouse x-position is in the range [X, X + Width]
                ↪  and if the mouse y-position is in the range [Y, Y + height]
75              if ((pt.X >= X && pt.X <= (X + Width)) && (pt.Y) >= Y && pt.Y <= (Y +
                ↪  Height))
76              {
77                  return true;
78              }
79              else
80              {
81                  return false;
82              }
83          }
84
85          public override void SaveTo(StreamWriter writer)
86          {
87              writer.WriteLine("Rectangle");
88              base.SaveTo(writer);
89              writer.WriteLine(Width);
90              writer.WriteLine(Height);
91          }
92
93          public override void LoadFrom(StreamReader reader)
94          {
95              base.LoadFrom(reader);
96              Width = reader.ReadInteger();
97              Height = reader.ReadInteger();
98          }
99      }
100 }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;

namespace DifferentShapes
{
    public class MyCircle : Shape
    {
        private int _radius; //radius field to store the radius of the circle

        //radius property, allows the user to get and set the radius
        public int Radius
        {
            get
            {
                return _radius;
            }
            set
            {
                _radius = value;
            }
        }

        //default constructor calling the other constructor and passing the color
        ↪  blue and radius 50
        public MyCircle() : this(Color.Blue, 50)
        {}

        //constructor with parameters
        public MyCircle(Color clr, int radius) : base (clr)
        {
            _radius = radius;
        }

        //override method to draw circles only
        public override void Draw()
        {
            if (Selected)
            {
                DrawOutline();
            }
            SplashKit.FillCircle(Color, X, Y, _radius);
        }

        //override method to draw the outline for the circle
        public override void DrawOutline()
        {
            //drawing outline with a radius bigger than the circle by 2 pixels
            SplashKit.DrawCircle(Color.Black, X, Y, _radius + 2);
```

```
53          }
54
55          public override bool IsAt(Point2D pt)
56          {
57              //checking if the point is at the circle
58              //since for the circle, X and Y are the coordinates of the circle, then
                ↪  we need to cover the entier circle
59              //checking if the mouse x-position is in the range [X - radius, X +
                ↪  radius] and if the mouse y-position is in the range [Y - radius, Y
                ↪  + radius]
60              Circle circle = SplashKit.CircleAt(X, Y, _radius);
61              if (SplashKit.PointInCircle(pt, circle))
62              {
63                  return true;
64              }
65              else
66              {
67                  return false;
68              }
69          }
70
71          public override void SaveTo(StreamWriter writer)
72          {
73              writer.WriteLine("Circle");
74              base.SaveTo(writer);
75              writer.WriteLine(Radius);
76          }
77
78          public override void LoadFrom(StreamReader reader)
79          {
80              base.LoadFrom(reader);
81              Radius = reader.ReadInteger();
82          }
83      }
84  }
```

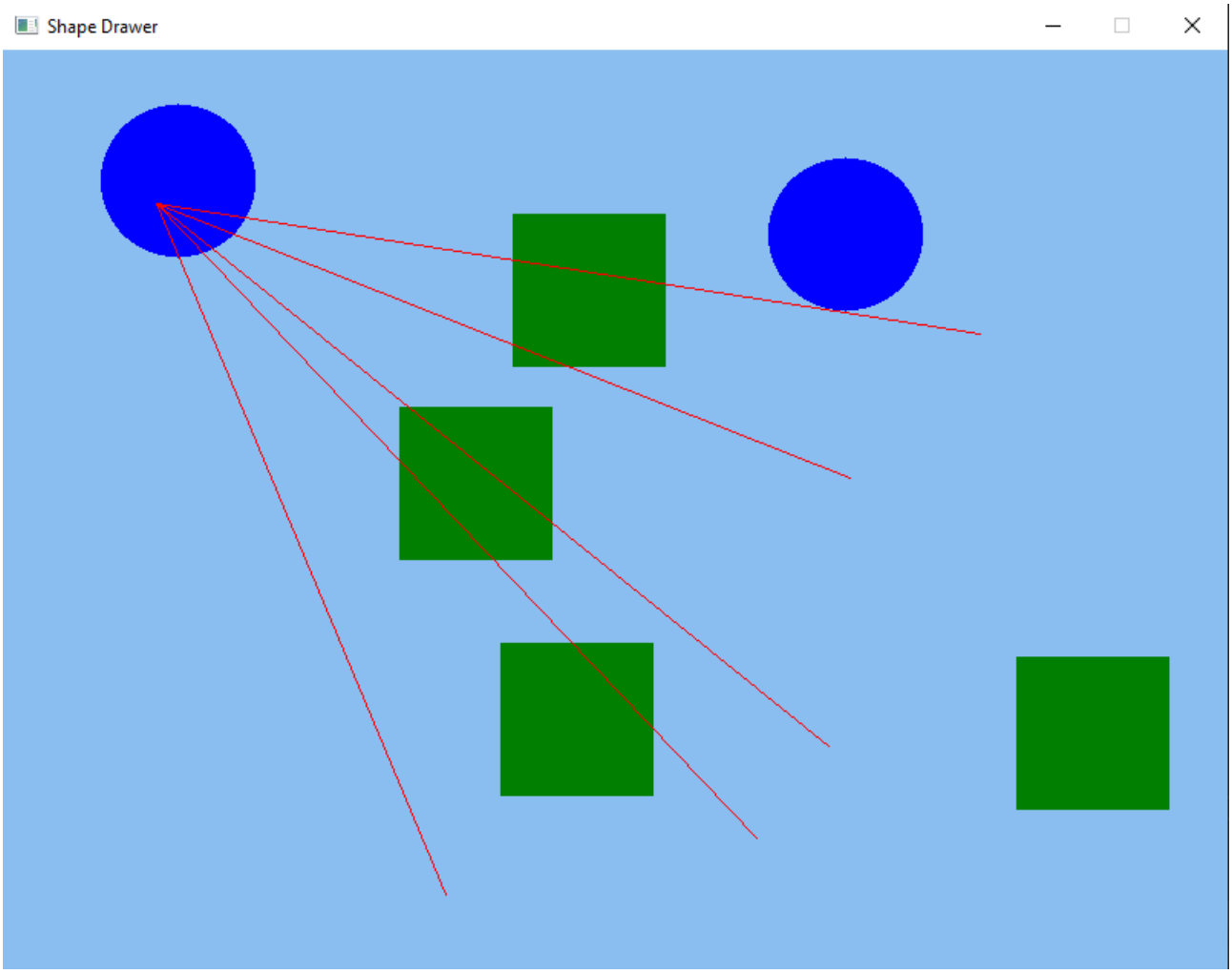
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;

namespace DifferentShapes
{
    public class MyLine : Shape
    {
        private float _endX;
        private float _endY;

        //default constructor
        public MyLine() : this(Color.Red, 0, 0, 100, 100)
        {}

        //constructor that takes in parameters
        public MyLine(Color clr, float starX, float starY, float endX, float endY)
        ↪  : base (clr)
        {
            X = starX;
            Y = starY;
            EndX = endX;
            EndY = endY;
        }

        //EndX property
        public float EndX
        {
            get
            {
                return _endX;
            }
            set
            {
                _endX = value;
            }
        }

        //EndY property
        public float EndY
        {
            get
            {
                return _endY;
            }
            set
            {
                _endY = value;
            }
```

```
53          }
54
55          //override draw method to draw lines
56          public override void Draw()
57          {
58              if(Selected)
59              {
60                  DrawOutline();
61              }
62              SplashKit.DrawLine(Color, X, Y, EndX, EndY);
63          }
64
65          //override draw outline method to draw two filled circles at both ends of
            ↪  the line using the coordinates (X,Y) and (EndX, EndY)
66          public override void DrawOutline()
67          {
68              SplashKit.FillCircle(Color, X, Y, 3);
69              SplashKit.FillCircle(Color, EndX, EndY, 3);
70          }
71
72          public override bool IsAt(Point2D pt)
73          {
74              //defining the line with the given coordinates (X,Y) and (EndX, EndY)
                ↪  using SplashKit's LineFrom
75              Line line = SplashKit.LineFrom(X, Y, EndX, EndY);
76              //checking if the point is close to the line, less than 10 pixels
77              if(SplashKit.PointLineDistance(pt, line) < 10)
78              {
79                  return true;
80              }
81              else
82              {
83                  return false;
84              }
85          }
86
87          public override void SaveTo(StreamWriter writer)
88          {
89              writer.WriteLine("Line");
90              base.SaveTo(writer);
91              writer.WriteLine(EndX);
92              writer.WriteLine(EndY);
93          }
94
95          public override void LoadFrom(StreamReader reader)
96          {
97              base.LoadFrom(reader);
98              EndX = reader.ReadInteger();
99              EndY = reader.ReadInteger();
100         }
101     }
102 }
```

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

namespace DifferentShapes
{
    public class Program
    {
        //creating ShaeKind that will determine what type of shape the user
        ↪  requested to draw
        private enum ShapeKind
        {
            Rectangle,
            Circle,
            Line
        }
        public static void Main()
        {
            //Opens a new window called Shape Drawer with dimensions 800 x 600
            new Window("Shape Drawer", 800, 600);

            Drawing myDrawing = new Drawing();
            //setting the default shapeKind to circle
            ShapeKind kindToAdd = ShapeKind.Circle;


            //leaves the window open unless requested to be closed
            do
            {
                SplashKit.ProcessEvents(); //allows Splashkit to react to user
                ↪  interactions
                SplashKit.ClearScreen();
                //if the user presses the R Key, then they've chosen to draw a
                ↪  rectangle
                if(SplashKit.KeyTyped(KeyCode.RKey))
                {
                    kindToAdd = ShapeKind.Rectangle;
                }
                //if the user presses the C Key, then they've chosen to draw a
                ↪  circle
                else if (SplashKit.KeyTyped(KeyCode.CKey))
                {
                    kindToAdd = ShapeKind.Circle;
                }
                //if the user presses the L Key, then they've chosen to draw a line
                else if (SplashKit.KeyTyped(KeyCode.LKey))
                {
                    kindToAdd = ShapeKind.Line;
                }

                //drawing the shapes when the mouse's left button is pressed, after
                ↪  taking in the user selection
                if (SplashKit.MouseClicked(MouseButton.LeftButton))
```

```
49              {
50                  Shape newShape;
51
52                  if(kindToAdd == ShapeKind.Circle)
53                  {
54                      //add a new Shape to your Drawing object based on the
                        ↪  mouse's location.
55                      MyCircle newCircle = new MyCircle();
56
57                      newShape = newCircle;
58                  }
59                  else if(kindToAdd == ShapeKind.Rectangle)
60                  {
61                      //add a new Shape to your Drawing object based on the
                        ↪  mouse's location.
62                      MyRectangle newRect = new MyRectangle();
63
64                      newShape = newRect;
65                  }
66                  else
67                  {
68                      MyLine newLine = new MyLine();
69                      newShape = newLine;
70                  }
71
72                  newShape.X = SplashKit.MouseX();
73                  newShape.Y = SplashKit.MouseY();
74                  myDrawing.AddShape(newShape);
75
76              }
77
78              //setting the background color to a random color when the user
                ↪  presses space
79              if (SplashKit.KeyTyped(KeyCode.SpaceKey) == true)
80              {
81                  myDrawing.Background = SplashKit.RandomRGBColor(255);
82              }
83
84              //if the mouse's right button is clicked, select the shape (if the
                ↪  mouse is at a shape)
85              if (SplashKit.MouseClicked(MouseButton.RightButton) == true)
86              {
87                  myDrawing.SelectShapeAt(SplashKit.MousePosition());
88              }
89
90              //pressing backspace or delete, allows the user to delete the
                ↪  selected shapes
91              if (SplashKit.KeyTyped(KeyCode.BackspaceKey) == true ||
                ↪  SplashKit.KeyTyped(KeyCode.DeleteKey) == true)
92              {
93                  List<Shape> selectedShapes = myDrawing.SelectedShapes;
94                  foreach (Shape s in selectedShapes)
95                  {
```

```
96                          myDrawing.RemoveShape(s);
97                      }
98                  }
99
100                 if(SplashKit.KeyTyped(KeyCode.SKey))
101                 {
102                     myDrawing.Save("TestDrawing.txt");
103                 }
104
105                 if (SplashKit.KeyTyped(KeyCode.OKey))
106                 {
107                     try
108                     {
109                         myDrawing.Load("TestDrawing.txt");
110                     } catch(Exception e)
111                     {
112                         Console.Error.WriteLine("Error loading file : {0}",
                            ↪  e.Message);
113                     }
114
115                 }
116
117                 myDrawing.Draw();
118
119                 //refresh
120                 SplashKit.RefreshScreen();
121
122             } while (!SplashKit.WindowCloseRequested("Shape Drawer"));
123         }
124     }
125 }
```


Shape Drawer

15 Distinction Task 5.3: Drawing Program - Flexible Saving

Drawing programs have a natural affinity with object oriented design and programming, with easy to see roles and functionality. In this task you will start to create an object oriented drawing program.

Date	Author	Comment
2021/09/07 00:35	Marella Morad	Demo video link
2021/09/07 00:35	Marella Morad	https://youtu.be/5uM3V7_u32w

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Distinction Task 5.3: Drawing Program - Flexible Saving

Submitted By:
Marella Morad
103076428
2021/09/07 00:32

Tutor:
Joshua Wright

September 7, 2021

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;

namespace DifferentShapes
{
    public abstract class Shape
    {
        private static Dictionary<string, Type> _ShapeClassRegistry = new
        ↪ Dictionary<string, Type>();

        public static void RegisterShape(string name, Type t)
        {
            _ShapeClassRegistry[name] = t;
        }

        public static Shape CreateShape(string name)
        {
            return (Shape)Activator.CreateInstance(_ShapeClassRegistry[name]);
        }

        static string GetKey(Type type)
        {
            foreach(string key in _ShapeClassRegistry.Keys)
            {
                if(_ShapeClassRegistry[key] == type)
                {
                    return key;
                }
            }
            return null;
        }

        private Color _color;
        private float _x, _y;
        private bool _selected;

        //Shape constructor
        public Shape() : this(Color.Yellow)
        {

        }

        public Shape(Color color)
        {
            _color = color;
        }

        public Color Color
```

```
53          {
54              get
55              {
56                  return _color;
57              }
58              set
59              {
60                  _color = value;
61              }
62          }
63
64          public float X
65          {
66              get
67              {
68                  return _x;
69              }
70              set
71              {
72                  _x = value;
73              }
74          }
75
76          public float Y
77          {
78              get
79              {
80                  return _y;
81              }
82              set
83              {
84                  _y = value;
85              }
86          }
87
88
89          public abstract void Draw();
90
91          public abstract bool IsAt(Point2D pt);
92
93          public bool Selected
94          {
95              get
96              {
97                  return _selected;
98              }
99              set
100             {
101                 _selected = value;
102             }
103         }
104
105         public abstract void DrawOutline();
```

```
106
107         public virtual void SaveTo(StreamWriter writer)
108         {
109             writer.WriteLine(GetKey(this.GetType()));
110             writer.WriteColor(Color);
111             writer.WriteLine(X);
112             writer.WriteLine(Y);
113         }
114
115         public virtual void LoadFrom(StreamReader reader)
116         {
117             Color = reader.ReadColor();
118             X = reader.ReadInteger();
119             Y = reader.ReadInteger();
120         }
121     }
122 }
```

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

namespace DifferentShapes
{
    public class Program
    {
        //creating ShaeKind that will determine what type of shape the user
        ↪  requested to draw
        private enum ShapeKind
        {
            Rectangle,
            Circle,
            Line
        }
        public static void Main()
        {
            //Registering the shapes
            Shape.RegisterShape("Rectangle", typeof(MyRectangle));
            Shape.RegisterShape("Circle", typeof(MyCircle));
            Shape.RegisterShape("Line", typeof(MyLine));

            //Opens a new window called Shape Drawer with dimensions 800 x 600
            new Window("Shape Drawer", 800, 600);

            Drawing myDrawing = new Drawing();
            //setting the default shapeKind to circle
            ShapeKind kindToAdd = ShapeKind.Circle;


            //leaves the window open unless requested to be closed
            do
            {
                SplashKit.ProcessEvents(); //allows Splashkit to react to user
                ↪  interactions
                SplashKit.ClearScreen();
                //if the user presses the R Key, then they've chosen to draw a
                ↪  rectangle
                if(SplashKit.KeyTyped(KeyCode.RKey))
                {
                    kindToAdd = ShapeKind.Rectangle;
                }
                //if the user presses the C Key, then they've chosen to draw a
                ↪  circle
                else if (SplashKit.KeyTyped(KeyCode.CKey))
                {
                    kindToAdd = ShapeKind.Circle;
                }
                //if the user presses the L Key, then they've chosen to draw a line
                else if (SplashKit.KeyTyped(KeyCode.LKey))
                {
                    kindToAdd = ShapeKind.Line;
```

```
50                  }
51
52                  //drawing the shapes when the mouse's left button is pressed, after
                    ↪  taking in the user selection
53                  if (SplashKit.MouseClicked(MouseButton.LeftButton))
54                  {
55                      Shape newShape;
56
57                      if(kindToAdd == ShapeKind.Circle)
58                      {
59                          //add a new Shape to your Drawing object based on the
                            ↪  mouse's location.
60                          MyCircle newCircle = new MyCircle();
61
62                          newShape = newCircle;
63                      }
64                      else if(kindToAdd == ShapeKind.Rectangle)
65                      {
66                          //add a new Shape to your Drawing object based on the
                            ↪  mouse's location.
67                          MyRectangle newRect = new MyRectangle();
68
69                          newShape = newRect;
70                      }
71                      else
72                      {
73                          MyLine newLine = new MyLine();
74                          newShape = newLine;
75                      }
76
77                      newShape.X = SplashKit.MouseX();
78                      newShape.Y = SplashKit.MouseY();
79                      myDrawing.AddShape(newShape);
80
81                  }
82
83                  //setting the background color to a random color when the user
                    ↪  presses space
84                  if (SplashKit.KeyTyped(KeyCode.SpaceKey) == true)
85                  {
86                      myDrawing.Background = SplashKit.RandomRGBColor(255);
87                  }
88
89                  //if the mouse's right button is clicked, select the shape (if the
                    ↪  mouse is at a shape)
90                  if (SplashKit.MouseClicked(MouseButton.RightButton) == true)
91                  {
92                      myDrawing.SelectShapeAt(SplashKit.MousePosition());
93                  }
94
95                  //pressing backspace or delete, allows the user to delete the
                    ↪  selected shapes
96                  if (SplashKit.KeyTyped(KeyCode.BackspaceKey) == true ||
                    ↪  SplashKit.KeyTyped(KeyCode.DeleteKey) == true)
```

```
97                  {
98                      List<Shape> selectedShapes = myDrawing.SelectedShapes;
99                      foreach (Shape s in selectedShapes)
100                     {
101                         myDrawing.RemoveShape(s);
102                     }
103                 }
104
105                 if(SplashKit.KeyTyped(KeyCode.SKey))
106                 {
107                     myDrawing.Save("TestDrawing.txt");
108                 }
109
110                 if (SplashKit.KeyTyped(KeyCode.OKey))
111                 {
112                     try
113                     {
114                         myDrawing.Load("TestDrawing.txt");
115                     } catch(Exception e)
116                     {
117                         Console.Error.WriteLine("Error loading file : {0}",
                            ↪  e.Message);
118                     }
119
120                 }
121
122                 myDrawing.Draw();
123
124                 //refresh
125                 SplashKit.RefreshScreen();
126
127             } while (!SplashKit.WindowCloseRequested("Shape Drawer"));
128         }
129     }
130 }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;

namespace DifferentShapes
{
    public class MyRectangle : Shape //MyRectangle class now inherits all the features from the Shape class
    {
        private int _height, _width;

        //default constructor
        public MyRectangle() : this(Color.Green, 0, 0, 100, 100)
        {}

        //constructor that takes in parameters
        public MyRectangle(Color clr, float x, float y, int width, int height) : base (clr)
        {
            X = x;
            Y = y;
            Width = width;
            Height = height;
        }

        //Width property
        public int Width
        {
            get
            {
                return _width;
            }
            set
            {
                _width = value;
            }
        }

        //Height property
        public int Height
        {
            get
            {
                return _height;
            }
            set
            {
                _height = value;
            }
```

```
52          }
53
54          //override draw method to draw rectangles only
55          public override void Draw()
56          {
57              if (Selected)
58              {
59                  DrawOutline();
60              }
61              SplashKit.FillRectangle(Color, X, Y, _width, _height);
62          }
63
64          //override draw method to draw outlines for rectangles
65          public override void DrawOutline()
66          {
67              SplashKit.DrawRectangle(Color.Black, X-2, Y-2, Width + 4, Height + 4);
68          }
69
70          //checking if the cursor is at a drawn rectangle
71          public override bool IsAt(Point2D pt)
72          {
73              //the X and Y are the top left corner of the rectangle
74              //therefore, check if mouse x-position is in the range [X, X + Width]
                ↪  and if the mouse y-position is in the range [Y, Y + height]
75              if ((pt.X >= X && pt.X <= (X + Width)) && (pt.Y) >= Y && pt.Y <= (Y +
                ↪  Height))
76              {
77                  return true;
78              }
79              else
80              {
81                  return false;
82              }
83          }
84
85          public override void SaveTo(StreamWriter writer)
86          {
87              base.SaveTo(writer);
88              writer.WriteLine(Width);
89              writer.WriteLine(Height);
90          }
91
92          public override void LoadFrom(StreamReader reader)
93          {
94              base.LoadFrom(reader);
95              Width = reader.ReadInteger();
96              Height = reader.ReadInteger();
97          }
98      }
99  }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;

namespace DifferentShapes
{
    public class MyLine : Shape
    {
        private float _endX;
        private float _endY;

        //default constructor
        public MyLine() : this(Color.Red, 0, 0, 100, 100)
        {}

        //constructor that takes in parameters
        public MyLine(Color clr, float starX, float starY, float endX, float endY)
        ↪  : base (clr)
        {
            X = starX;
            Y = starY;
            EndX = endX;
            EndY = endY;
        }

        //EndX property
        public float EndX
        {
            get
            {
                return _endX;
            }
            set
            {
                _endX = value;
            }
        }

        //EndY property
        public float EndY
        {
            get
            {
                return _endY;
            }
            set
            {
                _endY = value;
            }
```

```
53          }
54
55          //override draw method to draw lines
56          public override void Draw()
57          {
58              if(Selected)
59              {
60                  DrawOutline();
61              }
62              SplashKit.DrawLine(Color, X, Y, EndX, EndY);
63          }
64
65          //override draw outline method to draw two filled circles at both ends of
            ↪  the line using the coordinates (X,Y) and (EndX, EndY)
66          public override void DrawOutline()
67          {
68              SplashKit.FillCircle(Color, X, Y, 3);
69              SplashKit.FillCircle(Color, EndX, EndY, 3);
70          }
71
72          public override bool IsAt(Point2D pt)
73          {
74              //defining the line with the given coordinates (X,Y) and (EndX, EndY)
                ↪  using SplashKit's LineFrom
75              Line line = SplashKit.LineFrom(X, Y, EndX, EndY);
76              //checking if the point is close to the line, less than 10 pixels
77              if(SplashKit.PointLineDistance(pt, line) < 10)
78              {
79                  return true;
80              }
81              else
82              {
83                  return false;
84              }
85          }
86
87          public override void SaveTo(StreamWriter writer)
88          {
89              base.SaveTo(writer);
90              writer.WriteLine(EndX);
91              writer.WriteLine(EndY);
92          }
93
94          public override void LoadFrom(StreamReader reader)
95          {
96              base.LoadFrom(reader);
97              EndX = reader.ReadInteger();
98              EndY = reader.ReadInteger();
99          }
100     }
101 }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;

namespace DifferentShapes
{
    public class MyCircle : Shape
    {
        private int _radius; //radius field to store the radius of the circle

        //radius property, allows the user to get and set the radius
        public int Radius
        {
            get
            {
                return _radius;
            }
            set
            {
                _radius = value;
            }
        }

        //default constructor calling the other constructor and passing the color
        ↪  blue and radius 50
        public MyCircle() : this(Color.Blue, 50)
        {}

        //constructor with parameters
        public MyCircle(Color clr, int radius) : base (clr)
        {
            _radius = radius;
        }

        //override method to draw circles only
        public override void Draw()
        {
            if (Selected)
            {
                DrawOutline();
            }
            SplashKit.FillCircle(Color, X, Y, _radius);
        }

        //override method to draw the outline for the circle
        public override void DrawOutline()
        {
            //drawing outline with a radius bigger than the circle by 2 pixels
            SplashKit.DrawCircle(Color.Black, X, Y, _radius + 2);
```

```
53          }
54
55          public override bool IsAt(Point2D pt)
56          {
57              //checking if the point is at the circle
58              //since for the circle, X and Y are the coordinates of the circle, then
                ↪  we need to cover the entier circle
59              //checking if the mouse x-position is in the range [X - radius, X +
                ↪  radius] and if the mouse y-position is in the range [Y - radius, Y
                ↪  + radius]
60              Circle circle = SplashKit.CircleAt(X, Y, _radius);
61              if (SplashKit.PointInCircle(pt, circle))
62              {
63                  return true;
64              }
65              else
66              {
67                  return false;
68              }
69          }
70
71          public override void SaveTo(StreamWriter writer)
72          {
73              base.SaveTo(writer);
74              writer.WriteLine(Radius);
75          }
76
77          public override void LoadFrom(StreamReader reader)
78          {
79              base.LoadFrom(reader);
80              Radius = reader.ReadInteger();
81          }
82      }
83  }
```

16 Credit Task 6.1: Case Study - Iterations 5 and 6

Object oriented programming makes best sense with larger programs. The case study will be your opportunity to create a larger program and better see how these abstractions make it easier to create software solutions.

Date	Author	Comment
2021/09/15 00:33	Marella Morad	Demo video link
2021/09/15 00:33	Marella Morad	https://youtu.be/L2YvTr2c6Js

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Credit Task 6.1: Case Study - Iterations 5 and 6

Submitted By:
Marella Morad
103076428
2021/10/25 14:45

Tutor:
Joshua Wright

October 25, 2021

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

namespace SwinAdventure
{
    class Program
    {
        static void Main(string[] args)
        {
            //Create the objects of the game:
            Player player;
            Location hall, garden, lab;
            Path pathHtoL, pathHtoG, pathGtoH, pathGtoL, pathLtoH, pathLtoG;
            CommandProcessor command;
            Item shovel, sword, gem, pc, rope, scope, goggles, coat;
            Bag bag, cart, handbag;

            //Add locations
            hall = new Location(new string[] { "hallway" }, "Hallway", "This is a long well lit Hallway");

            garden = new Location(new string[] { "garden" }, "Garden", "This is a big garden with a lot of secret spots");

            lab = new Location(new string[] { "lab" }, "Laboratory", "This is where the magic is created");

            //Setup the paths
            pathHtoL = new Path(new string[] { "south", "s", "down" }, "South", "slide", lab);
            pathHtoG = new Path(new string[] { "east", "e", "right" }, "East", "small door", garden);

            pathGtoH = new Path(new string[] { "west", "w", "left" }, "West", "small door", hall);
            pathGtoL = new Path(new string[] { "sw", "south_west" }, "South West", "roller coaster", lab);

            pathLtoH = new Path(new string[] { "n", "north", "up" }, "North", "ladder", hall);
            pathLtoG = new Path(new string[] { "ne", "north_east" }, "North East", "roller coaster", garden);

            //Add the paths to each location
            hall.AddPath(pathHtoG);
            hall.AddPath(pathHtoL);

            garden.AddPath(pathGtoL);
            garden.AddPath(pathGtoH);

```

```
45              lab.AddPath(pathLtoG);
46              lab.AddPath(pathLtoH);
47
48              //Create the items
49              shovel = new Item(new string[] { "shovel" }, "shovel", "This is a might
                ↪  fine shovel");
50              sword = new Item(new string[] { "sword" }, "bronze sword", "This is a
                ↪  shiny sword");
51              gem = new Item(new string[] { "gem" }, "red gem", "This is a shiny red
                ↪  gem");
52              pc = new Item(new string[] { "pc" }, "small computer", "This is a
                ↪  computer from the future");
53              rope = new Item(new string[] { "rope" }, "rope", "This is the strongest
                ↪  rope ever");
54              scope = new Item(new string[] { "scope" }, "hand scope", "This is a
                ↪  golden scope with x1000 magnification");
55              goggles = new Item(new string[] { "goggles" }, "IR goggles", "These are
                ↪  the best night vision goggles");
56              coat = new Item(new string[] { "coat" }, "lab coat", "This is a white
                ↪  lab coat");
57
58              //Create the bags
59              bag = new Bag(new string[] { "bag" }, "plastic bag", "This is a clear
                ↪  plastic bag");
60              cart = new Bag(new string[] { "cart" }, "gardening cart", "This is a
                ↪  big gardening cart that fits large items");
61              handbag = new Bag(new string[] { "handbag" }, "handbag", "This is a
                ↪  leather handbag");
62
63              //Setup the game by adding items to locations and bags
64              hall.Inventory.Put(sword);
65              handbag.Inventory.Put(gem);
66              hall.Inventory.Put(handbag);
67
68              garden.Inventory.Put(scope);
69              garden.Inventory.Put(rope);
70              cart.Inventory.Put(shovel);
71              garden.Inventory.Put(cart);
72
73              lab.Inventory.Put(coat);
74              lab.Inventory.Put(pc);
75              bag.Inventory.Put(goggles);
76              lab.Inventory.Put(bag);
77
78              command = new CommandProcessor();
79
80              Console.WriteLine("Please enter your player's name: ");
81              string playerName = Console.ReadLine();
82              //getting the player's description from Console (user)
83              Console.WriteLine("Please enter your player's description: ");
84              string playerDesc = Console.ReadLine();
85              //creating the player using info from the user:
86              player = new Player(playerName, playerDesc);
```

