A Case Study on  
Designing and Developing a Rental Bike System  
for Community

In Partial Fulfillment

of the Requirement of the Subject

**Object-Oriented Programming**

*Submitted by:*

**Guy, Lawrence Adrian B.**

**Surio, Rovil Jr. M.**

**Sustento, Myke Alvin E.**

**Vallarta, Troy Joaquin G.**

*Schedule:*

**THURSDAY / 10:30 A.M to 1:30 P.M / ROOM or Blackboard Ultra**

*Submitted to:*

**Engr. Jordan Vhane D. Sardalla**

Instructor

**Abstract**

**Acknowledgment**

First and foremost, praise and thank God, the Almighty, for His showers of blessings throughout the case study to complete it successfully.

The members would like to express their deep and sincere gratitude to their database design and development lab professor, Engr. Jordan Vhane Sardalla, for teaching us the required knowledge necessary to do this case study. The members would also like to express their appreciation and gratitude to their family and friends for their undying love and support on the member’s endeavors in their computer engineering courses.

May this case study serve as fruition of all the support and encouragement provided by the people in the researcher’s lives.

**Table of Contents**

|  |  |
| --- | --- |
| Abstract ………………………………………………………………………………..... | 2 |
| Acknowledgment ………………………………………………………………………... | 3 |
| The Problem and Its Background ……………………………………………………….. | 6 |
| Background of the Study ………………………………………………………... | 6 |
| Objectives of the Study …………………………………………………………. | 6 |
| Significance of the Study ……………………………………………….………. | 7 |
| Scope and Delimitation …………………………………………………………. | 7 |
| Operational Definition of Terms …………………………………………..……. | 7 |
| Review of Related Literature and Studies……………………………………………….. | 8 |
| Methodology………………………………………………………………………..…… | 10 |
| General Method Used …………………………………………………………... | 10 |
| Procedure ……………………………………………………………………….. | 10 |
| Results and Discussions………………………………………………………….……… | 12 |
| The Developed System …………………………………………………….…… | 12 |
| Verification and Testing Result …………………………………………...……. | 19 |
| Summary of Findings, Conclusions, and Recommendations…………………………… | 29 |
| Summary of Findings …………………………………………………………... | 29 |
| Conclusions ……………………………………………….……….……….…… | 29 |
| Recommendations ……………………………………………….……….……... | 29 |
| Appendices …………………………………………………………………………….... | 31 |
| References ……………………………………………………………………... | 32 |
| Source Code ………………………………………….……….………….…… | 33 |
| Researcher’s Profile …………………………………………….……………… | 43 |

**List of Figures**

|  |  |
| --- | --- |
| Figure 1 – Agile method …………………………………………………….…………… | 10 |
| Figure 2 – Python Main file………………………………………………….…………… | 12 |
| Figure 3 – Python Menu() class…………………………………………………………… | 13 |
| Figure 4 – Python Item() class…………………………..………………….…...………… | 13 |
| Figure 5 – Python Game() class………………………..………………….……………… | 13 |
| Figure 6 – Python Game() class part 2……………………………………….…………… | 14 |
| Figure 7 – Python Gameconsole() class………………………….……….………….…… | 14 |
| Figure 8 – Added clear() function on sell\_cart and buy\_cart after transaction….…..……. | 15 |
| Figure 9 – C# Main and Menu ()…………………………………………….……….…… | 16 |
| Figure 10 – C# BaseDetails class……………………………….………………………… | 16 |
| Figure 11 – C# GameConsole class part 1………………………..………….…………… | 17 |
| Figure 12 – C# GameConsole class part 2……………………………….……………..… | 17 |
| Figure 13 – C# Game class part 1………………………….……….………………..…… | 18 |
| Figure 14 – C# Game class part 2……………………………….……………………...… | 18 |
| Figure 15 – Python Menu Output………………………..……….…………………..…… | 19 |
| Figure 16 – Python 2nd Hand Game Selections…………………………….………..…… | 19 |
| Figure 17 – Python Add to Cart………………………….……….…………………….… | 20 |
| Figure 18 – 18 Python Add to Cart part 2……………………………….……………...… | 20 |
| Figure 19 – Python Input Cashier Money………………………..……….…………….… | 21 |
| Figure 20 – Python Transaction Successful…………………………….………………… | 22 |
| Figure 21 – Python Loop to the Menu………………………….……….………………… | 23 |
| Figure 22 – Python Inclusion of the 2nd Hand Games to the Game Selection……....…… | 24 |
| Figure 23 – Python Transaction with 2nd Hand Game and Brand New Game……...…… | 25 |
| Figure 24 – C# Menu & Sell BrandNew/2ndHand Games………………….……….…… | 26 |
| Figure 25 – C# Buying a game………………………….……….……………………...… | 26 |
| Figure 26 – C# Total and change………………………….……….…………………...… | 27 |
| Figure 27 – C# Buying a game or console…….………………………………………..… | 27 |
| Figure 28 – C# Adding the item into inventory………………………………………...… | 27 |

**Chapter I**

**The Problem and Its Background**

The case study is about creating a game shop application using Python programming and C#. The game shop buys and sells brand new and used game or game consoles. The implementation of object-oriented concepts was observed.

**A. Background of the Study**

Different programming paradigms were developed through time and this considers different concepts to represent elements of programs. There are paradigms created to support another paradigm or to make something faster and accurate. These paradigms are advancing in a particular direction which is to reduce the workload. The object-oriented paradigm is an advanced programming approach that divides a complex system into manageable modules or smaller parts therefore the development process is much more readable or easy to grasp [1].

Object-oriented programming has many features but the four main features or also known as the four pillars of object-oriented programming are inheritance, polymorphism, abstraction, and encapsulation. Moreover, these four main pillars were used in this study to create the game shop application using the two different languages namely Python and C#.

The game shop application was created in order to process a transaction, organize and manage items such as games and game console with ease. With that being said, the study is needed to proffer knowledge about an object-oriented programming approach in the creation of an application that makes the management of items and transaction easier for an employee or user.

**B. Objectives of the Study**

*General Objective:* To create a game shop program.

*Specific Objective:*

1. To create a game shop system program that will be controlled by a cashier.
2. To create a game shop system that lets a customer buy and sell video games and game consoles.
3. To create a game shop system that automatically calculates the change given a specific amount of payment.

**C. Significance of the Study**

This study aims to create a game shop program that would be controlled by a cashier. The program should let a customer buy and sell video games and game consoles and it should also automatically calculate the change given a specific amount of payment. Furthermore, the study could be of importance to the following.

**Cashiers.** The program created from this study should help cashiers have a more efficient and smoother transaction with their customers.

**Customers.** The program created from this study should help customers have a more efficient and smoother transaction with video game shop cashiers.

**Future Researchers.** The ideas presented may be used by future researchers as a reference. This may help them create their own programs or improve the program created by the researchers.

**D. Scope and Delimitation**

The game shop application created mainly covers the basics of object-oriented programming using C# and Python programming but not limited to. A list data structure was used to store the objects or the products while a linear search algorithm was used to specify the object from the list and make a new algorithm out of it. The program does not have a graphical user interface hence it is only running through the console application.

**E. Operational Definition of Terms**

***Python -*** A programming language that was used in the creation of the game shop application. It was utilized to implement the object-oriented programming.

***C# -*** Another programming language that was used in the creation of the game shop application. The implementation of object-oriented programming can also be observed using this programming language.

***Game shop -*** The application created was about the game shop that can sell a game or console to a customer and also buy a game or console from a customer.

***Object-Oriented Programming -*** A programming paradigm that contains a concept of classes and objects. This is used to structure the code block of a program into a simple and organized line of codes.

**Chapter II**

**Review of Related Literature and Studies**

This chapter is designed to enumerate and identify some research related to the current study being developed.

OOP Paradigm

The object-oriented programming paradigm introduces a different approach to problem-solving, and it is by focusing on objects that make up the system rather than focusing on the problem itself. Objects are anything that can be named after the things you want to model within the program and therefore, the main goal of object-oriented programming it to represent or demonstrate the real world in the program. In addition, an object is an instance of a class. For example, in a store that sell games, the objects would be the games or other products within the store [2].

The objects can also be organized into classes and these are the objects that contains a certain characteristic that makes it unique. The classes from the word itself groups the objects that have similar characteristics with each other but different with other classes. Each class therefore contains an attributes that unique from other groups. Moreover, classes may be grouped into different types such as a super class which acts like a main class and a subclass which is a subset of the super class. An inheritance is the concept that passed the properties/attributes of a class into another class. With inheritance, the other created object or class can automatically have the properties/attributes of a superior object or class without needing to rewrite the code again hence it also promotes the reusability of the code within the program [3].

Polymorphism is one of the four pillars of OOP and this allows the usage of a method in a different interpretation depending on what the user wants. It basically gives the ability to redefine a method within the other class or derived class [4]. For example, in the game shop application created, the Createlist() method from an abstract class Basedetails was redefined to create a different list of items on the game and game console class.

Objects can be encapsulated meaning that the specific attributes of an object are hidden. “Encapsulation is the act of grouping data, and operations that affect the data, into a single objects, however, the content of the object could be visible to other objects” [5]. Moreover, the data that encapsulated can only be accessed by the functions that are permitted and defined. With that being said, the main purpose of encapsulation is to help in securing the data within the system.

One of the fundamentals in object-oriented paradigm is the abstraction. “Abstraction is the purposeful suppression, or hiding, of some details of a process or artifact, in order to bring out more clearly aspects, details, or structure” [6]. To understand it more clearly, think about when logging in any website such as Facebook or Twitter, after inputting all the necessary data and logging in, the process on how it is being verified by the system is hidden or abstracted to you.

