Programming II Assignment 1

# Table of Contents

[Part 1: Object oriented analysis and design 2](#_Toc460925378)

[Identifying classes, methods and attributes 2](#_Toc460925379)

[The Runner class 2](#_Toc460925380)

[The Professional class 2](#_Toc460925381)

[The Amateur class 2](#_Toc460925382)

[The sponsorship class/interface 3](#_Toc460925383)

[The FancyRunner class 3](#_Toc460925384)

[The CRC cards 3](#_Toc460925385)

[The Class diagram for the ‘GetFit’ Athletic Club 4](#_Toc460925386)

[Part2: Programming 5](#_Toc460925387)

[Question 1: Person 5](#_Toc460925388)

[Person.cs 5](#_Toc460925389)

[Program.cs 8](#_Toc460925390)

[Question 2: Hanoi tower 9](#_Toc460925391)

[HanoiTower.cs 9](#_Toc460925392)

[Program.cs 9](#_Toc460925393)

[Question 3: Tool 11](#_Toc460925394)

[Tool.cs 11](#_Toc460925395)

[Rock.cs 13](#_Toc460925396)

[Paper.cs 13](#_Toc460925397)

[Scissor.cs 13](#_Toc460925398)

[Program.cs 14](#_Toc460925399)

[Part 3: Short answer questions 15](#_Toc460925400)

[Strings in C# 15](#_Toc460925401)

[Exceptions 15](#_Toc460925402)

[Advantages and disadvantages of exception handling in C# 16](#_Toc460925403)

[Class vs Struct 16](#_Toc460925404)

[References 16](#_Toc460925405)

# Part 1: Object oriented analysis and design

## Identifying classes, methods and attributes

Classes are used to create objects, with attributes to store information regarding them. Methods are actions or messages displayed regarding the said objects.

Therefore, for the ‘GetFit’ Athletic Club, the classes are:

1. Runner
2. Professional
3. Amateur
4. FancyRunner
5. Sponsorshipform

The Runner class is the abstract class. The ‘*Professional*’ and ‘*Amateur*’ classes are children of the ‘*Runner* class’, and inherit the attributes and methods of the parent class. Furthermore, *‘FancyRunner’* class is a child class of the Amateur class. The ‘*SponsorshipForm*’ class is an interface of the ‘*Amateur*’ class, and can thus be accessed by ‘*FancyRunner*’.

### The Runner class

It is comprised of the following attributes:

* Number (to store the runner’s number)
* Hour
* Minute
* second
* Race result (did they finish the race?)

It also has the following methods:

* Record finish time
* Finished race

### The Professional class

It is comprised of the following attributes:

* World ranking

### The Amateur class

It also has the following attributes:

* Sponsorship form

### The sponsorship class/interface

It is comprised of the following attributes:

* Number of sponsors
* List of sponsors
* List of amounts sponsored

It also has the following methods:

* Add sponsor (sponsor name and amount)
* Print collection list (sponsors, sponsorship amounts, total raised)

### The FancyRunner class

It is comprised of the following attributes:

* Costume

## The CRC cards

These cards will help show the relationships and collaboration between classes.

|  |  |
| --- | --- |
| **Runner** | |
| * Stores details of runner * Keeps track of their finishing time * Did they finish the race? * Announces results of runners | * Professional class * Amateur class |

|  |  |
| --- | --- |
| **Professional** | |
| * Stores details of runner * Stores their world ranking |  |

|  |  |
| --- | --- |
| **Amateur** | |
| * Stores number of runner * Keeps track of their finishing time * Stores sponsorship details | * Sponsorship form * Fancy Runner |

|  |  |
| --- | --- |
| **Sponsorship** | |
| * Records a list of sponsors, along with the money sponsored * Add sponsors to the list * Prints details of sponsors, amounts, total raised |  |

|  |  |
| --- | --- |
| **Fancy Runner** | |
| * Adds the costume name to the runner details |  |

