/\*MILESTONE 2

Kelly Tiedt (602730) - Q2

Jaden Van der lely (600690) - Q4, Q2

Jonathan Joubert (578085) - Q1, Q3

Mamello Lelaka (577497) - Q2

Marco Brazao de Sousa (601587) - Q5

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace Retroslice\_M1

{

public class Details // Jonathan Joubert 578085

{

// Initialising applicant details

public string Name { get; set; }

public int Age { get; set; }

public int Rank { get; set; }

public DateTime Date { get; set; }

public int Pizza { get; set; }

public int Score { get; set; }

public bool Employment { get; set; }

public string SlushPreference { get; set; }

public int SlushConsumed { get; set; }

// Constructor to assign applicant details

public Details(string name, int age, int rank, DateTime date, int pizza, int score, bool employment, string slushPreference, int slushConsumed)

{

Name = name;

Age = age;

Rank = rank;

Date = date;

Pizza = pizza;

Score = score;

Employment = employment;

SlushPreference = slushPreference;

SlushConsumed = slushConsumed;

}

}

public class Retro // Jonathan Joubert 578085

{

// Method to capture the applicant's details and store them in a collection

public static List<Details> GetDetails()

{

var applicants = new List<Details>();

bool isTrue = true;

var currentColor = Console.ForegroundColor;

while (isTrue)

{

Console.WriteLine("");

Console.WriteLine("Type in the applicant's details");

Console.Write("Name: ");

string name = Console.ReadLine();

while (string.IsNullOrEmpty(name) || !IsAlphabetic(name))

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Your name must be in alphabetical characters.");

Console.ForegroundColor = currentColor;

Console.Write("Name: ");

name = Console.ReadLine();

}

Console.Write("Age: ");

string ageInput = Console.ReadLine();

int age;

while (!int.TryParse(ageInput, out age) || int.Parse(ageInput) > 99)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("You must enter a valid integer age.");

Console.ForegroundColor = currentColor;

Console.Write("Age: ");

ageInput = Console.ReadLine();

}

Console.Write("Rank: ");

string rankInput = Console.ReadLine();

int rank;

while (!int.TryParse(rankInput, out rank))

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("You must enter a valid rank.");

Console.ForegroundColor = currentColor;

Console.Write("Rank: ");

rankInput = Console.ReadLine();

}

Console.Write("Starting date (YYYY/MM/DD): ");

string dateInput = Console.ReadLine();

DateTime date;

DateTime currentDate = DateTime.Now;

DateTime birthDate = currentDate.AddYears(-age);

while (!DateTime.TryParse(dateInput, out date) || birthDate > date)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("You must enter a valid date.");

Console.Write("Starting date (YYYY/MM/DD): ");

Console.ForegroundColor = currentColor;

dateInput = Console.ReadLine();

}

Console.Write("Amount of pizzas eaten: ");

string pizzaInput = Console.ReadLine();

int pizza;

while (!int.TryParse(pizzaInput, out pizza))

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("You must enter a valid integer number.");

Console.ForegroundColor = currentColor;

Console.Write("Amount of pizzas eaten: ");

pizzaInput = Console.ReadLine();

}

Console.Write("Bowling high score: ");

string scoreInput = Console.ReadLine();

int score;

while (!int.TryParse(scoreInput, out score))

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("You must enter a valid integer number.");

Console.ForegroundColor = currentColor;

Console.Write("Bowling high score: ");

scoreInput = Console.ReadLine();

}

Console.Write("Are they employed? (yes or no): ");

string employmentInput = Console.ReadLine().ToLower();

while (employmentInput != "yes" && employmentInput != "no")

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("You must enter yes or no.");

Console.ForegroundColor = currentColor;

Console.Write("Are they employed? (yes or no): ");

employmentInput = Console.ReadLine().ToLower();

}

bool employment = employmentInput == "yes";

Console.Write("Favourite slush puppy flavour: "); // This part needs to be fixed if you run the program it allows you to enter digits in this field (check here and check method)

string slushPreference = Console.ReadLine();

while (string.IsNullOrEmpty(slushPreference) || !IsAlphabetic(slushPreference))

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("You must enter a flavour/colour.");

Console.ForegroundColor = currentColor;

Console.Write("Favourite slush puppy flavour: ");

slushPreference = Console.ReadLine();

}

Console.Write("Amount of slush puppies drunk: ");

string slushConsumedInput = Console.ReadLine();

int slushConsumed;

while (!int.TryParse(slushConsumedInput, out slushConsumed))

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("You must enter an amount.");

