MOD002702 Main Coursework Assignment

Introduction

A marina manager has requested a boat bookings system. The program should be capable of:

- showing all bookings and remaining space in the 150m canal
- adding a booking
- removing a booking
- closing the program

Only boats meeting certain criteria can book. The canal is narrow and boats cannot pass each other, so the program should also detail boat movements that occur when a craft blocked in by other boats needs to leave.

Design

The program has three main classes – a UserInterface class, BoatList class and BoatNode class. The BoatNode class has three subclasses – Sailing, Motor and Narrow.

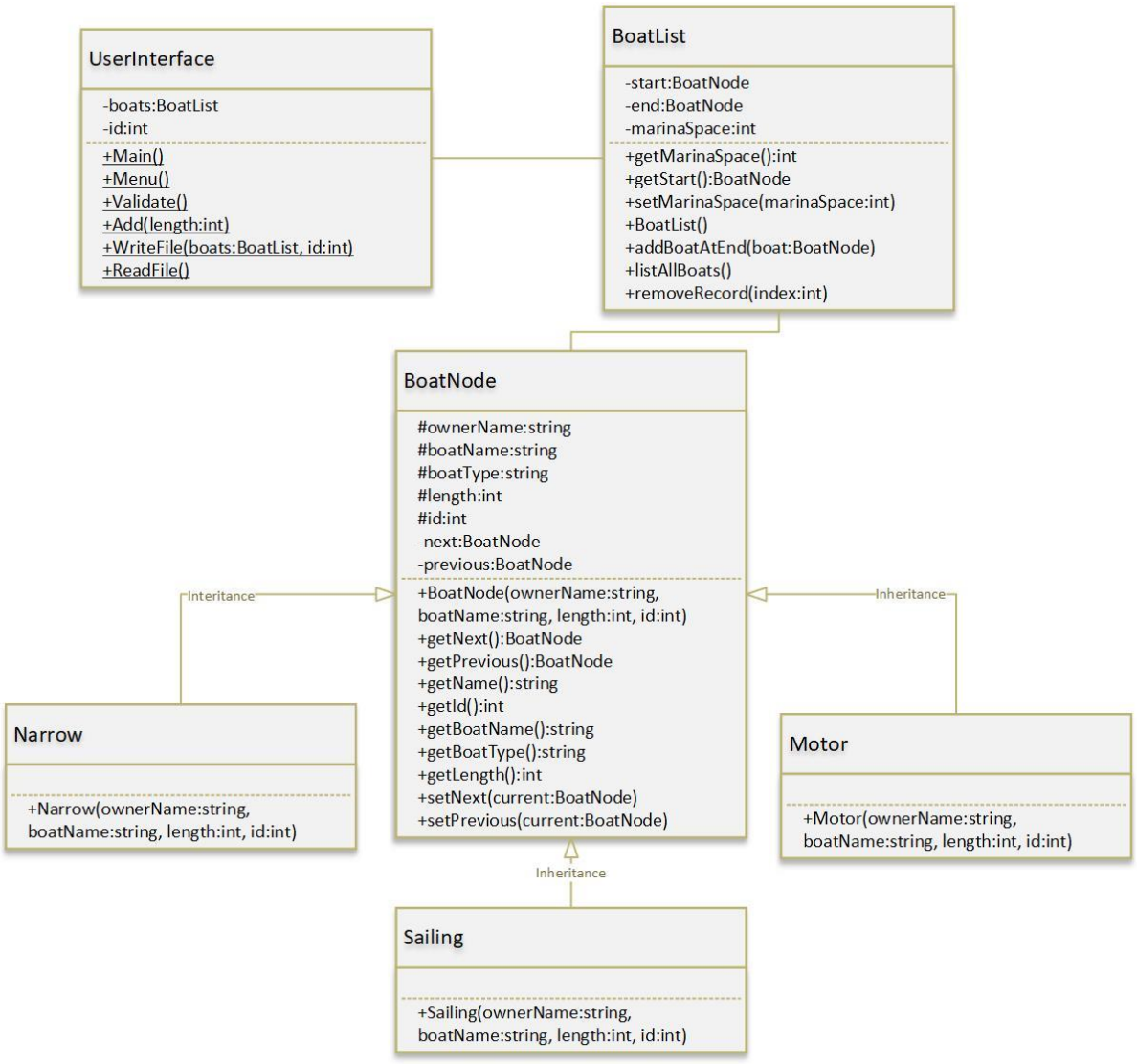

Figure 1: Class Diagram

When the program is opened, an empty linked list is created and an ID variable is initialised to zero. There is a check for a text file to see if any records already exist in the system. If the file is found, these records are added to the linked list, and the ID variable updated to the integer value stored on file. The user is then presented with a choice of Menu items – adding a record, viewing all records and berth space, removing a record or terminating the program.