The gathered related studies are all about the object-oriented paradigm, and it’s four main pillars. In this study, the object-oriented programming was used in order to develop an application that will help the user or employee to process and manage the things needed to run a small game shop and this study is different from the gathered related studies in a way that this focuses on creating the game shop application rather than discussing the object-oriented paradigm.

**Chapter III**

**Methodology**

The methodology and procedures used in designing and developing the project are chosen to consider the factors such as adaptability and flexibility. This chapter will primarily discuss the reason why the group decided to make use of the chosen methodology and procedures.

**A. General Method Used**

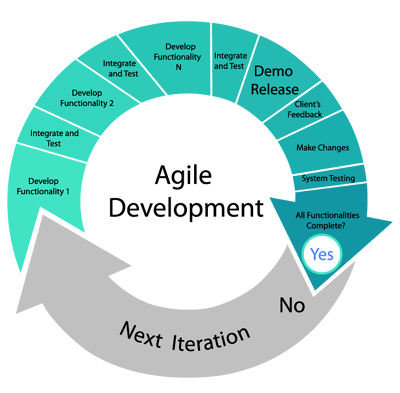


Figure 1 Agile method

The agile methodology was used since this method offers flexibility and adaptability. The group utilized making iterations for the development of the project to accommodate change and this makes the project more optimized since all files were being checked again per iterations. Overall, the program created using Python and C# goes to multiple testing until the desired output was achieved.

**B. Procedure**

**1. Design Phase**

The researchers planned before creating the program that it will be split evenly by the members; therefore, the C# language will be created by Rovil Surio Jr. and Troy Joaquin Vallarta and the python language of the program will be created by Adrian Lawrence Guy and Myke Alvin Sustento. The researchers planned the distribution of tasks evenly and everyone agreed with that idea.

Before creating the program, the researchers need a design phase to provide a direction, foundation, and blueprint for the flow of production of the program. First, the researchers created the UML diagram to give the team unity in implementing the program. The team needs the same wavelength of ideas to make progress efficiently, therefore the UML diagrams are needed in this situation especially when the researchers are not in the same room. The first UML diagram that was created is the sequence program; it provides a sequence or flow of a program that gives the researchers not having a misunderstanding when collaborating with a program. The next UML diagram is the use-case diagram it gives the researchers an idea of who is going to use the program and the process inside the program. Lastly, is the class diagram, which focuses mostly on creating the class of the program and the relationship of other classes, this provides a more in-depth representation of the codes. Therefore, this diagram is used before the codes are created.

**2. Implementation Phase**

After the design phase of the program, the researchers implemented the four pillars of OOP to apply all the lessons that the researchers learned. The first pillar implemented is encapsulation, which means bundling the data with the method, it is also used to hide values inside the class. The researchers implemented this pillar in creating the details of the game/ console such as title, platform, condition, and price. The next pillar implemented is Abstraction because encapsulation and abstraction go hand in hand to each other. This means that when you encapsulate the data, the data will become an abstraction because it hides the data. The next pillar created is the Inheritance, the researcher created the inheritance by inheriting classes into the main file of the researchers both in C# and Python. The last pillar that is created in this program is the polymorphism, after inheriting the data classes from another file, the researchers add some file to complement the needed data in the program, the difference that can be seen in this program is the game and console, which it has the platform for the game file.

**Chapter IV**

**Results and Discussions**

This chapter will discuss the developed output and the results of the tests discussed in the previous chapter, describing how the code was created and the methods that were used in order to make the application with regard to the objective of the case study. The functions were shown as well as its output to insure the reliability and consistency of its output.

**A. The Developed System**

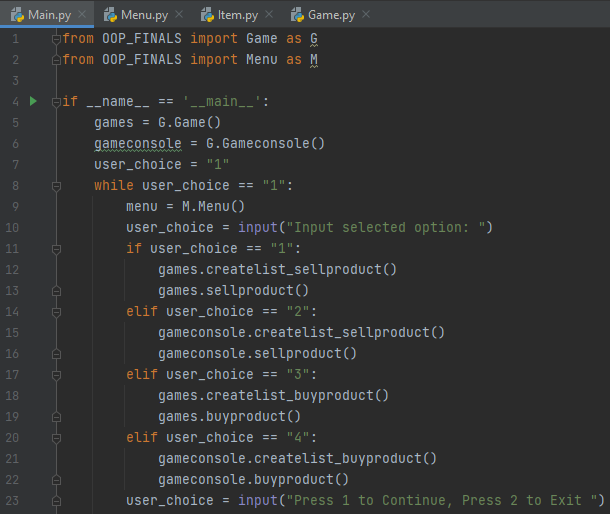
****

Figure 2 Python Main file

Figure 2 shows the main file for the Python code. The Main.py contains the main menu of the Python code. In the tabs, the different program files can be seen. The Game.py and Menu.py was imported into the main file so it can use its classes and methods. In this file, the 4 choices can be called and those 4 choices can be seen in the Menu.py file.

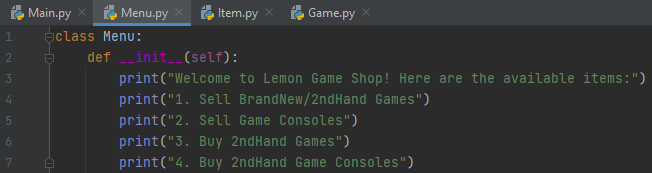


Figure 3 Python Menu() class

Figure 3 shows the Menu that will be called in the Main.py file. This class was created to promote a clean code in the Main.py file.

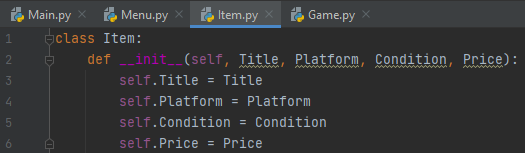


Figure 4 Python Item() class

Figure 4 shows the Item class. This item class was imported to Game.py to set the value of the products, game and game console.

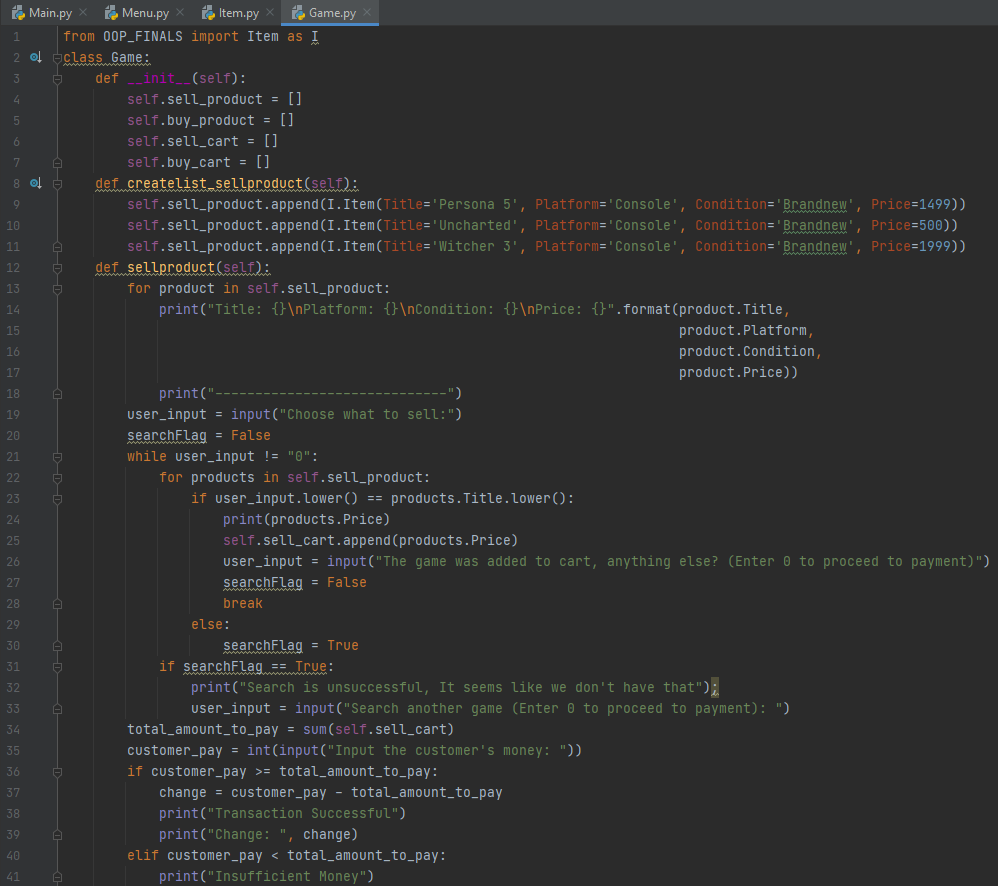
Figure 5 Python Game() class

Figure 5 shows the Game class. The Game() class contains the constructor which has four attributes called self.sell\_product, self.buy\_product, self.sell\_cart and self.buy\_cart which are used to store the games being sold and store the total amount to be paid by the customer or the cashier. It also contains the createlist\_sellproduct() method which adds a game to the self.sell\_product list. Lastly, it contains the sellproduct() method which is the code used to sell games to the customers and calculate change given a payment input.