Console.ForegroundColor = currentColor;

Console.Write("Amount of slush puppies drunk: ");

slushConsumedInput = Console.ReadLine();

}

var applicant = new Details(name, age, rank, date, pizza, score, employment, slushPreference, slushConsumed);

applicants.Add(applicant);

Console.WriteLine();

Console.Write("New application? (yes or no): ");

isTrue = Console.ReadLine().ToLower() == "yes";

}

return applicants;

}

private static bool IsAlphabetic(string input)

{

foreach (char c in input)

{

if (!char.IsLetter(c) && !char.IsWhiteSpace(c) && char.IsDigit(c))

{

return false;

}

}

return true;

}

// Method to find the youngest and oldest applicants

public static void GetAgeRange(List<Details> applicants)

{

if (applicants == null || applicants.Count == 0)

{

Console.WriteLine("No applicants to evaluate.");

return;

}

int youngest = int.MaxValue;

int oldest = int.MinValue;

foreach (var applicant in applicants)

{

if (applicant.Age < youngest)

{

youngest = applicant.Age;

}

if (applicant.Age > oldest)

{

oldest = applicant.Age;

}

}

Console.WriteLine($"Youngest Applicant: {youngest}");

Console.WriteLine($"Oldest Applicant: {oldest}");

}

}

public static class LoadingAnimationUtility

{

public static void ShowLoadingAnimation(int duration)

{

Console.Write("Loading");

for (int i = 0; i < duration / 250; i++) // Convert milliseconds to quarter-seconds for faster animation

{

Console.Write(".");

Thread.Sleep(250);

}

Console.WriteLine();

}

}

// Qualified customers method

public class CreditCheck // Kelly Tiedt 602730

{

public static List<Details> GetQualifiedApplicants(List<Details> applicants)

{

var qualifiedApplicants = new List<Details>();

//Loading animation

Console.WriteLine("Getting qualified applicants");

LoadingAnimationUtility.ShowLoadingAnimation(3000);

foreach (var applicant in applicants)

{

bool isQualified = true;

// If the customer is a long term loyal customer then they automatically qualify for credit

if (longtermloyal(applicant))

{

qualifiedApplicants.Add(applicant);

continue;

}

// Check employment status

if (!applicant.Employment && applicant.Age >= 18)

{

isQualified = false;

Console.Write(isQualified);

}

// Check loyalty duration (2 years)

DateTime startingDate = applicant.Date;

if ((DateTime.Now - startingDate).TotalDays / 365 < 2)

{

isQualified = false;

Console.Write(isQualified);

}

// Check high score rank and bowling high score

int highScoreRank = applicant.Rank;

int bowlingHighScore = applicant.Score;

if (highScoreRank <= 2000 || bowlingHighScore <= 1500)

{

if ((highScoreRank + bowlingHighScore) / 2 <= 1200)

{

isQualified = false;

}

}

// Check pizza consumption (at least 3 per month)

DateTime now = DateTime.Now;

int monthsAsCustomer = ((now.Year - startingDate.Year) \* 12) + now.Month - startingDate.Month;

if (monthsAsCustomer == 0) monthsAsCustomer = 1; // To avoid division by zero

if (applicant.Pizza / monthsAsCustomer < 3)

{

isQualified = false;

}

// Check slush-puppy consumption (at least 4 per month)

if (applicant.SlushConsumed / monthsAsCustomer < 4)

{

isQualified = false;

}

// Check slush-puppy preference

if (applicant.SlushPreference == "Gooey Gulp Galore")

{

isQualified = false;

}

if (isQualified)

{

qualifiedApplicants.Add(applicant);

}

}

return qualifiedApplicants;

}

public static bool longtermloyal(Details applicant)

{

DateTime datestart = applicant.Date;

if ((DateTime.Now - datestart).TotalDays / 365 >= 10)

{

return true;

}

return false;

}

}

public class StatsDisplay // Mamello Lelaka 577497

{

public static void ShowStats(List<Details> applicants, List<Details> qualifiedApplicants)

{

//Loading animation

Console.WriteLine("Displaying stats");

LoadingAnimationUtility.ShowLoadingAnimation(2000);

Console.WriteLine("Total Applicants: " + applicants.Count);

Console.WriteLine("Qualified Applicants: " + qualifiedApplicants.Count);

Console.WriteLine("Denied Applicants: " + (applicants.Count - qualifiedApplicants.Count));

var currentColor = Console.ForegroundColor;

if (qualifiedApplicants.Count > 0)

{

Console.WriteLine("");

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("Congradulations you have qualified");