If add is selected, the user is asked for boat dimensions and stay duration. Provided length is below 15m, depth is less than 5m and there's space for the boat in the canal, the user is presented with the booking cost to accept or reject. If they reject, they are returned to Menu. If they accept, the program requests further details which are then added in a boat object at the end of the linked list, along with a unique ID number generated by the program for that record.

If view records is selected, the information stored in each boat record of the linked list is displayed, along with the remaining marina space.

If delete is selected, the user is asked for the ID of the record to be deleted. If the user enters an ID that doesn't belong to an object in the linked list, they are prompted to enter the ID again. The record which has the ID input by the user is removed from the linked list. The user sees a display of the order that boats move out into the holding bay and then back into the marina so that the target boat can leave.

If quit is selected, the linked list and the ID number to be used for the next boat that books into the marina are written to file. The program terminates.

Conclusion

All program requirements specified are satisfied. Adding a unique ID for each record means it is possible to tell bookings apart should multiple boats share the same information.

The program would be improved by adding boat arrival and departure dates, allowing bookings to be made ahead of time. The manager could produce a calendar of how much space is available and when. As boats leave, the boats moved to the holding bay could re-enter the canal in a date determined order to minimise future movements.

Currently boat bookings are written to file on proper termination of the program. Should the program unexpectedly close, bookings or deletions made during that session are not saved. The program could be modified to write to file after every change to the linked list, however this would add extra processing time.

Word Count: 550

Appendix 1: User Instructions

Open the USB drive. Double click the “Coursework.exe – Shortcut” file. Follow the menu instructions to use booking program.

