Exam Code

```
using System;
public interface IExam
{
   int CheckExam(string[] correctAnswers, string[] studentAnswers);
}
public class Exam : IExam
{
   public int CheckExam(string[] correctAnswers, string[] studentAnswers)
   {
      int score = 0;
      int correctStreak = 0;
      int incorrectStreak = 0;
      int length = Math.Min(correctAnswers.Length, studentAnswers.Length);
      for (int i = 0; i < length; i++)
      {
         string correct = correctAnswers[i];
         string student = studentAnswers[i];
         if (string.IsNullOrEmpty(student))
         {
            // Blank answer: 0 points
            correctStreak = 0;
            incorrectStreak = 0;
         }
         else if (student == correct)
         {
            correctStreak++;
            score += 4 + (correctStreak - 1) * 2;
            incorrectStreak = 0;
         }
         else
         {
            incorrectStreak++;
            score -= (1 + (incorrectStreak - 1));
            correctStreak = 0;
         }
```

```
        }
        if (score < 0)
            return 0;
        int maxScore = correctAnswers.Length * 4;
        return Math.Min(16, maxScore);
    }
}
```

<u>CensusData (String)</u>

```
using System;
using System.Collections.Generic;
public class CensusData
{
    public int FamilySize { get; set; }
    public string Occupation { get; set; }
    public int Income { get; set; }
}
public class OccupationStats
{
    public int MinIncome { get; set; } = int.MaxValue;
    public int MaxIncome { get; set; } = int.MinValue;
    public int MinFamilySize { get; set; } = int.MaxValue;
    public int MaxFamilySize { get; set; } = int.MinValue;
}
public class CalculateOccupationStats
{
    public Dictionary<string, OccupationStats> CalculateStats(string[]
censusData, string[] uniqueOccupations)
    {
        var stats = new Dictionary<string, OccupationStats>();

        foreach (string entry in censusData)
        {
            var parts = entry.Split(',');
            int familySize = int.Parse(parts[1]);
            string occupation = parts[2];
            int income = int.Parse(parts[3]);

            if (!stats.ContainsKey(occupation))
                stats[occupation] = new OccupationStats();

            var occStat = stats[occupation];

            occStat.MinIncome = Math.Min(occStat.MinIncome, income);
            occStat.MaxIncome = Math.Max(occStat.MaxIncome, income);
            occStat.MinFamilySize = Math.Min(occStat.MinFamilySize,
familySize);
            occStat.MaxFamilySize = Math.Max(occStat.MaxFamilySize,
familySize);
        }
        return stats;
    }
}
```

<u>Computer</u>

```
using System;

public abstract class Computer
{
    protected string type;
    protected string model;
    protected string cpu;
    protected bool isTurnedOn = false;

    public Computer(string type, string model, string cpu)
    {
        this.type = type;
        this.model = model;
        this.cpu = cpu;
    }
    public string GetComputerType()
    {
        return type;
    }
    public string GetComputerModel()
    {
        return model;
    }
    public string GetComputerCpu()
    {
        return cpu;
    }
    public bool GetComputerStatus()
    {
        return isTurnedOn;
    }
    public void SwitchComputerStatus()
    {
        isTurnedOn = !isTurnedOn;
    }
}
public class PersonalComputer : Computer
{
    public PersonalComputer(string model, string cpu)
        : base("Personal Computer", model, cpu)
    {
    }
}
public class Notebook : Computer
{
    public Notebook(string model, string cpu)
        : base("Notebook", model, cpu)
    {
    }
}
```