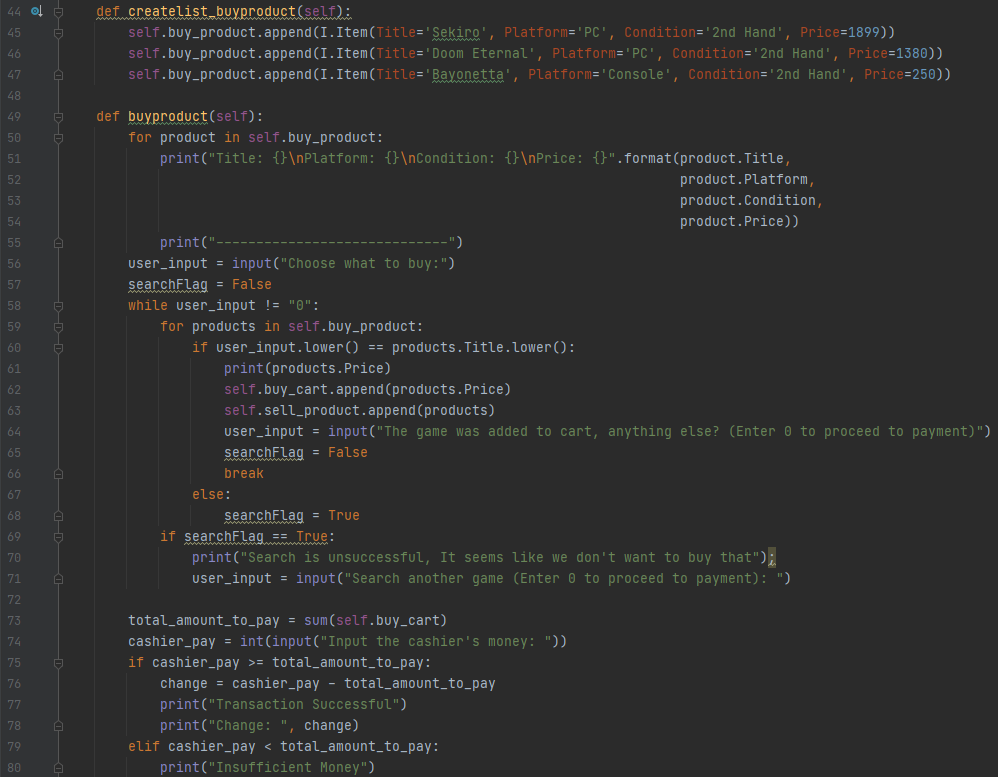


Figure 6 Python Game() class part 2

Figure 6 shows the continuation of the Game class. It contains the createlist\_buyproduct() and buyproduct() which is essentially the same as the createlist\_sellproduct() and sellproduct() methods but they are created to buy 2nd hand games from the customer.

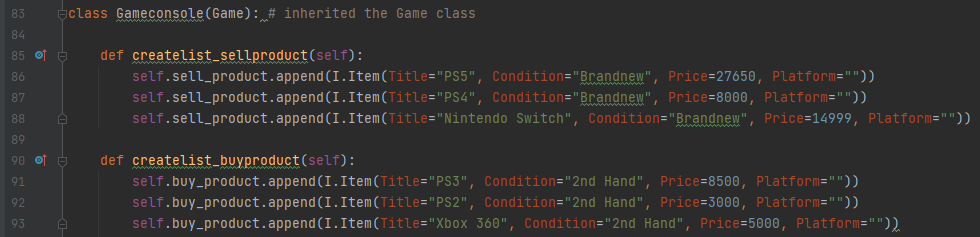
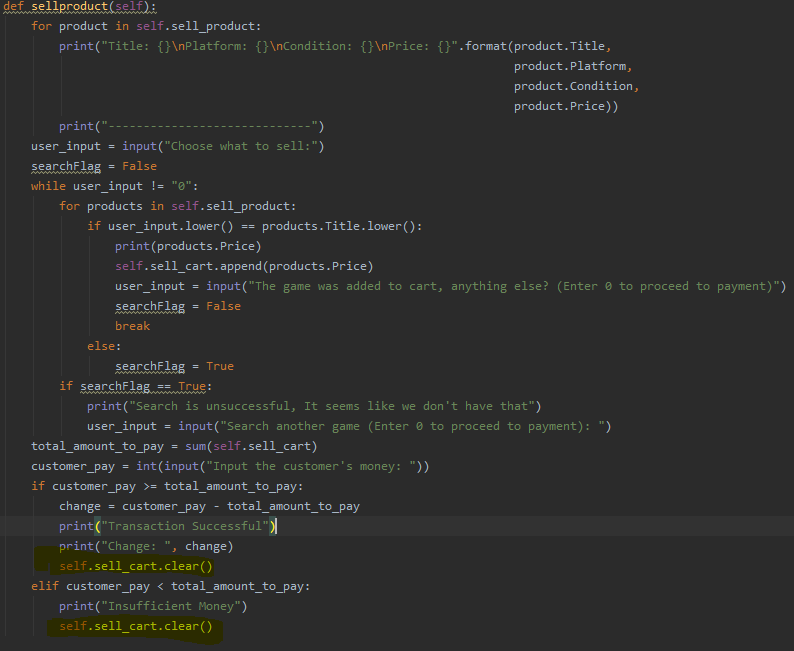


Figure 7 Python Gameconsole() class

Since the Gameconsole() class mostly uses the same methods as the Game() class, it is now considered a child class of the parent class, Game(). In order to specify that this class is selling and buying consoles instead of games, the createlist\_sellproduct() and createlist\_buyproduct() method was polymorphed to have different appended items.



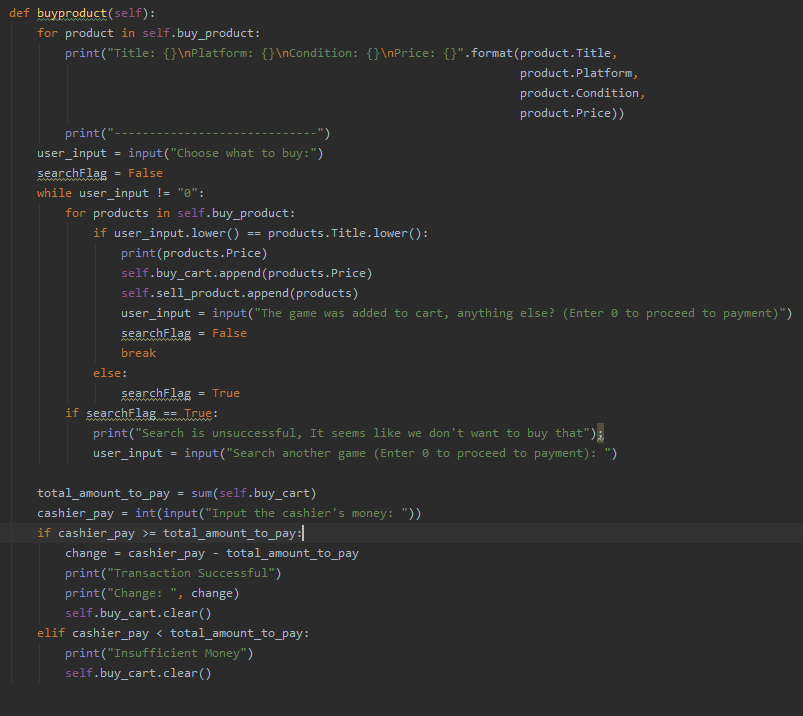


Figure 8 Added clear() function on sell\_cart and buy\_cart after transaction.

Added a clear function on self.sell\_cart and self.buy\_cart for each transaction so that the item appended in the cart will not duplicate after the next transaction.

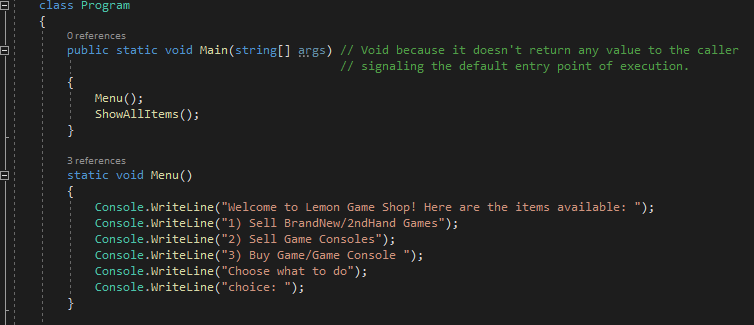


Figure 9 C# Main and Menu ()

Figure 9 Shows the Main and Menu method of the game shop console application created using C# programming language. The Main() is the default entry point upon execution of the program, and it calls the Menu() and ShowAllItems() during this time to show the possible things you can do with the program.

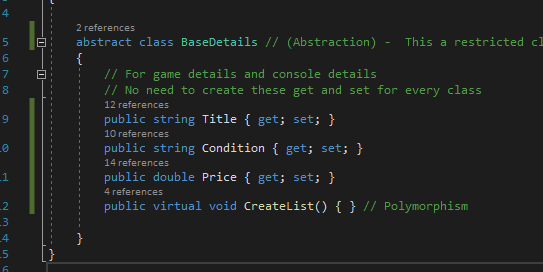


Figure 10 C# BaseDetails class

Figure 10 Shows the BaseDetails abstract class which contains the get and set methods for the fields. This class also contains the CreateList() for the creation of list of products. This class is being inherited by the Game and GameConsole class.