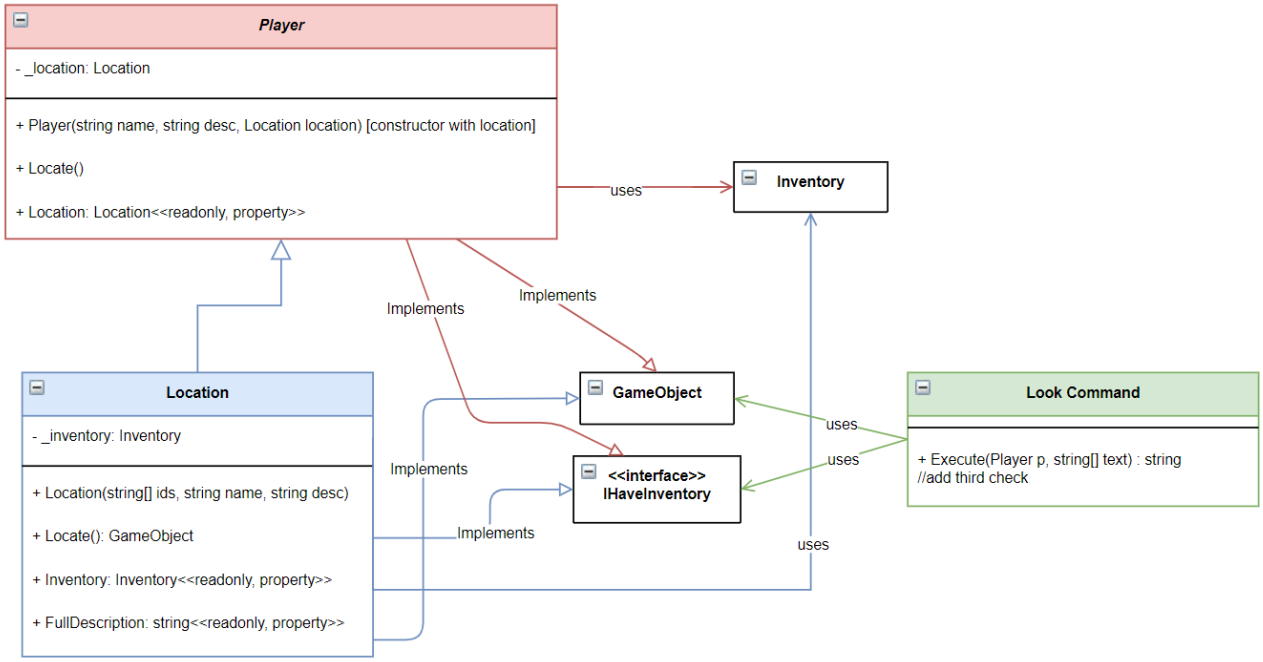
```
87
88              player.Location = hall; //default location of the player
89
90              Console.WriteLine(" ");
91              Console.WriteLine("Welcome to Swin-Adventure, {0}!", playerName);
92              Console.WriteLine("You have arrived in the " + player.Location.Name);
93
94              bool done = false; //to keep looping until the player chooses to stop
95              while(!done)
96              {
97                  Console.Write("Command -> ");
98                  string userCommand = Console.ReadLine(); //getting the command from
                    ↪ the user
99                  string[] commandArr = new[] { userCommand }; //converting into
                    ↪ string[] array
100                 commandArr = userCommand.Split(" "); //splitting the command based
                    ↪ on spaces between words
101                 Console.WriteLine(" ");
102                 string reply = command.Execute(player, commandArr);
103                 Console.WriteLine(reply); //executing command using command
                    ↪ processor
104                 //Console.WriteLine(" ");
105
106                 //checking if the player wants to enter another command
107
108                 if(reply.ToLower() == "good bye") //running for 10 times before
                    ↪ asking the user if they want to stop
109                 {
110                     done = true;
111                 }
112             }
113         }
114     }
115 }
```

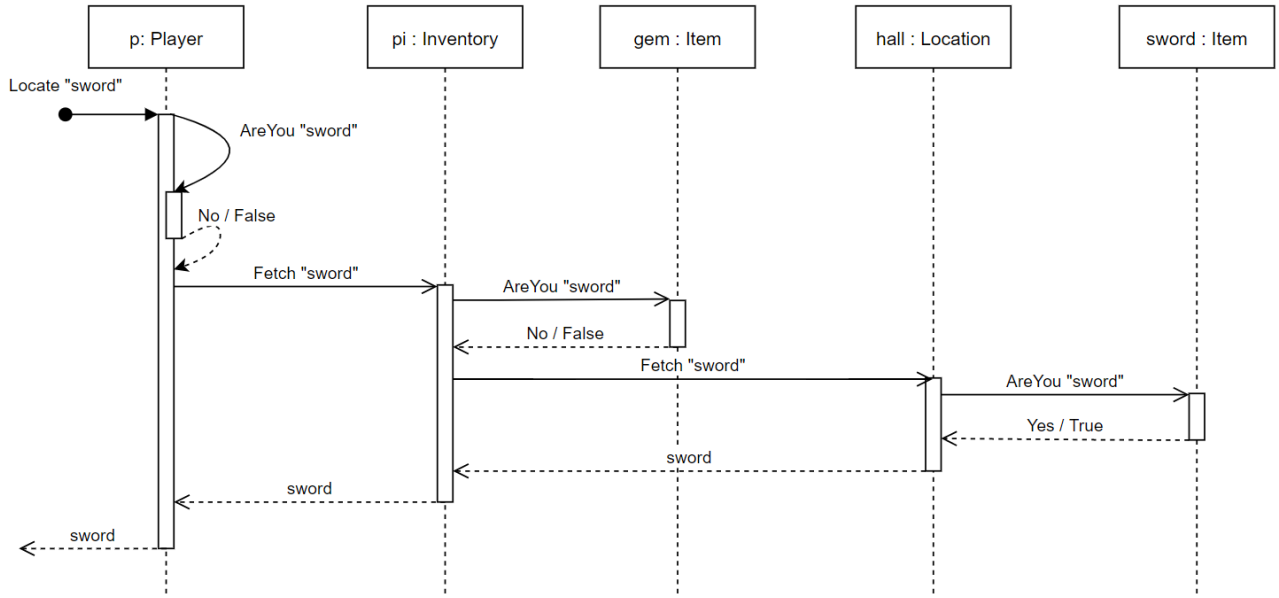
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Location : GameObject, IHaveInventory
    {
        private Inventory _inventory; //to conatin items
        private List<Path> _paths;

        public Location(string[] ids, string name, string desc) : base(ids, name,
        ↪ desc)
        {
            _inventory = new Inventory();
            _paths = new List<Path>();
        }

        public GameObject Locate(string id)
        {
            return _inventory.Fetch(id); //to locate items in the location inventory
        }

        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }

        public void AddPath(Path path)
        {
            _paths.Add(path); //adding the paths to the location
        }

        public Path GetPath(string id)
        {
            foreach(Path p in _paths)
            {
                if(p.AreYou(id))
                {
                    return p;
                }
            }

            return null;
        }

        public override string FullDescription
        {
```

```
53              get
54              {
55                  //adding the paths of a location
56                  string paths = "";
57                  int count = 0;
58                  foreach (Path p in _paths)
59                  {
60                      if(count == 0)
61                      {
62                          paths = p.Name.ToLower();
63                      }
64                      else
65                      {
66                          paths = paths + " and " + p.Name.ToLower();
67                      }
68                      count++;
69                  }
70
71                  string fullDesc = "You are in the " + Name + "\n" +
                    ↪  base.FullDescription
72                                  + "\nThere are exits to the " + paths //addition of
                                    ↪  exits available
73                                  + "\nIn this room you can see:\n" +
                                    ↪  _inventory.ItemList;
74
75                  return fullDesc;
76              }
77          }
78      }
79  }
```

```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SwinAdventure;

namespace SwinAdventureTests
{
    [TestFixture]
    public class LocationUnitTests
    {
        Item shovel;
        Item sword;
        Item gem;

        Player player;

        Bag b1;
        Bag b2;

        LookCommand look;

        Location location;

        [SetUp]
        public void Setup()
        {
            shovel = new Item(new string[] { "shovel" }, "shovel", "This is a might fine shovel");
            sword = new Item(new string[] { "sword" }, "bronze sword", "This is a shiny sword");
            gem = new Item(new string[] { "gem" }, "red gem", "This is a shiny red gem");

            b1 = new Bag(new string[] { "bag" }, "plastic bag", "This is a clear plastic bag");
            b2 = new Bag(new string[] { "wallet" }, "wallet", "This is a black wallet");

            player = new Player("Marella", "The amazing player");

            look = new LookCommand();

            b2.Inventory.Put(shovel);
            b1.Inventory.Put(b2);

            player.Inventory.Put(shovel);
            player.Inventory.Put(b1);
            player.Inventory.Put(gem);

            location = new Location(new string[] { "closet" }, "small Closet", "A small dark closet, with an odd smell");
```

```
48          }
49
50          [Test]
51          public void TestIdentifiableLocation()
52          {
53              Assert.IsTrue(location.AreYou("closet"), "Location is not
                ↪  identifiable"); //location is identifiable by AreYou()
54          }
55
56          [Test]
57          public void TestLocateItems()
58          {
59              location.Inventory.Put(gem);
60              Assert.AreEqual(location.Locate("gem"), gem, "Gem cannot be found in
                ↪  the location");
61          }
62
63          [Test]
64          public void TestLocationFullDescription()
65          {
66              location.Inventory.Put(gem);
67              string expected =
68              "You are in the small Closet\n" +
69              "A small dark closet, with an odd smell\n" +
70              "There are exits to the \n" +
71              "In this room you can see:\n" +
72              "     a red gem (gem)\n";
73              Assert.AreEqual(location.FullDescription, expected);
74          }
75      }
76  }
```


Player
- _location: Location
+ Player(string name, string desc, Location location) [constructor with location]
+ Locate()
+ Location: Location<<readonly, property>>
uses
Inventory
Implements
Implements
Location
- _inventory: Inventory
+ Location(string[] ids, string name, string desc)
+ Locate(): GameObject
+ Inventory: Inventory<<readonly, property>>
+ FullDescription: string<<readonly, property>>
GameObject
<<interface>>
IHaveInventory
Implements
Implements
uses
uses
uses
Look Command
+ Execute(Player p, string[] text) : string
//add third check


p: Player
pi : Inventory
gem : Item
hall : Location
sword : Item
Locate "sword"
AreYou "sword"
No / False
Fetch "sword"
AreYou "sword"
No / False
Fetch "sword"
AreYou "sword"
Yes / True
sword
sword
sword

17 Credit Task 6.2: Custom Program Plan and Design

A plan and initial design for a custom program.

Date	Author	Comment
2021/10/13 10:25	Joshua Wright	I would like to discuss this task further with you.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Credit Task 6.2: Custom Program Plan and Design

Submitted By:
Marella Morad
103076428
2021/10/09 23:17

Tutor:
Joshua Wright

October 9, 2021

Design Overview for Mario (for now)

Name: Marella Morad
Student ID: 103076428

Summary of Program

My custom program will be an adventure game, mostly based on Super Mario and Red ball games. The idea behind the game is to have a player (main character) who will go through a series of three levels to finish the game. Each of the levels will have a different superpower that's given to the player at the start of the level. There aren't any other characters in the game, only obstacles which will try and stop the player from finishing the level. Like Red ball, some of the blocks will have spikes in them leading to the failure of the level (the player - fails to jump over). The player will need to pass the obstacles and collect the **key** to open the door to the next level. *The key will most likely be at a high place that the player needs to get to.* (The game initial sketch is attached at the end of this document)

As mentioned earlier, in each of the levels, there will be a different main superpower that the player will have. For instance, level one, the player will have high jumping power which will allow him to jump over obstacles that a normal jump cannot achieve. Since the superpowers are customised to each level, the levels are also customized so that the superpower is used.

Level	**Superpower**	**Description**	Key to use	Obstacles
1	High jump	The player can jump with double the magnitude of the normal jump	Enter	Higher blocks
2	Magnetic suit	Attracts the player to the magnetic blocks (some will be moving)	Enter (hold Enter)	Water/fire pools to cross
4	Opera (singing power)	Breaks glass obstacles with voice	Hold Enter	Glass doors along the way that break when the player sings

These superpowers are inspired from a game called It Takes Two

Required Roles

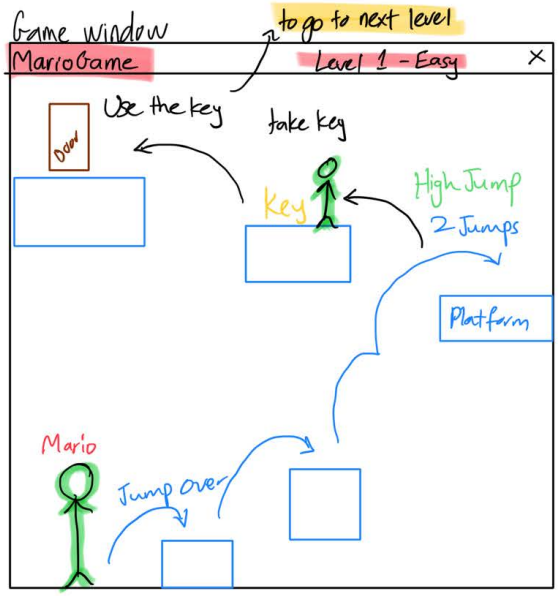
The required roles include:

- GameObject (abstract)
    - Holds properties and methods that are used in many other classes (such as draw, width, length and coordinates (x and y))
- Player
- Move (moves the player in either left, right or up (jump)) directions which will be specified in the identifiers of the Move Class.
- Block (different types of blocks) – the type of the block is specified in the constructor call
    - Normal blocks
    - Spiked blocks
- Supersuit (will provide the player with the superpower for the level)
    - HighJumpSuit
    - MagneticSuit
    - OperaSuit
        - All of the three classes will inherit from the Supersuit class (still incomplete in the UML diagram)
- Bag (the bag of the player + bag of the location)
- Key (item that the player needs to collect and use to open the door to the next level)
- Door (separate between the levels)
    - Open only when the player has the key.
- Level (three levels initially)
    - Level 1 uses blocks
    - Level 2 uses blocks
    - Level 3 uses blocks
- Size – abstract class will allow to change the size of the bitmap of the player to fit in smaller/larger places
    - Shrink
    - Enlarge
- Specific Obstacles:
    - GlassBorder
    - MagneticBlock

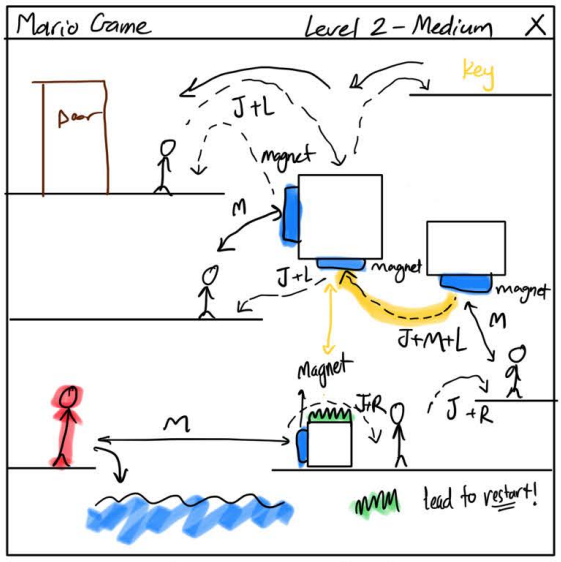
Table 1: Game details

Responsibility	Type Details	Notes
Player	_lastX, _lastY: double _supersuit: Supersuit _level: Level _nextLevel: Level _bag: Bag _playerBitmap: Bitmap	Move around the 2D game and finish the levels The player needs to be able to move in 2D (left and right) right to go forward and left to go backwards. And also be able to Jump. Based on the level, the player will be able to use the superpower by

		combining the letter S with the key responsible for the move. For example, high jump can be S and Space keys held together. The player then needs to collect a key when he is at the coordinates of the key, then use the key to pass the level.
Move	_x, _y: double (2-D movement) *Move left, right and up*	Hold the new coordinates of the player location The player jumps if the Boolean Jump in the Move constructor call is true, otherwise, will only move left or right
Size	Two size changes	The player can shrink or enlarge back to default
Supersuit	_suit: Suit (where the superpower will come from)	Puts the suit on the player (changes the bitmap of the player for visual purposes)
SuitType	Three suits in the game	Determines the suit type based on the level the player is in
Level	Three levels in the game	Start easy, then medium then hard.
Obstacles	Glass Border, MagneticBlock and High NormalBlock	Each level the player will be introduced to a new obstacle. First level – high normal blocks Second level – Magnetic blocks where the player can use their magnetic suit to be attracted to the magnet Last level – Glass border, the player needs to break the glass using their opera voice that comes with the suit for this level



High jump is Mario's superpower for this level

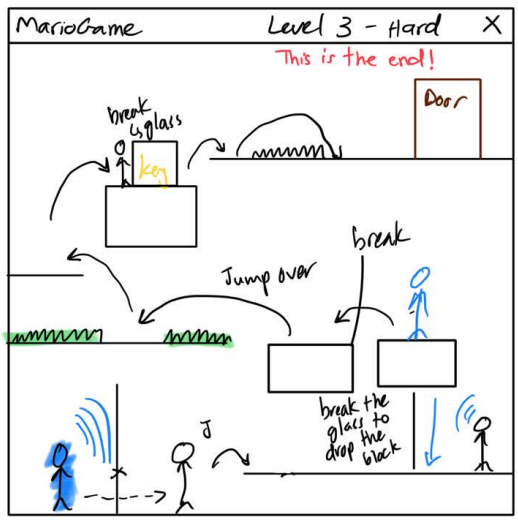


J - Jump (same x-coordinate)

J+R - Jump and move right

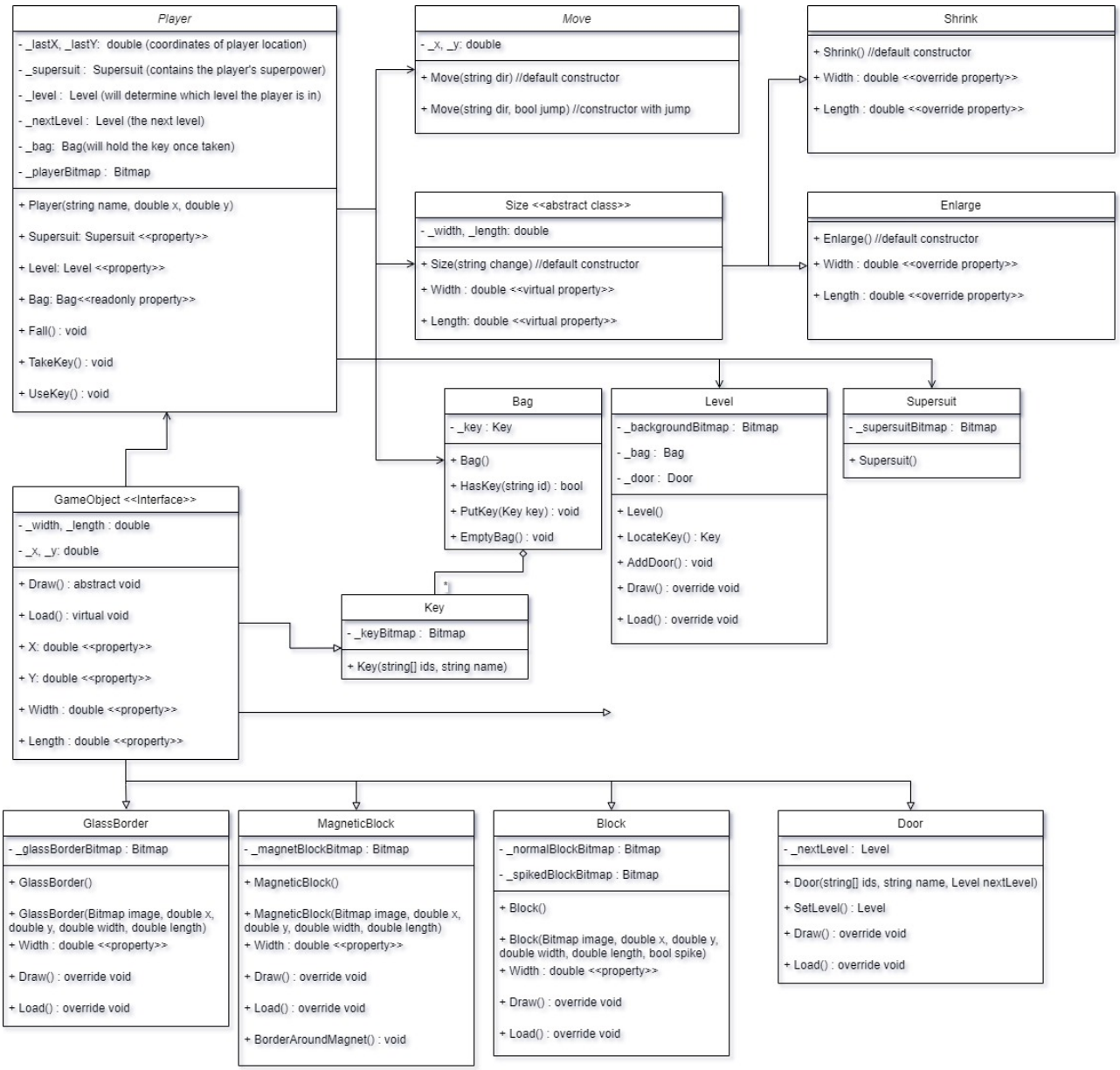
J+L - Jump and move left

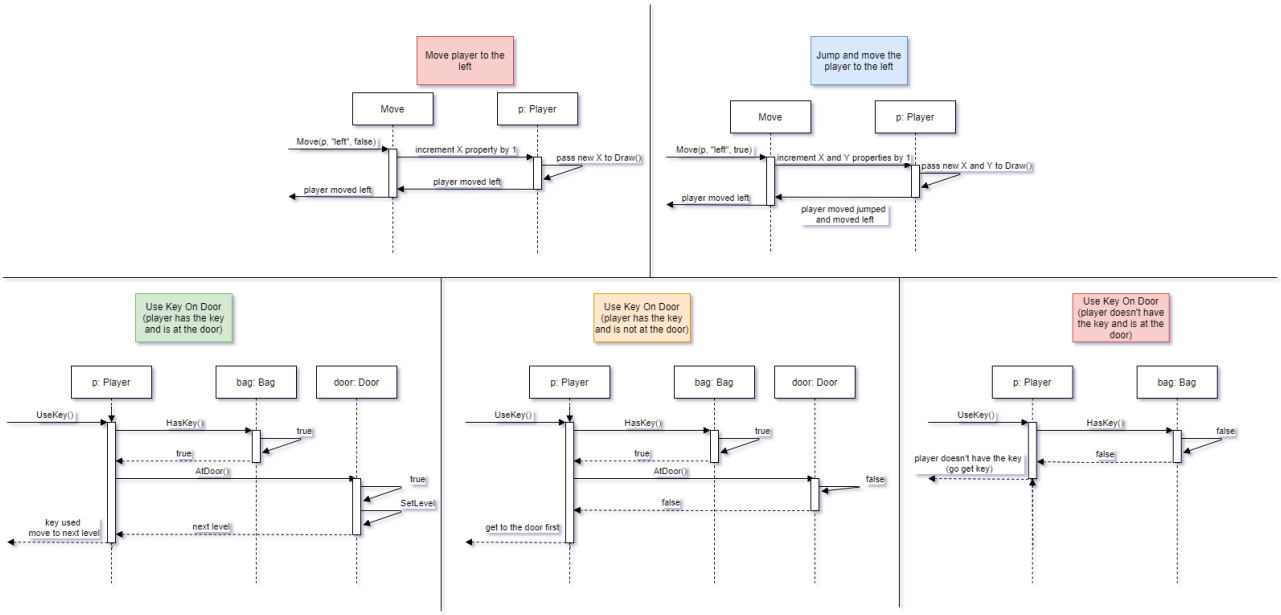
J+M+dir - Jump and use the suit and set the direction of the magnet



opera singing voice → Glass door - scream out it to break (disappear)

→ may add sound effects


Player
- _lastX, _lastY: double (coordinates of player location)
- _supersuit : Supersuit (contains the player's superpower)
- _level : Level (will determine which level the player is in)
- _nextLevel : Level (the next level)
- _bag: Bag(will hold the key once taken)
- _playerBitmap : Bitmap
+ Player(string name, double x, double y)
+ Supersuit: Supersuit <<property>>
+ Level: Level <<property>>
+ Bag: Bag<<readonly property>>
+ Fall() : void
+ TakeKey() : void
+ UseKey() : void
Move
- _x, _y: double
+ Move(string dir) //default constructor
+ Move(string dir, bool jump) //constructor with jump
Shrink
+ Shrink() //default constructor
+ Width : double <<override property>>
+ Length : double <<override property>>
Size <<abstract class>>
- _width, _length: double
+ Size(string change) //default constructor
+ Width : double <<virtual property>>
+ Length: double <<virtual property>>
Enlarge
+ Enlarge() //default constructor
+ Width : double <<override property>>
+ Length : double <<override property>>
Bag
- _key : Key
+ Bag()
+ HasKey(string id) : bool
+ PutKey(Key key) : void
+ EmptyBag() : void
Level
- _backgroundBitmap : Bitmap
- _bag : Bag
- _door : Door
+ Level()
+ LocateKey() : Key
+ AddDoor() : void
+ Draw() : override void
+ Load() : override void
Supersuit
- _supersuitBitmap : Bitmap
+ Supersuit()
GameObject <<Interface>>
- _width, _length : double
- _x, _y: double
+ Draw() : abstract void
+ Load() : virtual void
+ X: double <<property>>
+ Y: double <<property>>
+ Width : double <<property>>
+ Length : double <<property>>
Key
- _keyBitmap : Bitmap
+ Key(string[] ids, string name)
GlassBorder
- _glassBorderBitmap : Bitmap
+ GlassBorder()
+ GlassBorder(Bitmap image, double x, double y, double width, double length)
+ Width : double <<property>>
+ Draw() : override void
+ Load() : override void
MagneticBlock
- _magnetBlockBitmap : Bitmap
+ MagneticBlock()
+ MagneticBlock(Bitmap image, double x, double y, double width, double length)
+ Width : double <<property>>
+ Draw() : override void
+ Load() : override void
+ BorderAroundMagnet() : void
Block
- _normalBlockBitmap : Bitmap
- _spikedBlockBitmap : Bitmap
+ Block()
+ Block(Bitmap image, double x, double y, double width, double length, bool spike)
+ Width : double <<property>>
+ Draw() : override void
+ Load() : override void
Door
- _nextLevel : Level
+ Door(string[] ids, string name, Level nextLevel)
+ SetLevel() : Level
+ Draw() : override void
+ Load() : override void


Move player to the left
Move
p: Player
Move(p, "left", false)
increment X properly by 1
pass new X to Draw()
player moved left
player moved left
Jump and move the player to the left
Move
p: Player
Move(p, "left", true)
increment X and Y properties by 1
pass new X and Y to Draw()
player moved left
player moved jumped and moved left
Use Key On Door (player has the key and is at the door)
p: Player
bag: Bag
door: Door
UseKey()
HasKey()
true
true
AtDoor()
true
SetLevel
key used move to next level
next level
Use Key On Door (player has the key and is not at the door)
p: Player
bag: Bag
door: Door
UseKey()
HasKey()
true
true
AtDoor()
false
false
get to the door first
Use Key On Door (player doesn't have the key and is at the door)
p: Player
bag: Bag
UseKey()
HasKey()
false
false
player doesn't have the key (go get key)

18 Semester Test

Demonstrate you can pass a test on object oriented programming.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Semester Test

Submitted By:
Marella Morad
103076428
2021/09/28 22:29

Tutor:
Joshua Wright

September 28, 2021

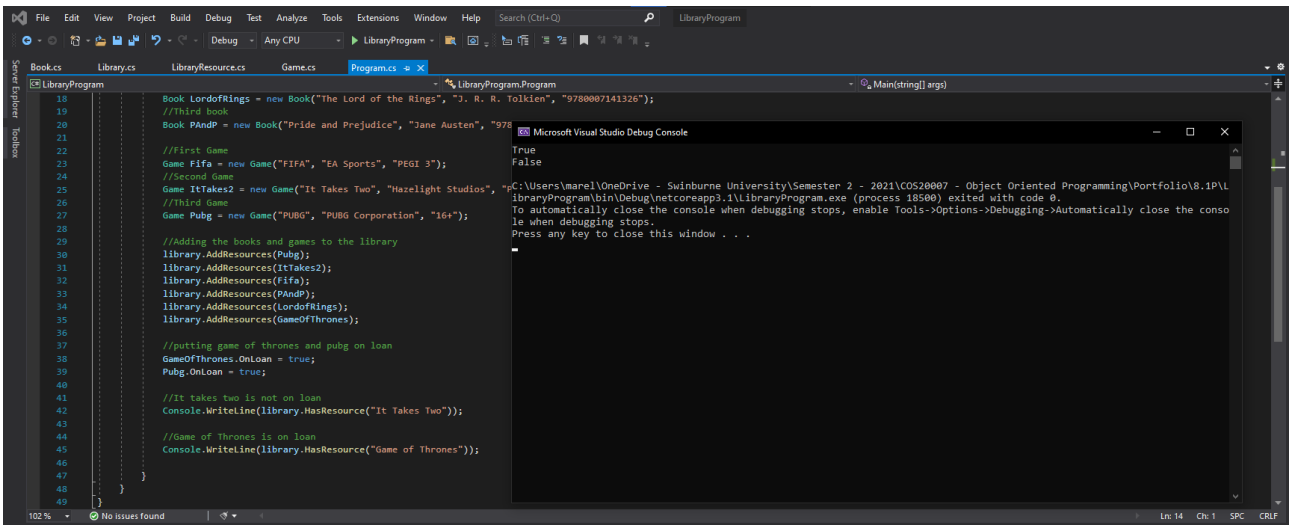
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibraryProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            Library library = new Library("Marella's Library");

            //First book
            Book GameOfThrones = new Book("Game of Thrones", "George R. R. Martin",
            ↪  "9780007237500");
            //Second book
            Book LordofRings = new Book("The Lord of the Rings", "J. R. R.
            ↪  Tolkien", "9780007141326");
            //Third book
            Book PAndP = new Book("Pride and Prejudice", "Jane Austen",
            ↪  "9783839893876");

            //First Game
            Game Fifa = new Game("FIFA", "EA Sports", "PEGI 3");
            //Second Game
            Game ItTakes2 = new Game("It Takes Two", "Hazelight Studios", "PEGI
            ↪  12");
            //Third Game
            Game Pubg = new Game("PUBG", "PUBG Corporation", "16+");

            //Adding the books and games to the library
            library.AddResources(Pubg);
            library.AddResources(ItTakes2);
            library.AddResources(Fifa);
            library.AddResources(PAndP);
            library.AddResources(LordofRings);
            library.AddResources(GameOfThrones);

            //putting game of thrones and pubg on loan
            GameOfThrones.OnLoan = true;
            Pubg.OnLoan = true;

            //It takes two is not on loan
            Console.WriteLine(library.HasResource("It Takes Two"));

            //Game of Thrones is on loan
            Console.WriteLine(library.HasResource("Game of Thrones"));

        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace LibraryProgram
{
    public class Library
    {
        //create a list of resources
        private List<LibraryResource> _resources;

        public Library(string name)
        {
            //instantiating the _resources list
            _resources = new List<LibraryResource>();
        }

        public void AddResources(LibraryResource resource)
        {
            _resources.Add(resource); //adding passed resources to the list
        }

        public bool HasResource(string name)
        {
            foreach(LibraryResource r in _resources) //looping through the
            ↪ _resources list
            {
                //HasResource returns true of the name matches one of the resources
                ↪ in the list and if the On Loan status is false (resource not on
                ↪ loan)
                if(r.Name == name && r.OnLoan == false) //cross checking with the
                ↪ passed name
                {
                    return true;
                }
            }

            return false;
        }
    }
}
```

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

namespace LibraryProgram
{
    public abstract class LibraryResource
    {
        //common fields in Game and Book are name and onloan
        private string _name;
        private bool _onLoan;

        public LibraryResource(string name, string creator)
        {
            _name = name;
            _onLoan = false;
        }

        //Name readonly property
        public string Name
        {
            get
            {
                return _name;
            }
        }

        //Creator abstract readonly property - to be overridden by children classes
        ↪ (Game & Book)
        public abstract string Creator
        {
            get;
        }

        //OnLoan property - can read and write
        public bool OnLoan
        {
            get
            {
                return _onLoan;
            }
            set
            {
                _onLoan = value;
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace LibraryProgram
{
    public class Game : LibraryResource
    {
        private string _contentRating;
        private string _developer;

        public Game(string name, string creator, string rating) : base(name,
        ↪  creator)
        {
            //initialise delveloper and content rating fields with given parameters
            _developer = creator;
            _contentRating = rating;
        }

        //ContentRating readonly property
        public string ContentRating
        {
            get
            {
               return _contentRating;
            }
        }

        //Creator override readonly property
        public override string Creator
        {
            get
            {
                return _developer;
            }
        }
    }
}
```

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

namespace LibraryProgram
{
    public class Book : LibraryResource
    {
        private string _isbn;
        private string _author;

        public Book(string name, string creator, string isbn) : base(name, creator)
        {
            //initialise the isbn and author fields with given parameters
            _isbn = isbn;
            _author = creator;
        }

        //ISBN readonly property
        public string ISBN
        {
            get
            {
                return _isbn;
            }
        }

        //Creator override readonly property
        public override string Creator
        {
            get
            {
                return _author;
            }
        }
    }
}
```


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
LibraryProgram
Debug
Any CPU
LibraryProgram
Book.cs
Library.cs
LibraryResource.cs
Game.cs
Program.cs
LibraryProgram
LibraryProgram.Program
Main(string[] args)
Book LordofRings = new Book("The Lord of the Rings", "J. R. R. Tolkien", "9780007141326");
//Third book
Book PAndP = new Book("Pride and Prejudice", "Jane Austen", "978
//First Game
Game Fifa = new Game("FIFA", "EA Sports", "PEGI 3");
//Second Game
Game ItTakes2 = new Game("It Takes Two", "Hazelight Studios", "p
//Third Game
Game Pubg = new Game("PUBG", "PUBG Corporation", "16+");
//Adding the books and games to the library
library.AddResources(Pubg);
library.AddResources(ItTakes2);
library.AddResources(Fifa);
library.AddResources(PAndP);
library.AddResources(LordofRings);
library.AddResources(GameOfThrones);
//putting game of thrones and pubg on loan
GameOfThrones.OnLoan = true;
Pubg.OnLoan = true;
//It takes two is not on loan
Console.WriteLine(library.HasResource("It Takes Two"));
//Game of Thrones is on loan
Console.WriteLine(library.HasResource("Game of Thrones"));
}
}
}
Microsoft Visual Studio Debug Console
True
False
C:\Users\marel\OneDrive - Swinburne University\Semester 2 - 2021\COS20007 - Object Oriented Programming\Portfolio\8.1P\L
ibraryProgram\bin\Debug\netcoreapp3.1\LibraryProgram.exe (process 18500) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
102 %
No issues found
Ln: 14 Ch: 1 SPC CRLF

Semester Test – Task 2

Written by: Marella Morad – Student ID: 103076428

For Task 1 of the semester test, I designed an abstract Library Resource class and redesigned the Game and Book classes from the original library design.