## The Class diagram for the ‘GetFit’ Athletic Club



# Part2: Programming

## Question 1: Person

### Person.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Person

{

class Person

{

private string firstName; // properties of Person class

private string lastName;

private string email;

private DateTime dateOfBirth;

public Person(string fname, string lname, string mail, int day, int month, int year) // constructor for all parameters

{

firstName = fname;

lastName = lname;

email = mail;

string dob = day.ToString()+"/"+month.ToString()+ "/"+year.ToString();

dateOfBirth = Convert.ToDateTime(dob);

}

public Person(string fname, string lname, string mail) // constructor for first name, last name, email

{

firstName = fname;

lastName = lname;

email = mail;

}

public Person(string fname, string lname, int day, int month, int year) // constructor for first name, last name, date of birth

{

firstName = fname;

lastName = lname;

string dob = day.ToString() + "/" + month.ToString() + "/" + year.ToString();

dateOfBirth = Convert.ToDateTime(dob);

}

public string getFirstName() //getter for first name

{

return "\n\nFirst Name : "+ firstName;

}

public string getLastName() //getter for last name

{

return "Last Name : "+ lastName;

}

public string getEmail() //getter for email

{

return "Email address : "+ email;

}

public string getDOB() //getter for date of birth

{

return "Date of birth : "+ dateOfBirth;

}

public bool checkIfAdult() // checks if person > 18. (current date - date of birth)

{

TimeSpan span = DateTime.Now.Subtract(dateOfBirth);

Console.WriteLine("\nIs person an adult?");

if (span.TotalDays / 365 < 18)

return false;

else

return true;

}

public string checkSunSign()

{

int month = dateOfBirth.Month; // gets value of month in 'int' (1 to 12)

Console.WriteLine("\nWestern Zodiac Sign:");

if (month == 1) // January

return "Capricorn";

if (month == 2) // February

return "Aquarius";

if (month == 3) // March

return "Pisces";

if (month == 4) // April

return "Aries";

if (month == 5) // May

return "Taurus";

if (month == 6) // June

return "Gemini";

if (month == 7) // July

return "Cancer";

if (month == 8) // August

return "Leo";

if (month == 9) // September

return "Virgo";

if (month == 10) // October

return "Libra";

if (month == 11) // November

return "Scorpio";

if (month == 12) // December

return "Sagittarius";

else

return ""; // to ensure that system compiles properly

}

public string checkChineseSign()

{

int year = dateOfBirth.Year;

Console.WriteLine("\nChinese Zodiac Sign:");

// since chinese zodiac is cycle of 12 years, year % 12 to get remainder to find sign

if (year % 12 == 1)

return "Rooster";

if (year % 12 == 2)

return "Dog";

if (year % 12 == 3)

return "Pig";

if (year % 12 == 4)

return "Rat";

if (year % 12 == 5)

return "Ox";

if (year % 12 == 6)

return "Tiger";

if (year % 12 == 7)

return "Rabbit";

if (year % 12 == 8)

return "Dragon";

if (year % 12 == 9)

return "Snake";

if (year % 12 == 10)

return "Horse";

if (year % 12 == 11)

return "Horse";

if (year % 12 == 0)

return "Monkey";

else

return ""; // to ensure that system compiles properly

}

public bool checkIfBirthday()

{

Console.WriteLine("\nIs it that person's birthday?");

int todayDayofYear = DateTime.Now.DayOfYear; // gets the date converted in a number from 1 to 366 representing it

int birthdayDOY = dateOfBirth.DayOfYear;

if (birthdayDOY == todayDayofYear)

return true;

else

return false;

}

}

}

### Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Person

{

class Program

{

static void Main(string[] args)

{

Person p1 = new Person("Francois", "Mindiel", "fgmind@hotmail.com", 05,09,1980);

Console.WriteLine(p1.getFirstName());

Console.WriteLine(p1.getLastName());

Console.WriteLine(p1.getEmail());

Console.WriteLine(p1.getDOB());

Console.WriteLine(p1.checkIfAdult());

Console.WriteLine(p1.checkSunSign());

Console.WriteLine(p1.checkChineseSign());

Console.WriteLine(p1.checkIfBirthday());