<u>RealEstate</u>

```
using System;
using System.Collections.Generic;
using System.Linq;

// Interface for RealEstateListing (optional, but good practice)
public interface IRealEstateListing
{
    int Id { get; set; }
    string Title { get; set; }
    string Description { get; set; }
    int Price { get; set; }
    string Location { get; set; }
}

public class RealEstateListing : IRealEstateListing
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public int Price { get; set; }
    public string Location { get; set; }

    public override string ToString()
    {
        return $"ID: {Id}, Title: {Title}, Price: {Price}, Location: {Location}";
    }
}
// Interface for the RealEstateApp
public interface IRealEstateApp
{
    void AddListing(RealEstateListing listing);
    void RemoveListing(int listingID);
    void UpdateListing(RealEstateListing listing);
    List<RealEstateListing> GetAllListings();
    List<RealEstateListing> GetListingsByLocation(string location);
    List<RealEstateListing> GetListingsByPriceRange(int minPrice, int
maxPrice);
}
```

```
// Class to manage real estate listings
public class RealEstateApp : IRealEstateApp
{
   private List<RealEstateListing> listings = new List<RealEstateListing>();

   public void AddListing(RealEstateListing listing)
   {
      listings.Add(listing);
   }

   public void RemoveListing(int listingID)
   {
      listings.RemoveAll(listing => listing.Id == listingID);
   }

   public void UpdateListing(RealEstateListing listing)
   {
      var existingListing = listings.FirstOrDefault(l => l.Id == listing.Id);
      if (existingListing != null)
      {
         existingListing.Title = listing.Title;
         existingListing.Description = listing.Description;
         existingListing.Price = listing.Price;
         existingListing.Location = listing.Location;
      }
   }

   public List<RealEstateListing> GetAllListings()
   {
      return listings;
   }

   public List<RealEstateListing> GetListingsByLocation(string location)
   {
      return listings.Where(listing => listing.Location.Equals(location,
StringComparison.OrdinalIgnoreCase)).ToList();
   }
```

```
    public List<RealEstateListing> GetListingsByPriceRange(int minPrice, int
maxPrice)
    {
        return listings.Where(listing => listing.Price >= minPrice && listing.Price
<= maxPrice).ToList();
    }
}
```

<u>Employee</u>

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

abstract class Employee
{
    protected string department;
    protected string name;
    protected string location;
    protected bool isOnVacation = false;
    public Employee(string department, string name, string location)
    {
        this.department = department;
        this.name = name;
        this.location = location;
    }
    public string GetDepartment()
    {
        return department;
    }
    public string GetName()
    {
        return name;
    }
    public string GetLocation()
    {
        return location;
    }
    public bool GetStatus()
    {
        return isOnVacation;
    }
    public void SwitchStatus()
    {
        isOnVacation = !isOnVacation;
    }
}
class FinanceEmployee : Employee
{
    public FinanceEmployee(string name, string location)
        : base("Finance", name, location) {}
}

class MarketingEmployee : Employee
{
    public MarketingEmployee(string name, string location)
        : base("Marketing", name, location) {}
}
```

<u>User (Admin & Moderator)</u>

```
using System;

public enum Gender
{
    Male,
    Female,
    Other
}

public abstract class User
{
    private string type;
    private string name;
    private Gender gender;
    private int age;

    public User(string type, string name, Gender gender, int age)
    {
        this.type = type;
        this.name = name;
        this.gender = gender;
        this.age = age;
    }

    public string GetUserType()
    {
        return type;
    }

    public string GetUserName()
    {
        return name;
    }

    public Gender GetGender()
    {
        return gender;
    }

    public int GetAge()
    {
        return age;
    }
}

public class Admin : User
{
    public Admin(string name, Gender gender, int age)
```

```
        : base("Admin", name, gender, age)
    {
    }
}

public class Moderator : User
{
    public Moderator(string name, Gender gender, int age)
        : base("Moderator", name, gender, age)
    {
    }
}
```

<u>Point2D</u>