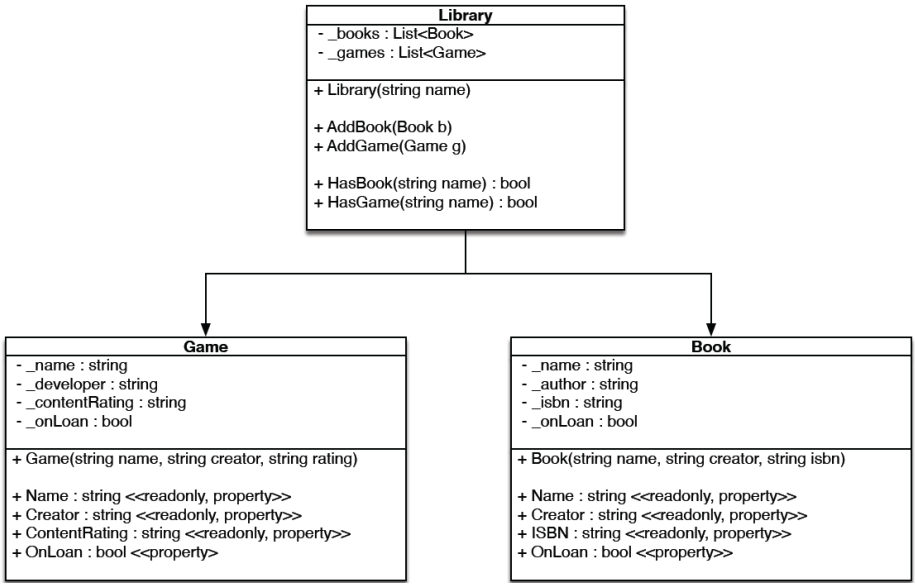


Figure 1 Original Design of the Library Program

In this report, I will talk about how I applied Abstraction and Polymorphism to achieve a better design.

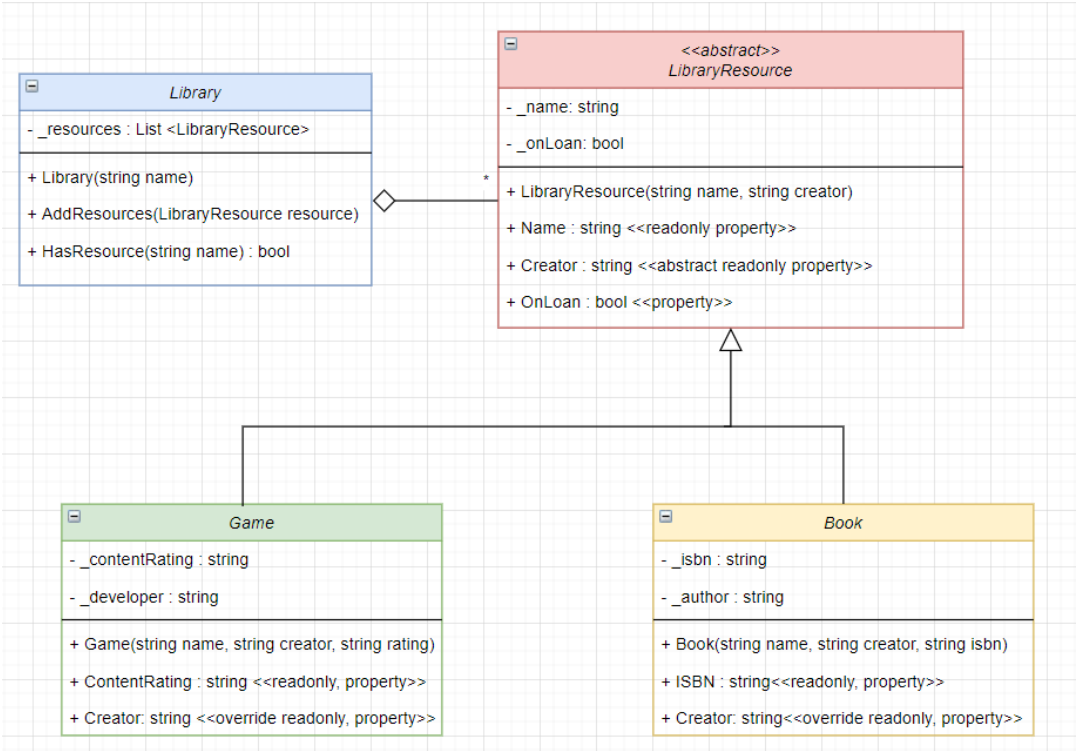


Figure 2 Redesign applying Abstraction and Polymorphism

Explanation

The concept of **Abstraction** is defined as the process of designing an object based on its Roles, Responsibilities, Classifications and Collaborations with other objects. For this task, I was required to design the Library Resource abstract class (see Figure 2). As outlined in the document, the `LibraryResource` class is an abstract class, which means it cannot instantiate an object. Meaning, `LibraryResource resource = new LibraryResource()` does not work (See Figure 3).

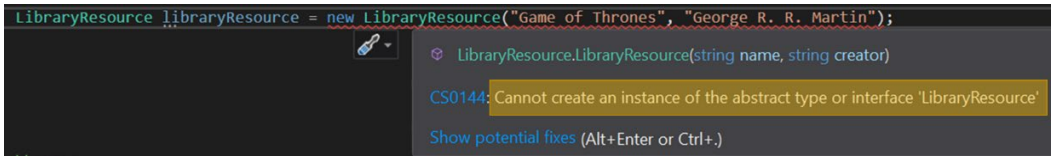


Figure 3 Evidence of how abstract classes cannot be instantiated

To design the `LibraryResource` class, I analysed the original design (Figure 1) to find similarities in the `Game` and `Book` classes which I will include as part of the `Library Resource` class.

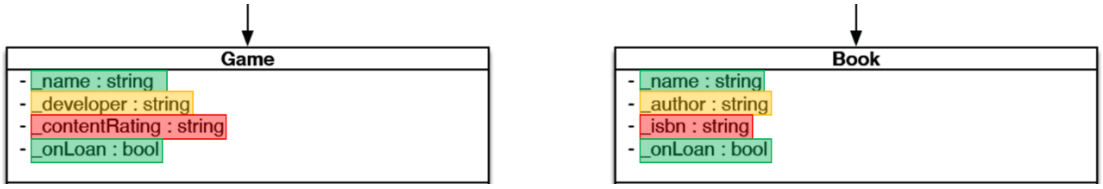


Figure 4 Comparing Game and Book classes {Same fields (Green), Similar fields (Yellow), Different Fields (Red)}

As shown in Figure 4, the `name` and `onLoan` fields are the exact same in both classes. While the `isbn` and `contentRating` are different fields. Leaving the author to be a shared but not the exact same field. Therefore, I decided to include the `name` and `onLoan` fields in the `LibraryResource` class and remove them from the `game` and `book` classes to reduce duplication of code and hence increase efficiency. I also added the `creator` property to the `LibraryResource` class, however, I made it an abstract property which will be overridden in the children classes (`Game` and `Book`). This concept of overriding elements comes from the inheritance relationship between these classes, where the `LibraryResource` class is the parent class, and the `Game` and `Book` classes are the children classes which inherit features from their parent class but also have their own specific features. Eventually the concept of polymorphism is applied by implementing multiple inheritance relationships between objects.

```
public class Game : LibraryResource
public class Book : LibraryResource
```

Figure 5 Inheritance in C#

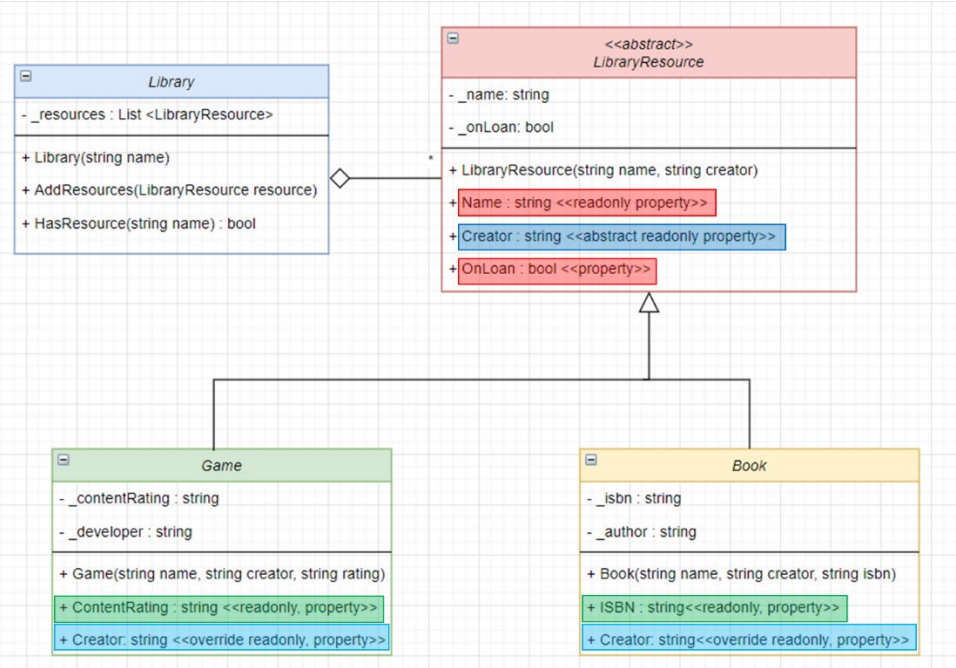
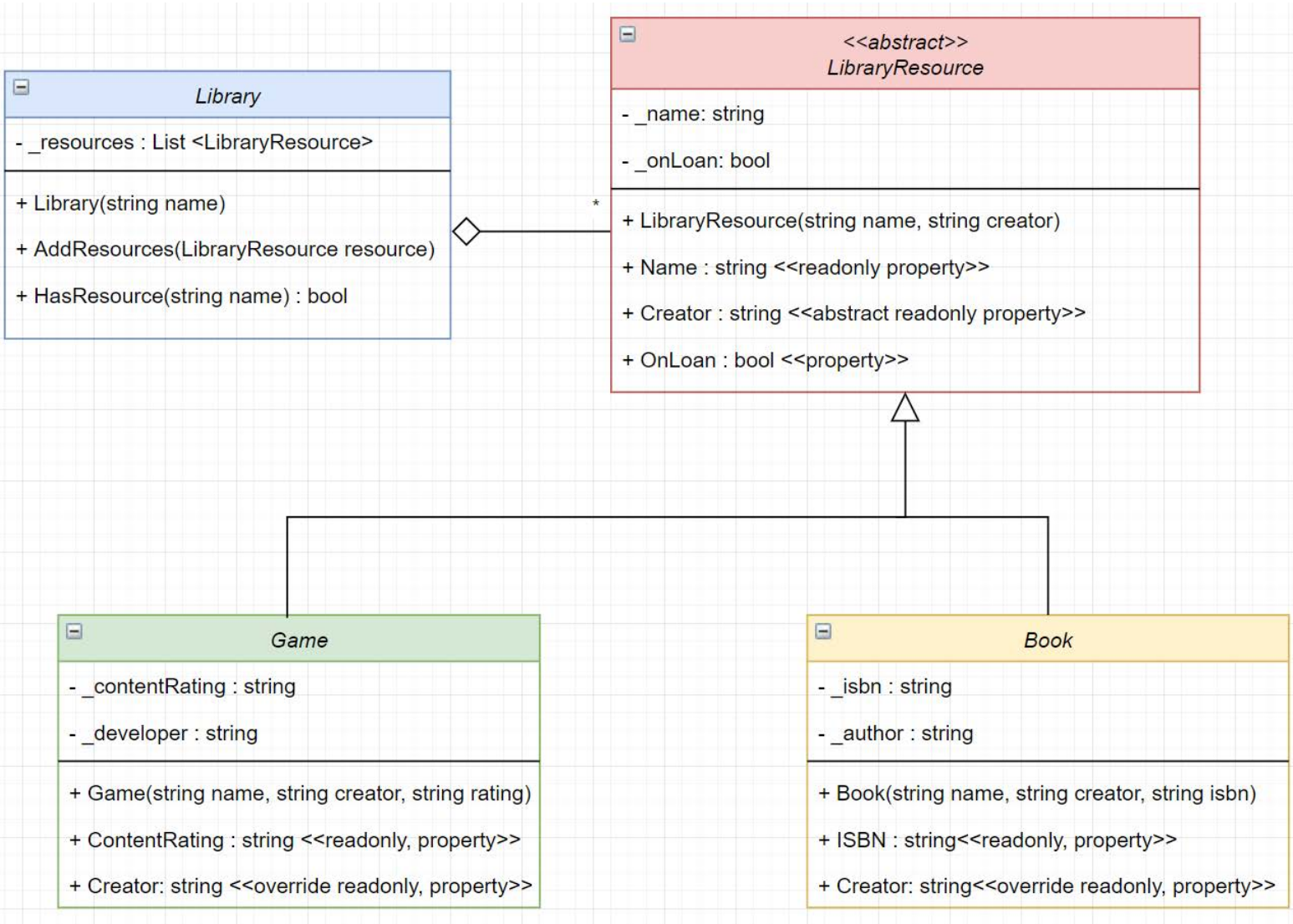


Figure 6 Inherited features (Name and OnLoan), Specific features (ContentRating and ISBN), and inherited but slightly changed features (Creator)

In addition, even though the abstract class cannot instantiate an object, it performs a powerful role in generalising other resources (`Game` and `Book`). The implementation of this abstract class initialises **Polymorphism** in our design. **Polymorphism** basically means many forms, and, in this case, it will generalise the `Game` and `Book` classes as being a `LibraryResource` so when we call one of the `AddResources` method from the `Library` class, we pass it either a Game or a Book object, and it will treat it as a `LibraryResource` object.

```
//Program.cs
Book GameOfThrones = new Book("Game of Thrones", "George R. R. Martin", "9780007237500");
Game Pubg = new Game("PUBG", "PUBG Corporation", "16+");

//Adding the books and games to the library
library.AddResources(Pubg);
library.AddResources(GameOfThrones);
//Library.cs
public void AddResources(LibraryResource resource)
{
    _resources.Add(resource); //adding passed resources to the list
}
```

Figure 7 Polymorphism example – AddResources() method from the Library class


Library
- _resources : List <LibraryResource>
+ Library(string name)
+ AddResources(LibraryResource resource)
+ HasResource(string name) : bool
*
<<abstract>>
LibraryResource
- _name: string
- _onLoan: bool
+ LibraryResource(string name, string creator)
+ Name : string <<readonly property>>
+ Creator : string <<abstract readonly property>>
+ OnLoan : bool <<property>>
Game
- _contentRating : string
- _developer : string
+ Game(string name, string creator, string rating)
+ ContentRating : string <<readonly, property>>
+ Creator: string <<override readonly, property>>
Book
- _isbn : string
- _author : string
+ Book(string name, string creator, string isbn)
+ ISBN : string<<readonly, property>>
+ Creator: string<<override readonly, property>>

19 Pass Task 7.1: Object Oriented Principles

You have been using object oriented programming to implement the programs in this unit. Express your understanding of the principles associated with this programming paradigm (abstraction, encapsulation, inheritance, and polymorphism).

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 7.1: Object Oriented Principles

Submitted By:
Marella Morad
103076428
2021/09/22 01:35

Tutor:
Joshua Wright

September 22, 2021

Object Oriented Principles

Written by: Marella Morad – Student ID: 103076428

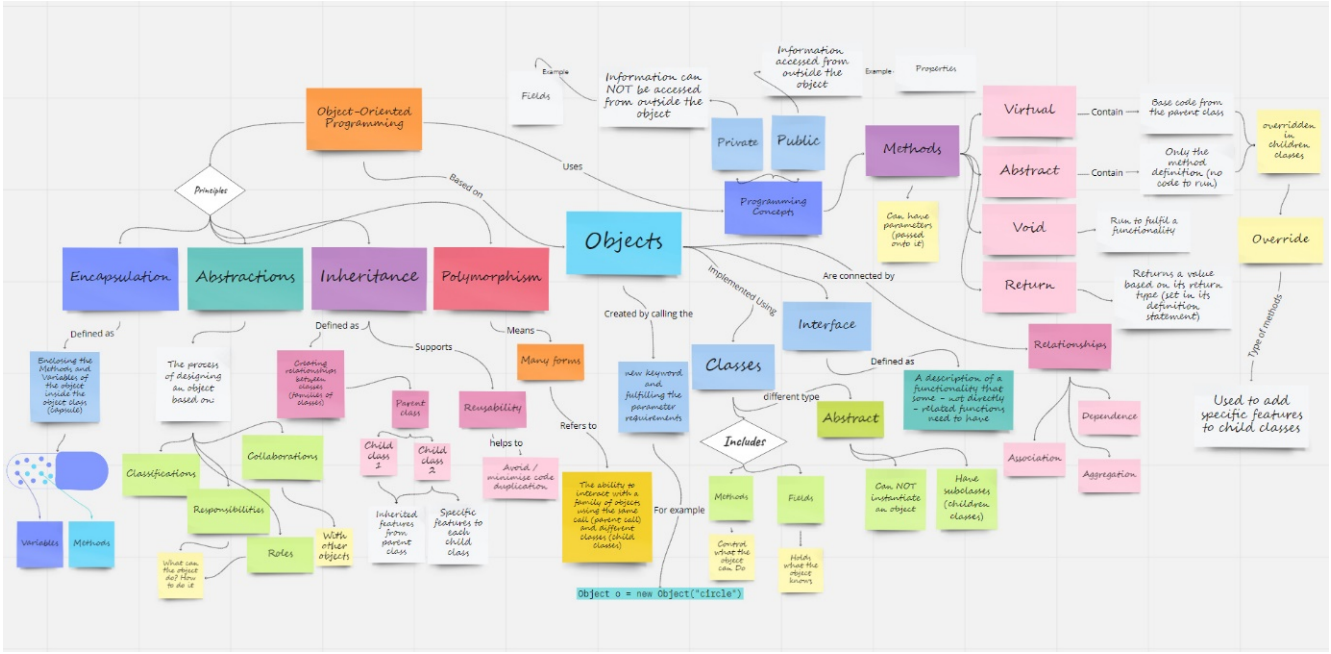


Table of Contents

1 Introduction:

Object Oriented Programming is a style of programming which organises a program/software into objects rather than separating the code into functions. The benefit of OOP becomes clearer with large and more complex designs. The use of objects is particularly useful to decrease the timeliness of a program and increase its efficiency (minimum code duplication)

2 OOP Principles:

Object Oriented Programming is based on four main principles.

2.1 Encapsulation

One of the key principles about OOP is **encapsulation**. This is basically the idea of enclosing the methods and fields of the object inside the object class – as a capsule – where the capsule cover represents the curly brackets at the start and end of the object (see Figure 1).

```
public class Message
{
        private string text;
        public Message(string txt)
        {
            text = txt;
        }
        public void Print()
        {
            Console.WriteLine(text);
        }
}
```

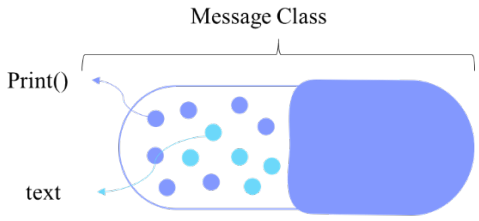


Figure 1 Encapsulation Principle explained with an example

2.2 Abstractions

Another key principle in OOP is **abstractions**. This is defined as the process of designing an object in relations to its Roles, Responsibilities, Classifications and Collaborations with other objects. Basically, defining what the object can do and how it can do it (with the implementation of methods and some relationships with other object which will be discussed later).

2.2.1 Cohesion

Cohesion is also an important concept in OOP. This concept is responsible for defining the objective of an object, or in other words, making sure that the object does not have a lot of different roles tied to it and is only responsible for the one single goal that it was made for. For example, in the shape drawer activity, although the Circle and Rectangle classes are both shapes, but we can argue that the goal of a Circle class is to draw a circle only not a rectangle, not a line.

2.3 Inheritance

The third main principle in OOP is **inheritance**. As the word suggests, a child inherits something from their parents. The same is applied in OOP. Inheritance is used to create families of classes. These classes will have a parent class and children's classes which inherit functionalities from their parent class. For example, from SwinAdventure, there are two types of commands that the game requires. See, two types of commands (command is the parent class here with functionalities that it passes on to its children's classes), see Figure 2 for code example.

```
namespace SwinAdventure
{
    public abstract class Command : IdentifiableObject
    {
        public Command(string[] ids) : base(ids)
        {

        }

        public abstract string Execute(Player p, string[] text);
    }
}
```

```
namespace SwinAdventure
{
    public class LookCommand : Command
    {
        public LookCommand() : base(new string[] { "look" })
        {

        }

        public override string Execute(Player p, string[] text)
        {
            if(text.Length !=1 && text.Length != 3 && text.Length != 5)
            {
                return "I don't know how to look like that";
            }
```

Figure 2 Inheritance LookCommand inherits Execute functionality from Command class

2.3.1 Reusability

The use of inheritance increases the reusability of the program as it groups objects into family which can call each other instead of having to run all the individual classes to get to a functionality (skips through the line).

2.4 Polymorphism

The final major principle of OOP is **polymorphism**. Polymorphism translates into "many forms", which refers to the ability to interact with a family of objects using the same call (parent call) and specifying the type (children class) in the call (See Figure 3).

```
//Polymorphism
//Creating a ResigsterShape method in the (parent class) Shape
//and registering the different types of shapes (child classes)
Shape.RegisterShape("Rectangle", typeof(MyRectangle));
Shape.RegisterShape("Circle", typeof(MyCircle));
Shape.RegisterShape("Line", typeof(MyLine));
```

Figure 3 Polymorphism of the Shape family

3 Objects

Now that we've overview OOP's principles, we can move to talking about OOP's backbone – Objects. At the end of the day OOP is a style of programming and so it obeys the basic programming principles.

3.1 Instantiation

Objects are instantiated only by calling the new keyword and fulfilling their constructor's parameter requirements as shown in Figure 4.

```
Item goggles = new Item(new string[] { "goggles" }, "IR goggles", "These are the
best night vision goggles");
```

Figure 4 Instantiating an Item by calling the new keyword with its constructor

3.2 Class

A class is how the object is constructed. It contains what the object knows and what it can do (fields and methods respectively). There are different types of classes, one of which is an Abstract class. Abstract classes are different as that they cannot instantiate an object, however, they have subclasses which can. If you look back at Figure 3, you can see that Shape class is an abstract class, so it only contains the information that it will pass on to its children classes.

3.3 Interface

Interface is only a description of a functionality that some – not directly related – classes need to have. For example the `IHaveInventory` interface from SwinAdventure, is implemented in both Player and Location classes to add the Locate() functionalities to both of them.

3.4 Relationships

After objects have been created, now is the time to define the relationships between them in order to have a fully functional program at the end. There are three types of relationships in OOP and these relationships are best described using a UML diagram (see Figure 5).

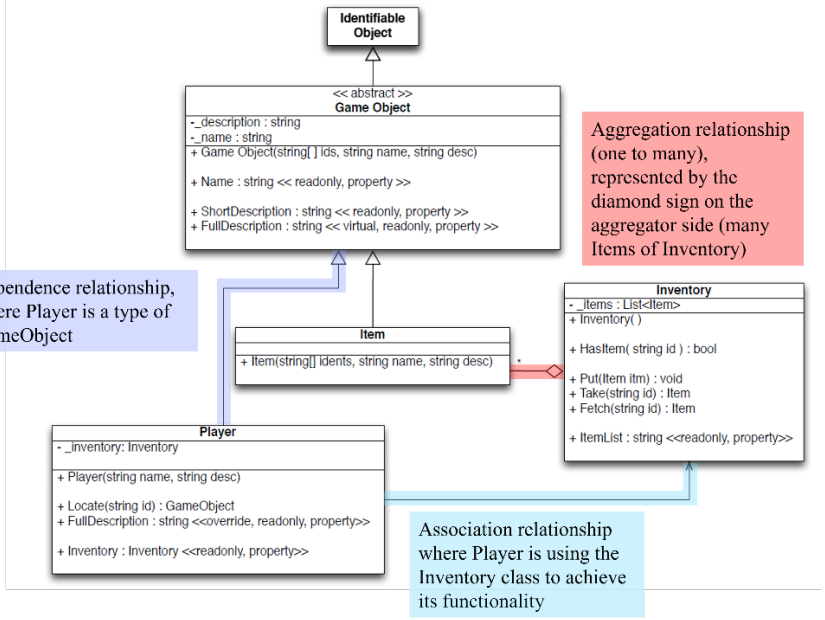


Figure 5 UML diagram with the three types of relationships

4 Programming Concepts / Terminology

4.1 Methods

There are a few types of methods used in OOP and all of them can have parameters passed onto them. A summary of the common types is in Table 1.

Table 1 Summary of Method Types used in OOP

Method Type	Definitions	Example
Void	Basic type of methods, does not return a value	`public void Draw()`
Return	Must have a return statement which returns a value based on the specified return type (in the method definition)	`static string GetKey(Type type)`
Abstract	Contains only the method definitions with no code to run	`public abstract string Execute(Player p, string[] text);`
Virtual	Contains the base code from the parent class	`public virtual void SaveTo(StreamWriter writer)`
Override	Used to override abstract and virtual methods in children's classes to add specific features.	`public override string Execute(Player p, string[] text)`

4.2 Scope

The last thing to talk about in this brief introduction to OOP is the scope of an element of a program.

Table 2 Scopes in Programming (used in OOP)

Type	**Scope**
Public	Can be accessed from anywhere in the program
Private	Can only be accessed from inside the object
Protected	Can be accessed from children classes which inherit it from the parent class
Static	Is shared through the entire class scope

Please refer to 7.2C Concept map which also outlines the relationships between the discussed concepts in a graphical manner.

5 References:

danijar 2012, “What is the difference between protected and private?,” *Stack Overflow*, viewed 21 September, 2021, <https://stackoverflow.com/questions/12784083/what-is-the-difference-between-protected-and-private>.

Francesco Lelli 2019, “Object Oriented Programming: A curated set of resources,” *Francesco Lelli*, viewed 21 September, 2021, <https://francescolelli.info/tutorial/object-oriented-programming-a-curated-set-of-resources/>.

Pavan Podila 2016, “10 Applications of Object Oriented Programming,” *Quickstart.com*, QuickStart, viewed 21 September, 2021, <https://www.quickstart.com/blog/10-applications-of-object-oriented-programming/>.

Gillis, AS & Lewis, S 2017, “object-oriented programming (OOP),” *SearchAppArchitecture*, TechTarget, viewed 21 September, 2021, <https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP>.

Object Oriented Programming Lectures 2021, “Lecture Slides and Materials: 2021-HS2-COS20007-Object Oriented Programming,” *Instructure.com*, viewed 21 September, 2021, <https://swinburne.instructure.com/courses/38543/pages/lecture-slides-and-materials?module_item_id=2306990>.

20 Credit Task 7.2: Object Oriented Programming Concept Map

A concept map visually shows the relationships between concepts. Think through the various relationships between the object oriented programming concepts and the associated artefacts, and to demonstrate your understanding.