Console.ForegroundColor = currentColor;

}

}

}

public class PizzaAvg //milestone 2 Jonathan Joubert 578085

{

public static double CalculateAveragePizzasEaten(List<Details> applicants)

{

//Loading animation

Console.WriteLine("Calculating average pizzas eaten...");

LoadingAnimationUtility.ShowLoadingAnimation(4000);

if (applicants == null || applicants.Count == 0)

{

return 0;

}

double days = 0;

double total = 0;

foreach (var applicant in applicants)

{

int customerdays = (DateTime.Now - applicant.Date).Days;

if (customerdays == 0) customerdays = 1; // this is to avoid dividing by 0 which will cause an error

total += applicant.Pizza;

days += customerdays;

}

double average = total / days;

return average;

}

internal class Program

{

// Enum defined

enum MenuOption

{

GetDetails = 1,

CheckCreditQualification,

ShowStats,

ShowPizzaAvg,

longterm,

ClearScreen,

GetAgeRange,

Exit

}

static void Main(string[] args)

{

// Created a list for saving applicants and also a list of customers who qualified

List<Details> applicants = new List<Details>();

List<Details> qualifiedApplicants = new List<Details>();

bool exit = false;

// The while loop will continue until the Exit option is chosen in the Menu

while (!exit)

{

Console.WriteLine("Retroslice Application Capture System");

Console.WriteLine("1. Capture Details");

Console.WriteLine("2. Check Game Token Credit Qualification");

Console.WriteLine("3. Show Current Arcade & Bowling Stats");

Console.WriteLine("4. Show the average pizzas consumed");

Console.WriteLine("5. Long term loyalty");

Console.WriteLine("6. Clear current page");

Console.WriteLine("7. Sort Ages");

Console.WriteLine("8. Exit");

Console.Write("Choose an option: ");

// Using parse to convert string to integer for user input

bool validInput = int.TryParse(Console.ReadLine(), out int input); // I added validation for the option so it stops crashing without proper input

if (!validInput || !Enum.IsDefined(typeof(MenuOption), input))

{

Console.WriteLine("");

Console.WriteLine("Please select a valid option!");

Console.WriteLine("");

continue;

}

MenuOption option = (MenuOption)input;

int milliseconds = 1000;

// Using a switch case method to determine user inputs using the enum

switch (option)

{

// Option 1 in Enum

case MenuOption.GetDetails:

applicants = Retro.GetDetails();

Console.WriteLine("");

Console.Write("Wait a moment while we process data ");

Thread.Sleep(milliseconds);

Console.Write(".");

Thread.Sleep(milliseconds);

Console.Write(".");

Thread.Sleep(milliseconds);

Console.WriteLine(".");

Thread.Sleep(milliseconds);

break;

// Option 2 of Enum

case MenuOption.CheckCreditQualification:

qualifiedApplicants = CreditCheck.GetQualifiedApplicants(applicants);

Console.WriteLine("");

Console.WriteLine("Credit qualification check completed.");

break;

// Option 3 of Enum

case MenuOption.ShowStats:

Console.WriteLine("");

StatsDisplay.ShowStats(applicants, qualifiedApplicants);

break;

//Option no.4 of the enum

case MenuOption.ShowPizzaAvg:

double averagePizzas = PizzaAvg.CalculateAveragePizzasEaten(applicants);

Console.WriteLine("");

Console.WriteLine("Average pizzas eaten everyday since join: " + Math.Round(averagePizzas, 2));

break;

//option 5 of the enum

case MenuOption.longterm:

foreach (var applicant in applicants)

{

bool criteria = CreditCheck.longtermloyal(applicant);

string qualified;

string meet;

if (criteria == false)

{

qualified = "Unfortunately, ";

meet = "does not meet ";

}

else

{

qualified = "Congratulations! ";

meet = "meets ";

}

Console.WriteLine("");

Console.WriteLine($"{qualified}{applicant.Name} {meet}the criteria for Long-Term Loyalty.");

}

break;

//option 6 of the enum

case MenuOption.ClearScreen:

Console.Clear();

break;

// Option 7 of Enum - New Option for Age Range

case MenuOption.GetAgeRange:

Retro.GetAgeRange(applicants);

break;

// Option 8 of Enum

case MenuOption.Exit:

Console.WriteLine("");

Console.WriteLine("Are you sure you want to exit. (yes/no)");

string ResponseExitCheck = Console.ReadLine();

if (ResponseExitCheck == "yes")

{

exit = true;

break;

}

break;

}

Console.WriteLine();

}

}

}

}

}