```
class Point2D {
    protected int x, y;

    public Point2D(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public virtual double dist2D(Point2D p) {
        int dx = this.x - p.x;
        int dy = this.y - p.y;
        return Math.Sqrt(dx * dx + dy * dy);
    }

    public virtual void printDistance(double d) {
        Console.WriteLine("2D distance = " + d);
    }
}

class Point3D : Point2D {
    protected int z;
    public Point3D(int x, int y, int z) : base(x, y) {
        this.z = z;
    }

    public double dist3D(Point3D p) {
        int dx = this.x - p.x;
        int dy = this.y - p.y;
        int dz = this.z - p.z;
        return Math.Sqrt(dx * dx + dy * dy + dz * dz);
    }

    public override void printDistance(double d) {
        Console.WriteLine("3D distance = " + Math.Ceiling(d));
    }
}
```

List<CensusData>

```
using System;
using System.Collections.Generic;
using System.Linq;

public class CensusData
{
    public string SSN { get; set; }
    public int FamilySize { get; set; }
    public string Occupation { get; set; }
    public int Income { get; set; }

    public CensusData(string ssn, int familySize, string occupation, int income)
    {
        SSN = ssn;
        FamilySize = familySize;
        Occupation = occupation;
        Income = income;
    }
```

```
}

public class OccupationStats
{
   public int MinIncome { get; set; }
   public int MaxIncome { get; set; }
   public int MinFamilySize { get; set; }
   public int MaxFamilySize { get; set; }

   public OccupationStats(int minIncome, int maxIncome, int minFamilySize,
int maxFamilySize)
   {
      MinIncome = minIncome;
      MaxIncome = maxIncome;
      MinFamilySize = minFamilySize;
      MaxFamilySize = maxFamilySize;
   }

   public override string ToString()
   {
      return $"Min Income: {MinIncome}, Max Income: {MaxIncome}, Min
Family Size: {MinFamilySize}, Max Family Size: {MaxFamilySize}";
   }
}

public class CalculateOccupationStats
{
   public static Dictionary<string, OccupationStats>
GetOccupationStats(List<CensusData> censusData, string[]
uniqueOccupations)
   {
      var stats = new Dictionary<string, OccupationStats>();
      foreach (var occupation in uniqueOccupations)
      {
         var filtered = censusData
            .Where(c => c.Occupation != null && c.Occupation.ToLower() ==
occupation.ToLower())
```

```
                .ToList();

            if (filtered.Any())
            {
                int minIncome = filtered.Min(c => c.Income);
                int maxIncome = filtered.Max(c => c.Income);
                int minFamilySize = filtered.Min(c => c.FamilySize);
                int maxFamilySize = filtered.Max(c => c.FamilySize);
                stats[occupation] = new OccupationStats(minIncome, maxIncome,
minFamilySize, maxFamilySize);
            }
        }
        return stats;
    }
}
-------------------------------------------------------------------------
```

<u>**Animal Zoo**</u>

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

public interface IAnimal
{
    int Id { get; set; }
    string Species { get; set; }
    string Name { get; set; }
    int Age { get; set; }
}

public interface IZoo
{
    void AddAnimal(IAnimal animal);
    void RemoveAnimal(int id);
    int CountAnimals();
    List<IAnimal> GetAnimalsBySpecies(string species);
    List<(int, List<IAnimal>)> GetAnimalsByAge();
}

public class Animal : IAnimal
{
    public int Id { get; set; }
    public string Species { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}

public class Zoo : IZoo
{
    private List<IAnimal> _animals = new List<IAnimal>();

    public void AddAnimal(IAnimal animal)
    {
        _animals.Add(animal);
    }

    public void RemoveAnimal(int id)
    {
        _animals.RemoveAll(a => a.Id == id);
    }

    public int CountAnimals()
    {
        return _animals.Count;
    }

    public List<IAnimal> GetAnimalsBySpecies(string species)
    {
        return _animals.Where(a => a.Species == species).ToList();
    }

    public List<(int, List<IAnimal>)> GetAnimalsByAge()
    {
        return _animals
            .GroupBy(a => a.Age)
            .Select(g => (g.Key, g.ToList()))
            .OrderByDescending(g => g.Key)
            .ToList();
    }
}
```