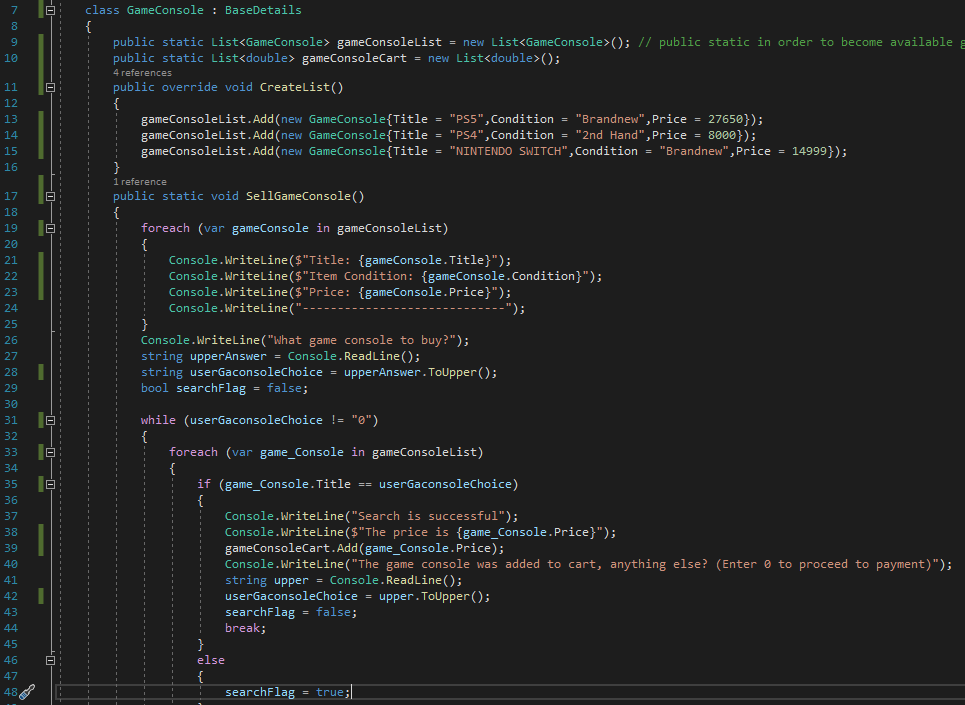


Figure 11 C# GameConsole class part 1

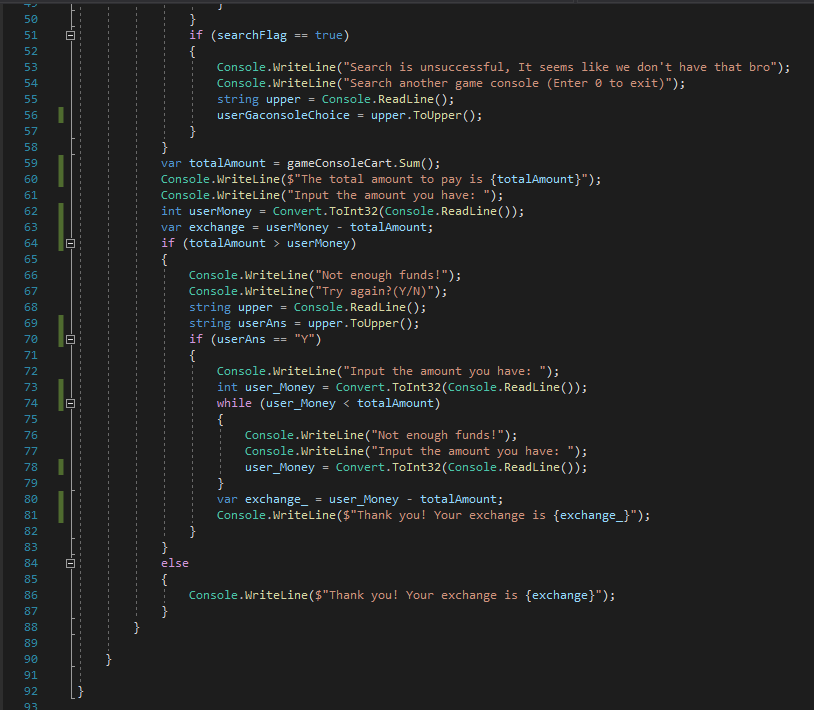


Figure 12 C# GameConsole class part 2

Figure 11 and 12 Show the class for the game console. This contains the method for the creation of a game console list and the method for selling a game console and the computation of the total amount to pay and change.

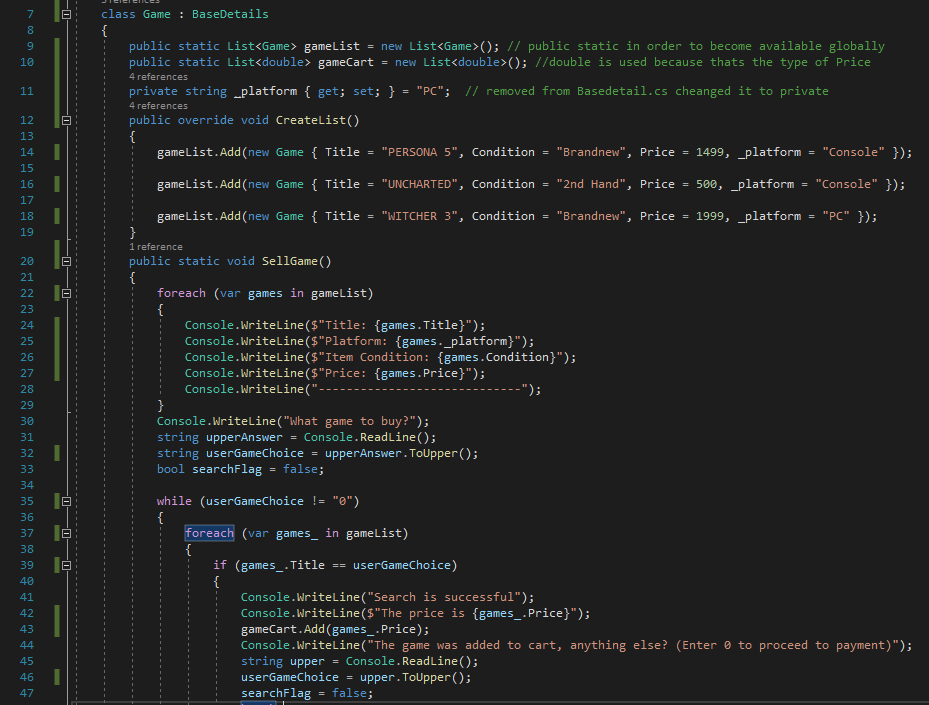


Figure 13 C# Game class part 1

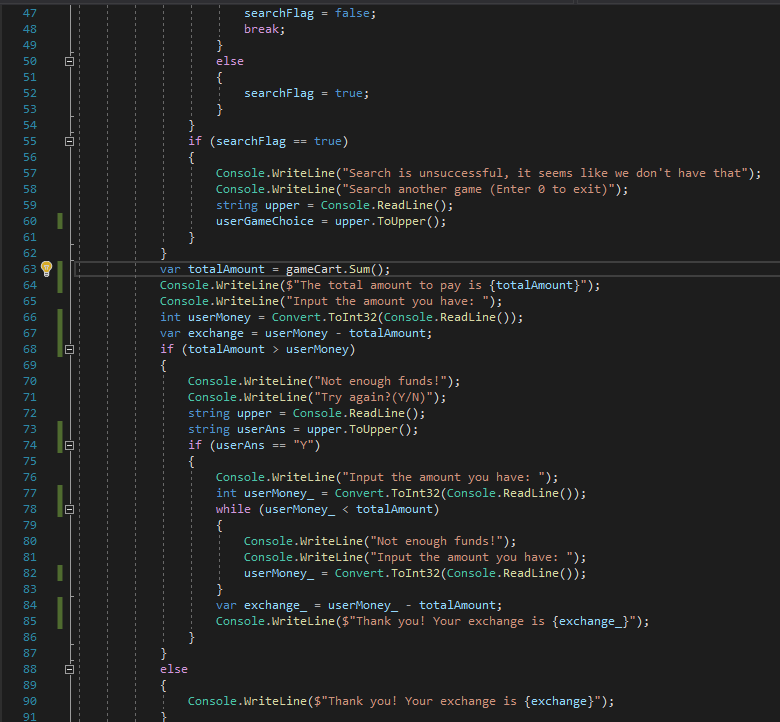


Figure 14 C# Game class part 2

Figure 13 and 14 Show the class for the games. This contains the method for the creation of a game list and the method for selling a game and the computation of the total amount to pay and change.

**B. Verification and Testing Result**

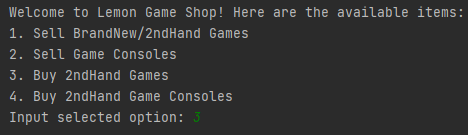
****

Figure 15 Python Menu Output

Figure 15 shows the Menu that was created in the Menu.py file. It shows 4 options which are to sell brand new/2ndhand games, sell game consoles, buy 2ndhand games, and buy 2ndhand game consoles. For this verification and testing, the inputted value will be “3” which is the “Buy 2ndHand Games” option. This is to check if the bought 2nd hand game will be included in the selection in the “Sell BrandNew/2ndHand Games” option.

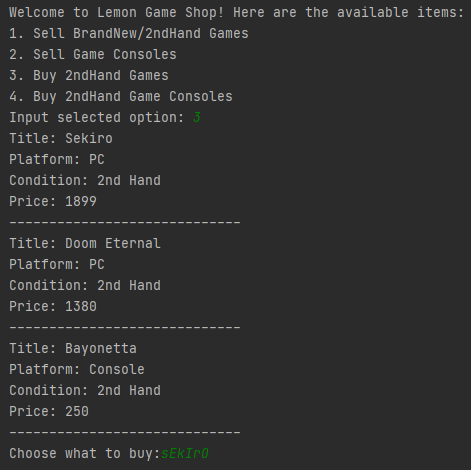
****

Figure 16 Python 2nd Hand Game Selections

Figure 16 shows the 2nd hand game selections. For this part, the 2nd hand game that will be chosen is sekiro and notice that it is written in jumbled uppercase and lowercase. This is to check if the system will still acknowledge the input and disregard case sensitivity.