Date	Author	Comment
2021/09/21 14:37	Marella Morad	Here is the link to the concept map online, for better clarity
2021/09/21 14:37	Marella Morad	https://miro.com/app/board/o9J_lvlxUoE=/

Swinburne University of Technology

Object Oriented Programming (2021 S2)

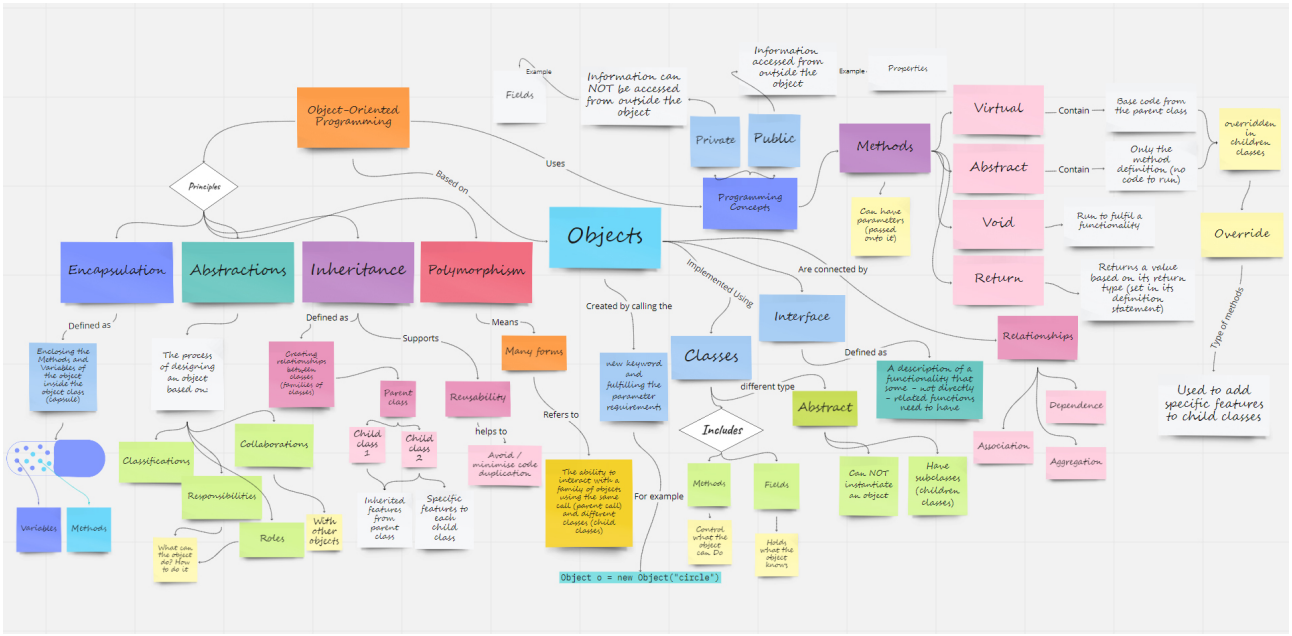
Doubtfire Submission

Credit Task 7.2: Object Oriented Programming Concept Map

Submitted By:
Marella Morad
103076428
2021/09/21 14:36

Tutor:
Joshua Wright

September 21, 2021


Object-Oriented Programming
Principles
Uses
Based on
Encapsulation
Abstractions
Inheritance
Polymorphism
Objects
Programming Concepts
Fields
Example
Information can NOT be accessed from outside the object
Private
Public
Information accessed from outside the object
Example
Properties
Methods
Can have parameters (passed into it)
Virtual
Abstract
Void
Return
Contain
Contain
Base code from the parent class
Only the method definition, (no code to run)
overridden in children classes
Run to fulfil a functionality
Returns a value based on its return type (set in its definition statement)
Override
Type of methods
Used to add specific features to child classes
Defined as
Enclosing the Methods and Variables of the object inside the object class (Capsule)
Variables
Methods
The process of designing an object based on
Classifications
Responsibilities
Roles
What can the object do? How to do it
Collaborations
With other objects
Defined as
Creating relationships between classes (families of classes)
Parent class
Child class 1
Child class 2
Inherited features from parent class
Specific features to each child class
Supports
Reusability
helps to
Avoid / minimise code duplication
Means
Many forms
Refers to
The ability to interface with a family of objects using the same call (parent call) and different classes (child classes)
For example
Object o = new Object("circle")
Created by calling the
new keyword and fulfilling the parameter requirements
Implemented Using
Classes
Includes
Methods
Control what the object can Do
Fields
Holds what the object knows
Are connected by
Interface
different type
Abstract
Can NOT instantiate an object
Have subclasses (children classes)
Defined as
A description of a functionality that some - not directly - related functions need to have
Relationships
Dependence
Association
Aggregation

21 Distinction Task 7.3: Case Study - Iterations 7 and 8

Object oriented programming makes best sense with larger programs. The case study will be your opportunity to create a larger program and better see how these abstractions make it easier to create software solutions.

Date	Author	Comment
2021/09/29 12:22	Joshua Wright	- Fix design issues with path as discussed
2021/10/02 23:44	Marella Morad	Video Demo link
2021/10/02 23:44	Marella Morad	https://youtu.be/iVSsknHgZE4
2021/10/06 12:07	Joshua Wright	Please refactor FullDescription for path as discussed.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Distinction Task 7.3: Case Study - Iterations 7 and 8

Submitted By:
Marella Morad
103076428
2021/10/25 14:47

Tutor:
Joshua Wright

October 25, 2021

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

namespace SwinAdventure
{
    public class Path : GameObject
    {
        private Location _destination;

        public Path(string[] ids, string name, string desc, Location destination) :
        ↪ base(ids, name, desc)
        {
            _destination = destination;
        }

        public Location SetDestination()
        {
            //returns the destentation which will be set to the player's location
            ↪ in the move command
            return _destination;
        }

        public override string FullDescription
        {
            get
            {
                return "You head " + this.Name +
                       "\nYou travel through a " + base.FullDescription +
                       "\nYou have arrived in the " + _destination.Name;
            }
        }
    }
}
```

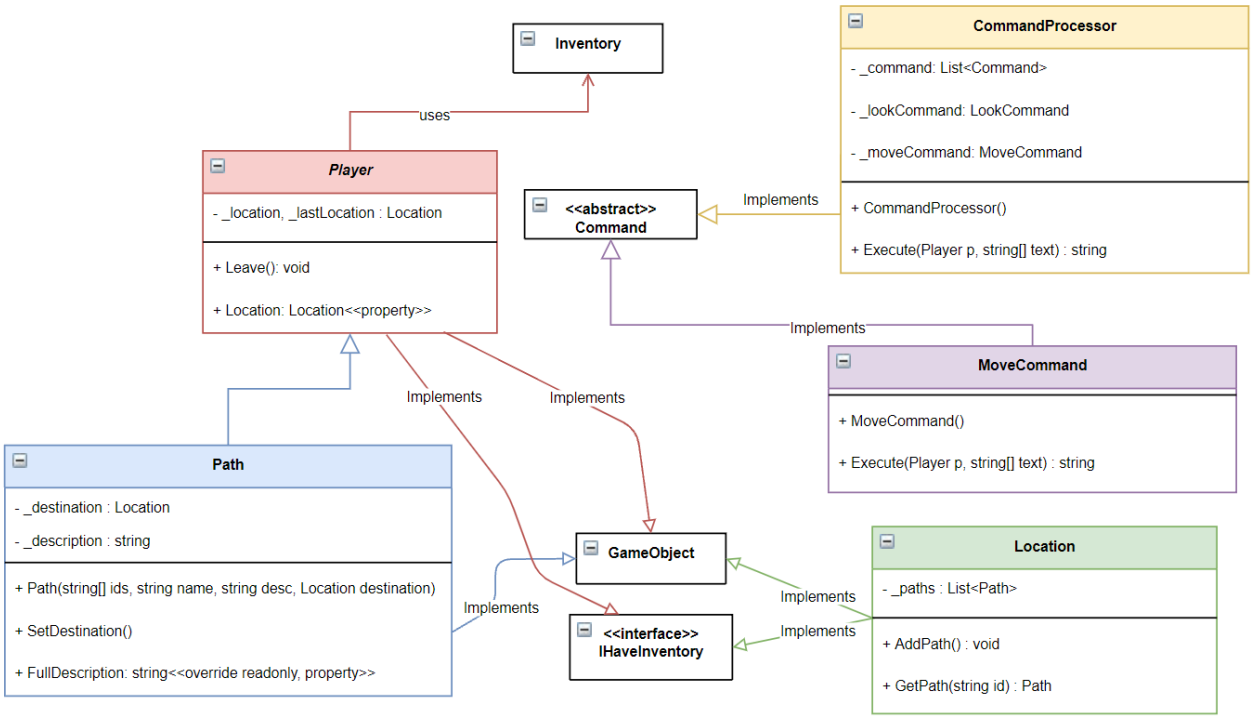
```
using NUnit.Framework;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SwinAdventure;

namespace SwinAdventureTests
{
    [TestFixture]
    public class PathUnitTests
    {
        Player player;
        Location hall, garden, lab;
        Path pathHtoL, pathHtoG, pathGtoH, pathGtoL, pathLtoH, pathLtoG;
        MoveCommand move;

        [SetUp]
        public void Setup()
        {
            player = new Player("Marella", "The amazing player");
            hall = new Location(new string[] { "hallway" }, "Hallway", "This is a long well lit Hallway");
            garden = new Location(new string[] { "garden" }, "Garden", "This is a big garden with a lot of secret spots");
            lab = new Location(new string[] { "lab" }, "Laboratory", "This is where the magic is created");

            pathHtoL = new Path(new string[] { "south", "s", "down" }, "South", "slide", lab);
            pathHtoG = new Path(new string[] { "east", "e" }, "East", "small door", garden);

            pathGtoH = new Path(new string[] { "west", "w" }, "West", "small door", hall);
            pathGtoL = new Path(new string[] { "sw", "south west" }, "South West", "roller coaster", lab);

            pathLtoH = new Path(new string[] { "n", "north", "up" }, "North", "ladder", hall);
            pathLtoG = new Path(new string[] { "ne", "north west" }, "North West", "roller coaster", garden);

            //setting the map
            /*
             * Hall -----> Garden
             * |
             * |
             * Lab
             */

            //Add the paths to each location
            hall.AddPath(pathHtoG);
```

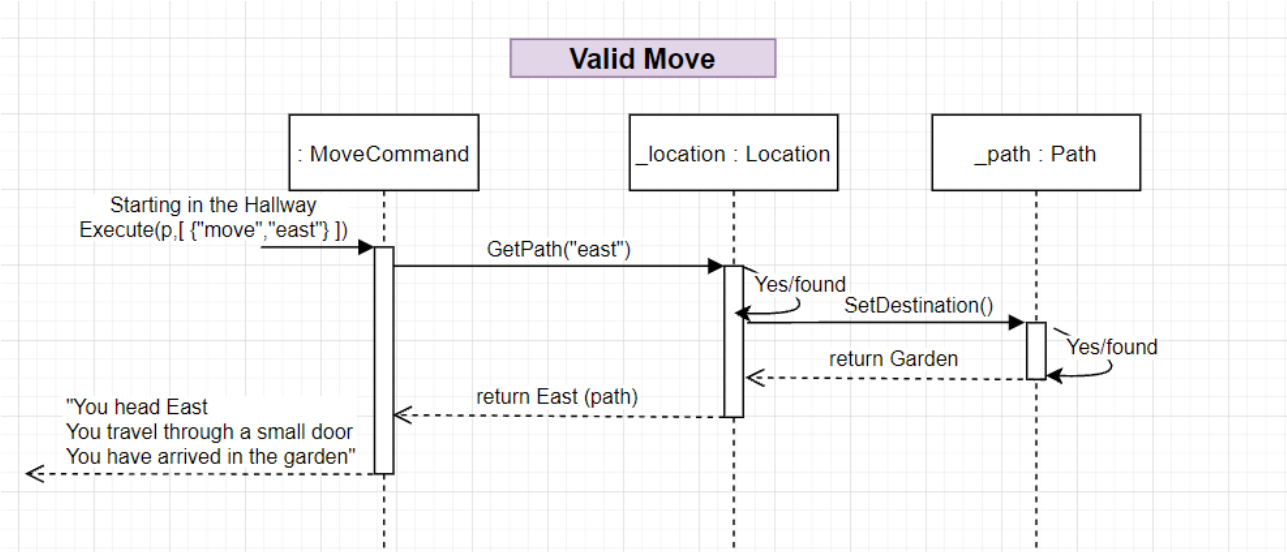
```
45              hall.AddPath(pathHtoL);
46
47              garden.AddPath(pathGtoL);
48              garden.AddPath(pathGtoH);
49
50              lab.AddPath(pathLtoG);
51              lab.AddPath(pathLtoH);
52
53
54              player.Location = hall; //default location of the player
55
56              move = new MoveCommand();
57          }
58
59          [Test]
60          public void TestMovePlayer()
61          {
62              string expected = "You head South\nYou travel through a slide\nYou have
                ↪  arrived in the Laboratory";
63              Assert.AreEqual(expected, move.Execute(player, new string[] { "move",
                ↪  "south" }), "Player cannot be moved south");
64          }
65
66          [Test]
67          public void TestGetPath()
68          {
69              Assert.AreEqual(pathGtoL, garden.GetPath("sw"));
70          }
71
72          [Test]
73          public void TestLeaveLocation()
74          {
75
76              move.Execute(player, new string[] { "move", "south" }); //player is now
                ↪  in the lab
77              Assert.AreEqual("You have headed back to the Hallway",
                ↪  move.Execute(player, new string[] { "leave" }));
78          }
79
80          [Test]
81          public void TestInvalidMove()
82          {
83              //invalid direction
84              Assert.AreEqual("This location has no path in the somewhere direction",
                ↪  move.Execute(player, new string[] { "move", "somewhere" }));
85              Assert.AreEqual(hall, player.Location, "Player left somewhere...");
86
87              //moving the player to different location and test again
88              move.Execute(player, new string[] { "go", "east" }); //go to garden
89              move.Execute(player, new string[] { "head", "south west" }); //go to lab
90
91              //wrong length of the move command
92              //One word command in only accepted for leave, not for move, go and head
```

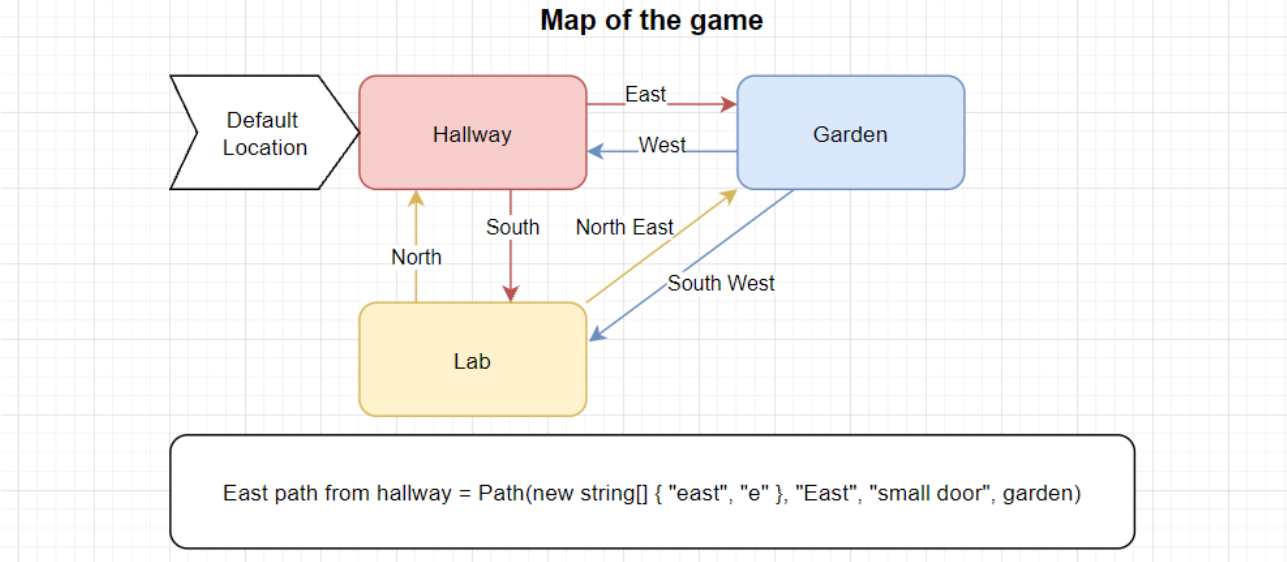
```
93              Assert.AreEqual("This location has no path in the south direction",
                ↪  move.Execute(player, new string[] { "move", "south" })); //cannot
                ↪  move south from the lab
94              Assert.AreEqual("Where do you want to go?", move.Execute(player, new
                ↪  string[] { "move" }));
95          }
96      }
97  }
```

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

namespace SwinAdventure
{
    public class Player : GameObject, IHaveInventory
    {
        private Location _location; //to allow player to be in a location
        private Inventory _inventory; //to manage the Player's items
        private Location _lastLocation; //to allow player to leave a location (go back to the previous one)

        //default constructor
        public Player(string name, string desc) : base(new string[] { "me", "inventory" }, name, desc)
        {
            _inventory = new Inventory();
            _location = new Location(new string[] { "hallway" }, "Hallway", "This is a long well lit Hallway");
        }

        public GameObject Locate(string id)
        {
            //three checks
            if (AreYou(id)) //if the player is what is to be located
            {
                return this; //retunring the player as a game object around the player
            }
            else if (_inventory.HasItem(id)) //if the player had what is being located
            {
                return _inventory.Fetch(id); //if the game object is not the player, then fetch it from inventory
            }
            else //if the item can be located where the player is
            {
                return _location.Locate(id);
            }
        }

        public override string FullDescription //overridden with the extra infromation
        {
            get
            {
                //returns the full description
                String fullDesc = "You are " + Name + " " + base.FullDescription + "\nYou are carrying:\n" + _inventory.ItemList;

                return fullDesc;
            }
```

```
46          }
47
48          public Inventory Inventory
49          {
50              get
51              {
52                  return _inventory;
53              }
54          }
55
56          public Location Location
57          {
58              get
59              {
60                  return _location;
61              }
62              set
63              {
64                  //setting the last location to the current location then changing
                    ↪  to the new location
65                  if(_location != null)
66                  {
67                      _lastLocation = _location;
68                  }
69                  else
70                  {
71                      _lastLocation = value;
72                  }
73
74                  _location = value;
75              }
76          }
77
78          public void Leave()
79          {
80              _location = _lastLocation; //returning to the last location
81          }
82      }
83  }
```


Inventory
uses
CommandProcessor
- _command: List<Command>
- _lookCommand: LookCommand
- _moveCommand: MoveCommand
+ CommandProcessor()
+ Execute(Player p, string[] text) : string
Player
- _location, _lastLocation : Location
+ Leave(): void
+ Location: Location<<property>>
<<abstract>>
Command
Implements
Implements
MoveCommand
+ MoveCommand()
+ Execute(Player p, string[] text) : string
Implements
Implements
Path
- _destination : Location
- _description : string
+ Path(string[] ids, string name, string desc, Location destination)
+ SetDestination()
+ FullDescription: string<<override readonly, property>>
GameObject
Implements
<<interface>>
IHaveInventory
Implements
Location
- _paths : List<Path>
+ AddPath() : void
+ GetPath(string id) : Path


Valid Move
: MoveCommand
_location : Location
_path : Path
Starting in the Hallway
Execute(p,[{"move","east"}])
GetPath("east")
Yes/found
SetDestination()
Yes/found
return Garden
return East (path)
"You head East
You travel through a small door
You have arrived in the garden"


Map of the game
Default Location
Hallway
East
West
Garden
South
North East
North
South West
Lab
East path from hallway = Path(new string[] { "east", "e" }, "East", "small door", garden)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SwinAdventure
{
    public class MoveCommand : Command
    {
        public MoveCommand() : base(new string[] { "move", "go", "head", "leave" })
        {

        }

        public override string Execute(Player p, string[] text)
        {
            //text can be "leave" or "move/head/go somewhere"
            if(text.Length != 2 && text.Length != 1)
            {
                return "I don't know how to move like that";
            }

            if(text[0] != "leave" && text.Length == 1)
            {
                return "Where do you want to go?";
            }

            if(text[0] == "leave")
            {
                p.Leave(); //goes back to the previous location
                string message;
                message = "You have headed back to the " + p.Location.Name;
                return message;
            }
            else //if text[0] = move, go or head - then move the player in the direction indicated
            {
                Location _location = p.Location; //getting the players location
                Path _path = _location.GetPath(text[1]); //getting the path of the location

                if(_path == null)
                {
                    return "This location has no path in the " + text[1] + " direction";
                }

                //if the location has paths:
                _location = _path.SetDestination(); //getting the locations of the path

                if (_location == null) //if the requested dirction does not exist
                {
```

```
50                      return "Location not found";
51                  }
52
53                  p.Location = _location;
54
55                  //if the path is found, return the new location
56                  return _path.FullDescription;
57
58              }
59          }
60      }
61  }
```

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

namespace SwinAdventure
{
    public class CommandProcessor : Command
    {
        private List<Command> _commands = new List<Command>();

        private LookCommand _lookCommand = new LookCommand();
        private MoveCommand _moveCommand = new MoveCommand();
        private TransferCommand _transferCommand = new TransferCommand();
        //private PutCommand _putCommand = new PutCommand();
        //private TakeCommand _takeCommand = new TakeCommand();
        private QuitCommand _quitCommand = new QuitCommand();

        public CommandProcessor() : base(new string[] {"command"}) //adding the
         ↪  _commands to the list
        {
            _commands.Add(_lookCommand);
            _commands.Add(_moveCommand);
            _commands.Add(_transferCommand);
            //_commands.Add(_putCommand);
            //_commands.Add(_takeCommand);
            _commands.Add(_quitCommand);
        }

        public override string Execute(Player p, string[] text)
        {
            foreach(Command c in _commands)
            {
                if(c.AreYou(text[0])) //identify the command
                {
                    return c.Execute(p, text); //execute the command based on the
                     ↪  text passed in
                }
            }

            string search = "";

            foreach(string txt in text)
            {
                search = search + txt + " ";
            }

            return "I don't understand " + search.TrimEnd(); //if the command is
             ↪  not found / error in input _commands (not move nor
        }
    }
}
```

22 Distinction Task 9.1: Case Study - Take, GUI and DLL

Demonstrate design skills in tackling the put/take commands, a mastery of C# libraries with a simple GUI (WinForms or WPF), and the ability to create class libraries.

Date	Author	Comment
2021/10/04 17:50	Marella Morad	I have used two froms for the GUI and since I cannot upload both to doubtfire I have screen recorded them to show you.
2021/10/04 17:53	Marella Morad	https://youtu.be/KU1v4pdeCOI
2021/10/04 17:53	Marella Morad	9.1D Demonstration link
2021/10/04 17:53	Marella Morad	https://youtu.be/KR-2_65lUH8
2021/10/06 12:13	Joshua Wright	Please refactor TransferCommand to remove code duplication as discussed.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Distinction Task 9.1: Case Study - Take, GUI and DLL

Submitted By:
Marella Morad
103076428
2021/10/06 18:11

Tutor:
Joshua Wright

October 6, 2021

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

namespace SwinAdventure
{
    public class TransferCommand : Command
    {
        public TransferCommand() : base(new string[] {"take", "pickup", "put",
        ↪  "drop"})
        {
        }

        public override string Execute(Player p, string[] text)
        {
            //accepted inputs are:
            //take item, take item from bag, pickup item
            // put item in bag, drop item
            if (text.Length != 2 && text.Length != 4)
            {
                return "I don't know how to transfer like that";
            }

            //if text.Length == 2 or 4, the code continues
            Item _item; //holds the item to be transfered
            if (text.Length == 2)
            {
                if(text[0].ToLower() == "take" || text[0].ToLower() == "pickup") //
                ↪  take / pickup item
                {
                    _item = Transfer(p.Location, p, text[1]);
                    if (_item != null)
                        return "You have taken the " + _item.Name + " from the " +
                        ↪  p.Location.Name;
                    return text[1] + " was not found in the " + p.Location.Name;
                    ↪  //if _item is null
                }


                if(text[0].ToLower() == "drop" || text[0].ToLower() == "put") //
                ↪  put / drop item
                {
                    _item = Transfer(p, p.Location, text[1]);
                    if (_item != null)
                        return "You have dropped the " + _item.Name + " in the " +
                        ↪  p.Location.Name;
                    return text[1] + " was not found in your inventory"; //if _item
                    ↪  is null
                }


            }

```

```
47              //if text.Length == 4
48              IHaveInventory container;
49              if (p.Locate(text[3]) != null)
50              {
51                  container = FetchContainer(p, text[3]) as IHaveInventory;
52                  if ((text[0].ToLower() == "take" || text[0].ToLower() == "pickup"))
                    ↪ //take/pickup item from container
53                  {
54                      if (text[2].ToLower() != "from")
55                          return "Where do you want to take the " + text[1] + "
                            ↪ from?";
56                      _item = Transfer(container, p, text[1]);
57                      if (_item != null)
58                          return "You have taken the " + _item.Name + " from the " +
                            ↪ container.Name;
59                      return text[1] + " was not found in the " + container.Name;
                        ↪ //if _item is null
60                  }
61                  else if ((text[0].ToLower() == "put" || text[0].ToLower() ==
                    ↪ "drop")) //put/drop item in container
62                  {
63                      if (text[2].ToLower() != "in")
64                          return "Where do you want to put the " + text[1] + "?";
65                      _item = Transfer(p, container, text[1]);
66                      if (_item != null)
67                          return "You have put the " + _item.Name + " in the " +
                            ↪ container.Name;
68                      return text[1] + " was not found in your inventory"; //if _item
                        ↪ is null
69                  }
70
71
72              }
73              return text[3] + " was not found";
74          }
75
76          public Item Transfer(IHaveInventory source, IHaveInventory destination,
            ↪ string itemID)
77          {
78              if (source.Inventory.HasItem(itemID))
79              {
80                  Item item = source.Inventory.Take(itemID); ;
81                  destination.Inventory.Put(item);
82                  return item;
83              }
84
85              return null;
86          }
87
88          public GameObject FetchContainer(Player p, string containerId)
89          {
90              return p.Locate(containerId);
91          }
```

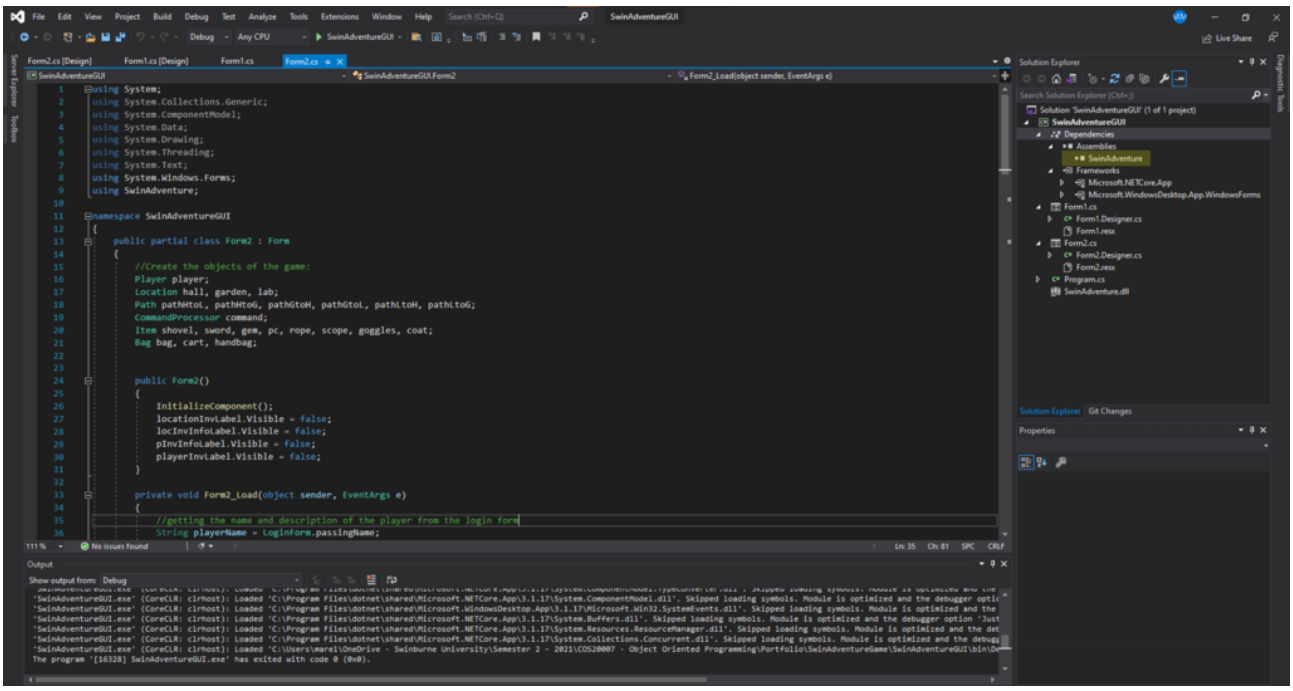
```
92      }
93  }
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Threading;
using System.Text;
using System.Windows.Forms;
using SwinAdventure;

namespace SwinAdventureGUI
{
    public partial class Form2 : Form
    {
        //Create the objects of the game:
        Player player;
        Location hall, garden, lab;
        Path pathHtoL, pathHtoG, pathGtoH, pathGtoL, pathLtoH, pathLtoG;
        CommandProcessor command;
        Item shovel, sword, gem, pc, rope, scope, goggles, coat;
        Bag bag, cart, handbag;


        public Form2()
        {
            InitializeComponent();
            locationInvLabel.Visible = false;
            locInvInfoLabel.Visible = false;
            pInvInfoLabel.Visible = false;
            playerInvLabel.Visible = false;
        }

        private void Form2_Load(object sender, EventArgs e)
        {
            //getting the name and description of the player from the login form
            String playerName = LoginForm.passingName;
            String playerDesc = LoginForm.passingDesc;
            player = new Player(playerName, playerDesc);

            SetupGame();

            player.Location = hall;

            //print welcome message
            outputLabel.Text = "Welcome to SwinAdventure, " + player.Name + "!\n" +
            ↪  "You have arrived in the " + player.Location.Name;
        }

        private void executeButton_Click(object sender, EventArgs e)
        {
            locationInvLabel.Visible = true;
            locInvInfoLabel.Visible = true;
            pInvInfoLabel.Visible = true;
```

```
53              playerInvLabel.Visible = true;
54
55              //strings to handle input from user (command)
56              string reply, userCommand;
57              string[] commandArr;
58
59              //Validate input
60              userCommand = commandTextBox.Text; //getting the command from the user
61              if (String.IsNullOrEmpty(userCommand))
62              {
63                  MessageBox.Show("Enter your command");
64              }
65              else
66              {
67                  commandArr = new[] { userCommand }; //converting into string[] array
68                  commandArr = userCommand.Split(" "); //splitting the command based
                    ↪ on spaces between words
69
70                  reply = command.Execute(player, commandArr); //executing command
                    ↪ using command processor
71                  outputLabel.Text = reply;
72
73                  //checking if the player wants to quit
74                  //quit command returns Good bye
75                  if (reply.ToLower() == "good bye")
76                  {
77                      MessageBox.Show(reply);
78                      this.Hide();
79                      Application.Exit();
80                  }
81              }
82
83              //displaying the player inventory and location invnetory and exits for
                ↪ refernce
84              pInvInfoLabel.Text = player.Name + "'s Inventory";
85              playerInvLabel.Text = player.Inventory.ItemList;
86              locInvInfoLabel.Text = player.Location.Name + "'s Description";
87              locationInvLabel.Text = player.Location.FullDescription;
88
89              commandTextBox.Text = String.Empty; //clear the textbox
90          }
91
92          void SetupGame()
93          {
94              //Add locations
95              hall = new Location(new string[] { "hallway" }, "Hallway", "This is a
                ↪ long well lit Hallway");
96
97              garden = new Location(new string[] { "garden" }, "Garden", "This is a
                ↪ big garden with a lot of secret spots");
98
99              lab = new Location(new string[] { "lab" }, "Laboratory", "This is where
                ↪ the magic is created");
```