Person p2 = new Person("John", "Smith", "john@smith.com");

Console.WriteLine(p2.getFirstName());

Console.WriteLine(p2.getLastName());

Console.WriteLine(p2.getEmail());

// cannot test date methods as no date parameter was passed to p2

Person p3 = new Person("Jane", "Dow", 04, 09, 2000); // change date parameters to check birthday validation

Console.WriteLine(p3.checkIfBirthday());

}

}

}

## Question 2: Hanoi tower

### HanoiTower.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HanoiTower

{

class HanoiTower //class containing method to move discs from 1 peg to the other

{

public void moveDiscs(int n, String a, String b, String c)

{

if (n == 1)

{

Console.WriteLine("Move Disc from " + a + " to " + b);

}

else

{

moveDiscs(n - 1, a, c, b); // reloads method and decrements n, also swaps pegs

Console.WriteLine("Move Disc from " + a + " to " + b);

moveDiscs(n - 1, c, b, a); // will loop through method until n = 1

}

}

}

}

### Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HanoiTower

{

class Program

{

static void Main(string[] args)

{

HanoiTower hanoi = new HanoiTower();

Console.WriteLine("Enter number of discs: ");

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

hanoi.moveDiscs(n, "Peg A", "Peg C", "Peg B"); // had to swap Peg B and C...

}

}

}

## Question 3: Tool

### Tool.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tool

{

class Tool

{

protected int strength;

protected char type;

protected string name;

// didn't create a constructor for 'tool' as it has to be either Rock,Paper or scissor...

public void setStrength(int strength\_) // sets strength of tool

{

strength = strength\_;

}

public int getStrength()

{

return strength;

}

public char getType()

{

return type;

}

public string getName()

{

return name;

} // To differentiate Name/type during fight

public bool fight(Tool temp1, Tool temp2)

{

int temp1Strength = temp1.strength; // used to store objects' strength

int temp2Strength = temp2.strength;

if (temp1.type == 'r') // if rock

{

if (temp2.type == 'p')

temp1Strength = temp1Strength / 2; // strength temporarily halved

if (temp2.type == 's')

temp1Strength = temp1Strength \* 2; // strength temporarily doubled

}

if (temp1.type == 'p') // if paper

{

if (temp2.type == 's')

temp1Strength = temp1Strength / 2; // strength temporarily halved

if (temp2.type == 'r')

temp1Strength = temp1Strength \* 2; // strength temporarily doubled

}

if (temp1.type == 's') // if scissor

{

if (temp2.type == 'r')

temp1Strength = temp1Strength / 2; // strength temporarily halved

if (temp2.type == 'p')

temp1Strength = temp1Strength \* 2; // strength temporarily doubled

}

// MATCH TIME!!!

Console.WriteLine("\n\n" + temp1.getName() + " vs " + temp2.getName() + "\n");

Console.WriteLine("Strength of " + temp1.getName() + " = " + temp1.getStrength());

Console.WriteLine("Type of " + temp1.getName() + " is " + temp1.getType());

Console.WriteLine("Strength of " + temp2.getName() + " = " + temp2.getStrength());

Console.WriteLine("Type of " + temp2.getName() + " is " + temp2.getType());

Console.WriteLine("Does " + temp1.getName() + " beat " + temp2.getName() + "?");

if (temp1Strength > temp2Strength)

{

return true;

}

else

return false;

}

}

}

### Rock.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tool

{

class Rock : Tool

{

public Rock(int strength\_)

{

strength = strength\_;

type = 'r';

name = "Rock"; // makes fight better looking on screen :)

}

}

}

### Paper.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tool

{

class Paper : Tool

{

public Paper(int strength\_)

{

strength = strength\_;

type = 'p';

name = "Paper"; // makes fight better looking on screen :)

}

}

}

### Scissor.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tool

{

class Scissor : Tool

{

public Scissor(int strength\_)

{

strength = strength\_;

type = 's';

name = "Scissor"; // makes fight better looking on screen :)

}

}

}

### Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tool

{

class Program

{

static void Main(string[] args)

{

Rock r1 = new Rock(99); // creating Rock

Scissor s1 = new Scissor(7); // creating Scissor

Paper p1 = new Paper(3); // creating Paper

r1.setStrength(4); // changes strength of rock to 4.