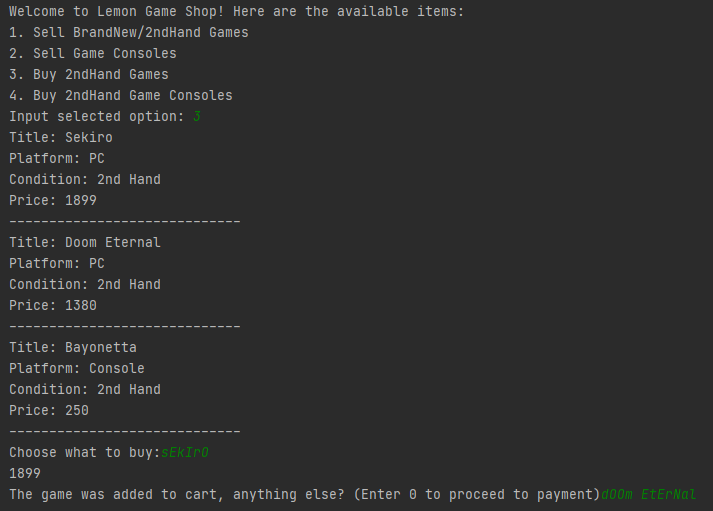


Figure 17 Python Add to Cart

Figure 17 shows the price of the sekiro game and it was stated that the game was added to cart. The program prompts the user if they would like to add more games or proceed to payment. For this testing, we are going to add another game which is doom eternal.

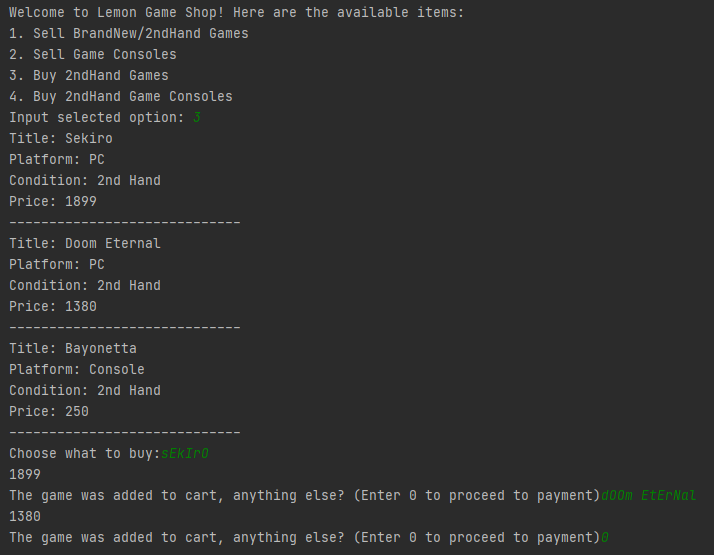


Figure 18 Python Add to Cart part 2

Figure 18 shows the price of the sekiro and doom eternal game. It also shows that both games are added to cart. After this, the program will proceed to payment by inputting 0.

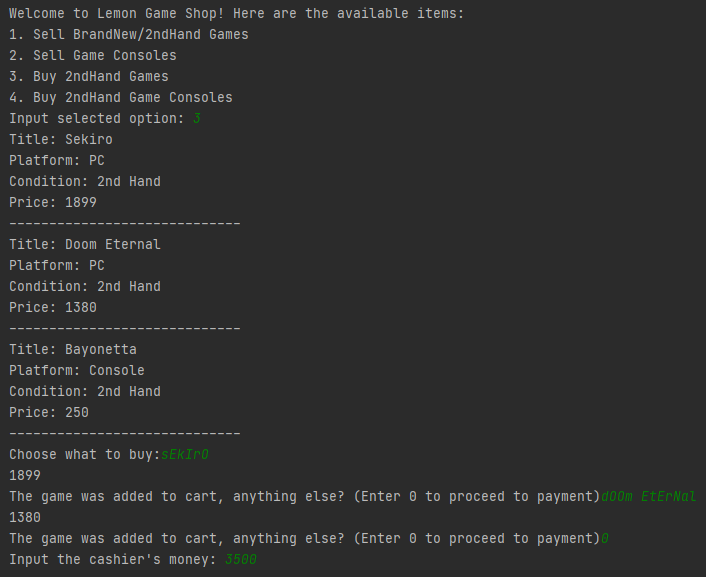


Figure 19 Python Input Cashier Money

Figure 19 shows the program prompting the user to input a certain amount of money. Since the total of both games are ₱3,279.00, the inputted value will be 3500 to have a sufficient fund for the games.

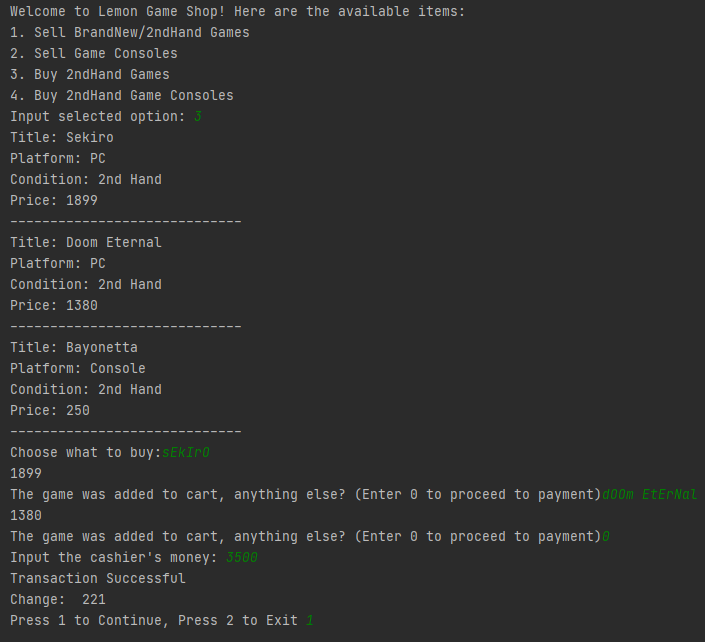


Figure 20 Python Transaction Successful

Figure 20 shows the user that the transaction was successful. It also shows the accurate amount of change which is ₱221. There are 2 options left after a transaction which is to continue or exit. For this test, the program will continue since the input was 1.

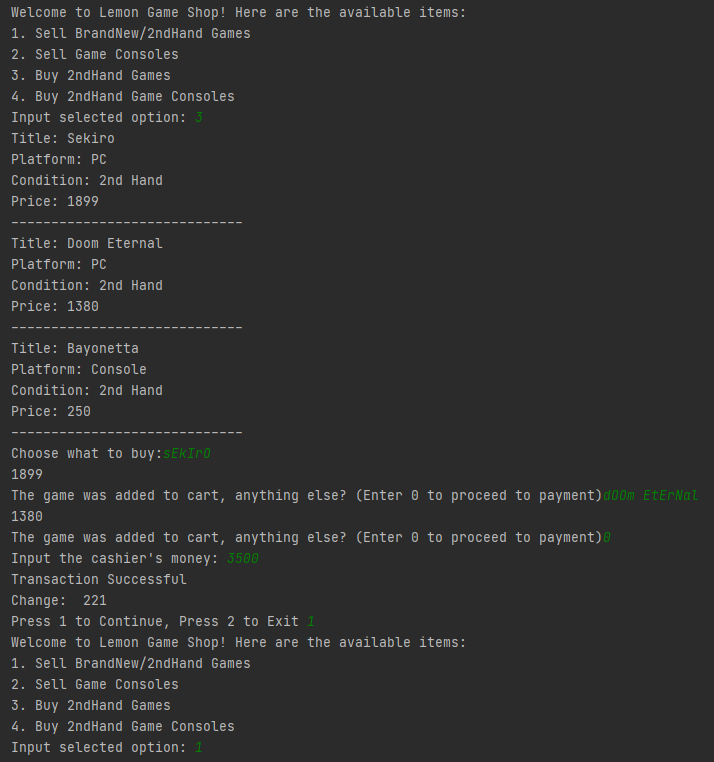


Figure 21 Python Loop to the Menu

Figure 21 shows the loop of the program back to the menu. Instead of pressing the “Buy 2ndHand Games” again, the testing would select the “Sell BrandNew/2ndHand Games” by pressing 1 to verify if they bought 2nd hand games, sekiro and doom eternal, will be available for selling after the previous transaction.

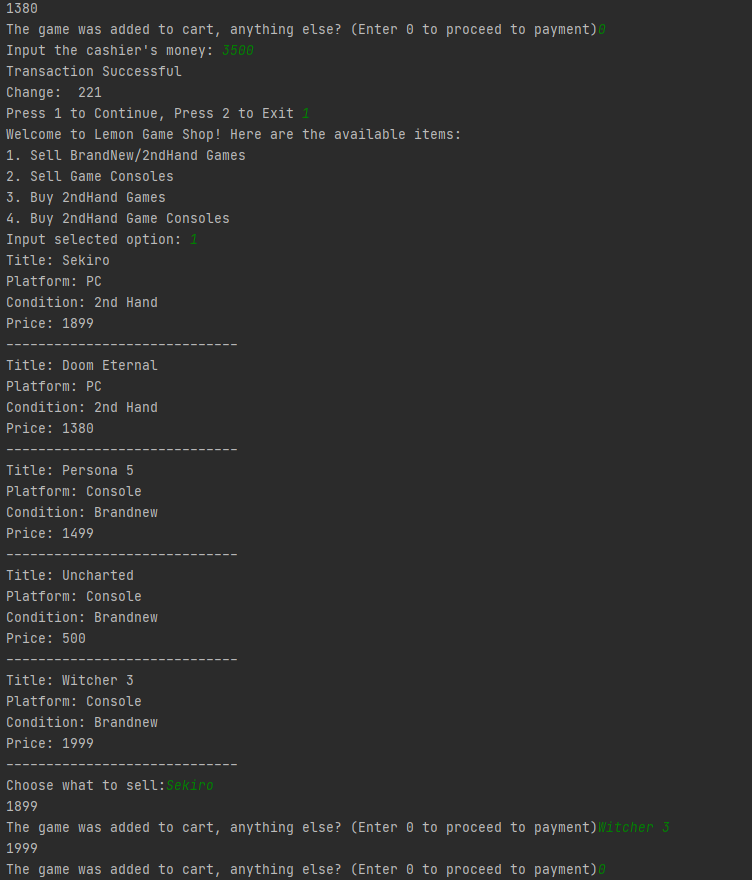


Figure 22 Python Inclusion of the 2nd Hand Games to the Game Selection

As can be seen in Figure 22, both the bought games, sekiro and doom eternal, can be seen in the selection. In order to check if these 2nd hand games can be sold, the inputted game to sell will be sekiro and a brand new game, Witcher 3. After adding the games into the cart, it is time to proceed to payment by pressing 0.