Appendix 2: C# Source Code

```
using System;
using System.IO;


namespace Coursework
{
	class UserInterface
	{
		private static BoatList boats = new BoatList();
		private static int id = 1;

		public static void Main()
		{
			// if data.txt file exists
			if (File.Exists("data.txt"))
			{
				UserInterface.ReadFile();
			}
			UserInterface.Menu();
			return;
		}

		public static void Menu()
		{
			char input = ' ';

			while (input != 'Q')
			{
				Console.WriteLine("");
				Console.WriteLine("-----------------------------------------");
				Console.WriteLine("What would you like to do? Select");
				Console.WriteLine("Q: Quit");
				Console.WriteLine("A: Add a booking");
				Console.WriteLine("V: View all records and available berth space");
				Console.WriteLine("D: Delete specific record");
				Console.WriteLine("-----------------------------------------");

				input = Console.ReadLine().ToUpper()[0];

				switch (input)
				{
					case 'A':
						UserInterface.Validate();
						break;
					case 'V':
						Console.WriteLine("View records and available berth
space");
						Console.WriteLine("ID: Owner - Boat - Type - Length (m)");
						Console.WriteLine("-----------------------------------
");
						boats.listAllBoats();
						Console.WriteLine("The marina space available is " +
boats.getMarinaSpace() + " metres.");
						break;
					case 'D':
						Console.WriteLine("Enter the id number of the record to be
deleted: ");
						try
						{

	boats.removeRecord(Convert.ToInt16(Console.ReadLine()));
						}
```

```
                                        catch (FormatException)
                                        {
                                                Console.WriteLine("Not a valid id number, returning
to Menu");
                                        }
                                        catch (NullReferenceException)
                                        {
                                                Console.WriteLine("ID number does not exist,
returning to Menu");
                                        }
                                        break;
                                case 'Q':
                                        break;
                                default:
                                        Console.WriteLine("Invalid input - returning to Menu.");
                                        break;
                        }
                }
                UserInterface.WriteFile(boats, id);
                Console.WriteLine("Choosing to quit");
                return;
        }

        public static void Validate()
        {
                int depth, length=0;
                bool lengthValid = false;
                while (lengthValid is false)
                {
                        try
                        {
                                Console.WriteLine("What is the boat length (to the nearest
meter)?");
                                length = Convert.ToInt32(Console.ReadLine());
                                lengthValid = true;
                                if (length > 15)
                                {
                                        Console.WriteLine("The boat is too long, the Marina only
accepts boats shorter than 15m.");
                                        return;
                                }

                                if (boats.getMarinaSpace() < length)
                                {
                                        Console.WriteLine("There is not enough space in the
Marina.");
                                        return;
                                }
                        }
                        catch (FormatException)
                        {
                                Console.WriteLine("Input an integer for boat length: ");
                        }
                }
                bool depthValid = false;
                while (depthValid is false)
                {
                        try
                        {
                                Console.WriteLine("What is the boat depth (to the nearest
meter)?");
                                depth = Convert.ToInt32(Console.ReadLine());
                                depthValid = true;
                                if (depth > 5)
                                {
```

```
                                        Console.WriteLine("The boat is too deep, the Marina only
accepts boats with a maximum depth of 5m");
                                        return;
                                }
                        }
                        catch (FormatException)
                        {
                                Console.WriteLine("Input an integer for depth: ");
                        }
                }
                UserInterface.Add(length);
        }

        public static void Add(int length)
        {
                char answer;
                int months=0, cost;
                string owner, boatName;
                bool boatType = false, monthsValid = false;

                while (monthsValid is false)
                {
                        try
                        {
                                Console.WriteLine("How many months is the booking for?");
                                months = Convert.ToInt32(Console.ReadLine());
                                monthsValid = true;
                        }
                        catch (FormatException)
                        {
                                Console.WriteLine("Input an integer for number of months.");
                        }
                }
                cost = 15 * length * months;
                Console.WriteLine("The cost of the booking is £" + cost + ". Would you like to
book?");
                Console.WriteLine("Enter Yes or No");
                answer = Console.ReadLine().ToUpper()[0];
                if (answer != 'Y')
                {
                        return;
                }
                Console.WriteLine("What's the name of the owner?");
                owner = Console.ReadLine();
                Console.WriteLine("What's the name of the boat?");
                boatName = Console.ReadLine();
                while (boatType == false)
                {
                        Console.WriteLine("What type of boat is it?");
                        Console.WriteLine("Enter N for narrow, S for sailing, M for Motor: ");
                        switch (Console.ReadLine().ToUpper()[0])
                        {
                                case 'N':
                                        Narrow narrow = new Narrow(owner, boatName, length, id);
                                        boats.addBoatAtEnd(narrow);
                                        id += 1;
                                        boatType = true;
                                        break;
                                case 'S':
                                        Sailing sailing = new Sailing(owner, boatName, length,
id);
                                        boats.addBoatAtEnd(sailing);
                                        id += 1;
                                        boatType = true;
                                        break;
```

```
                                case 'M':
                                        Motor motor = new Motor(owner, boatName, length, id);
                                        boats.addBoatAtEnd(motor);
                                        id += 1;
                                        boatType = true;
                                        break;
                                default:
                                        Console.WriteLine("Incorrect boat type input, please try
again.");
                                        break;
                        }
                }

        }

        public static void WriteFile(BoatList boats, int id)
        {