Console.WriteLine(r1.getStrength()); // to check if strength has been changed

Console.WriteLine(r1.fight(r1, s1));

Console.WriteLine(r1.fight(r1, p1));

Console.WriteLine(r1.fight(s1, p1));

}

}

}

# Part 3: Short answer questions

## Strings in C#

Strings in C# (and most object-oriented languages) are used to store a sequence of characters, or an array of characters. The string class (System.String) inherits from the System.Object class. The characteristics of the string class are that it can store any ASCII character. Furthermore, with with the appropriate methods, we can retrieve the desired character stored in the string, or get the length of the string.

Common methods for the String class are:

Concat: which concatenates two or more strings into one (eg public static string Concat(string string1, string2, string3)).

The copy function: creates a new string with the specified value(eg public static string Copy (string string\_value)).

Equals: which compares whether 2 strings are of the same value.

ToLower / ToUpper: converts all characters to either lower or upper case.

The String class has much more methods and purposes, but these, along with variable declarations, are the main ones commonly used.

## Exceptions

An exception is a problem that arises during the execution of a program. In C# it is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero (C# - Exception Handling, 2016). It inherits from the System.Object class and has many predefined system exceptions. Exception handling is how the system reacts when an exception occurs. As an example, when dividing by zero the system instead of crashing, the exception handler would stop the program and inform the user that a division by zero exception has occurred. Exception handling is a good and safe way to tackle with issues arising in a program, and can also be customised, or new exceptions created.

When designing a banking system for example, the amount withdrawn by someone cannot be more than the amount in the account. An exception would be thrown should that happen, the user informed via a custom message.

Exceptions are caught when they are called to validate an input, process or output. They are implemented using:

* **Try**: performs a process by calling a method (e.g. result = n1 / n2)
* **Catch**: declares what we want to ‘catch’ and what to do. For example, catching a division by zero exception, and warning the user.
* **Finally**: what the program does after it has checked for exceptions.

## Advantages and disadvantages of exception handling in C#

Advantages of exception handling are:

* Exceptions can be thrown to prevent the system from crashing or exiting unexpectedly
* Messages can be customised to the user’s need
* It can pick up invalid inputs, or valid inputs that may cause issues later on in the program
* It is a good way to cover bases by ensuring that the program works as desired. It doesn’t debug per se but helps in identifying some issues in the code.

Disadvantages of exception handling:

* Exceptions would have to be imported in the program, and implemented in several places depending on the size and complexity of the latter. This may result in a bulky code.
* Exception handling would have to be created for validation of every input, method, output of the program. This issue might not apply to simple codes, but complex programming would require this, to make it fail safe.
* That also means that more memory would be required for running the program, and this could cause lengthier runtimes.

## Class vs Struct

A class is used to create, define and what it does (methods). It is a reference type, meaning that when an object is created, a reference to where it is in the memory is created.

A Struct is nearly the same as a class object, except that it a value type. This means that a value is created in the memory for the object.

Both have their usefulness. Deciding on whether to use one over the other should be done by applying the following:

Is it immutable? If the value stays the same, then a Struct is appropriate. If the value is to be changed often, it is better to go with class to avoid unexpected behaviour.

Complexity: if it is simple, for example defining time by hour, minute, second, a struct would be best. A value would be created and stored. Not much data manipulation would be done on it, or often.

Embedding: if it is simple and is embedded in another object (for example struct time can be stored in a class containg working hours of staff), we could go for a Struct.

If an object is more complex, for example it will store staff name, address, wages, work time; then a class is better suited to the situation. It will probably contain a value for work time, which would be created by calling a struct. Also the class will contain more methods to manipulate the data.

# References

*C# - Exception Handling*. (2016, 09 05). Retrieved from Tutorialspoint: http://www.tutorialspoint.com/csharp/csharp\_exception\_handling.htm