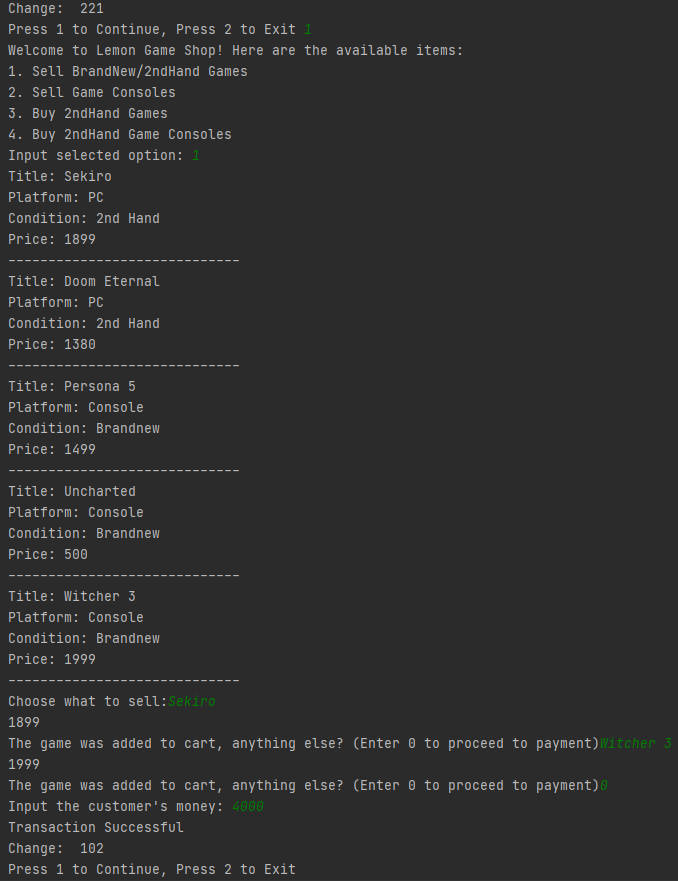


Figure 23 Python Transaction with 2nd Hand Game and Brand New Game

In figure 23, it can be seen that the transaction was successful with the correct amount of change given an input. All of these processes in the game can be also recreated in the game console since the game console inherits most of the functions of the game in the programming part. The only difference is the product which is game consoles instead of a game.

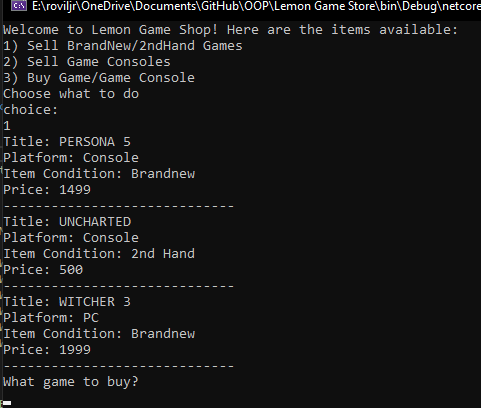


Figure 24 C# Menu & Sell BrandNew/2ndHand Games

Figure 24 shows the menu and the result upon choosing the number 1 option. This option prints out the available games and from here, the user can choose what to buy. If the 2 option was chosen then the available game console will show up instead.

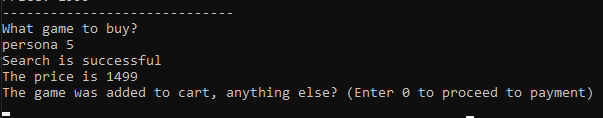


Figure 25 C# Buying a game

Figure 25 shows how the program works upon processing a game. After the user inputted the chosen game, it will print either the search is successful if the game is on the list or unsuccessful it is not. If in the list then it will print the price of it and the notice that it was added to a list and a statement if it will proceed to payment or add another game again.

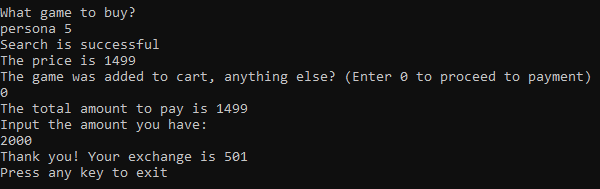


Figure 26 C# Total and change

Figure 26 shows how the program process the calculation of the total amount to pay and the exchange. After entering 0, the code proceeds to the code block created for the computation and this just adds prices from the cart, and it will ask the user to input the amount of money the user have and that value will just be subtracted to the total amount to compute the change. This process is also the same with the game console.

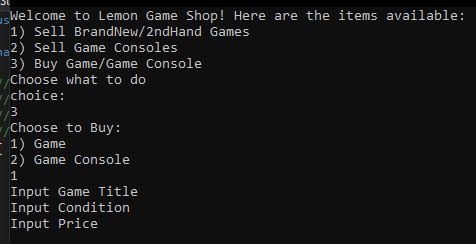


Figure 27 C# Buying a game or console

Figure 27 shows how buying a used or new games and console works on the program created. Upon choosing the number 3 from the menu, it will ask the user to choose what to buy and after choosing, it will ask for the details of that item, then it will be directly added into the list of that particular item.

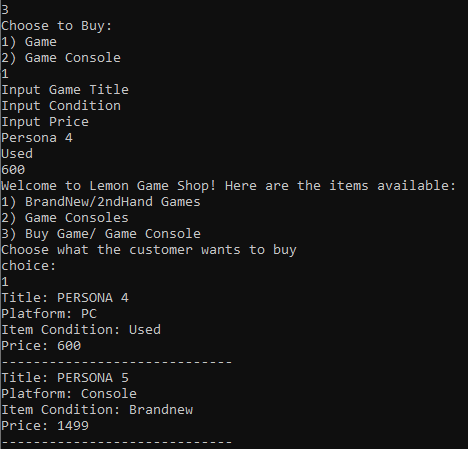


Figure 28 C# Adding the item into inventory

Figure 28 shows that after inputting the details of the game, the menu will show up and when the available games of the shop was check, it shows that the bought game was added into available games list or inventory.

**Chapter V**

**Summary of Findings, Conclusions, and Recommendations**

This chapter will discuss the findings, conclusions and recommendations of the created program Game Shop.

**A. Summary of Findings**

**A background in Programming Logic & Design will make the development of the Game Shop program easier.** Through learning basic programming logics, the researchers are able to create a functional program using different loops and conditional statements. The researchers also used functions and lists to shorten the program.

**Knowledge in Object-Oriented Programming is a must in creating the Game Shop Program.** The program’s main focus is to see the proficiency of the researchers in creating a program by utilizing the classes and the pillars of Object-Oriented Programming. Using the pillars of Object-Oriented Programming the researchers are able to shorten the program and make it more visually pleasing.

**B. Conclusions**

Object-Oriented Programming is one of the most important topics in coding, it is used to give the programmer the convention of creating code from bottom to up or creating the program piece by piece. Giving it an analogy of creating a Lego structure, just like in Lego the creator slowly creates the structure little by little and the result is the creator can add more features in the process and improvise as the structure finishes. This shows the importance of OOP in programming, the researchers concluded that in creating the program they implemented the previous lesson PLD and OOP at the same time.

The researchers acquired the knowledge of being a multi-language programmer such as C++, C#, and python. Also, learning the four pillars of OOP and these are Encapsulation, Abstraction, Inheritance, and Polymorphism. The researchers gathered information that can be useful in the work field, making them adaptive in programming language and using contributive programming just like what OOP lesson is. This project also provides the researchers to learn how inventory works and can be useful when handling business.

**C. Recommendations**

Future researchers may add a function wherein the quantity of product can be seen and the quantity can be deducted or added depending on the transaction. In addition, future researchers can add a Graphical User Interface by using Tkinter, PyQT5, etc. to make the application more visually pleasing and also user-friendly. Further recommendation is to add a login system so that the application would be secure and only those that are employed or administer the store can access the application. Lastly, future researchers can look to implement databases to further make storing data easier and more secure. It can also help in reducing errors and fix these errors faster.

**Appendices**

**Appendix A**

**References**

[1]R. Yilmaz, A. Sezgin, S. Kurnaz and Y. Arslan, "Object-Oriented Programming in Computer Science", *Encyclopedia of Information Science and Technology, Fourth Edition*, pp. 7470-7480, 2018. Available: 10.4018/978-1-5225-2255-3.ch650 [Accessed 24 April 2021].

[2]"Isaac Computer Science", *Isaaccomputerscience.org*, 2021. [Online]. Available: https://isaaccomputerscience.org/concepts/prog\_oop\_paradigm. [Accessed: 25- Apr- 2021].

[3]W. Newman and A. Hendrickson, "The object-oriented paradigm as an implementation of systems theory in IS,"Journal of International Information Management: Vol. 5: Iss. 2, Article 7.", *Core.ac.uk*, 1996. [Online]. Available: https://core.ac.uk/download/pdf/55331824.pdf. [Accessed: 25- Apr- 2021].

[4]F. Onu and M. Ezeji, "Comparative Study of Structured and OOP Paradigms", *Ijarcsse.com*, 2016. [Online]. Available: https://ijarcsse.com/Before\_August\_2017/docs/papers/Volume\_6/8\_August2016/V6I8-01805.pdf. [Accessed: 25- Apr- 2021].

