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

namespace Refactoring
{
    public class Account
    {
        public Account(string accountHolderName, int accountNumber)
        {
            TransactionList = new List<Transaction>();
            AccountHolderName = accountHolderName;
            AccountNumber = accountNumber;
        }
        private int AccountNumber { get; set; }
        private string AccountHolderName { get; set; }
        private List<Transaction> TransactionList { get; set; }
        private decimal Balance { get; set; }
        private DateTime? LastTransactionDate { get; set; }
        public decimal MaxCreditAmount { get; set; }
        public DateTime BillingCycleStartDate { get; set; }
        public int BillingCycleDays { get; set; }

        public DateTime GetNextBillingCycleStart()
        {
            var currentDate = DateTime.Now.Date;
            var iteratingDate = BillingCycleStartDate.Date;
            while (iteratingDate <= currentDate)
            {
                iteratingDate = iteratingDate.AddDays(BillingCycleDays);
            }
            return iteratingDate;
        }

        public Transaction GetLastTransaction()
        {
            return TransactionList.LastOrDefault();
        }

        public Transaction GetTransactionAt(int index)
        {
            return TransactionList.ElementAtOrDefault(index);
        }
        public int GetTransactionCount()
        {
            return TransactionList.Count;
        }

        public void Credit(decimal amount, string recipient)
        {
            Balance += amount;
            var creditTransaction = new CreditTransaction(false, amount);
```

```csharp
            creditTransaction.SetRecipient(recipient);
            creditTransaction.SetSender(AccountHolderName);
            TransactionList.Add(creditTransaction);
            LastTransactionDate = DateTime.Now;
        }

        public void Debit(decimal amount, string recipient)
        {
            Balance -= amount;
            var debitTransaction = new DebitTransaction(true, amount);
            debitTransaction.SetRecipient(recipient);
            debitTransaction.SetSender(AccountHolderName);
            TransactionList.Add(debitTransaction);
            LastTransactionDate = DateTime.Now;
        }
        public string SummaryCreditChargedMonthly(decimal totalAmount, string recipient, int numberOfMonths ,decimal maxCreditAmount, double rateOfInterest, int numberOfYears)
        {
            var baseMonthlyTotal = totalAmount/numberOfMonths;
            Balance += baseMonthlyTotal;
            var creditTransaction = new CreditTransaction(false, baseMonthlyTotal);
            creditTransaction.SetRecipient(recipient);
            creditTransaction.SetSender(AccountHolderName);
            TransactionList.Add(creditTransaction);
            if (Balance > maxCreditAmount)
            {
                Balance -= baseMonthlyTotal;
                TransactionList.RemoveAt(TransactionList.Count-1);
                return "Your credit transaction was initially rejected because you reached your max balance";
            }
            var nextCreditTransactionValue = new CreditTransaction(false, baseMonthlyTotal).CalculateInterest(rateOfInterest, numberOfYears, "Month");
            Balance += nextCreditTransactionValue;
            var nextCreditTransaction = new CreditTransaction(false, nextCreditTransactionValue);
            nextCreditTransaction.SetRecipient(recipient);
            nextCreditTransaction.SetSender(AccountHolderName);
            TransactionList.Add(nextCreditTransaction);
            if (Balance > maxCreditAmount)
            {
                Balance -= baseMonthlyTotal;
                TransactionList.RemoveAt(TransactionList.Count - 1);
                return "Your credit transaction was completely rejected because you reached your max balance";
            }
            return "Your transaction was accepted";
        }
        public decimal GetBalance()
        {
            return Balance;
        }
        public DateTime? GetLastTransactionDate()
        {
            return LastTransactionDate;
        }
    }
}

```

```
using System;

namespace Refactoring
{
    public class Car : Vehicle
    {
        public string CarBrand { get; set; }
        public int NumberOfWheels { get; set; }
        public Car(string carBrand)
        {
            CarBrand = carBrand;
            NumberOfWheels = 4;
        }

        private Driver Driver { get; set; }
        public void SetDriver(CarDriver driver)
        {
            Driver = driver;
        }

        public override string Drive()
        {
            return "I am driving a car";
        }

        public override int GetNumberOfWheels()
        {
            return 4;
        }
        public string VerifyOwnership()
        {
            var result = "This car has no owner";
            if (Driver != null)
            {
                result = "This car has an owner";
                if (!String.IsNullOrWhiteSpace(Driver.FormattedAddress()))
                {
                    result += "\nThe owner's address is:
                     \n"+Driver.FormattedAddress();
                }
            }
            return result;
        }