```
100
101             //Setup the paths
102             pathHtoL = new Path(new string[] { "south", "s", "down" }, "South",
                ↪  "slide", lab);
103             pathHtoG = new Path(new string[] { "east", "e", "right" }, "East",
                ↪  "small door", garden);
104
105             pathGtoH = new Path(new string[] { "west", "w", "left" }, "West",
                ↪  "small door", hall);
106             pathGtoL = new Path(new string[] { "sw", "south_west" }, "South West",
                ↪  "roller coaster", lab);
107
108             pathLtoH = new Path(new string[] { "n", "north", "up" }, "North",
                ↪  "ladder", hall);
109             pathLtoG = new Path(new string[] { "ne", "north_east" }, "North East",
                ↪  "roller coaster", garden);
110
111             //Add the paths to each location
112             hall.AddPath(pathHtoG);
113             hall.AddPath(pathHtoL);
114
115             garden.AddPath(pathGtoL);
116             garden.AddPath(pathGtoH);
117
118             lab.AddPath(pathLtoG);
119             lab.AddPath(pathLtoH);
120
121             //Create the items
122             shovel = new Item(new string[] { "shovel" }, "shovel", "This is a might
                ↪  fine shovel");
123             sword = new Item(new string[] { "sword" }, "bronze sword", "This is a
                ↪  shiny sword");
124             gem = new Item(new string[] { "gem" }, "red gem", "This is a shiny red
                ↪  gem");
125             pc = new Item(new string[] { "pc" }, "small computer", "This is a
                ↪  computer from the future");
126             rope = new Item(new string[] { "rope" }, "rope", "This is the strongest
                ↪  rope ever");
127             scope = new Item(new string[] { "scope" }, "hand scope", "This is a
                ↪  golden scope with x1000 magnification");
128             goggles = new Item(new string[] { "goggles" }, "IR goggles", "These are
                ↪  the best night vision goggles");
129             coat = new Item(new string[] { "coat" }, "lab coat", "This is a white
                ↪  lab coat");
130
131             //Create the bags
132             bag = new Bag(new string[] { "bag" }, "plastic bag", "This is a clear
                ↪  plastic bag");
133             cart = new Bag(new string[] { "cart" }, "gardening cart", "This is a
                ↪  big gardening cart that fits large items");
134             handbag = new Bag(new string[] { "handbag" }, "handbag", "This is a
                ↪  leather handbag");
135
```

```
136             //Setup the game by adding items to locations and bags
137             hall.Inventory.Put(sword);
138             handbag.Inventory.Put(gem);
139             hall.Inventory.Put(handbag);
140 
141             garden.Inventory.Put(scope);
142             garden.Inventory.Put(rope);
143             cart.Inventory.Put(shovel);
144             garden.Inventory.Put(cart);
145 
146             lab.Inventory.Put(coat);
147             lab.Inventory.Put(pc);
148             bag.Inventory.Put(goggles);
149             lab.Inventory.Put(bag);
150 
151             command = new CommandProcessor();
152         }
153 
154         private void Form2_FormClosed(object sender, FormClosedEventArgs e)
155         {
156             Application.Exit();
157         }
158     }
159 }
```


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Threading;
using System.Text;
using System.Windows.Forms;
using SwinAdventure;
namespace SwinAdventureGUI
public partial class Form2 : Form
//Create the objects of the game:
Player player;
Location hall, garden, lab;
Path pathHtoL, pathHtoG, pathGtoH, pathGtoL, pathLtoH, pathLtoG;
CommandProcessor command;
Item shovel, sword, gem, pc, rope, scope, goggles, coat;
Bag bag, cart, handbag;
public Form2()
InitializeComponent();
locationInvLabel.Visible = false;
locInvInfoLabel.Visible = false;
pInvInfoLabel.Visible = false;
playerInvLabel.Visible = false;
private void Form2_Load(object sender, EventArgs e)
//getting the name and description of the player from the login form
String playerName = LoginForm.passingName;

23 Pass Task 10.1: Clock in Another Language

When learning a new language it is always best to create a small program that you are familiar with. In this task you will create the Clock class from the previous task in a new programming language.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Pass Task 10.1: Clock in Another Language

Submitted By:
Marella Morad
103076428
2021/09/20 23:14

Tutor:
Joshua Wright

September 20, 2021

```
class Counter
{
  private int _count;
  private String _name;

  public String getName()
  {
    return _name;
  }
  public void setName(String name)
  {
    _name = name;
  }

  public int getTicks()
  {
    return _count;
  }

  public void Increment()
  {
    _count = _count + 1;
  }

  public void Reset()
  {
    _count = 0;
  }
}

class Clock
{
  Counter _hours = new Counter();
  Counter _minutes = new Counter();
  Counter _seconds = new Counter();

  public void Tick()
  {
    _seconds.Increment();
    if(_seconds.getTicks() > 59)
    {
      _seconds.Reset();
      _minutes.Increment();
      if(_minutes.getTicks() > 59)
      {
        _minutes.Reset();
        _hours.Increment();
        if(_hours.getTicks() > 23)
        {
          _hours.Reset();
        }
      }
    }
```

```
54    }
55
56    public void Reset()
57    {
58      _hours.Reset();
59      _minutes.Reset();
60      _seconds.Reset();
61    }
62
63    public String getTime()
64    {
65      return _hours.getTicks() + ":" + _minutes.getTicks() + ":" +
        ↪  _seconds.getTicks();
66    }
67  }
68
69  public class Program{
70    public static void main(String[] args)
71    {
72      Clock clock = new Clock();
73      //Testing intialisation of the clock
74      System.out.println("Initialise Clock...");
75      System.out.println("Experceted Output: 0:0:0");
76      System.out.println("Actual output:     " + clock.getTime());
77      System.out.println();
78
79      //Testing seconds
80      for(int i = 0; i < 59; i++)
81      {
82        clock.Tick();
83      }
84
85      System.out.println("Seconds...");
86      System.out.println("Experceted Output: 0:0:59");
87      System.out.println("Actual output:     " + clock.getTime());
88      System.out.println();
89
90      //Testing seconds turning into minutes
91      clock.Tick();
92      System.out.println("One second later...");
93      System.out.println("Experceted Output: 0:1:0");
94      System.out.println("Actual output:     " + clock.getTime());
95      System.out.println();
96
97      //Testing minutes
98      clock.Reset();
99      for(int i = 0; i < 3599; i++)
100     {
101       clock.Tick();
102     }
103
104     System.out.println("Minutes...");
105     System.out.println("Experceted Output: 0:59:59");
```

```
106     System.out.println("Actual output:     " + clock.getTime());
107     System.out.println();
108 
109     //Testing minutes turning into hours
110     clock.Tick();
111     System.out.println("One second later...");
112     System.out.println("Experceted Output: 1:0:0");
113     System.out.println("Actual output:     " + clock.getTime());
114     System.out.println();
115 
116     //Testing hours
117     clock.Reset();
118     for(int i = 0; i < 86399; i++)
119     {
120       clock.Tick();
121     }
122 
123     System.out.println("Hours...");
124     System.out.println("Experceted Output: 23:59:59");
125     System.out.println("Actual output:     " + clock.getTime());
126     System.out.println();
127 
128     //Testing 24 hours
129     clock.Tick();
130     System.out.println("One second later...");
131     System.out.println("Experceted Output: 0:0:0");
132     System.out.println("Actual output:     " + clock.getTime());
133     System.out.println();
134 
135     //Random time then Reset
136     for(int i = 0; i < 85400; i++)
137     {
138       clock.Tick();
139     }
140 
141     System.out.println("Random Time...");
142     //Time calculations:
143     //85400 / 3600 = 23h
144     //85400 - 23*3600 = 2600s / 60 = 43m
145     //2600 - 43*60 = 20s
146     System.out.println("Experceted Output: 23:43:20");
147     System.out.println("Actual output:     " + clock.getTime());
148     System.out.println();
149 
150     clock.Reset();
151     System.out.println("Reset...");
152     System.out.println("Experceted Output: 0:0:0");
153     System.out.println("Actual output:     " + clock.getTime());
154     System.out.println();
155   }
156 }
```

```
$javac Program.java

$java -Xmx128M -Xms16M Program
Initialise Clock...
Experceted Output: 0:0:0
Actual output:     0:0:0

Seconds...
Experceted Output: 0:0:59
Actual output:     0:0:59

One second later...
Experceted Output: 0:1:0
Actual output:     0:1:0

Minutes...
Experceted Output: 0:59:59
Actual output:     0:59:59

One second later...
Experceted Output: 1:0:0
Actual output:     1:0:0

Hours...
Experceted Output: 23:59:59
Actual output:     23:59:59

One second later...
Experceted Output: 0:0:0
Actual output:     0:0:0

Random Time...
Experceted Output: 23:43:20
Actual output:     23:43:20

Reset...
Experceted Output: 0:0:0
Actual output:     0:0:0
```

24 Distinction Task 10.2: Other Language Report

So far we have looked at object oriented programming using the C# language. In this task you will compare the features of C# with that of another language to demonstrate similarities as well as differences in how languages are used.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

Distinction Task 10.2: Other Language Report

Submitted By:
Marella Morad
103076428
2021/10/15 02:01

Tutor:
Joshua Wright

October 15, 2021

C# vs Python

In Object Oriented Programming

Distinction Task 10.2

2021

Written by Marella Morad

Student ID: 103076428

Introduction

Programming languages are basically the way that we communicate with computers, giving them instructions to how to do a task and then asking them to complete it. Just like the languages we speak, there aren't two languages that are the exact same, there are always difference, some could be minor, and others major differences. This report will focus on two programming languages, C#, since it's the core language in this course, and a language that has some key differences with C#. This language is Python. There will be a comparison between the two languages in relation to main language features, but more specifically, into how the two languages apply object-oriented programming principles.

Main Language Features

C#

C-Sharp is a **statically typed** language, developed by Microsoft, Inc. It first appeared in the year 2000 and is now one of the most known object-oriented programming languages. This is mainly because when C# was first developed, it was based on OOP principles, which make applying these principles using C# considerably easy.

C# supports language **interoperability**, which means that it can access code written in any .NET compliant language and can also inherit classes written in such languages, since C# itself is a .NET language (C# Corner 2021). In addition, C# is a **structured language**, allowing the developer to break code down into functions, which improves the efficiency of the program given that code duplication can be basically replaced by a call of a function. It also features a **rich library**, providing the developer with many built-in functions, also increasing the speed of the program (JavaTPoint 2011).

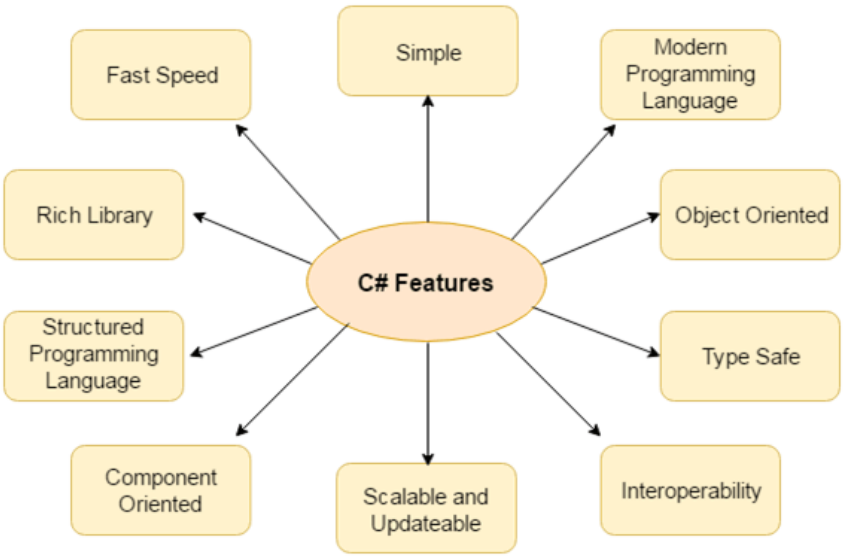


Figure 1 Top 10 C# language features (JavaTPoint 2011)

Python

Python is a dynamically typed, interpreted, general-purpose language, which supports Object-Oriented programming. It developed by Python Software Foundation in early 1991. However, in 2000, Python Consortium members signed an agreement for an Open-Source License, allowing developers to modify/improve on the language (Python.org 2020).

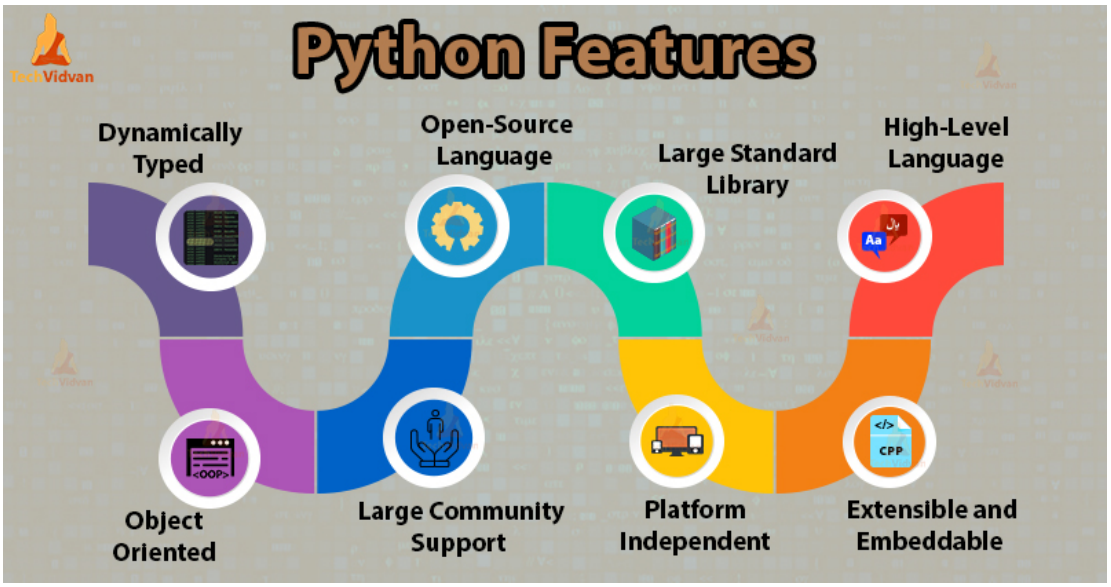


Figure 2 Python Language Features (TechVidvan 2019)

Python is an **Extensible and Embeddable** language, which means it can implement code from other languages, and can also be embedded into other language's code. A feature that sets Python aside is being an **open-source** language. Being an open source, free language, gives it the advantage of being constantly improved by the community and not only relying on improvements/bug fixes from the original developer, like Microsoft in C#'s case. Moreover, Python is a **platform independent** language, meaning that the same program will run on multiple platforms such as Windows, Linux, and Mac OS. It features a **large standard library** providing developers with many build-in functions.

Key Differences

After the brief introduction to each of the two languages, now we will discuss some key differences that are related to OOP implementation in both languages.

Static vs Dynamic Languages

The first main difference between Python and C# is that Python is a dynamically typed language, while C# is statically typed. The main differences arise as Statically typed languages are better with error handling, since the code is first compiled, and error checked before it runs. C# specifically has a built-in error detector that points out the specific lines that caused an error and will not compile in case there are errors in the code. Compiler errors include either not specifying the data type of a variable, or incorrectly specifying the data type (for example, declaring an int variable and assigning it to a string). On the other hand, dynamically typed languages, can run multiple functions simultaneously, resulting in a more difficult approach to locating and fixing errors, but a simpler way of writing the actual code (Friedman 2019). As mentioned earlier, Python is an interpreted language, and not a compiled language, which is why detecting errors before runtime does not happen in Python. Also, variable types are determined during runtime, no need for type declaration in Python (Simon Bonello 2021).

Syntax

The first obvious syntax difference between the two languages is that C# implements namespaces, classes, and functions to run the code. Any C# program has a Main() function which is where the order of the code run is defined, only functions/methods called in Main will run when the program is executed. Looking at a simple program in both languages in Table 1, we can see that python is much more efficient in terms of lines of code.

Table 1 A simple program in C# vs Python

C#	Python

`using System;`
`class Program`
`{`
`  public static void Main (string[] args)`
`  {`
`  int age = 21;`
`  string name = "Marella";`
`  Console.WriteLine(name + " is " + age + " years old");`
`  }`
`}`
	`age = 21`
	`name = "Marella"`
	`print(name + " is " + str(age) + " years old")`

Another main difference is that in Python, indentations are a must, while in C# they are only used for clarity, and are replaced by curly brackets. Additionally, C# and Python are very different in terms of syntax. As mentioned earlier, in Python, there is no need to declare a variable before using it, however, in C#, you must declare the variable correctly first, only then you'll be able to use it (see Table 1).

OOP Differences

The syntax differences discussed in the previous section will have a huge impact on the implementation of OOP principle in the two languages. In this report, I will only cover encapsulation differences.

Encapsulation

One of the main OOP principles is **encapsulation**, which means that the user can only access and change the fields that are relevant to their action (Schults 2018). For example, setting the name of a character in a game is only done once, and the user does not need to change it. Therefore, the name property of the character class should be a read-only (getter) property, that will only return the character's name. In C#, encapsulation is done very easily through **properties** that can be read-only, write-only, or even read and write properties (see Table 2). These properties set rules for user's interactions with the class's private fields. On the other hand, Python has a different implementation of properties, which function more like a method than a property compared to C# syntax. As shown in Table 2, the user can access the private field __name, by calling the name method. Unlike C#, we cannot have a getter and setter (read and write) property in Python, they need to be two different methods (see `age_getter(self)` and `age_setter(self, age)` in Table 2).

Sample Code Illustrating The Difference

Table 2 Encapsulation in C# vs Python

C#	Python

`using System;`
 	
`class Program`
`{`
`  public static void Main (string[] args)`
`  {`
`  Dog buddy = new Dog("Buddy", 9);`
`  Console.WriteLine(buddy.Name);`
`  Console.WriteLine(buddy.Age);`
`  buddy.Age = 15;`
`  Console.WriteLine(buddy.Age);`
`  }`
`}`
 	
`class Dog`
	`class Dog:`
	`  def __init__(self, name, age):`
	`  self.__name = name`
	`  self.__age = age`
	
	`  def name(self):`
	`  return self.__name`
	
	`  def age_getter(self):`
	`  return self.__age`
	
	`  def age_setter(self, age):`
	`  self.__age = age`
	
	`buddy = Dog("Buddy", 9)`
	`print(buddy.name())`

```
{
  private string _name;
  private int _age;

  public Dog(string name, int age)
  {
    _name = name;
    _age = age;
  }

  public string Name
  {
    get
    {
      return _name;
    }
  }

  public int Age
  {
    get
    {
      return _age;
    }
    set
    {
      _age = value;
    }
  }
}
```

```
print(buddy.age_getter())
buddy.age_setter(15)
print(buddy.age_getter())
```

Reference List

C# Corner 2021, “Top 10 Most Important Features Of C#,” *C-sharpcorner.com*, viewed 14 October, 2021, <https://www.c-sharpcorner.com/article/top-10-most-important-features-of-C-Sharp-programming/>.

Friedman, J 2019, “C# vs Python: Two Object-Oriented Languages With Key Differences,” *C# Station*, viewed 14 October, 2021, <https://csharp-station.com/c-vs-python/>.

JavaTPoint 2011, “C# Features - javatpoint,” *www.javatpoint.com*, viewed 14 October, 2021, <https://www.javatpoint.com/csharp-features>.

Python Tutorial 2020, “Understanding Python Encapsulation Clearly By Practical Examples,” *Python Tutorial - Master Python Programming For Beginners from Scratch*, viewed 14 October, 2021, <https://www.pythontutorial.net/python-oop/python-private-attributes/>.

Python.org 2020, “Welcome to Python.org,” *Python.org*, Python.org, viewed 14 October, 2021, <https://www.python.org/search/?q=open&page=1>.

Schults, C 2018, “C# Features: An Exhaustive List of the Best Ones - NDepend,” *NDepend*, viewed 14 October, 2021, <https://blog.ndepend.com/c-features-best-ones/>.

Simon Bonello 2021, “C# vs Python. Which language is best for you? - Chubby Developer,” *Chubby Developer*, viewed 14 October, 2021, <https://www.chubbydeveloper.com/csharp-vs-python/>.

TechVidvan 2019, “Features of Python - Explore the essence of Python - TechVidvan,” *TechVidvan*, viewed 14 October, 2021, <https://techvidvan.com/tutorials/features-of-python/>.

25 High Distinction Task 6.4: HD Custom Program

To be eligible for a High Distinction grade you must demonstrate that you can use the skills you have learnt to create high quality software solutions that demonstrate the highest programming and design standards. This is a place to upload your HD level custom program once it is completed. Note: if you are doing a HD level custom program you skip the D custom program upload.

Swinburne University of Technology

Object Oriented Programming (2021 S2)

Doubtfire Submission

High Distinction Task 6.4: HD Custom Program

Submitted By:
Marella Morad
103076428
2021/11/07 00:01

Tutor:
Joshua Wright

November 7, 2021

```
///Blocks Family Classes
//Block
using SplashKitSDK;

namespace MarioGame
{
    /// <summary>
    /// Abstract class, specific block types inherit from it
    /// </summary>
    public abstract class Block : GameObject, IDrawable
    {
        private Bitmap _blockBitmap;
        private string _type;
        private CollisionProcessor _collision;

        /// <summary>
        /// Block defaul constructor
        /// </summary>
        /// <param name="name"></param>
        /// <param name="x"></param>
        /// <param name="y"></param>
        public Block(string name, double x, double y)  : base(name, x, y)
        {
            _collision = new CollisionProcessor();
        }

        /// <summary>
        /// Bitmap property
        /// </summary>
        public Bitmap Bitmap
        {
            get
            {
                return _blockBitmap;
            }
            set
            {
                _blockBitmap = value;
            }
        }

        /// <summary>
        /// to distinguise between the different types of blocks, set in child
        ↪  classes
        /// </summary>
        public string Type
        {
            get
            {
                return _type;
            }
            set
            {
```

```
                _type = value;
            }
        }

        /// <summary>
        /// Runs the collision processor to check for collisions between the player and a block
        /// </summary>
        /// <param name="p"></param>
        public void PlayerBlockCollision(Player p)
        {
            _collision.Check(p, this);
        }

        /// <summary>
        /// Implementation of the IDrawable interface Draw method to draw the block on the screen
        /// </summary>
        public void Draw()
        {
            SplashKit.DrawBitmap(Bitmap, X, Y);
        }
    }
}

//GlassBlock
using SplashKitSDK;

namespace MarioGame
{
    public class GlassBlock : Block
    {
        /// <summary>
        /// GlassBlock Default constructor
        /// </summary>
        /// <param name="name"></param>
        /// <param name="x"></param>
        /// <param name="y"></param>
        public GlassBlock(string name, double x, double y) : base(name, x, y)
        {
            Type = "glass"; //setting the type of the block
            Bitmap = SplashKit.LoadBitmap(name, "glassBlock.png");
            X = x;
            Y = y;
        }
    }
}

//LavaBlock
using SplashKitSDK;

namespace MarioGame
{
```

```
104     public class LavaBlock : Block
105     {
106         /// <summary>
107         /// LavaBlock Default constructor
108         /// </summary>
109         /// <param name="name"></param>
110         /// <param name="x"></param>
111         /// <param name="y"></param>
112         public LavaBlock(string name, double x, double y) : base(name, x, y)
113         {
114             Type = "lava";
115             Bitmap = SplashKit.LoadBitmap(name, "lavaBlock.png");
116             X = x;
117             Y = y;
118         }
119     }
120 }
121
122 //MagneticBlock
123 using SplashKitSDK;
124
125 namespace MarioGame
126 {
127     public class MagneticBlock : Block
128     {
129         /// <summary>
130         /// MagneticBlock Default constructor
131         /// </summary>
132         /// <param name="name"></param>
133         /// <param name="x"></param>
134         /// <param name="y"></param>
135         public MagneticBlock(string name, double x, double y) : base(name, x, y)
136         {
137             Type = "magnet";
138             Bitmap = SplashKit.LoadBitmap(name, "magneticBlock.png");
139             X = x;
140             Y = y;
141         }
142     }
143 }
144
145 //NormalBlock
146 using SplashKitSDK;
147
148 namespace MarioGame
149 {
150     public class NormalBlock : Block
151     {
152         /// <summary>
153         /// Normal Blcok Default Constructor
154         /// </summary>
155         /// <param name="name"></param>
156         /// <param name="x"></param>
```

```
157         /// <param name="y"></param>
158         public NormalBlock(string name, double x, double y) : base(name, x, y)
159         {
160             Type = "normal";
161             Bitmap = SplashKit.LoadBitmap(name, "normalBlock.png");
162             X = x;
163             Y = y;
164         }
165     }
166 }
167 
168 //SpikedBlock
169 using SplashKitSDK;
170 
171 namespace MarioGame
172 {
173     public class SpikedBlock : Block
174     {
175         /// <summary>
176         /// SpikedBlock Default Constructor
177         /// </summary>
178         /// <param name="name"></param>
179         /// <param name="x"></param>
180         /// <param name="y"></param>
181         public SpikedBlock(string name, double x, double y) : base(name, x, y)
182         {
183             Type = "spiked";
184             Bitmap = SplashKit.LoadBitmap(name, "spikedBlock.png");
185             X = x;
186             Y = y;
187         }
188     }
189 }
190 
191 ///Collisions Family Classes
192 //CollisionProcessor
193 using System.Collections.Generic;
194 
195 namespace MarioGame
196 {
197     public class CollisionProcessor : ICollision
198     {
199         private List<ICollision> _collisions = new List<ICollision>(); //to
            ↪ register the different collisions
200 
201         private VerticalCollision _verticalCollision = new VerticalCollision();
202         private HorizontalCollision _horizontalCollision = new
            ↪ HorizontalCollision();
203 
204         /// <summary>
205         /// Default CollisionProcessor constructor to register the different
            ↪ collisions
206         /// </summary>
```

```
207         public CollisionProcessor()
208         {
209             _collisions.Add(_verticalCollision);
210             _collisions.Add(_horizontalCollision);
211         }
212 
213         /// <summary>
214         /// Runs the collisions check taking in the player and the block as
            ↪ parameters
215         /// Implements the Check method from the ICollision interface
216         /// </summary>
217         /// <param name="p"></param>
218         /// <param name="block"></param>
219         public void Check(Player p, Block block)
220         {
221             foreach (ICollision c in _collisions)
222             {
223                 c.Check(p, block);
224             }
225         }
226     }
227 }
228 
229 //HorizontalCollision
230 using SplashKitSDK;
231 
232 namespace MarioGame
233 {
234     public class HorizontalCollision : ICollision
235     {
236         public HorizontalCollision()
237         {
238 
239         }
240 
241         /// <summary>
242         /// Check for horizontal collisions between the player and a block using
            ↪ bounding rectangles
243         /// Applying different treatments to different block types
244         /// </summary>
245         /// <param name="p"></param>
246         /// <param name="block"></param>
247         public void Check(Player p, Block block)
248         {
249             Rectangle playerRec = p.Bitmap.BoundingRectangle(p.X, p.Y); //getting
                ↪ the bounding rectangle of the player
250             Rectangle blockRec = block.Bitmap.BoundingRectangle(block.X, block.Y);
                ↪ //get the bounding rectangle of the block
251             Rectangle intersection = SplashKit.Intersection(playerRec, blockRec);
                ↪ //get the intersection rectangle and a save it as a rectangle
252             if (intersection.Height > intersection.Width) //horizontal collision
253             {
254                 //glass blocks break when the player collides with them horizontally
```

```
255                 if (block.Type == "glass" && p.Sing) //Sing is true when the player
                    ↪  uses their superpower (level 3)
256                 {
257                     block.Bitmap = SplashKit.LoadBitmap("brokenGlass",
                        ↪  "brokenGlassBlock.png"); //change the block bitmap to
                        ↪  broken glass
258                     block.Type = "broken"; //change the type to broken
259                     block.Y += 160; //lower the block to sit on the ground/platform
260                     p.Sing = false; //turn sing to false
261                 }
262                 //Checking player intersection with the block from the LHS
263                 if (SplashKit.RectangleRight(playerRec) >
                    ↪  SplashKit.RectangleLeft(blockRec) &&
                    ↪  SplashKit.RectangleRight(playerRec) <
                    ↪  SplashKit.RectangleRight(blockRec))
264                 {
265                     p.X -= intersection.Width; //subtracting the intersection width
                        ↪  from the player's X coordinate
266                 }
267                 //Checking player intersection with the block from the RHS
268                 else
269                 {
270                     p.X += intersection.Width; //adding the intersection width to
                        ↪  the player's X coordinate
271                 }
272             }
273         }
274     }
275 }
276
277 //VerticalCollision
278 using SplashKitSDK;
279
280 namespace MarioGame
281 {
282     public class VerticalCollision : ICollision
283     {
284         public VerticalCollision()
285         {
286         }
287
288         /// <summary>
289         /// Check for vertical collisions between the player and a block using
            ↪  bounding rectangles
290         /// Applying different treatments to different block types
291         /// </summary>
292         /// <param name="p"></param>
293         /// <param name="block"></param>
294         public void Check(Player p, Block block)
295         {
296             Rectangle playerRec = p.Bitmap.BoundingRectangle(p.X, p.Y); //getting
                ↪  the bounding rectangle of the player
297             Rectangle blockRec = block.Bitmap.BoundingRectangle(block.X, block.Y);
                ↪  //get the bounding rectangle of the block
```

```
298             Rectangle intersection = SplashKit.Intersection(playerRec, blockRec);
                ↪ //get the intersection rectangle and a save it as a rectangle
299             if (intersection.Width > intersection.Height) //vertical collision
300             {
301                 //Checking player intersection with the block from the bottom
                    ↪ (players feet)
302                 if (SplashKit.RectangleBottom(playerRec) >
                    ↪ SplashKit.RectangleTop(blockRec) &&
                    ↪ SplashKit.RectangleBottom(playerRec) <
                    ↪ SplashKit.RectangleBottom(blockRec))
303                 {
304                     if (block.Type != "magnet") //landing on magnetic blocks is not
                        ↪ allowed
305                     {
306                         if (intersection.Height > 1)
307                         {
308                             p.Y -= intersection.Height; //adjust the player Y based
                                ↪ on the intersection height
309                             p.DY = 0; //stop the player from falling
310                             p.Landed = true; //setting landed to true, the player
                                ↪ has landed on the platfrom
311                         }
312
313                         //collision with a lava or spiked block will lead to
                            ↪ resetting the player (player dies)
314                         if (block.Type == "lava" || block.Type == "spiked")
315                         {
316                             p.Reset();
317                         }
318                     }
319                 }
320                 else //collision with a block from top (head of the player)
321                 {
322                     p.Y += intersection.Height; //adjust the player Y based on the
                        ↪ intersection height
323                     //this collision with magneticBlocks in level two allows the
                        ↪ player to be attrackted to the block and hang from it
324                     //if the superpower is used (i.e. p.Attract is true)
325                     if (block.Type == "magnet" && p.Attract)
326                     {
327                         p.Y = block.Y;
328                         p.Attract = false; //if the player stops using the pwoer,
                            ↪ he will fall (stops holding ctrl)
329                     }
330                 }
331             }
332         }
333     }
334 }
335
336 //ICollision
337 namespace MarioGame
338 {
```