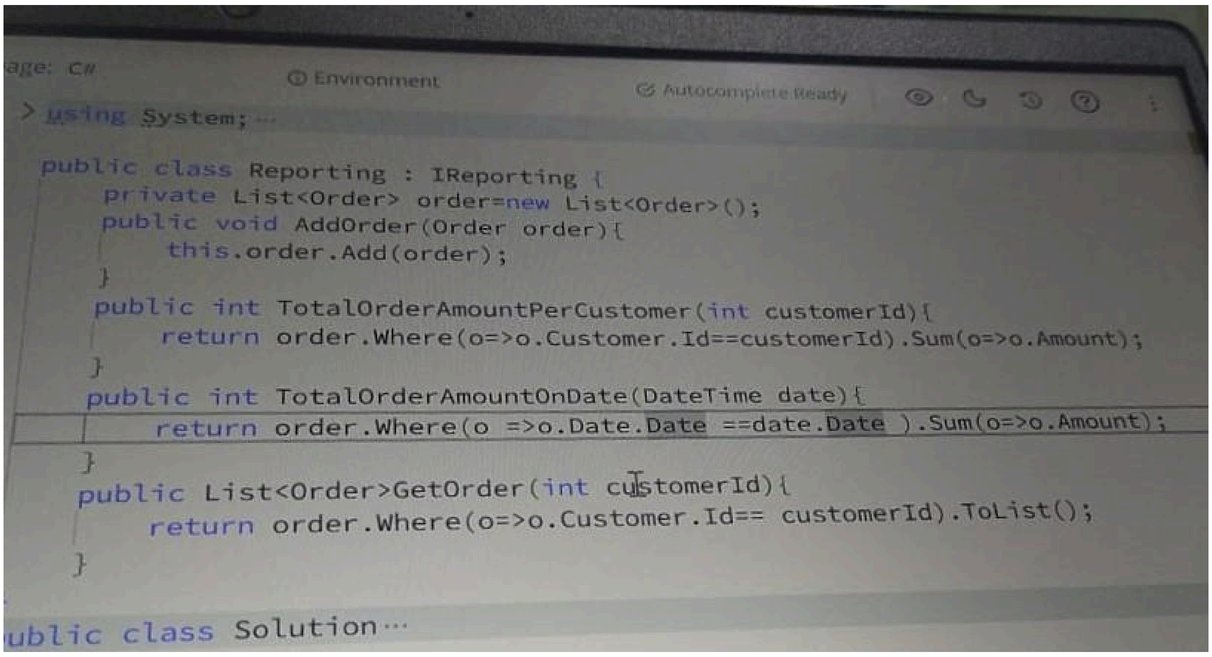
Edge

```
using System;

public class Edge
{
    public int FromNode { get; }
    public int ToNode { get; }

    private Edge(int fromNode, int toNode)
    {
        FromNode = fromNode;
        ToNode = toNode;
    }

    public static Edge CreateEdge(int fromNode, int toNode)
    {
        return new Edge(fromNode, toNode);
    }

    public override bool Equals(object obj)
    {
        if (obj == null || GetType() != obj.GetType())
        {
            return false;
        }

        Edge other = (Edge)obj;
        return (FromNode == other.FromNode && ToNode == other.ToNode);
    }

    public int CompareTo(Edge other)
    {
        if (FromNode != other.FromNode)
        {
            return FromNode.CompareTo(other.FromNode);
        }
        return ToNode.CompareTo(other.ToNode);
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(FromNode, ToNode);
    }

    public override string ToString()
    {
        return $"From node: <{FromNode}> To node: <{ToNode}>";
    }
```

StockPrediction

```
using System;

public class StockPrediction
{
    public int Gain { get; set; }
    public int StockPrice { get; set; }

    public StockPrediction(int gain, int stockPrice)
    {
        Gain = gain;
        StockPrice = stockPrice;
    }
    // Default implementation of expectedValue
    public int ExpectedValue()
    {
        return StockPrice + Gain;
    }

    public int ExpectedValue(int transactionCosts)
    {
        return (StockPrice + Gain) - transactionCosts;
    }

    // Overload 2: Takes a string transactionCosts as input
    public int ExpectedValue(string transactionCosts)
    {
        if (int.TryParse(transactionCosts, out int costs))
        {
            return (StockPrice + Gain) - costs;
        }
        else
        {
            Console.WriteLine("Error: Invalid transaction costs format.");
            return StockPrice + Gain;
        }
    }
}
```