[5]P. Paetau, "ON THE BENEFITS AND PROBLEMS OF THE OBJECT-ORIENTED PARADIGM INCLUDING A FINISH STUDY", *Citeseerx.ist.psu.edu*, 2005. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.3903&rep=rep1&type=pdf. [Accessed: 25- Apr- 2021].

[6]"Chapter 2 Abstraction", *Web.engr.oregonstate.edu*, 2021. [Online]. Available: https://web.engr.oregonstate.edu/~budd/Books/oopintro3e/info/chap02.pdf. [Accessed: 25- Apr- 2021].

**Appendix B**

**Source Code**

**Program.cs**

using System;

namespace Lemon\_Game\_Store

/\*At the outermost level of a program, types are organized into namespaces. The

using directive was used to make the System namespace available to our application,

to use the Console class We defined all our classes within Lemon\_Game\_Store namespace\*/

{

class Program

{

public static void Main(string[] args) // Void because it doesn't return any value to the caller

// signaling the default entry point of execution.

{

Menu();

ShowAllItems();

}

static void Menu()

{

Console.WriteLine("Welcome to Lemon Game Shop! Here are the items available: ");

Console.WriteLine("1) Sell BrandNew/2ndHand Games");

Console.WriteLine("2) Sell Game Consoles");

Console.WriteLine("3) Buy Game/Game Console ");

Console.WriteLine("Choose what to do");

Console.WriteLine("choice: ");

}

static void BuyItems()

{

var user\_choice = Console.ReadLine();

switch (user\_choice)

{

case "1":

Console.WriteLine("Input Game Title");

Console.WriteLine("Input Condition");

Console.WriteLine("Input Price");

string upper = Console.ReadLine();

Game.gameList.Add(new Game

{

Title = upper.ToUpper(),

Condition = Console.ReadLine(),

Price = Convert.ToInt32(Console.ReadLine())

});

Menu();

ShowAllItems();

break;

case "2":

Console.WriteLine("Input Console Name");

Console.WriteLine("Input Condition");

Console.WriteLine("Input Price");

string bupper = Console.ReadLine();

GameConsole.gameConsoleList.Add(new GameConsole

{

Title = bupper.ToUpper(),

Condition = Console.ReadLine(),

Price = Convert.ToInt32(Console.ReadLine())

});

Menu();

ShowAllItems();

break;

default:

Console.WriteLine("Please select a number that is in the menu!");

ShowAllItems();

break;

}

}

static void ShowAllItems()

{

var user\_choice = Console.ReadLine();

switch (user\_choice)

{

case "1":

Game cl = new Game();

cl.CreateList();

Game.SellGame();

break;

case "2":

GameConsole gc = new GameConsole();

gc.CreateList();

GameConsole.SellGameConsole();

break;

case "3":

Console.WriteLine("Choose to Buy:");

Console.WriteLine("1) Game");

Console.WriteLine("2) Game Console");

BuyItems();

break;

default:

Console.WriteLine("Please select a number that is in the menu!");

ShowAllItems();

break;

}

// To keep the console open while debugging

Console.WriteLine("Press any key to exit");

Console.ReadKey();

}

}

}

**Gameconsole.cs**

using System;

using System.Collections.Generic; //for the list

using System.Linq; //for the sum

namespace Lemon\_Game\_Store

{

class GameConsole : BaseDetails

{

public static List<GameConsole> gameConsoleList = new List<GameConsole>(); // public static in order to become available globally

public static List<double> gameConsoleCart = new List<double>();

public override void CreateList()

{

gameConsoleList.Add(new GameConsole{Title = "PS5",Condition = "Brandnew",Price = 27650});

gameConsoleList.Add(new GameConsole{Title = "PS4",Condition = "2nd Hand",Price = 8000});

gameConsoleList.Add(new GameConsole{Title = "NINTENDO SWITCH",Condition = "Brandnew",Price = 14999});

}

public static void SellGameConsole()

{

foreach (var gameConsole in gameConsoleList)

{

Console.WriteLine($"Title: {gameConsole.Title}");

Console.WriteLine($"Item Condition: {gameConsole.Condition}");

Console.WriteLine($"Price: {gameConsole.Price}");

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

}

Console.WriteLine("What game console to buy?");

string upperAnswer = Console.ReadLine();

string userGaconsoleChoice = upperAnswer.ToUpper();

bool searchFlag = false;

while (userGaconsoleChoice != "0")

{

foreach (var game\_Console in gameConsoleList)

{

if (game\_Console.Title == userGaconsoleChoice)

{

Console.WriteLine("Search is successful");

Console.WriteLine($"The price is {game\_Console.Price}");

gameConsoleCart.Add(game\_Console.Price);

Console.WriteLine("The game console was added to cart, anything else? (Enter 0 to proceed to payment)");

string upper = Console.ReadLine();

userGaconsoleChoice = upper.ToUpper();

searchFlag = false;

break;

}

else

{

searchFlag = true;

}

}

if (searchFlag == true)

{

Console.WriteLine("Search is unsuccessful, It seems like we don't have that bro");

Console.WriteLine("Search another game console (Enter 0 to exit)");

string upper = Console.ReadLine();

userGaconsoleChoice = upper.ToUpper();

}

}

var totalAmount = gameConsoleCart.Sum();

Console.WriteLine($"The total amount to pay is {totalAmount}");

Console.WriteLine("Input the amount you have: ");

int userMoney = Convert.ToInt32(Console.ReadLine());

var exchange = userMoney - totalAmount;

if (totalAmount > userMoney)

{

Console.WriteLine("Not enough funds!");

Console.WriteLine("Try again?(Y/N)");

string upper = Console.ReadLine();

string userAns = upper.ToUpper();

if (userAns == "Y")

{

Console.WriteLine("Input the amount you have: ");

int user\_Money = Convert.ToInt32(Console.ReadLine());

while (user\_Money < totalAmount)

{

Console.WriteLine("Not enough funds!");

Console.WriteLine("Input the amount you have: ");

user\_Money = Convert.ToInt32(Console.ReadLine());

}

var exchange\_ = user\_Money - totalAmount;

Console.WriteLine($"Thank you! Your exchange is {exchange\_}");

}

}

else

{

Console.WriteLine($"Thank you! Your exchange is {exchange}");

}

}

}

}

**Game.cs**

using System;

using System.Collections.Generic; //for the list

using System.Linq; //for the sum

namespace Lemon\_Game\_Store

{

class Game : BaseDetails

{

public static List<Game> gameList = new List<Game>(); // public static in order to become available globally

public static List<double> gameCart = new List<double>(); //double is used because thats the type of Price

private string \_platform { get; set; } = "PC"; // removed from Basedetail.cs cheanged it to private

public override void CreateList()

{

gameList.Add(new Game { Title = "PERSONA 5", Condition = "Brandnew", Price = 1499, \_platform = "Console" });

gameList.Add(new Game { Title = "UNCHARTED", Condition = "2nd Hand", Price = 500, \_platform = "Console" });

gameList.Add(new Game { Title = "WITCHER 3", Condition = "Brandnew", Price = 1999, \_platform = "PC" });

}

public static void SellGame()

{

foreach (var games in gameList)

{

Console.WriteLine($"Title: {games.Title}");

Console.WriteLine($"Platform: {games.\_platform}");

Console.WriteLine($"Item Condition: {games.Condition}");

Console.WriteLine($"Price: {games.Price}");

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

}

Console.WriteLine("What game to buy?");

string upperAnswer = Console.ReadLine();

string userGameChoice = upperAnswer.ToUpper();

bool searchFlag = false;

while (userGameChoice != "0")

{

foreach (var games\_ in gameList)

{

if (games\_.Title == userGameChoice)

{

Console.WriteLine("Search is successful");

Console.WriteLine($"The price is {games\_.Price}");

gameCart.Add(games\_.Price);

Console.WriteLine("The game was added to cart, anything else? (Enter 0 to proceed to payment)");

string upper = Console.ReadLine();

userGameChoice = upper.ToUpper();

searchFlag = false;

break;

}

else

{

searchFlag = true;

}

}

if (searchFlag == true)

{

Console.WriteLine("Search is unsuccessful, it seems like we don't have that");

Console.WriteLine("Search another game (Enter 0 to exit)");

string upper = Console.ReadLine();

userGameChoice = upper.ToUpper();

}

}

var totalAmount = gameCart.Sum();

Console.WriteLine($"The total amount to pay is {totalAmount}");

Console.WriteLine("Input the amount you have: ");

int userMoney = Convert.ToInt32(Console.ReadLine());

var exchange = userMoney - totalAmount;

if (totalAmount > userMoney)

{

Console.WriteLine("Not enough funds!");

Console.WriteLine("Try again?(Y/N)");

string upper = Console.ReadLine();

string userAns = upper.ToUpper();

if (userAns == "Y")

{

Console.WriteLine("Input the amount you have: ");

int userMoney\_ = Convert.ToInt32(Console.ReadLine());

while (userMoney\_ < totalAmount)

{

Console.WriteLine("Not enough funds!");

Console.WriteLine("Input the amount you have: ");

userMoney\_ = Convert.ToInt32(Console.ReadLine());

}

var exchange\_ = userMoney\_ - totalAmount;

Console.WriteLine($"Thank you! Your exchange is {exchange\_}");

}

}

else

{

Console.WriteLine($"Thank you! Your exchange is {exchange}");

}

}

}

}

**Basedetails.cs**

namespace Lemon\_Game\_Store

{

abstract class BaseDetails // (Abstraction) - This a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