```
339     public interface ICollision
340     {
341         //provides the collision check method to be used in CollisionProcessor,
            ↪ Horizontal and Vertical Collision classes
342         public abstract void Check(Player p, Block block);
343     }
344 }
345 
346 ///Game Family Classes
347 //Door
348 using SplashKitSDK;
349 
350 namespace MarioGame
351 {
352     public class Door : GameObject, IDrawable
353     {
354         private Bitmap _doorBitmap;
355         private Level _nextLevel;
356         private Superpower _nextSuperpower;
357         private bool _passedLevel;
358 
359         /// <summary>
360         /// Default constructor for a door
361         /// </summary>
362         /// <param name="name"></param>
363         /// <param name="x"></param>
364         /// <param name="y"></param>
365         public Door(string name, double x, double y) : base(name, x, y)
366         {
367             _doorBitmap = SplashKit.LoadBitmap(name, "door.png");
368             X = x;
369             Y = y;
370             _passedLevel = false;
371         }
372 
373         /// <summary>
374         /// Constructor with next level parameter, supplied from Game class
375         /// </summary>
376         /// <param name="name"></param>
377         /// <param name="x"></param>
378         /// <param name="y"></param>
379         /// <param name="nextLevel"></param>
380         public Door(string name, double x, double y, Level nextLevel) : this(name,
            ↪ x, y)
381         {
382             _nextLevel = nextLevel;
383         }
384 
385         /// <summary>
386         /// Constructor with superpower parameter, supplied from Game class
387         /// </summary>
388         /// <param name="name"></param>
389         /// <param name="x"></param>
```

```
390         /// <param name="y"></param>
391         /// <param name="nextLevel"></param>
392         /// <param name="superpower"></param>
393         public Door(string name, double x, double y, Level nextLevel, Superpower
            ↪ superpower) : this(name, x, y, nextLevel)
394         {
395             _nextSuperpower = superpower;
396         }
397 
398         /// <summary>
399         /// Bitmap proeprty from IDrawable interface
400         /// </summary>
401         public Bitmap Bitmap
402         {
403             get
404             {
405                 return _doorBitmap;
406             }
407         }
408 
409         /// <summary>
410         /// Checks if the player has arrived at the door and has the key to open it
411         /// </summary>
412         /// <param name="p"></param>
413         public void ArrivedAtDoor(Player p)
414         {
415             Rectangle playerRec = p.Bitmap.BoundingRectangle(p.X, p.Y); //getting
                ↪ the bounding rectangle of the player
416             Rectangle doorRec = _doorBitmap.BoundingRectangle(X, Y); //getting the
                ↪ bounding rectangle of the door
417 
418             //checks if the player is at the door and has the key
419             if (SplashKit.RectanglesIntersect(playerRec, doorRec) && p.HasKey)
420             {
421                 _passedLevel = true; //turns passed level to true
422                 SetLevel(p); //sets the player's level to the next level
423                 p.Reset(); //resets the player position and appearance
424                 p.HasKey = false; //sets HasKey to false
425             }
426         }
427 
428         /// <summary>
429         /// Checks if the player has successfully passed the level and moves him to
            ↪ the next level
430         /// also sets the new superpower
431         /// </summary>
432         /// <param name="p"></param>
433         public void SetLevel(Player p)
434         {
435             if (_passedLevel)
436             {
437                 p.Level = _nextLevel;
438                 p.Superpower = _nextSuperpower;
```

```
439             }
440         }
441 
442         /// <summary>
443         /// Implementation of the IDrawable interface Draw method to draw the door
            ↪ on the screen
444         /// </summary>
445         public void Draw()
446         {
447             SplashKit.DrawBitmap(Bitmap, X, Y);
448         }
449     }
450 }
451 
452 //Game
453 using SplashKitSDK;
454 using System;
455 
456 namespace MarioGame
457 {
458     //game class applying the singleton pattern
459     class Game
460     {
461         private static Game _instance; //to create a single instance of the game
462         //objects for the contruction of the game
463         int windowWidth;
464         double ground;
465         Player player;
466         Level startWindow, exitWindow, levelOne, levelTwo, levelThree;
467         Superpower levelOneSuper, levelTwoSuper, levelThreeSuper;
468         Key levelOneKey, levelTwoKey, levelThreeKey;
469         Door doorOne, doorTwo, doorThree;
470         Block NormBlock1_1, NormBlock1_2, NormBlock1_3, NormBlock1_4, NormBlock1_5;
471         Block LavaBlock2_0, LavaBlock2_1, LavaBlock2_2;
472         Block NormBlock2_1, NormBlock2_2, NormBlock2_3, NormBlock2_4, NormBlock2_5,
            ↪ NormBlock2_6, NormBlock2_7, NormBlock2_8, NormBlock2_9;
473         Block MagBlock2_1, MagBlock2_2, MagBlock2_3;
474         Block GlassBlock3_1, GlassBlock3_2, GlassBlock3_3;
475         Block NormBlock3_1, NormBlock3_2, NormBlock3_3, NormBlock3_4, NormBlock3_5,
            ↪ NormBlock3_6, NormBlock3_7, NormBlock3_8;
476         Block SpikeBlock3_1, SpikeBlock3_2;
477 
478         /// <summary>
479         /// Game defaul constructor applying the singleton design patters (cannot
            ↪ be instantiated outside Game class)
480         /// </summary>
481         private Game()
482         {
483         }
484 
485         /// <summary>
486         /// Used to allow creating one instance only of the game class (i.e.
            ↪ singleton)
```

```
487         /// If an instance hasn't been created, it creates and returns it
488         /// If it has been created, it will return the created one (won't create a
            ↪ new one)
489         /// </summary>
490         /// <returns></returns>
491         public static Game getInstance()
492         {
493             if (_instance == null)
494             {
495                 _instance = new Game();
496             }
497 
498             return _instance;
499         }
500 
501         /// <summary>
502         /// Holds the construction and instantiation of the game (player, levels
            ↪ and their objects)
503         /// </summary>
504         public void CreateGame()
505         {
506             windowWidth = 800;
507             //creating the player
508             player = new Player("mario", 0, 467);
509             ground = SplashKit.WindowHeight("Mario Game") - player.Bitmap.Height -
                ↪ 25;
510 
511             //Creating start window
512             startWindow = new Level("startWindow", "The start window", 0, 0);
513             player.Level = startWindow; //setting the player's level to the start
                ↪ window
514 
515             //Creating Level 1
516             levelOne = new Level("levelOne", "Level One - Easy", 0, 0);
517             levelOneSuper = new SuperJump("Super jump"); //level one's super power
                ↪ is the super jump
518 
519             //creating the construction blocks of level 1 (All normal blocks - easy
                ↪ level)
520             NormBlock1_1 = new NormalBlock("norm1.1", 150, 440);
521             NormBlock1_2 = new NormalBlock("norm1.2", 400, 400);
522             NormBlock1_3 = new NormalBlock("norm1.3", 650, 225);
523             NormBlock1_4 = new NormalBlock("norm1.4", 200, 225);
524             NormBlock1_5 = new NormalBlock("norm1.5", 0, 125);
525 
526             //Adding the blocks to the level to allow the player to interact with
                ↪ them
527             levelOne.AddBlock(NormBlock1_1);
528             levelOne.AddBlock(NormBlock1_2);
529             levelOne.AddBlock(NormBlock1_3);
530             levelOne.AddBlock(NormBlock1_4);
531             levelOne.AddBlock(NormBlock1_5);
532 
```

```
533             //creating and adding the key to level 1
534             levelOneKey = new Key("levelOneKey", 670, 170);
535             levelOne.Key = levelOneKey;
536 
537             //Creatig Level 2
538             levelTwo = new Level("levelTwo", "Level Two - Medium", 0, 0);
539             levelTwoSuper = new SuperMagnet("Super Magnet");
540 
541             //creating the construction blocks of level 2 (lava, magnetic and
                ↪  normal blocks - medium level)
542             LavaBlock2_0 = new LavaBlock("lava2.0", 135, 565);
543             LavaBlock2_1 = new LavaBlock("lava2.1", 190, 565);
544             LavaBlock2_2 = new LavaBlock("lava2.2", 650, 565);
545             NormBlock2_1 = new NormalBlock("norm2.1", 700, 420);
546             NormBlock2_2 = new NormalBlock("norm2.2", 0, 300);
547             NormBlock2_3 = new NormalBlock("norm2.3", 150, 300);
548             NormBlock2_4 = new NormalBlock("norm2.4", 0, 125);
549             NormBlock2_5 = new NormalBlock("norm2.5", 620, 168);
550             NormBlock2_6 = new NormalBlock("norm2.6", 700, 30);
551             NormBlock2_7 = new NormalBlock("norm2.7", 300, 300);
552             NormBlock2_8 = new NormalBlock("norm2.7", 470, 168);
553             NormBlock2_9 = new NormalBlock("norm2.7", 770, 168);
554             MagBlock2_1 = new MagneticBlock("mag2.1", 65, 332);
555             MagBlock2_2 = new MagneticBlock("mag2.2", 470, 200);
556             MagBlock2_3 = new MagneticBlock("mag2.3", 120, 0);
557 
558             //Adding the blocks to the level to allow the player to interact with
                ↪  them
559             levelTwo.AddBlock(LavaBlock2_0);
560             levelTwo.AddBlock(LavaBlock2_1);
561             levelTwo.AddBlock(LavaBlock2_2);
562             levelTwo.AddBlock(NormBlock2_1);
563             levelTwo.AddBlock(NormBlock2_2);
564             levelTwo.AddBlock(NormBlock2_3);
565             levelTwo.AddBlock(NormBlock2_4);
566             levelTwo.AddBlock(NormBlock2_5);
567             levelTwo.AddBlock(NormBlock2_6);
568             levelTwo.AddBlock(NormBlock2_7);
569             levelTwo.AddBlock(NormBlock2_8);
570             levelTwo.AddBlock(NormBlock2_9);
571             levelTwo.AddBlock(MagBlock2_1);
572             levelTwo.AddBlock(MagBlock2_2);
573             levelTwo.AddBlock(MagBlock2_3);
574 
575             //creating and adding the key to level 2
576             levelTwoKey = new Key("levelTwoKey", 725, 80);
577             levelTwo.Key = levelTwoKey;
578 
579             //Creating level 3
580             levelThree = new Level("levelThree", "Level Three - Hard", 0, 0);
581             levelThreeSuper = new SuperVoice("Super Voice");
582 
583             //creating the construction blocks of level 3 (glass, spiked and normal
                ↪  blocks - hard level)
```

```
584             GlassBlock3_1 = new GlassBlock("glass3.1", 200, 390);
585             GlassBlock3_2 = new GlassBlock("glass3.2", 650, 172);
586             GlassBlock3_3 = new GlassBlock("glass3.3", 225, -55);
587             NormBlock3_1 = new NormalBlock("norm2.4", 650, 125);
588             NormBlock3_2 = new NormalBlock("norm3.2", 650, 460);
589             NormBlock3_3 = new NormalBlock("norm3.3", 530, 352);
590             NormBlock3_4 = new NormalBlock("norm3.4", 354, 352);
591             NormBlock3_5 = new NormalBlock("norm3.5", 150, 250);
592             NormBlock3_6 = new NormalBlock("norm3.6", 0, 250);
593             NormBlock3_7 = new NormalBlock("norm3.7", 120, 125);
594             NormBlock3_8 = new NormalBlock("norm3.8", 380, 80);
595             SpikeBlock3_1 = new SpikedBlock("spike3.1", 334, 317);
596             SpikeBlock3_2 = new SpikedBlock("spike3.2", 505, 317);
597
598             //Adding the blocks to the level to allow the player to interact with
                ↪  them
599             levelThree.AddBlock(GlassBlock3_1);
600             levelThree.AddBlock(GlassBlock3_2);
601             levelThree.AddBlock(GlassBlock3_3);
602             levelThree.AddBlock(NormBlock3_1);
603             levelThree.AddBlock(NormBlock3_2);
604             levelThree.AddBlock(NormBlock3_3);
605             levelThree.AddBlock(NormBlock3_4);
606             levelThree.AddBlock(NormBlock3_5);
607             levelThree.AddBlock(NormBlock3_6);
608             levelThree.AddBlock(NormBlock3_7);
609             levelThree.AddBlock(NormBlock3_8);
610             levelThree.AddBlock(SpikeBlock3_1);
611             levelThree.AddBlock(SpikeBlock3_2);
612
613             //creating and adding the key to level 3
614             levelThreeKey = new Key("levelThreeKey", 700, 500);
615             levelThree.Key = levelThreeKey;
616
617             //creating the exit/finish window
618             exitWindow = new Level("exitWindow", "The exit window", 0, 0);
619
620             //creating and adding the door to level 1
621             //passing in level two as the next level, and levelTwoSuper as the next
                ↪  superpower
622             doorOne = new Door("doorOne", 0, 0, levelTwo, levelTwoSuper);
623             levelOne.Door = doorOne;
624
625             //creating and adding the door to level 2
626             //passing in level three as the next level, and levelThreeSuper as the
                ↪  next superpower
627             doorTwo = new Door("doorTwo", 0, 0, levelThree, levelThreeSuper);
628             levelTwo.Door = doorTwo;
629
630             //creating and adding the door to level 3
631             //passing in the exit window as the next level
632             doorThree = new Door("doorTwo", 700, 0, exitWindow);
633             levelThree.Door = doorThree;
```

```
634
635             Console.WriteLine("WELCOME TO SUPER MARIO 11.29\nPRESS ENTER TO
                ↪  START\nMOVING KEYS:\n\tARROWS TO MOVE LEFT AND RIGHT" +
636                 "\n\tSPACE OR UP ARROW TO JUMP\n\tS KEY TO SHRINK\n\tE KEY TO
                    ↪  ENLARGE\n\tCTRL TO USE YOUR SUPERPOWER\nPRESS I FOR LEVEL
                    ↪  INSTRUCTIONS");
637         }
638
639         /// <summary>
640         /// Handles keyboard input from the user
641         /// will be run in the main loop of the game
642         /// </summary>
643         public void Run()
644         {
645             //Checking if the player's current level is the start window
646             //if the user presses enter, move to the first level
647             if (player.Level == startWindow &&
                ↪  SplashKit.KeyTyped(KeyCode.ReturnKey))
648             {
649                 player.Reset();
650                 player.Level = levelOne;
651                 player.Superpower = levelOneSuper;
652             }
653
654             //Checking if the player's current level is the exit window
655             //if the user presses enter, exit the game
656             if (player.Level == exitWindow && SplashKit.KeyTyped(KeyCode.ReturnKey))
657             {
658                 Console.WriteLine("\nNICE WORK, GOOD BYE!");
659                 Environment.Exit(0);
660             }
661
662             //jump if the user presses Space or up key
663             if (SplashKit.KeyTyped(KeyCode.SpaceKey) ||
                ↪  SplashKit.KeyTyped(KeyCode.UpKey))
664             {
665                 player.Jump();
666                 //if the player presses ctrl left or right (check if they are in
                    ↪  level one)
667                 if ((SplashKit.KeyDown(KeyCode.LeftCtrlKey) ||
                    ↪  SplashKit.KeyDown(KeyCode.RightCtrlKey) && player.Level ==
                    ↪  levelOne))
668                 {
669                     player.Superpower.Use(player); //use the superpower (super jump
                        ↪  for level 1)
670                 }
671             }
672
673             //Display specific level instructions for levels 1, 2 and 3 (not start
                ↪  window or exit window)
674             //infromation is displayed when the i key is pressed
675             if ((player.Level == levelOne || player.Level == levelTwo ||
                ↪  player.Level == levelThree) && SplashKit.KeyTyped(KeyCode.IKey))
```

```
676             {
677                 Console.WriteLine(player.Instructions);
678             }
679 
680             //if the player is in levels two or three, pressing ctrl will activate
                ↪  the super power
681             //this is different from level one, because in level one the super
                ↪  power is associated with the jump
682             if (player.Level == levelTwo || player.Level == levelThree)
683             {
684                 if (SplashKit.KeyDown(KeyCode.LeftCtrlKey) ||
                    ↪  SplashKit.KeyDown(KeyCode.RightCtrlKey))
685                 {
686                     player.Superpower.Use(player);
687                 }
688             }
689 
690             //moving the player to the right
691             if (SplashKit.KeyDown(KeyCode.RightKey))
692             {
693                 player.Move("right", windowWidth);
694             }
695 
696             //moving the player to the left
697             if (SplashKit.KeyDown(KeyCode.LeftKey))
698             {
699                 player.Move("left", windowWidth);
700             }
701 
702             //shrink the player using S
703             if (SplashKit.KeyTyped(KeyCode.SKey))
704             {
705                 player.Resize.Shrink(player);
706             }
707 
708             //Enlarge the player using E
709             if (SplashKit.KeyTyped(KeyCode.EKey))
710             {
711                 player.Resize.Enlarge(player);
712             }
713 
714             //run the level methods
715             player.Level.Run(player);
716             //run the player methods
717             player.Run(ground);
718         }
719     }
720 }
721 
722 //GameObject
723 namespace MarioGame
724 {
725     public abstract class GameObject
```

```
726     {
727         //private fields to hold the object's coordinates in the game + their width
            ↪  and height
728         private double _x, _y;
729         private string _name;
730 
731         /// <summary>
732         /// Default GameObject constructor
733         /// </summary>
734         /// <param name="name"></param>
735         /// <param name="x"></param>
736         /// <param name="y"></param>
737         public GameObject(string name, double x, double y)
738         {
739             _x = x;
740             _y = y;
741             _name = name;
742         }
743 
744         /// <summary>
745         /// The x coordinate of the game object
746         /// </summary>
747         public double X
748         {
749             get
750             {
751                 return _x;
752             }
753             set
754             {
755                 _x = value;
756             }
757         }
758 
759         /// <summary>
760         /// The y coordinate of the game object
761         /// </summary>
762         public double Y
763         {
764             get
765             {
766                 return _y;
767             }
768             set
769             {
770                 _y = value;
771             }
772         }
773 
774         /// <summary>
775         /// The length property of the game object
776         /// </summary>
777         public string Name
```

```
        {
            get
            {
                return _name;
            }
            set
            {
                _name = value;
            }
        }
    }
}

//IDrawable
using SplashKitSDK;

namespace MarioGame
{
    public interface IDrawable
    {
        /// <summary>
        /// Bitmap property
        /// </summary>
        public Bitmap Bitmap
        {
            get;
        }

        /// <summary>
        /// abstract draw method
        /// </summary>
        public abstract void Draw();
    }
}

//Key
using SplashKitSDK;

namespace MarioGame
{
    public class Key : GameObject, IDrawable
    {
        private Bitmap _keyBitmap;
        private bool _disappear;

        /// <summary>
        /// Default Key Constructor
        /// </summary>
        /// <param name="name"></param>
        /// <param name="x"></param>
        /// <param name="y"></param>
        public Key(string name, double x, double y) : base(name, x, y)
        {
```

```
831             _keyBitmap = SplashKit.LoadBitmap("key", "key.png");
832         }
833 
834         /// <summary>
835         /// Bitmap property from IDrawable interface
836         /// </summary>
837         public Bitmap Bitmap
838         {
839             get
840             {
841                 return _keyBitmap;
842             }
843         }
844 
845         /// <summary>
846         /// Checks if the player intersect with the key, if so allows the player to
        ↪  take the key
847         /// </summary>
848         /// <param name="p"></param>
849         public void TakeKey(Player p)
850         {
851             Rectangle playerRec = p.Bitmap.BoundingRectangle(p.X, p.Y); //getting
            ↪  the bounding rectangle of the player
852             Rectangle keyRec = Bitmap.BoundingRectangle(X, Y); //getting the
            ↪  bounding rectangle of the key
853             if (SplashKit.RectanglesIntersect(playerRec, keyRec)) //check if the
            ↪  player intersects with the key
854             {
855                 _disappear = true; //to remove the key from the display
856                 p.HasKey = true; //sets the HasKey property of the player to true
857             }
858         }
859 
860         /// <summary>
861         /// Property to return the _disapper boolean
862         /// </summary>
863         public bool Disappear
864         {
865             get
866             {
867                 return _disappear;
868             }
869         }
870 
871         /// <summary>
872         /// Implementation of the IDrawable interface Draw method to draw the block
        ↪  on the screen
873         /// </summary>
874         public void Draw()
875         {
876             SplashKit.DrawBitmap(Bitmap, X, Y);
877         }
878     }
```

```
879 }
880
881 //Level
882 using SplashKitSDK;
883 using System.Collections.Generic;
884
885 namespace MarioGame
886 {
887     public class Level : GameObject, IDrawable
888     {
889         //illustration fields
890         private Bitmap _backgroundBitmap;
891         private List<Block> _blocks;
892         private Door _door;
893         private Key _key;
894         private string _desc;
895
896         /// <summary>
897         /// Level default constructor
898         /// </summary>
899         /// <param name="name"></param>
900         /// <param name="desc"></param>
901         /// <param name="x"></param>
902         /// <param name="y"></param>
903         public Level(string name, string desc, double x, double y) : base(name, x,
            ↪ y)
904         {
905             _backgroundBitmap = SplashKit.LoadBitmap(name, name + ".png");
906             _blocks = new List<Block>();
907             _desc = desc;
908         }
909
910         /// <summary>
911         /// Key propetry (used in game to set the key of the level)
912         /// </summary>
913         public Key Key
914         {
915             set
916             {
917                 _key = value;
918             }
919         }
920
921         /// <summary>
922         /// Bitmap property from IDrawable interface
923         /// </summary>
924         public Bitmap Bitmap
925         {
926             get
927             {
928                 return _backgroundBitmap;
929             }
930         }
```

```
931
932         /// <summary>
933         /// to add blocks to the list of blocks to be drawn on the screen
934         /// </summary>
935         /// <param name="block"></param>
936         public void AddBlock(Block block)
937         {
938             _blocks.Add(block);
939         }
940
941         /// <summary>
942         /// Blocks property (used in Resize class)
943         /// </summary>
944         public List<Block> Blocks
945         {
946             get
947             {
948                 return _blocks;
949             }
950         }
951
952         /// <summary>
953         /// Retruns the description of the level
954         /// </summary>
955         public string Description
956         {
957             get
958             {
959                 return _desc.ToUpper();
960             }
961         }
962
963         /// <summary>
964         /// Door propetry (used in game to set the door of the level)
965         /// </summary>
966         public Door Door
967         {
968             set
969             {
970                 _door = value;
971             }
972         }
973
974         /// <summary>
975         /// Calls the PlayerBlockCollision method for each of the blocks in the
            ↪  level
976         /// </summary>
977         /// <param name="p"></param>
978         public void CollisionsResponder(Player p)
979         {
980             foreach (Block block in _blocks)
981             {
982                 block.PlayerBlockCollision(p);
```

```
            }
        }

        /// <summary>
        /// Draws the level, including, the background, blocks, door and key
        /// </summary>
        public void Draw()
        {
            SplashKit.DrawBitmap(_backgroundBitmap, X, Y);
            if (_blocks != null)
            {
                foreach (Block block in _blocks)
                {
                    block.Draw();
                }
            }
            if (_door != null)
            {
                _door.Draw();
            }
            if (_key != null)
            {
                if (!_key.Disappear)
                {
                    _key.Draw();
                }
            }
        }

        /// <summary>
        /// Calls the methods that will need to be run continuously
        /// </summary>
        /// <param name="p"></param>
        public void Run(Player p)
        {
            Draw();
            CollisionsResponder(p);
            if (_key != null)
            {
                _key.TakeKey(p);
            }
            if (_door != null)
            {
                _door.ArrivedAtDoor(p);
            }
        }
    }
}


//Program
using System;
using SplashKitSDK;
```

```

namespace MarioGame
{
    public class Program
    {
        public static void Main()
        {
            int windowHeight = 600;
            int windowWidth = 800;

            //Creating the start window
            new Window("Mario Game", windowWidth, windowHeight);

            Game game = Game.getInstance();
            game.CreateGame(); //create the game

            //leaves the window open unless requested to be closed
            do
            {
                SplashKit.ProcessEvents(); //allows Splashkit to react to user
                ↪  interactions
                SplashKit.ClearScreen();

                game.Run(); //run the game

                //refresh
                SplashKit.RefreshScreen();

            } while (!SplashKit.WindowCloseRequested("Mario Game"));
        }
    }
}

///PlayerChanges Family Classes
//Direction
using SplashKitSDK;

namespace MarioGame
{
    public class Direction
    {
        private Bitmap _leftBitmap, _rightBitmap, _leftShrunkBitmap,
        ↪  _rightShrunkBitmap;

        /// <summary>
        /// Direction default constructor, loads the four bitmaps
        /// </summary>
        public Direction()
        {
            _leftBitmap = SplashKit.LoadBitmap("mario_left", "mario_left.png");
            _leftShrunkBitmap = SplashKit.LoadBitmap("mario_shrunk_left",
            ↪  "mario_shrunk_left.png");
            _rightBitmap = SplashKit.LoadBitmap("mario", "mario.png");
```

```
1086            _rightShrunkBitmap = SplashKit.LoadBitmap("mario_shrunk",
                ↪  "mario_shrunk.png");
1087        }
1088
1089        /// <summary>
1090        /// LeftBitmap readonly property, retunrs the player bitmap facing left
                ↪  (enlarged)
1091        /// </summary>
1092        public Bitmap LeftBitmap
1093        {
1094            get
1095            {
1096                return _leftBitmap;
1097            }
1098        }
1099
1100        /// <summary>
1101        /// LeftShrunkBitmap readonly property, retunrs the player bitmap facing
                ↪  left (shrunk)
1102        /// </summary>
1103        public Bitmap LeftShrunkBitmap
1104        {
1105            get
1106            {
1107                return _leftShrunkBitmap;
1108            }
1109        }
1110
1111        /// <summary>
1112        /// RightBitmap readonly property, retunrs the player bitmap facing right
                ↪  (enlarged)
1113        /// </summary>
1114        public Bitmap RightBitmap
1115        {
1116            get
1117            {
1118                return _rightBitmap;
1119            }
1120        }
1121
1122        /// <summary>
1123        /// RightShrunkBitmap readonly property, retunrs the player bitmap facing
                ↪  right (shrunk)
1124        /// </summary>
1125        public Bitmap RightShrunkBitmap
1126        {
1127            get
1128            {
1129                return _rightShrunkBitmap;
1130            }
1131        }
1132
1133        /// <summary>
```

```
1134            /// checks if the player is not already facing left, and sets the player's
                ↪  bitmap to the left bitmap
1135            /// </summary>
1136            /// <param name="p"></param>
1137            public void FacingLeft(Player p)
1138            {
1139                if (!p.Left)
1140                {
1141                    p.Bitmap = _leftBitmap;
1142                }
1143            }
1144
1145            /// <summary>
1146            /// checks if the player is not already facing left, and sets the player's
                ↪  bitmap to the left bitmap (shrunk)
1147            /// </summary>
1148            /// <param name="p"></param>
1149            public void FacingLeftShrunk(Player p)
1150            {
1151                if (!p.Left)
1152                {
1153                    p.Bitmap = _leftShrunkBitmap;
1154                }
1155            }
1156
1157            /// <summary>
1158            ///  checks if the player is not already facing right, and sets the
                ↪  player's bitmap to the right bitmap
1159            /// </summary>
1160            /// <param name="p"></param>
1161            public void FacingRight(Player p)
1162            {
1163                if (p.Left)
1164                {
1165                    p.Bitmap = _rightBitmap;
1166                }
1167            }
1168
1169            /// <summary>
1170            ///  checks if the player is not already facing right, and sets the
                ↪  player's bitmap to the right bitmap (shrunk)
1171            /// </summary>
1172            /// <param name="p"></param>
1173            public void FacingRightShrunk(Player p)
1174            {
1175                if (p.Left)
1176                {
1177                    p.Bitmap = _rightShrunkBitmap;
1178                }
1179            }
1180        }
1181    }
1182
```

```
1183 //Player
1184 using SplashKitSDK;
1185
1186 namespace MarioGame
1187 {
1188     public class Player : GameObject, IDrawable
1189     {
1190         private Bitmap _playerBitmap; //, _DieBitmap; Add die bitmap later
1191         private double _dy, _dx, _gravity;
1192         private bool _jumping, _hasKey, _landed, _attract, _sing, _left;
1193         private Level _level;
1194         private Resize _resize;
1195         private Superpower _superpower;
1196         private Direction _direction;
1197
1198         /// <summary>
1199         /// Default constructor of the player, responsible for initialising the
              ↪ fields, such as,
1200         /// setting the default player bitmap.
1201         /// </summary>
1202         /// <param name="name"></param>
1203         /// <param name="x"></param>
1204         /// <param name="y"></param>
1205         public Player(string name, double x, double y) : base(name, x, y)
1206         {
1207             _playerBitmap = SplashKit.LoadBitmap(name, name + ".png");
1208             _dx = 0.2;
1209             _gravity = 0.003;
1210             _jumping = false;
1211             _hasKey = false;
1212             _resize = new Resize();
1213             _direction = new Direction();
1214             _landed = false;
1215             _attract = false;
1216             _sing = false;
1217         }
1218
1219         /// <summary>
1220         /// Bitmap property used in other classes to get and set the displayed
              ↪ player bitmap
1221         /// </summary>
1222         public Bitmap Bitmap
1223         {
1224             get
1225             {
1226                 return _playerBitmap;
1227             }
1228             set
1229             {
1230                 _playerBitmap = value;
1231             }
1232         }
1233
```

```
1234        /// <summary>
1235        /// Resize property applying the resize class which is responsible for
            ↪ player enlarge and shrink
1236        /// </summary>
1237        public Resize Resize
1238        {
1239            get
1240            {
1241                return _resize;
1242            }
1243        }
1244
1245        /// <summary>
1246        /// Superpower property, used to get the player's currecnt superpower, and
            ↪ set the next one (next level)
1247        /// </summary>
1248        public Superpower Superpower
1249        {
1250            get
1251            {
1252                return _superpower;
1253            }
1254            set
1255            {
1256                _superpower = value;
1257            }
1258        }
1259
1260        /// <summary>
1261        /// Level property, used to get and set the player's level (in game and
            ↪ door classes)
1262        /// </summary>
1263        public Level Level
1264        {
1265            get
1266            {
1267                return _level;
1268            }
1269            set
1270            {
1271                _level = value;
1272            }
1273        }
1274
1275        /// <summary>
1276        /// Landed bool property, is used when the player lands on a platform to
            ↪ stop him from falling
1277        /// (Vertical collision class)
1278        /// </summary>
1279        public bool Landed
1280        {
1281            set
1282            {
```