                StreamWriter writer;
                try
                {
                        writer = new StreamWriter("data.txt");
                        writer.WriteLine(id);

                        BoatNode current = boats.getStart();
                        while (current != null)
                        {
                                writer.WriteLine(current.getId().ToString() + "," +
current.getName() + "," + current.getBoatName() + "," + current.getBoatType() + "," +
current.getLength().ToString());
                                current = current.getNext();
                        }
                        writer.Close();
                }
                catch (IOException e)
                {
                        Console.WriteLine("Error writing to file.");
                }
        }

        public static void ReadFile()
        {
                string line;
                string[] inputs;
                try
                {
                        StreamReader reader = new StreamReader(@"data.txt");
                        id = Convert.ToInt16(reader.ReadLine());

                        while (reader.EndOfStream == false)
                        {
                                line = reader.ReadLine();
                                inputs = line.Split(',');
                                if (inputs[3] == "Sailing")
                                {
                                        Sailing boat = new Sailing(inputs[1], inputs[2],
Convert.ToInt16(inputs[4]), Convert.ToInt16(inputs[0]));
                                        boats.addBoatAtEnd(boat);
                                }
                                else if (inputs[3] == "Narrow")
                                {
                                        Narrow boat = new Narrow(inputs[1], inputs[2],
Convert.ToInt16(inputs[4]), Convert.ToInt16(inputs[0]));
                                        boats.addBoatAtEnd(boat);
                                }
                                else if (inputs[3] == "Motor")
```

```
                                {
                                        Motor boat = new Motor(inputs[1], inputs[2],
Convert.ToInt16(inputs[4]), Convert.ToInt16(inputs[0]));
                                        boats.addBoatAtEnd(boat);
                                }
                        }
                        reader.Close();
                }
                catch(IOException e)
                {
                        Console.WriteLine("File input error");
                }
        }
}
class BoatNode
{
        protected string ownerName, boatName, boatType = " ";
        protected int length, id;
        private BoatNode next, previous;

        public BoatNode(string ownerName, string boatName, int length, int id)
        {
                this.id = id;
                this.ownerName = ownerName;
                this.boatName = boatName;
                this.length = length;
        }

        public BoatNode getNext()
        {
                return next;
        }

        public BoatNode getPrevious()
        {
                return previous;
        }

        public string getName()
        {
                return ownerName;
        }
        public int getId()
        {
                return id;
        }
        public string getBoatName()
        {
                return boatName;
        }
        public string getBoatType()
        {
                return boatType;
        }
        public int getLength()
        {
                return length;
        }
        public void setNext(BoatNode current)
        {
                next = current;
        }
        public void setPrevious(BoatNode current)
        {
                previous = current;
```

```
		}
	}
	class Narrow : BoatNode
	{
		public Narrow(string ownerName, string boatName, int length, int id) : base(ownerName,
boatName, length, id)
		{
			boatType = "Narrow";
		}
	}
	class Sailing : BoatNode
	{
		public Sailing(string ownerName, string boatName, int length, int id) :
base(ownerName, boatName, length, id)
		{
			boatType = "Sailing";
		}
	}
	class Motor : BoatNode
	{
		public Motor(string ownerName, string boatName, int length, int id) : base(ownerName,
boatName, length, id)
		{
			boatType = "Motor";
		}
	}
	class BoatList
	{
		private BoatNode start;
		private BoatNode end;
		private int marinaSpace = 150;

		public int getMarinaSpace()
		{
			return marinaSpace;
		}
		public BoatNode getStart()
		{
			return start;
		}

		public void setMarinaSpace(int marinaSpace)
		{
			this.marinaSpace = marinaSpace;
		}

		public BoatList()
		{
			start = null;
			end = null;
		}
		public void addBoatAtEnd(BoatNode boat)
		{
			BoatNode current = boat;

			if (end == null)
			{
				start = current;
				end = current;
			}
			else
			{
				boat.setPrevious(end);
				end.setNext(current);
				end = current;
```

```
                }
                setMarinaSpace(getMarinaSpace() - boat.getLength());
        }
        public void listAllBoats()
        {
                BoatNode current = start;
                while (current != null)
                {
                        Console.WriteLine(current.getId() + ": " + current.getName() + " - " +
current.getBoatName() + " - " + current.getBoatType() + " - " + current.getLength());
                        current = current.getNext();
                }
        }

        public void removeRecord(int index)
        {
                BoatNode current = end;
                if (current == null)
                {
                        Console.WriteLine("There are no boat booking records to be removed");
                        return;
                }

                if (current.getId() == index) // if boat to be removed is at the end of the
list
                {
                        setMarinaSpace(marinaSpace + current.getLength());
                        end = current.getPrevious();
                        end.setNext(null);
                        Console.WriteLine("Boat " + current.getBoatName() + " leaves without any
others needing to move out of the way.");
                }

                else
                {
                        while (current != null)
                        {
                                Console.WriteLine("Boat " + current.getBoatName() + " moves to
holding bay.");
                                if (current.getPrevious().getId() == index)
                                {
                                        if (current.getPrevious() == start)
                                        {
                                                start = current;
                                        }
                                        else
                                        {

        current.getPrevious().getPrevious().setNext(current);
                                        }
                                        Console.WriteLine("Boat " +
current.getPrevious().getBoatName() + " leaves the Marina.");
                                        setMarinaSpace(marinaSpace +
current.getPrevious().getLength());
                                        current.setPrevious(current.getPrevious().getPrevious());
                                        while (current != null)
                                        {
                                                Console.WriteLine("Boat " + current.getBoatName() +
" moves back into the Marina.");
                                                current = current.getNext();
                                        }
                                        return;
                                }
                                else
                                {
```

```
                                        current = current.getPrevious();
                                    }
                                }
                            }
                        }
                    }
}
```