{

/\*For game details and console details, no need to create these get and set for every class on these properties \*/

public string Title { get; set; }

public string Condition { get; set; }

public double Price { get; set; }

public virtual void CreateList() { } // Polymorphism

}

}

**Main.py**

import Game as G  
import Menu as M  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 games = G.Game()  
 gameconsole = G.Gameconsole()  
 user\_choice = "1"  
 while user\_choice == "1":  
 menu = M.Menu()  
 user\_choice = input("Input selected option: ")  
 if user\_choice == "1":  
 games.createlist\_sellproduct()  
 games.sellproduct()  
 elif user\_choice == "2":  
 gameconsole.createlist\_sellproduct()  
 gameconsole.sellproduct()  
 elif user\_choice == "3":  
 games.createlist\_buyproduct()  
 games.buyproduct()  
 elif user\_choice == "4":  
 gameconsole.createlist\_buyproduct()  
 gameconsole.buyproduct()  
 user\_choice = input("Press 1 to Continue, Press 2 to Exit ")

**Menu.py**

class Menu:  
 def \_\_init\_\_(self):  
 print("Welcome to Lemon Game Shop! Here are the available items:")  
 print("1. Sell BrandNew/2ndHand Games")  
 print("2. Sell Game Consoles")  
 print("3. Buy 2ndHand Games")  
 print("4. Buy 2ndHand Game Consoles")

**Item.py**

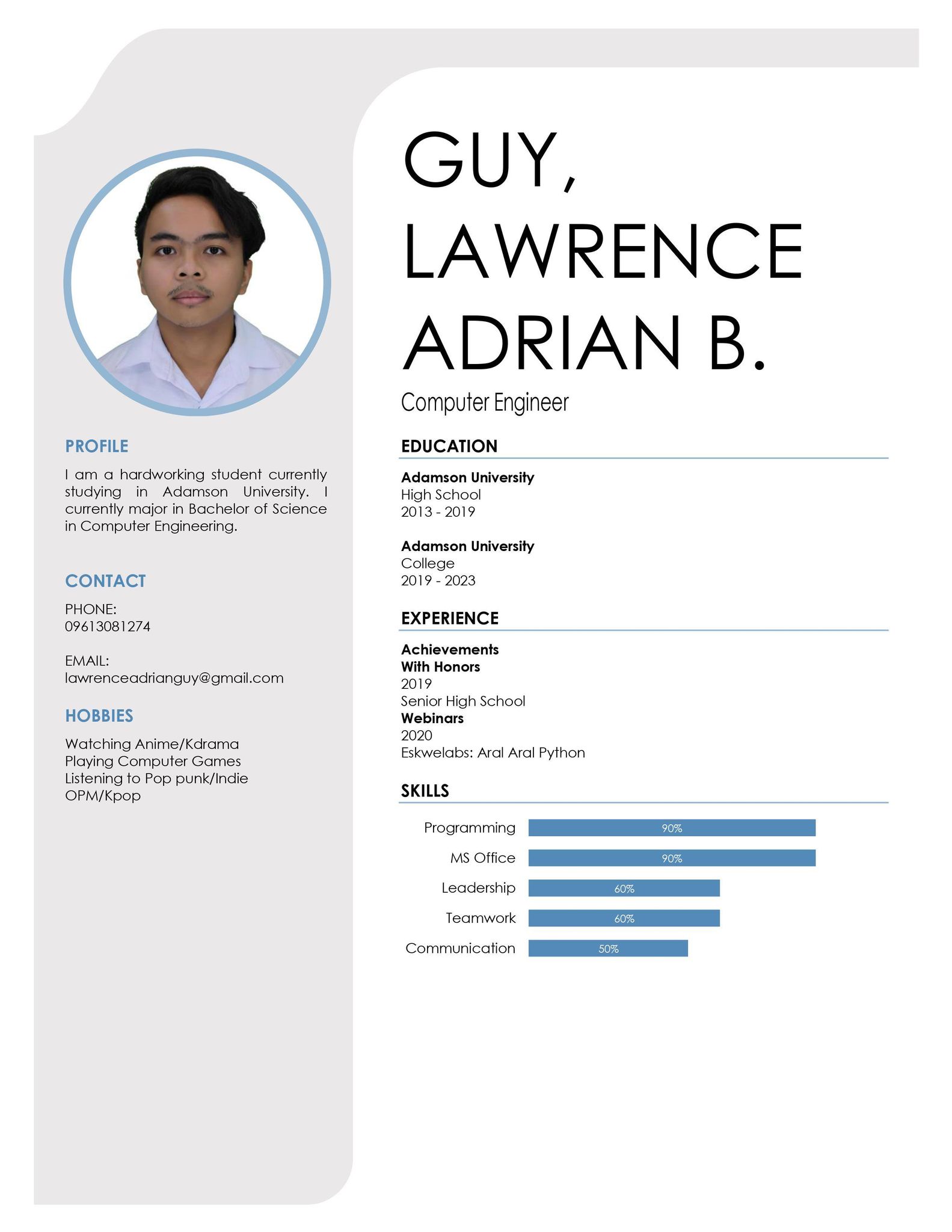
class Item:  
 def \_\_init\_\_(self, Title, Platform, Condition, Price):  
 self.Title = Title  
 self.Platform = Platform  
 self.Condition = Condition  
 self.Price = Price

**Game.py**

import Item as I  
class Game:  
 def \_\_init\_\_(self):  
 self.sell\_product = []  
 self.buy\_product = []  
 self.sell\_cart = []  
 self.buy\_cart = []  
 def createlist\_sellproduct(self):  
 self.sell\_product.append(I.Item(Title='Persona 5', Platform='Console', Condition='Brandnew', Price=1499))  
 self.sell\_product.append(I.Item(Title='Uncharted', Platform='Console', Condition='Brandnew', Price=500))  
 self.sell\_product.append(I.Item(Title='Witcher 3', Platform='Console', Condition='Brandnew', Price=1999))  
 def sellproduct(self):  
 for product in self.sell\_product:  
 print("Title: {}\nPlatform: {}\nCondition: {}\nPrice: {}".format(product.Title,  
 product.Platform,  
 product.Condition,  
 product.Price))  
 print("-----------------------------")  
 user\_input = input("Choose what to sell:")  
 searchFlag = False  
 while user\_input != "0":  
 for products in self.sell\_product:  
 if user\_input.lower() == products.Title.lower():  
 print(products.Price)  
 self.sell\_cart.append(products.Price)  
 user\_input = input("The game was added to cart, anything else? (Enter 0 to proceed to payment)")  
 searchFlag = False  
 break  
 else:  
 searchFlag = True  
 if searchFlag == True:  
 print("Search is unsuccessful, It seems like we don't have that")  
 user\_input = input("Search another game/game console (Enter 0 to proceed to payment): ")  
 total\_amount\_to\_pay = sum(self.sell\_cart)  
 customer\_pay = int(input("Input the customer's money: "))  
 if customer\_pay >= total\_amount\_to\_pay:  
 change = customer\_pay - total\_amount\_to\_pay  
 print("Transaction Successful")  
 print("Change: ", change)  
 self.sell\_cart.clear()  
 elif customer\_pay < total\_amount\_to\_pay:  
 print("Insufficient Money")  
 self.sell\_cart.clear()  
  
  
 def createlist\_buyproduct(self):  
 self.buy\_product.append(I.Item(Title='Sekiro', Platform='PC', Condition='2nd Hand', Price=1899))  
 self.buy\_product.append(I.Item(Title='Doom Eternal', Platform='PC', Condition='2nd Hand', Price=1380))  
 self.buy\_product.append(I.Item(Title='Bayonetta', Platform='Console', Condition='2nd Hand', Price=250))  
  
 def buyproduct(self):  
 for product in self.buy\_product:  
 print("Title: {}\nPlatform: {}\nCondition: {}\nPrice: {}".format(product.Title,  
 product.Platform,  
 product.Condition,  
 product.Price))  
 print("-----------------------------")  
 user\_input = input("Choose what to buy:")  
 searchFlag = False  
 while user\_input != "0":  
 for products in self.buy\_product:  
 if user\_input.lower() == products.Title.lower():  
 print(products.Price)  
 self.buy\_cart.append(products.Price)  
 self.sell\_product.append(products)  
 user\_input = input("The game was added to cart, anything else? (Enter 0 to proceed to payment)")  
 searchFlag = False  
 break  
 else:  
 searchFlag = True  
 if searchFlag == True:  
 print("Search is unsuccessful, It seems like we don't want to buy that");  
 user\_input = input("Search another game/game console (Enter 0 to proceed to payment): ")  
  
 total\_amount\_to\_pay = sum(self.buy\_cart)  
 cashier\_pay = int(input("Input the cashier's money: "))  
 if cashier\_pay >= total\_amount\_to\_pay:  
 change = cashier\_pay - total\_amount\_to\_pay  
 print("Transaction Successful")  
 print("Change: ", change)  
 self.buy\_cart.clear()  
 elif cashier\_pay < total\_amount\_to\_pay:  
 print("Insufficient Money")  
 self.buy\_cart.clear()  
  
  
  
class Gameconsole(Game): # inherited the Game class  
  
 def createlist\_sellproduct(self):  
 self.sell\_product.append(I.Item(Title="PS5", Condition="Brandnew", Price=27650, Platform=""))  
 self.sell\_product.append(I.Item(Title="PS4", Condition="Brandnew", Price=8000, Platform=""))  
 self.sell\_product.append(I.Item(Title="Nintendo Switch", Condition="Brandnew", Price=14999, Platform=""))  
  
 def createlist\_buyproduct(self):  
 self.buy\_product.append(I.Item(Title="PS3", Condition="2nd Hand", Price=8500, Platform=""))  
 self.buy\_product.append(I.Item(Title="PS2", Condition="2nd Hand", Price=3000, Platform=""))  
 self.buy\_product.append(I.Item(Title="Xbox 360", Condition="2nd Hand", Price=5000, Platform=""))

**Appendix C**

**Researcher’s Profile**



Graphical user interface, text, application

Description automatically generated