Reporting



```
> using System;...

public class Reporting : IReporting {
    private List<Order> order=new List<Order>();
    public void AddOrder(Order order){
        this.order.Add(order);
    }
   public int TotalOrderAmountPerCustomer(int customerId){
       return order.Where(o=>o.Customer.Id==customerId).Sum(o=>o.Amount);
   }
  public int TotalOrderAmountOnDate(DateTime date){
      return order.Where(o =>o.Date.Date ==date.Date ).Sum(o=>o.Amount);
  }
  public List<Order>GetOrder(int customerId){
      return order.Where(o=>o.Customer.Id== customerId).ToList();
  }

ublic class Solution...
```

<u>Film</u>

```
using System
public class Film:IFilm
{
   public string Title {get;set;}
   public int Year {get;set;}
   public string Genre {get;set;}
   public bool OscarNominated {get;set;}
   public int StreamingCount {get;set;}
}
public class FilmLibrary
{
   private List<Film> films=new List<Film>();
```

```
    public void AddFilm(Film film)
    {
        films.Add(film);
    }
    public List<Film> GetFilms()
    {
        return films;
    }
    public List<Film> FilterByGenre(string genre)
    {
        return films.Where(f =>
f.Genre.Equals(genre,StringComparision.OrdinalIgnoreCase)).ToList();
    }
    public List<Film> FilterByYear(int year)
    {
        return films.Where(f => f.Year == year).ToList();
    }
    public List<Film> FilterByOscarNominations(bool isNominted)
    {
        return films.Where(f => f.OscarNominated == isNominated).ToList();
    }
    public List<Film> FilterByStreamingCount(int streamingCount)
    {
        return films.Where(f => f.StreamingCount == streamingCount).ToList();
    }
}
```

Coffee ShopBill

```
using System;
using System.Collections.Generic;
using System.Linq;

// Abstract base class
public abstract class CoffeeShopBillBase
{
    public Dictionary<string, decimal> Prices { get; set; }
    public Dictionary<string, int> Discounts { get; set; }
    public CoffeeShopBillBase(Dictionary<string, decimal> prices,
Dictionary<string, int> discounts)
    {
        Prices = prices;
        Discounts = discounts;
    }

    public abstract List<List<object>> Calculate(List<List<object>>
shoppingList);
}

// Your implementation class
public class CoffeeShopBill : CoffeeShopBillBase
{
    public CoffeeShopBill(Dictionary<string, decimal> prices, Dictionary<string,
int> discounts)
        : base(prices, discounts)
    {
    }

    public override List<List<object>> Calculate(List<List<object>>
shoppingList)
    {
        // Aggregate quantities for duplicate items
        Dictionary<string, int> quantities = new Dictionary<string, int>();
        foreach (var entry in shoppingList)
        {
            string item = (string)entry[0];
            int qty = Convert.ToInt32(entry[1]);

            if (quantities.ContainsKey(item))
                quantities[item] += qty;
            else
                quantities[item] = qty;
        }
        // Prepare the result list
        List<List<object>> bill = new List<List<object>>();

        // Calculate bill with discount
        foreach (var kvp in quantities.OrderBy(k => k.Key))
        {
            string item = kvp.Key;
            int qty = kvp.Value;
            decimal pricePerUnit = Prices.ContainsKey(item) ? Prices[item] : 0m;
            int discountPercent = Discounts.ContainsKey(item) ? Discounts[item] :
0;

            decimal totalPrice = qty * pricePerUnit * (1m - discountPercent / 100m);
            bill.Add(new List<object> { item, pricePerUnit, totalPrice });
        }

        return bill;
    }
}
```