```
            _landed = value;
        }
    }

    /// <summary>
    /// Attract property, used with the supermagnet super power to get and set the player's
    /// attraction status to the magnetic block
    /// </summary>
    public bool Attract
    {
        get
        {
            return _attract;
        }
        set
        {
            _attract = value;
        }
    }

    /// <summary>
    /// Left property, used to get and set the player's facing directoin to determine which
    /// bitmap to be displayed on the screen (works with the Resize class)
    /// </summary>
    public bool Left
    {
        get
        {
            return _left;
        }
        set
        {
            _left = value;
        }
    }

    /// <summary>
    /// Sing property, used in level 3, when the player has the power to break glass using his voice
    /// Used in Supervoice class
    /// </summary>
    public bool Sing
    {
        get
        {
            return _sing;
        }
        set
        {
            _sing = value;
        }
```

```
        }

        /// <summary>
        /// Moves the player in 1D (left or right only)
        /// </summary>
        /// <param name="dir"></param>
        /// <param name="width"></param>
        public void Move(string dir, int width)
        {
            //making sure the player stays in the borders of the game
            if (dir == "left" && X > 0)
            {
                //checking which Bitmap to display
                if (_resize.Shrunk)
                {
                    _direction.FacingLeftShrunk(this);
                }
                else
                {
                    _direction.FacingLeft(this);
                }
                _left = true;
                X -= _dx;
            }
            else if (dir == "right" && X < width -
             ↪  SplashKit.BitmapWidth(_playerBitmap))
            {
                if (_resize.Shrunk)
                {
                    _direction.FacingRightShrunk(this);
                }
                else
                {
                    _direction.FacingRight(this);
                }
                _left = false;
                X += _dx;
            }
        }

        /// <summary>
        /// To allow the player to jump if he is not already jumping or has landed
         ↪  on a platform
        /// </summary>
        public void Jump()
        {
            if (!_jumping || _landed)
            {
                _dy = -1; //setting the player's vertical speed, which will allow
                 ↪  for the projectile motion
                          //of the player jumping up and falling down
                _jumping = true; //set jump to true, which will be used in update
                 ↪  method
```

```
                _landed = false; //only allow one jump
            }
        }

        /// <summary>
        /// This method will keep running throughout the game to update the
        ↪  vertical postion of the player
        /// </summary>
        /// <param name="ground"></param>
        public void Update(double ground)
        {
            //make the jumping suitable for both mario sizes (normal and shrunk)
            if (_resize.Shrunk)
            {
                ground += 35;
            }

            if (_jumping)
            {
                _dy += _gravity; //change the vertical speed by a factor of gravity
            }

            Y += _dy; //change the player's vertical position by the dy value

            if (Y > ground) //to prevent the player from falling below ground
            {
                Y = ground;
                _jumping = false;
            }
        }

        /// <summary>
        /// DY writeonly property used in the superjump class
        /// </summary>
        public double DY
        {
            set
            {
                _dy = value;
            }
        }

        /// <summary>
        /// bool property used to check if the player has the key of the level
        /// </summary>
        public bool HasKey
        {
            get
            {
                return _hasKey;
            }
            set
            {
```

```
                _hasKey = value;
            }
        }

        /// <summary>
        /// To print instructions for the user in console
        /// </summary>
        public string Instructions
        {
            get
            {
                string instructions;
                //the instructions includes infromation about the current level, superpower, and general infromation about how the game works
                instructions = Level.Description.ToUpper() + ":\n" + Superpower.Description +
                    "\nYOUR ROLE IS TO AVOID THE OBSTACLES, COLLECT THE KEY AND USE IT TO OPEN THE DOOR TO THE NEXT LEVEL\n\tENJOY!";

                return instructions;
            }
        }

        /// <summary>
        /// Implementation of the IDrawable interface Draw method to draw the player on the screen
        /// </summary>
        public void Draw()
        {
            SplashKit.DrawBitmap(_playerBitmap, X, Y);
        }

        /// <summary>
        /// Resets the player its starting postions
        /// Also resets the player's superpower status
        /// Sets the player back to being enlarge (default bitmap)
        /// </summary>
        public void Reset()
        {
            X = 0;
            Y = 467;
            _attract = false;
            _sing = false;
            _resize.Enlarge(this);
        }

        /// <summary>
        /// Calls the methods that will need to be run continuously
        /// </summary>
        /// <param name="ground"></param>
        public void Run(double ground)
        {
            Draw();
```

```
            Update(ground);
        }
    }
}

//Resize
using SplashKitSDK;

namespace MarioGame
{
    public class Resize
    {
        private bool _shrunk;
        private Direction _direction;

        /// <summary>
        /// Resize Default constructor
        /// </summary>
        public Resize()
        {
            _shrunk = false;
            _direction = new Direction();
        }

        /// <summary>
        /// Shrink method, to shrink the player when called (player presses S)
        /// </summary>
        /// <param name="p"></param>
        public void Shrink(Player p)
        {
            if (!_shrunk) //checks if the player is not already shrunk
            {
                //checks the direction the player is facing to set the
                ↪ corresponding bitmap
                if (p.Left)
                {
                    p.Bitmap = _direction.LeftShrunkBitmap;
                }
                else
                {
                    p.Bitmap = _direction.RightShrunkBitmap;
                }
                p.Y += 35; //lower the player by 35 (difference in height between
                ↪ englarged and shrunk bitmaps)
                _shrunk = true; //set shrunk to true
            }
        }

        /// <summary>
        /// Boolean Shrunk property, returns _shrunk bool
        /// </summary>
        public bool Shrunk
        {
```

```
1534            get
1535            {
1536                return _shrunk;
1537            }
1538        }
1539
1540        /// <summary>
1541        /// Enlarge method, to enlarge the player when called (player presses E)
1542        /// </summary>
1543        /// <param name="p"></param>
1544        public void Enlarge(Player p)
1545        {
1546            if (_shrunk && !BlockEnlarge(p)) //checks if the player is shrunk and
                ↪  enlarging is not blocked
1547            {
1548                //checks the direction the player is facing to set the
                    ↪  corresponding bitmap
1549                if (p.Left)
1550                {
1551                    p.Bitmap = _direction.LeftBitmap;
1552                }
1553                else
1554                {
1555                    p.Bitmap = _direction.RightBitmap;
1556                }
1557
1558                p.Y -= 35; //rises the player by 35 pixels
1559                _shrunk = false;
1560            }
1561        }
1562
1563        /// <summary>
1564        /// Checks if the player will collide with a block if requests enlrage and
            ↪  blocks it if that's the case
1565        /// </summary>
1566        /// <param name="p"></param>
1567        /// <returns></returns>
1568        public bool BlockEnlarge(Player p)
1569        {
1570            foreach (Block block in p.Level.Blocks)
1571            {
1572                Rectangle playerEnlargeRec = SplashKit.RectangleFrom(p.X, p.Y - 36,
                    ↪  p.Bitmap.Width, _direction.RightBitmap.Height);
1573                Rectangle blockRec = block.Bitmap.BoundingRectangle(block.X,
                    ↪  block.Y); //get the bounding rectangle of the block
1574
1575                //checks if the enlarged player will intersect with a block (before
                    ↪  allowing enlarge)
1576                if (SplashKit.RectanglesIntersect(playerEnlargeRec, blockRec))
1577                {
1578                    //if the player is already shrunk, and an enlrage will lead to
                        ↪  a collision
```

```
                    if (_shrunk && (SplashKit.RectangleTop(playerEnlargeRec) <
                    ↪  SplashKit.RectangleBottom(blockRec) ||
                    ↪  SplashKit.RectangleTop(playerEnlargeRec) <
                    ↪  SplashKit.RectangleTop(blockRec)))
                    {
                        return true;
                    }
                }
            }
            return false;
        }
    }
}

///Powerups Family Classes
//Superpower
namespace MarioGame
{
    public abstract class Superpower
    {
        /// <summary>
        /// Name property
        /// </summary>
        public string Name
        {
            get;
            set;
        }

        /// <summary>
        /// Description proprety
        /// </summary>
        public string Description
        {
            get;
            set;
        }

        /// <summary>
        /// abstract use method
        /// </summary>
        /// <param name="p"></param>
        public abstract void Use(Player p);
    }
}

//SuperMagnet
namespace MarioGame
{
    public class SuperMagnet : Superpower
    {
        /// <summary>
        /// default constructor, sets the name and description of the superpower
```

```
1629            /// </summary>
1630            /// <param name="name"></param>
1631            public SuperMagnet(string name)
1632            {
1633                Name = name;
1634                Description = "YOU HAVE THE POWER TO HANG FROM THE MAGNETIC RAIL USING
                    ↪  YOUR MAGNETIC HELMET\n" +
1635                    "TO USE YOUR POWER, PRESS AND HOLD CTRL (LEFT OR RIGHT) AND YOU CAN
                        ↪  MOVE FORWARD/BACKWARDS USING THE ARROWS";
1636            }
1637
1638            /// <summary>
1639            /// overridden Use method
1640            /// </summary>
1641            /// <param name="p"></param>
1642            public override void Use(Player p)
1643            {
1644                if (!p.Attract)
1645                {
1646                    p.Attract = true; //sets p.Attract to true when this method is
                        ↪  called
1647                }
1648            }
1649        }
1650    }
1651
1652    //SuperVoice
1653    namespace MarioGame
1654    {
1655        public class SuperVoice : Superpower
1656        {
1657            /// <summary>
1658            /// default constructor, sets the name and description of the superpower
1659            /// </summary>
1660            /// <param name="name"></param>
1661            public SuperVoice(string name)
1662            {
1663                Name = name;
1664                Description = "YOU HAVE THE POWER TO BREAK GLASS USING YOUR BEAUTIFUL
                    ↪  OPERA VOICE\n" +
1665                    "TO USE YOUR POWER, PRESS AND HOLD CTRL (LEFT OR RIGHT) AS YOU
                        ↪  APPROACH THE GLASS DOOR TO BREAK IT";
1666            }
1667
1668            /// <summary>
1669            /// overridden Use method
1670            /// </summary>
1671            /// <param name="p"></param>
1672            public override void Use(Player p)
1673            {
1674                if (!p.Sing)
1675                {
1676                    p.Sing = true; //sets p.Sing to true when this method is called
```

```
            }
        }
    }
}


//SuperJump
namespace MarioGame
{
    public class SuperJump : Superpower
    {
        /// <summary>
        /// SuperJump default constructor, sets the name and description of the superpower
        /// </summary>
        /// <param name="name"></param>
        public SuperJump(string name)
        {
            Name = name;
            Description = "YOU HAVE THE POWER TO SUPER JUMP\nTO USE THE SUPER JUMP, PRESS SPACE + CTRL (LEFT OR RIGHT)" +
                "\nTHIS WILL ALLOW YOU TO REACH THE HIGHER BLOCKS";
        }

        /// <summary>
        /// overridden Use method
        /// </summary>
        /// <param name="p"></param>
        public override void Use(Player p)
        {
            p.DY = -1.2; //changes the vertical speed of the player to be -1.2 (higher jump)
        }
    }
}
```

Object Oriented Programming – 6.4HD Task

Super Mario 29.11 - Design Report[1]

By Marella Morad – 103076428

[1] *All rights reserved to the original Super Mario Bros game. This game has not been developed to be published, only to demonstrate OOP and C# programming skills.*

Table of Contents

Overview:

Super Mario 29.11 is an adventure game, inspired by Super Mario Bros, initially released in 1983 (Super Mario Bros 2021), in combination with some features inspired by a game called, It Takes Two, which is also an adventure game, published in 2021 (Electronic Arts 2021).

Super Mario 29.11 features one main character, Mario, whose role is to avoid and get past all the obstacles in the level to get the key to the next level, which he can then use to open the door to the next level. This game consists of three levels, and as the game progresses, the levels get harders. However, the new feature is that Mario will be given a new superpower (powerup) for each of the levels that will help him finish the level successfully.

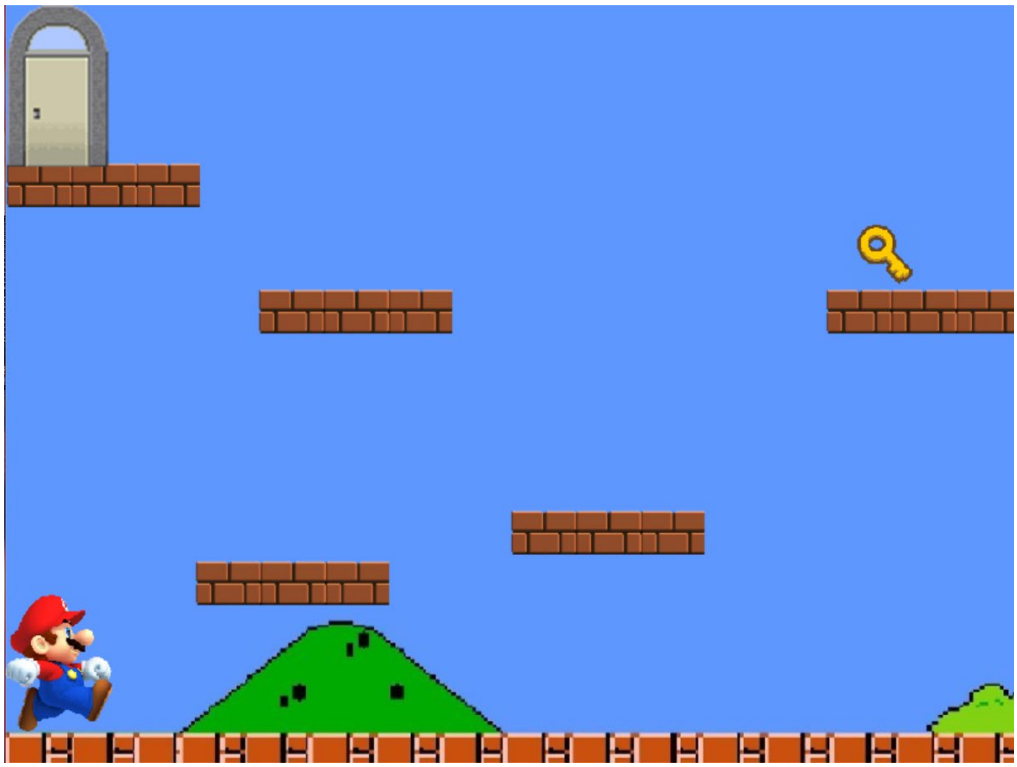
Obstacles in the game include, high normal blocks, lava blocks, spiked blocks and glass blocks. On the other hand, Mario will have three superpowers, corresponding to the three levels as follows:

1. SuperJump in Level 1
2. SuperMagnet in Level 2
3. SuperVoice in Level 3

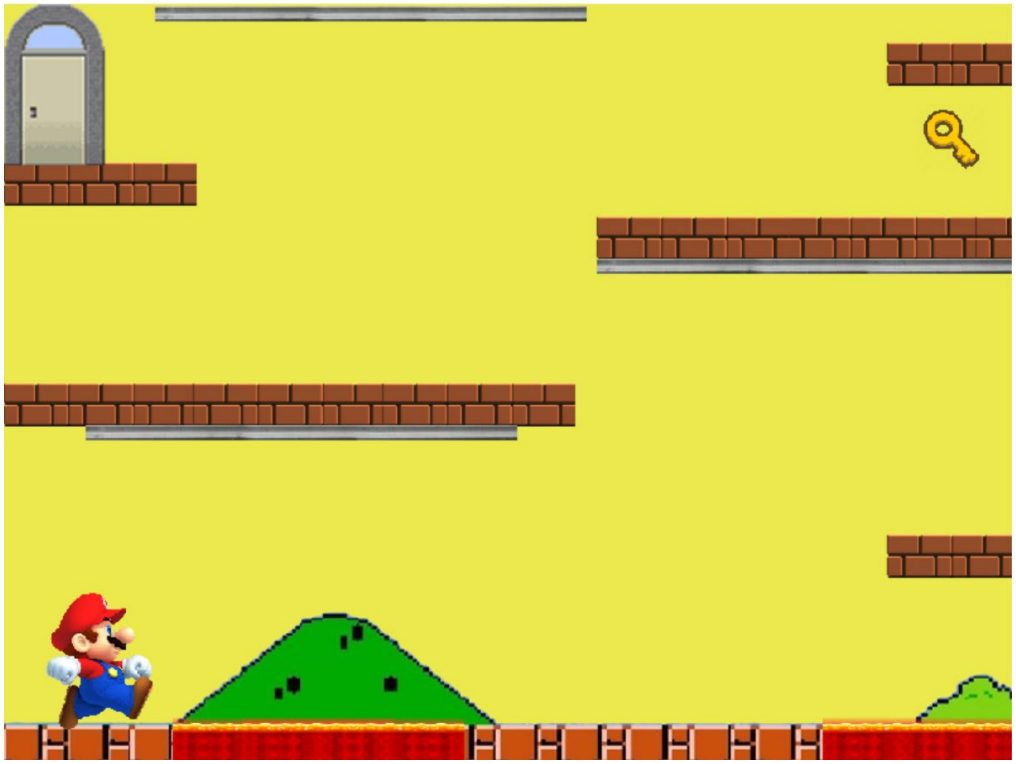
Level Appearance:

Start Window:

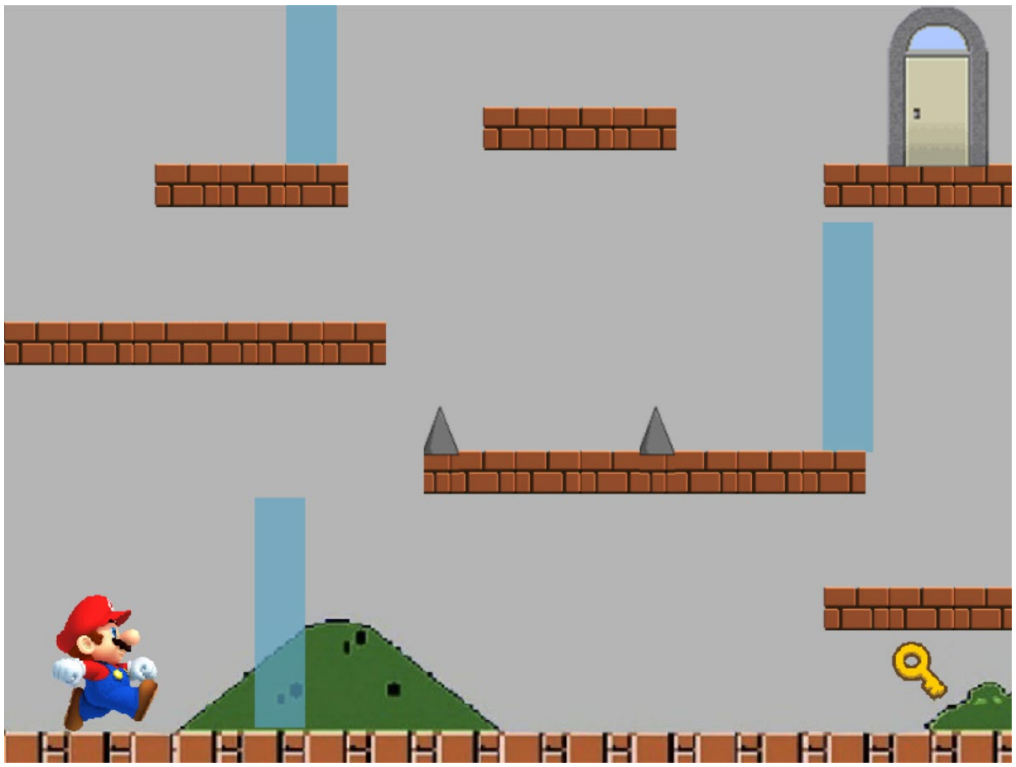
Level One:



Level Two:



Level Three:



Exit/Finish Window:

Classes and Interfaces List:

Table 1 Classes and Interfaces list, including responsibilities

Name	Type	Responsibility
Inheritance and Polymorphism		
Block	Abstract Class	Generalised form of the different types of blocks in the game (polymorphism)
Glass Block	Class	All these classes inherit from the Block class. Their responsibilities differ based on the collision they have with the player (explained later)
Lava Block	Class	
Magnetic Block	Class	
Normal Block	Class	
Spiked Block	Class	
Following the Strategy Design Pattern		
CollisionProcessor	Class	Responsible for running the two types of collisions on each of the blocks in a level
HorizontalCollision	Class	Checks for horizontal collision between the player and a block using their bounding rectangles, and an intersection rectangle for adjustments (left and right collisions only)
		Performs different functionality when the block is "glass", as it works with the superpower (SuperVoice) to break the glass block when the player collides with it having used his superpower.
Vertical Collision	Class	Checks for vertical collision between the player and a block using their bounding rectangles, and an intersection rectangle for adjustments (top and bottom collisions only)
		For bottom collisions (player lands on a block), the player is permitted to land on normal and glass blocks only. The player will fall if he tries to land on a magnetic block and will die (Reset) if he lands on a lava or spiked block. For top collisions (player's head collides with a block), the player can hang from a magnetic block, only in level 2, given that he is using his superpower (SuperMagnet)
ICollision	Interface	Provides the collision check method to be used in CollisionProcessor, Horizontal and Vertical Collision classes
Door	Class	The door of a level (three doors in total in the game). Is responsible for checking if the player has ArrivedAtDoor, by checking intersections between the player and the door's bounding rectangles and if the player HasKey. It is also responsible for setting the next level and next superpower (SetLevel), which are

		passed to it as parameters when the door is created (using its third constructor)
Following the Singleton Design Pattern		
Game	Class	Responsible for the construction of the levels, CreateGame() and running the functionalities based on the user's keyboards input
Inheritance and Polymorphism		
GameObject	Abstract Class	Most generalised object type in the game, child classes inherit properties (Name, X and Y) from it
IDrawable	Interface	Provides the ability for the object to be drawn when implemented by implementing its Bitmap property, and its Draw() void method
Key	Class	The key of a level (three in total), responsible for taking key functionality, which is achieved when the player collides with the key (turns the player's HasKey property to True), as well as the appearance of the key (once the key is taken by the player, it disappears from the level)
Level	Class	One of the major classes in the game, it is responsible for drawing the blocks (added to it from game class), drawing the key and the door (if not null, i.e. not the start and finish windows). It also runs the CollisionResponder functionality, which runs PlayerBlockCollision for each of the blocks in the level (including both vertical and horizontal collisions), along with key and door functionalities.
Direction	Class	Purely for display purposes, it checks the direction the player is facing, and displays the corresponding Bitmap
Player	Class	The biggest and most important class of the game, it handles the player's movement (left, right and jumping). It uses other classes to perform changes on the player (such as direction and resize classes). It is also responsible for displaying level specific instructions for the player in console window.
Resize	Class	Uses Direction class to identify the direction the player is facing, and also gets the Bitmaps from it to display the correct bitmap when the player requests to Enlarge and/or Shrink. It is also responsible for blocking enlarge if it will cause a collision between a block and the enlarged Mario Bitmap.
Inheritance and Polymorphism		
Superpower	Abstract Class	Provides the Name and Description properties for the three superpowers, and also the abstract Use method which is overridden in all of the three child classes
SuperJump	Class	Gives the player the ability to jump higher by changing the player's vertical speed (DY)

SuperVoice	Class	Give the player the ability to break glass using his voice (by setting the player's Sing field to true)
SuperMagnet	Class	Gives the player the ability to hang from a magnetic block (by setting the player's Attract field to true)
All three superpowers are used when Ctrl button is pressed (input handled in Game class)		

UML Class Diagram

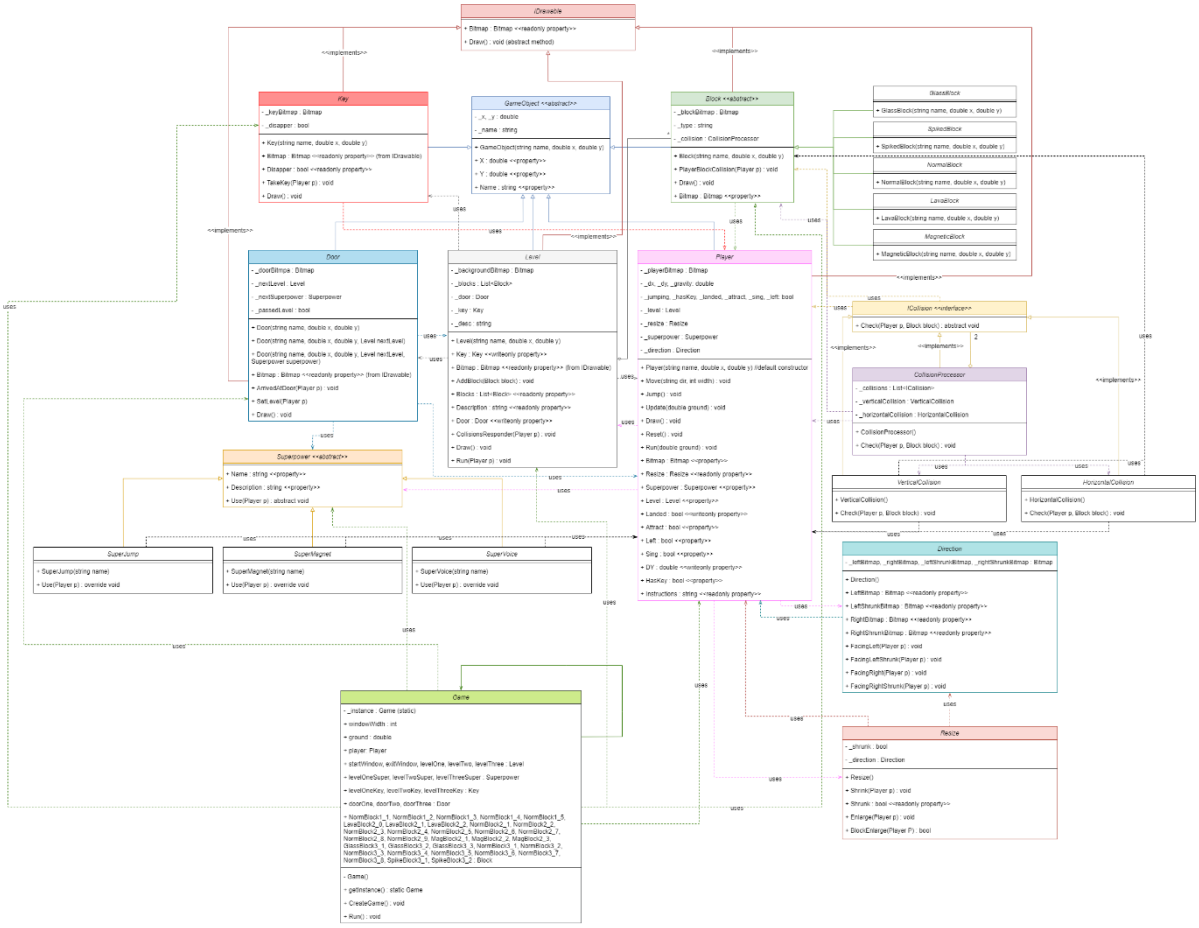


Figure 1 UML Class diagram with all associations marked and colour coded for clarity

Please note: The main UML Class diagram will be submitted separately, with only the main associations displayed (some uses associations will be removed for clarity)

Main functions descriptions

The functions will be explained in order of playing the game (Level 1 to Level 3)

Moving the Player Left and Right

The main method responsible for moving the player, is the Move(string dir, int windowWidth) method contained in the player class. Figure 2 shows a sequence diagram of how the player moves to the left, same sequence is followed when the player moves to the right

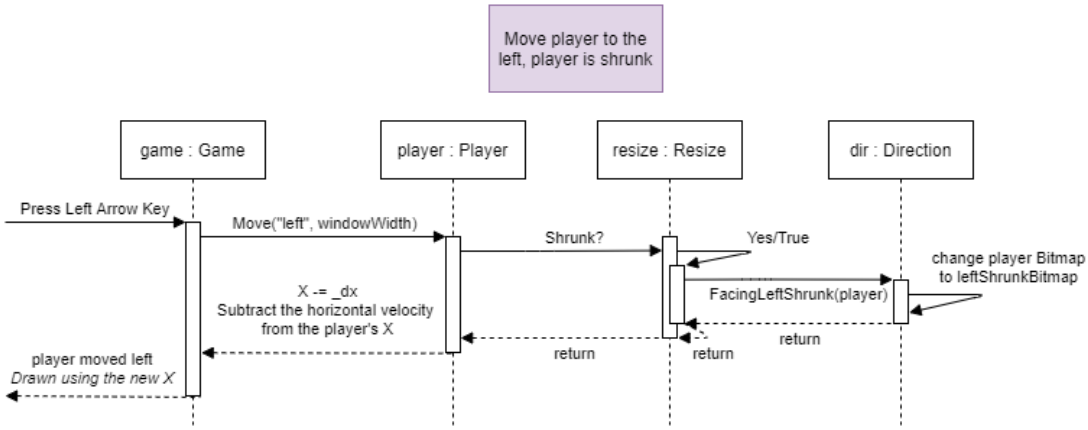


Figure 2 UML Sequence diagram of the move command (case: move player left, player is shrunk)

An additional check is in place, to check if the player is going beyond the game borders (which is why I pass in the window width as a parameter for the move method). Initially my design had both the jump and the move functionalities under one method, however I decided to break them down into two methods to make the methods more specific.

Jumping

This functionality required the use of physics to be achieved as the player undergoes projectile motion when jumping. Therefore, after some research, I was able to implement physics in my game. I used three fields that are responsible for creating a realistic jump:

- _jumping Boolean field: is responsible for only allowing one jump at a time
- _dy double field: is responsible for changing the player's Y coordinate
- _gravity double field: is responsible for changing the DY of the player, ultimately affecting the player's Y coordinate

These three fields work together inside two methods, as shown in Figure 3

- Jump() void method: called in Game class when the player presses space or up
- Update(double ground) void method: called continuously in Game Run() method

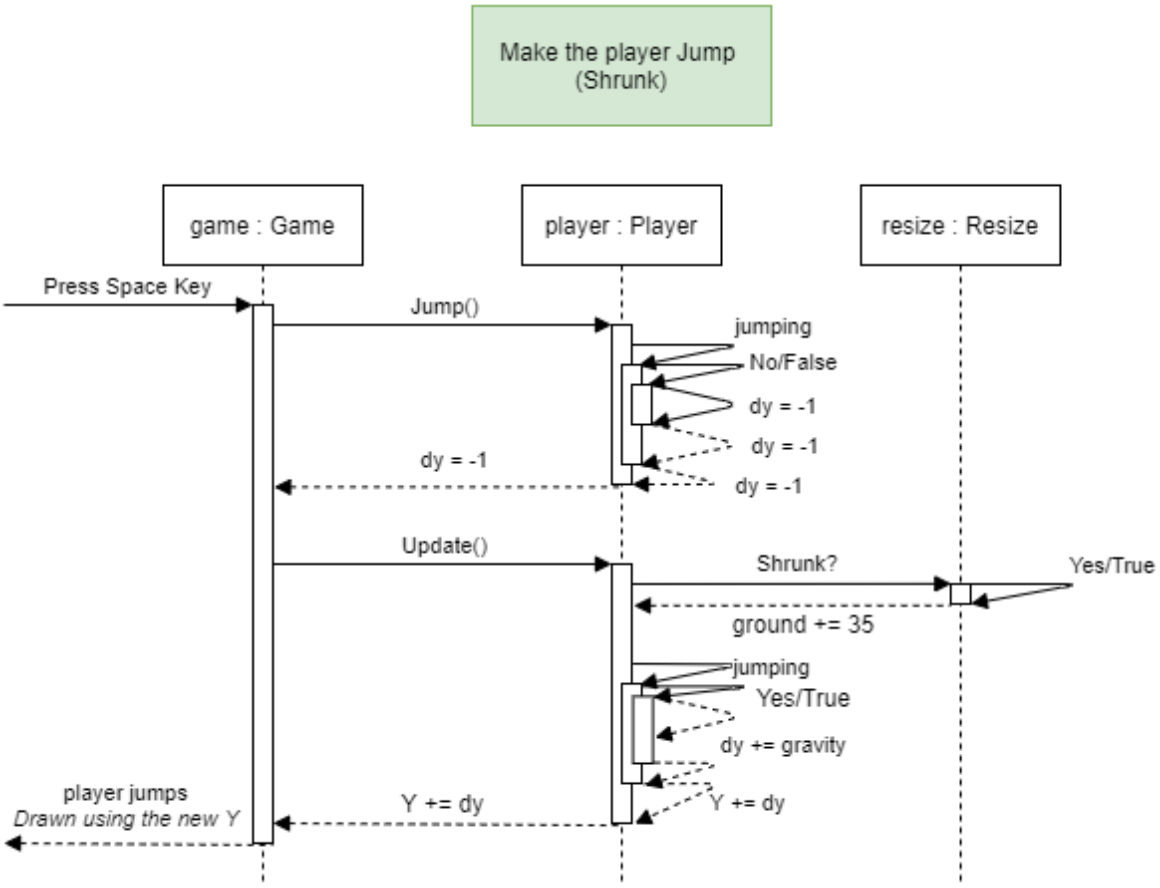


Figure 3 UML Sequence Diagram of the Jumping functionality (case: make the player jump, player is shrunk)

Collisions with Blocks

Collisions in this game are implemented using the strategy design pattern, since the two collisions only differ in their behaviour which is defined in the check method in each of the classes (implemented from the ICollision interface). The collisions are processed in the CollisionProcessor class which calls the Check methods for both the vertical and horizontal classes.

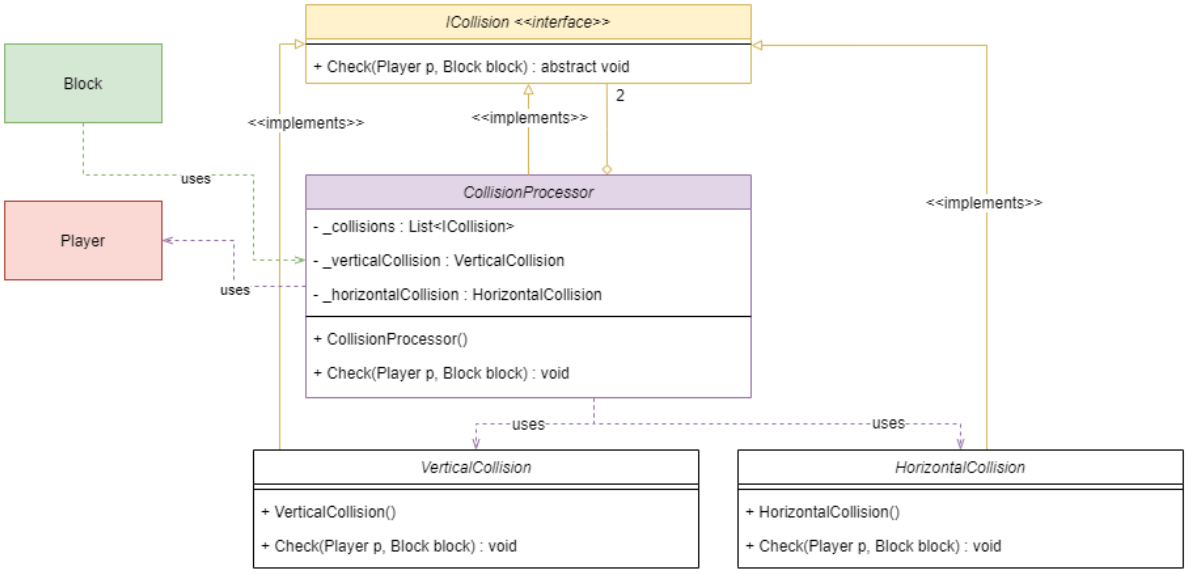


Figure 4 UML Class diagram describing collisions functionality

One of the player's basic collisions is which a normal block. If the plyer collides with a normal block either horizontally or vertically, it will be adjusted based on the intersection rectangle created using the Intersection function from Splashkit library, which returns a rectangle that represents the intersection of two rectangles.

Table 2 Snippet from the Check() method from HorizontalCollision class showing the logic of bounding rectangles intersection and player adjustment

```
public void Check(Player p, Block block)
{
    Rectangle playerRec = p.Bitmap.BoundingRectangle(p.X, p.Y); //getting the
bounding rectangle of the player
    Rectangle blockRec = block.Bitmap.BoundingRectangle(block.X, block.Y); //get
the bounding rectangle of the block
    Rectangle intersection = SplashKit.Intersection(playerRec, blockRec); //get
the intersection rectangle and a save it as a rectangle
    if (intersection.Height > intersection.Width) //horizontal collision
    {
        //Checking player intersection with the block from the LHS
        if (SplashKit.RectangleRight(playerRec) >
SplashKit.RectangleLeft(blockRec) && SplashKit.RectangleRight(playerRec) <
SplashKit.RectangleRight(blockRec))
        {
            p.X -= intersection.Width; //subtracting the intersection width from
the player's X coordinate
        }
        //Checking player intersection with the block from the RHS
        else
        {
            p.X += intersection.Width; //adding the intersection width to the
player's X coordinate
        }
    }
}
```

In a more simplified way, the adjustment of the player's coordinates is done by adding or subtracting the width and height of the intersection rectangle from the player's X and Y coordinates respectively as shown in Figure 5.

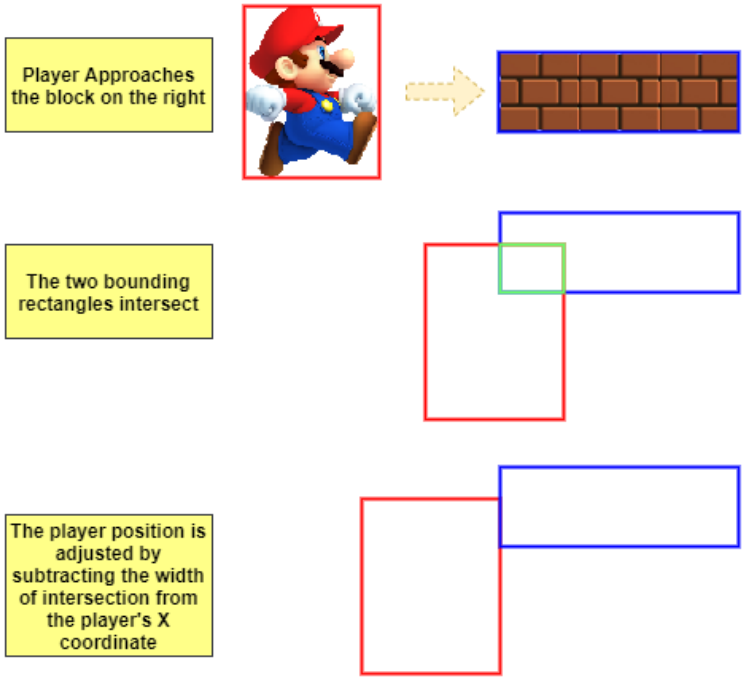


Figure 5 Collision Adjustment steps

Another main player collision is with a resetting block (i.e. Lava or Spiked Block). If the player collides with one of these blocks vertically from the bottom (player's feet collide with the block), the player is reset to its default position and appearance (using the Reset method in Player class), see Figure 6 for more details.

The same sequence is followed when the block type is "spiked"

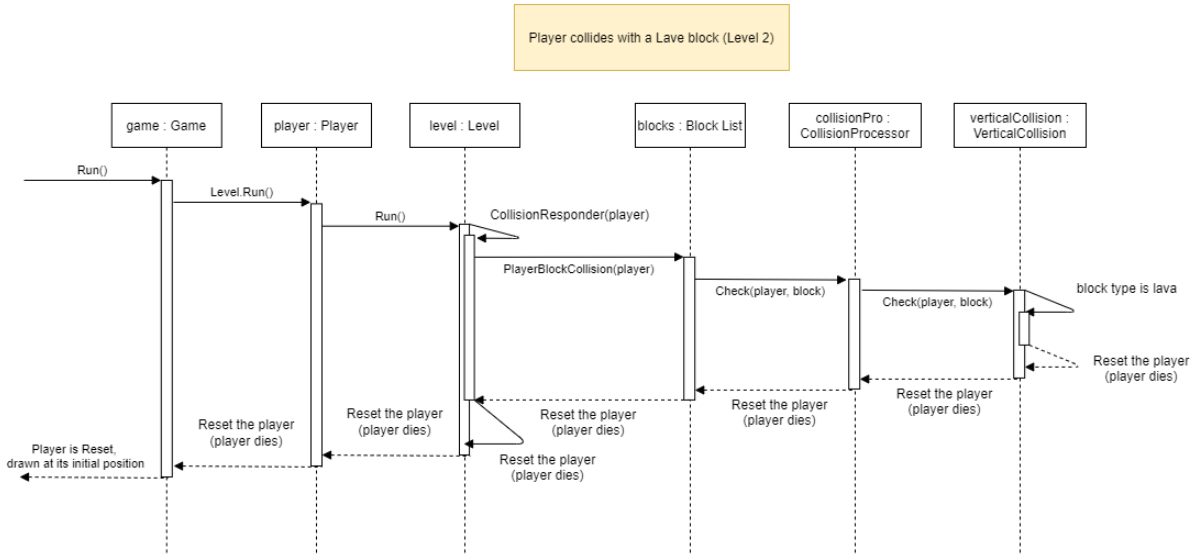


Figure 6 UML Sequence diagram of player collision (case: player collides vertically with a lava block)

Player Resize (Enlarge and Shrink)

I added the enlarge and shrink functionalities, to allow for added complexity of the game, as well as give the user a bigger control over its player. After adding the enlarge and shrink functionalities, I created some situations throughout the game that will require the player to be shrunk to achieve them. One of these situations is shown in Figure 7.

Figure 7 A situation that requires Mario to be shrunk

In some other cases, the player's enlarging will lead to a collision with a block above it, and so, the Resize class has a method to block enlarging in such cases. Figure 9 shows the sequence of calls to deny a player's enlarge request.

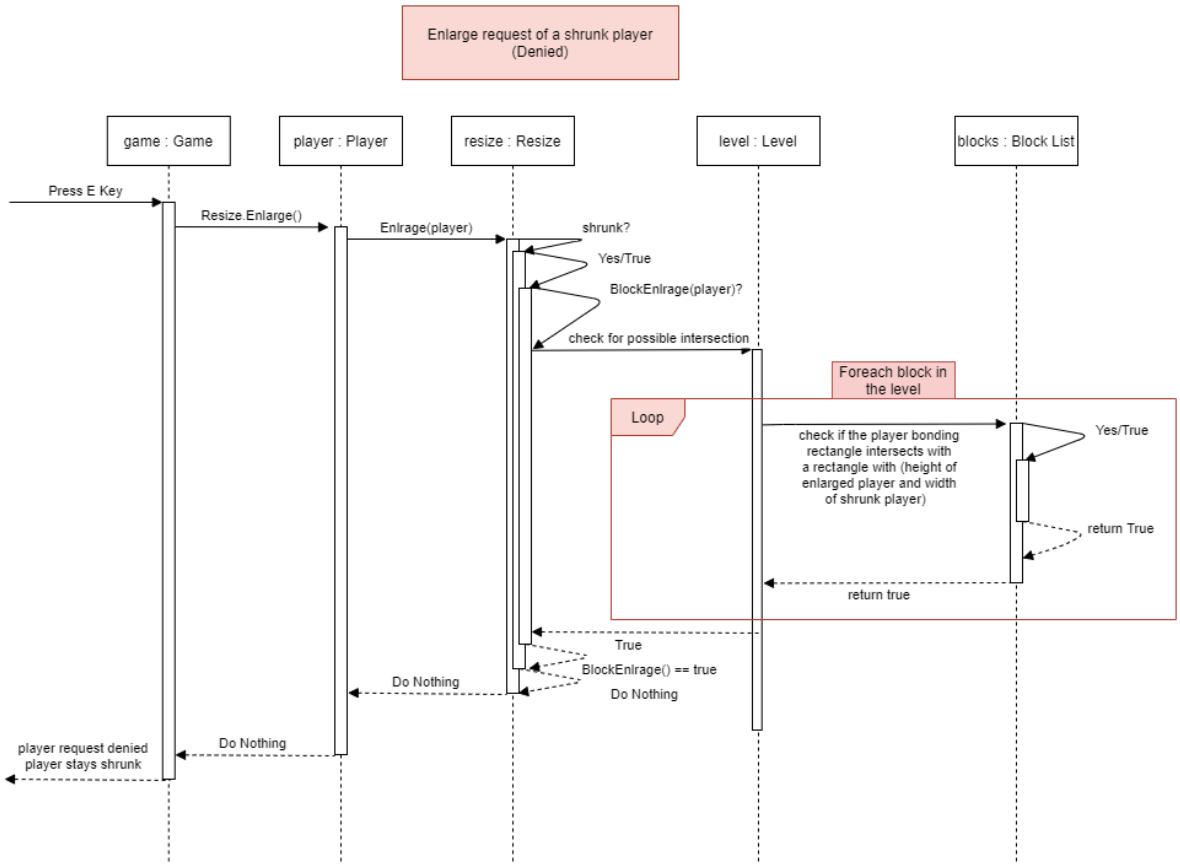


Figure 8 UML Sequence Diagram of Resizing request (case: player enlarge request is denied)

Door and Key Interactions

For the player to move to the next level, he needs to collect the key `key.TakeKey()` and get to the door to open it `door.ArrivedAtDoor()` and `door.SetLevel()` as shown in Figure 9.

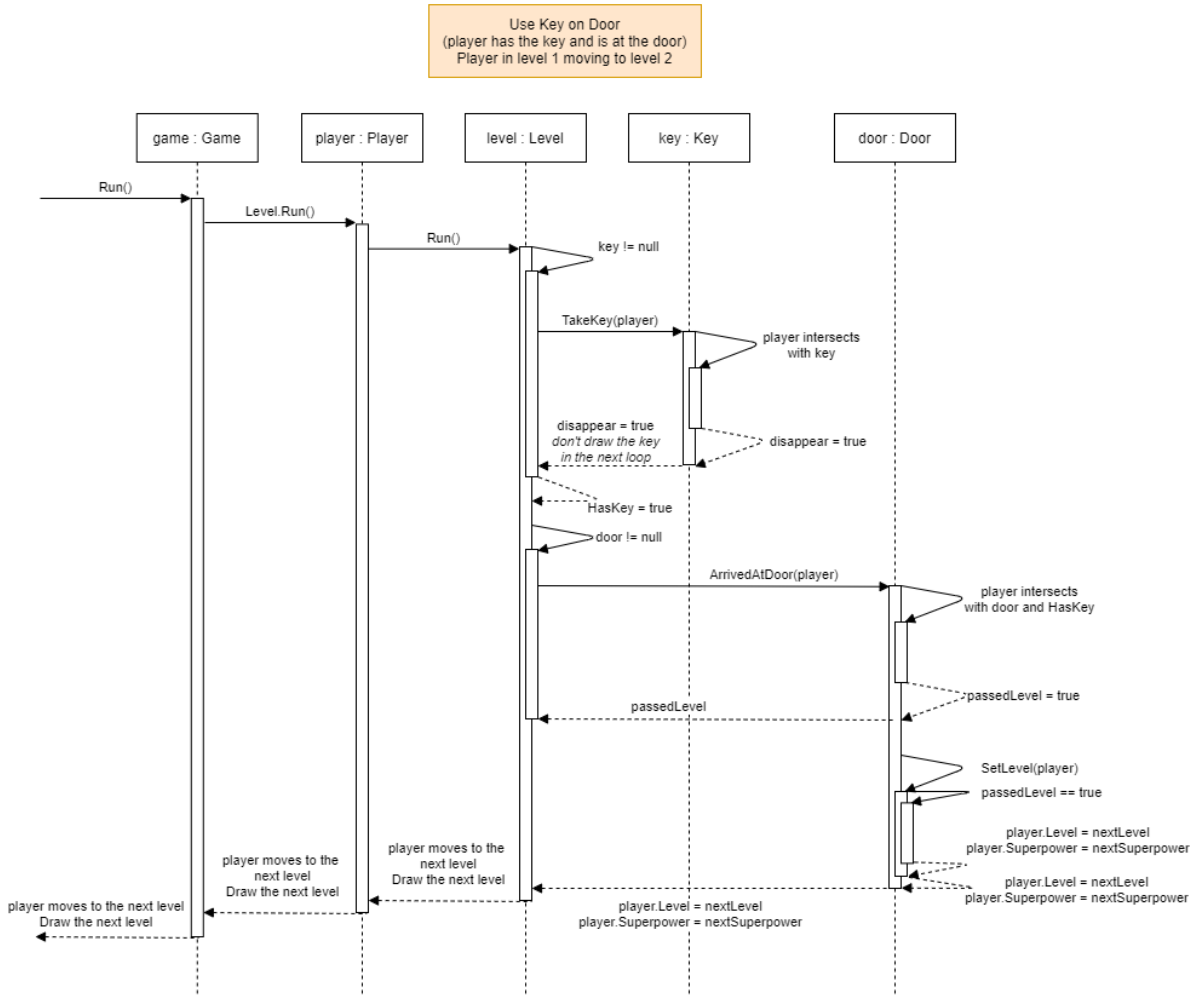


Figure 9 UML Sequence Diagram of using the key to open the door (case: player has the key and is at the door)

Superpowers

As mentioned earlier, the player is provided with a superpower as soon as he enters the level. I will show the sequence diagram for the using the SuperMagnet and the other two superpowers follow the same sequence.

SuperJump in Level 1

This superpower modifies the player's vertical speed, i.e., dy, by enlarging its magnitude (changing it from -1 to -1.2). This gives the player the ability to jump higher to reach blocks that using a normal jump is not sufficient.

SuperMagnet in Level 2

This superpower modifies the player's Attract property and sets it to true when called. The sequence of steps for using the SuperMagnet is show in Figure 10.

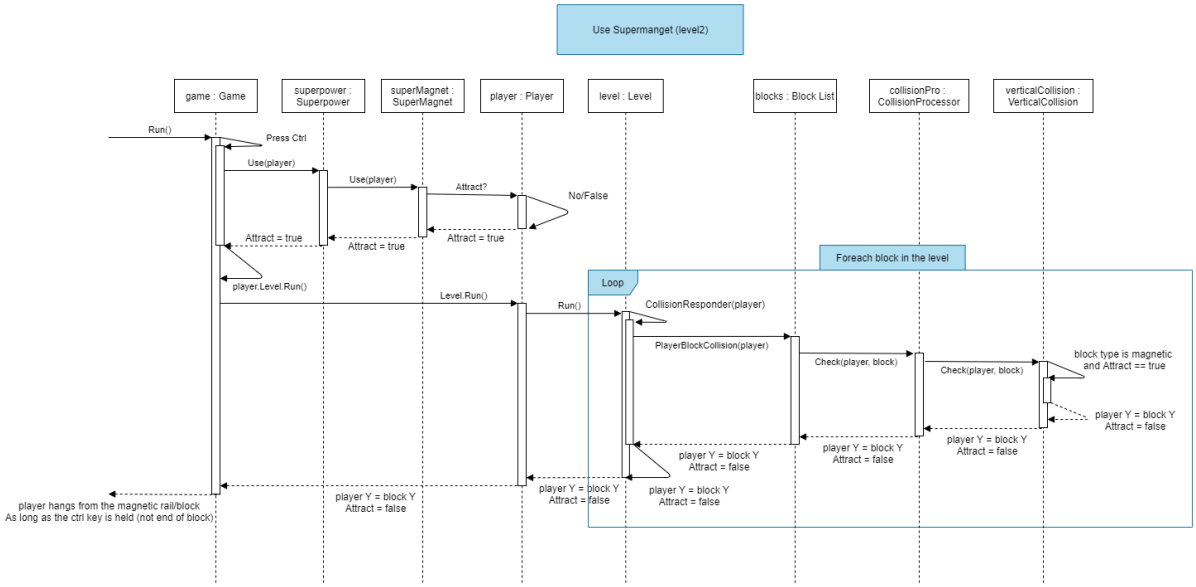


Figure 10 UML Sequence Diagram of using superpowers (case: player uses his super magnet power in Level 2)

SuperVoice in Level 3

This superpower modifies the player's Sing property and sets it to true when called. The sequence of steps for using the SuperVoice is very similar to the one shown in Figure 10.

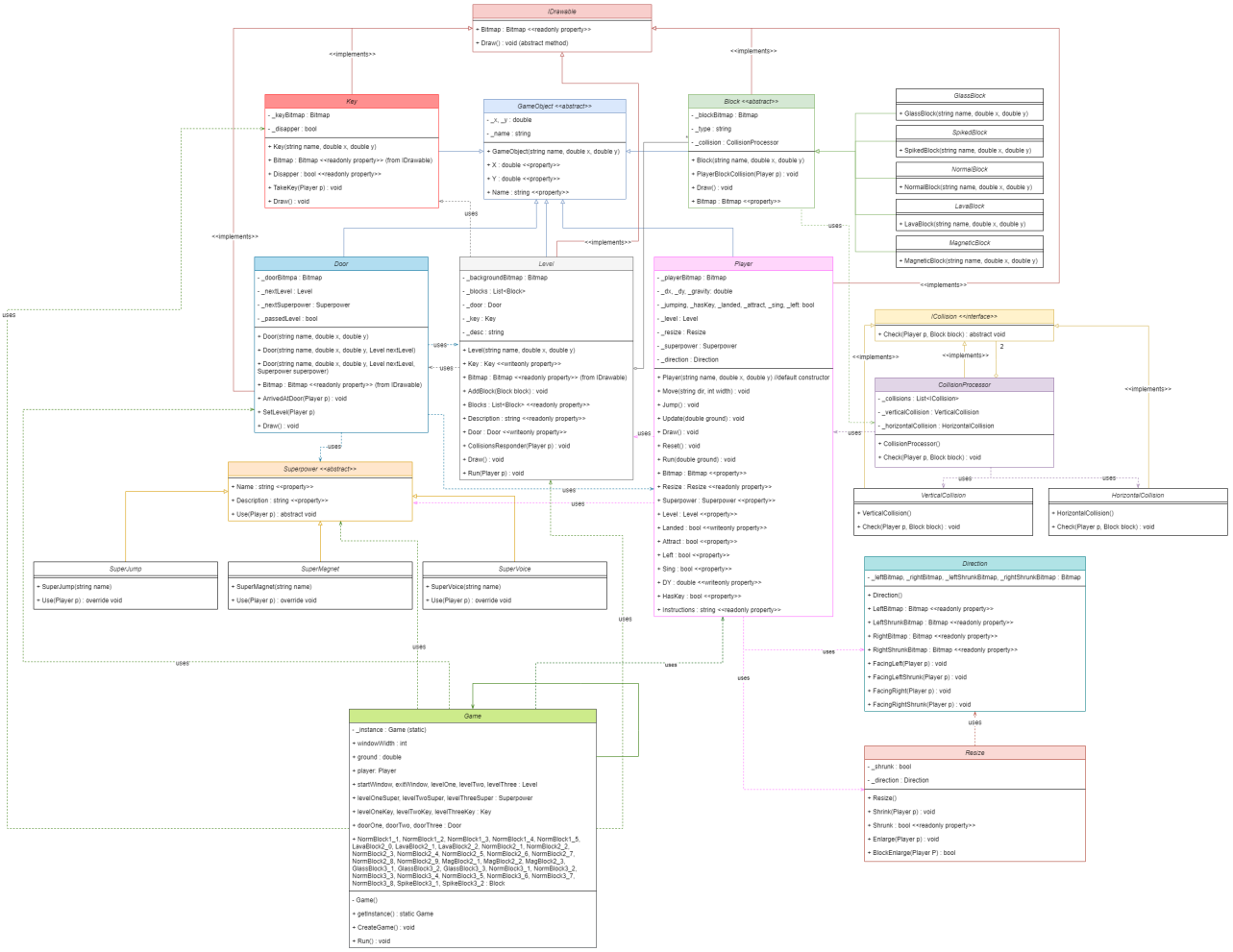
Video Demonstration Link

Game Demonstration YouTube Link

Please note, Mario Bros music has only been added to the video as a background music and is not part of the game.

Full Game ZIP file Link (including images and diagrams link)

OneDrive Game Link


IDrawable
+ Bitmap : Bitmap <<readonly property>>
+ Draw() : void (abstract method)
<<implements>>
Key
- _keyBitmap : Bitmap
- _disappear : bool
+ Key(string name, double x, double y)
+ Bitmap : Bitmap <<readonly property>> (from IDrawable)
+ Disappear : bool <<readonly property>>
+ TakeKey(Player p) : void
+ Draw() : void
GameObject <<abstract>>
- _X, _y : double
- _name : string
+ GameObject(string name, double x, double y)
+ X : double <<property>>
+ Y : double <<property>>
+ Name : string <<property>>
Block <<abstract>>
- _blockBitmap : Bitmap
- _type : string
- _collision : CollisionProcessor
+ Block(string name, double x, double y)
+ PlayerBlockCollision(Player p) : void
+ Draw() : void
+ Bitmap : Bitmap <<property>>
GlassBlock
+ GlassBlock(string name, double x, double y)
SpikedBlock
+ SpikedBlock(string name, double x, double y)
NormalBlock
+ NormalBlock(string name, double x, double y)
LavaBlock
+ LavaBlock(string name, double x, double y)
MagneticBlock
+ MagneticBlock(string name, double x, double y)
uses
Door
- _doorBitmpa : Bitmap
- _nextLevel : Level
- _nextSuperpower : Superpower
- _passedLevel : bool
+ Door(string name, double x, double y)
+ Door(string name, double x, double y, Level nextLevel)
+ Door(string name, double x, double y, Level nextLevel, Superpower superpower)
+ Bitmap : Bitmap <<readonly property>> (from IDrawable)
+ ArrivedAtDoor(Player p) : void
+ SetLevel(Player p)
+ Draw() : void
Level
- _backgroundBitmap : Bitmap
- _blocks : List<Block>
- _door : Door
- _key : Key
- _desc : string
+ Level(string name, double x, double y)
+ Key : Key <<writeonly property>>
+ Bitmap : Bitmap <<readonly property>> (from IDrawable)
+ AddBlock(Block block) : void
+ Blocks : List<Block> <<readonly property>>
+ Description : string <<readonly property>>
+ Door : Door <<writeonly property>>
+ CollisionsResponder(Player p) : void
+ Draw() : void
+ Run(Player p) : void
Player
- _playerBitmap : Bitmap
- _dx, _dy, _gravity: double
- _jumping, _hasKey, _landed, _attract, _sing, _left: bool
- _level : Level
- _resize : Resize
- _superpower : Superpower
- _direction : Direction
+ Player(string name, double x, double y) //default constructor
+ Move(string dir, int width) : void
+ Jump() : void
+ Update(double ground) : void
+ Draw() : void
+ Reset() : void
+ Run(double ground) : void
+ Bitmap : Bitmap <<property>>
+ Resize : Resize <<readonly property>>
+ Superpower : Superpower <<property>>
+ Level : Level <<property>>
+ Landed : bool <<writeonly property>>
+ Attract : bool <<property>>
+ Left : bool <<property>>
+ Sing : bool <<property>>
+ DY : double <<writeonly property>>
+ HasKey : bool <<property>>
+ Instructions : string <<readonly property>>
<<implements>>
ICollision <<interface>>
+ Check(Player p, Block block) : abstract void
2
CollisionProcessor
- _collisions : List<ICollision>
- _verticalCollision : VerticalCollision
- _horizontalCollision : HorizontalCollision
+ CollisionProcessor()
+ Check(Player p, Block block) : void
VerticalCollision
+ VerticalCollision()
+ Check(Player p, Block block) : void
HorizontalCollision
+ HorizontalCollision()
+ Check(Player p, Block block) : void
Superpower <<abstract>>
+ Name : string <<property>>
+ Description : string <<property>>
+ Use(Player p) : abstract void
SuperJump
+ SuperJump(string name)
+ Use(Player p) : override void
SuperMagnet
+ SuperMagnet(string name)
+ Use(Player p) : override void
SuperVoice
+ SuperVoice(string name)
+ Use(Player p) : override void
Direction
- _leftBitmap, _rightBitmap, _leftShrunkBitmap, _rightShrunkBitmap : Bitmap
+ Direction()
+ LeftBitmap : Bitmap <<readonly property>>
+ LeftShrunkBitmap : Bitmap <<readonly property>>
+ RightBitmap : Bitmap <<readonly property>>
+ RightShrunkBitmap : Bitmap <<readonly property>>
+ FacingLeft(Player p) : void
+ FacingLeftShrunk(Player p) : void
+ FacingRight(Player p) : void
+ FacingRightShrunk(Player p) : void
Resize
- _shrunk : bool
- _direction : Direction
+ Resize()
+ Shrink(Player p) : void
+ Shrunk : bool <<readonly property>>
+ Enlarge(Player p) : void
+ BlockEnlarge(Player P) : bool
Game
- _instance : Game (static)
+ windowWidth : int
+ ground : double
+ player: Player
+ startWindow, exitWindow, levelOne, levelTwo, levelThree : Level
+ levelOneSuper, levelTwoSuper, levelThreeSuper : Superpower
+ levelOneKey, levelTwoKey, levelThreeKey : Key
+ doorOne, doorTwo, doorThree : Door
+ NormBlock1_1, NormBlock1_2, NormBlock1_3, NormBlock1_4, NormBlock1_5, LavaBlock2_0, LavaBlock2_1, LavaBlock2_2, NormBlock2_1, NormBlock2_2, NormBlock2_3, NormBlock2_4, NormBlock2_5, NormBlock2_6, NormBlock2_7, NormBlock2_8, NormBlock2_9, MagBlock2_1, MagBlock2_2, MagBlock2_3, GlassBlock3_1, GlassBlock3_2, GlassBlock3_3, NormBlock3_1, NormBlock3_2, NormBlock3_3, NormBlock3_4, NormBlock3_5, NormBlock3_6, NormBlock3_7, NormBlock3_8, SpikeBlock3_1, SpikeBlock3_2 : Block
- Game()
+ getInstance() : static Game
+ CreateGame() : void
+ Run() : void