    }
}
```

```
using System;
namespace Refactoring
{
    public class CarDriver : Driver
    {
        public CarDriver(DateTime dateOfBirth, int pointsOnLicense, string
          licenseNumber, DateTime licenseExpireDate, string carBrand) : base
          (dateOfBirth, pointsOnLicense, licenseNumber, licenseExpireDate)
        {
            CarBrand = carBrand;
            Car = new Car(CarBrand);
        }
        public Car Car { get; set; }
        private string CarBrand { get; set; }
        public string GetCarBrand()
        {
            return CarBrand;
        }
        public string Drive()
        {
            return Car.Drive();
        }
        public string BuySpareWheel()
        {
            while (Car.NumberOfWheels <= 4) Car.NumberOfWheels ++;
            return String.Format("My car now has {0} number of wheels",
              Car.NumberOfWheels);
        }
    }
    public class BycicleDriver : Driver
    {
        public BycicleDriver(DateTime dateOfBirth, int pointsOnLicense, string
          licenseNumber, DateTime licenseExpireDate, string bycicleModel) : base
          (dateOfBirth, pointsOnLicense, licenseNumber, licenseExpireDate)
        {
            BycicleModel = bycicleModel;
            Bycicle = new Bycicle(BycicleModel);
        }

        public Bycicle Bycicle { get; set; }
        private string BycicleModel { get; set; }
        public string GetBycicleModel()
        {
            return BycicleModel;
        }
        public string Drive()
        {
            return Bycicle.Drive();
        }
    }
}
```

```
using System;
namespace Refactoring
{
    public class CreditTransaction : Transaction
    {
        public CreditTransaction(bool isDebit, decimal amount) : base(isDebit, amount){}
        private string Recipient { get; set; }
        private string Sender { get; set; }
        public void SetRecipient(string recipient)
        {
            Recipient = recipient;
        }
        public string GetRecipient() { return Recipient; }
        public void SetSender(string sender)
        {
            Sender = sender;
        }
        public string GetSender()
        {
            return Sender;
        }
        public string GetSummary()
        {
            return String.Format("This is a credit transaction for ${0} from {1} to {2}", Amount, Sender, Recipient);
        }
        public decimal CalculateInterest(double rateOfInterest, int numberOfYears, string interestPeriod)
        {
            double numberOfPeriodsPerYear = 0;
            switch (interestPeriod)
            {
                case "Day":
                    numberOfPeriodsPerYear = 365;
                    break;
                case "Month":
                    numberOfPeriodsPerYear = 12;
                    break;
                case "Semester":
                    numberOfPeriodsPerYear = 2;
                    break;
                case "Year":
                    numberOfPeriodsPerYear = 1;
                    break;
            }
            return Math.Round((decimal)((double) Amount*Math.Pow(1 + rateOfInterest/numberOfPeriodsPerYear, numberOfPeriodsPerYear*numberOfYears)), 2);
        }
    }
}
```

```
using System.Collections.Generic;

namespace Refactoring
{
    public class Customer
    {
        public Customer()
        {
            PersonalAccounts = new List<Account>();
        }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Title { get; set; }
        public List<Account> PersonalAccounts { get; set; }
    }
}

```

```
using System;

namespace Refactoring
{
    public class DebitTransaction : Transaction
    {
        public DebitTransaction(bool isDebit, decimal amount) : base(isDebit, amount)
        {
        }
        private string Recipient { get; set; }
        private string Sender { get; set; }
        public void SetRecipient(string recipient)
        {
            Recipient = recipient;
        }
        public string GetRecipient()
        {
            return Recipient;
        }
        public void SetSender(string sender)
        {
            Sender = sender;
        }
        public string GetSender()
        {
            return Sender;
        }

        public string GetSummary()
        {
            return String.Format("This is a debit transaction for ${0} from {1} to {2}", Amount, Sender, Recipient);
        }

        public decimal CalculateInterest(double rateOfInterest, int numberOfYears, string interestPeriod)
        {
            double numberOfPeriodsPerYear = 0;
            switch (interestPeriod)
            {
                case "Day":
                    numberOfPeriodsPerYear = 365;
                    break;
                case "Month":
                    numberOfPeriodsPerYear = 12;
                    break;
                case "Semester":
                    numberOfPeriodsPerYear = 2;
                    break;
                case "Year":
                    numberOfPeriodsPerYear = 1;
```

```
                        break;
                }
                var initialAmount = (double)Amount;
                for (var i = 0; i < numberOfYears; i++)
                {
                    var periodRate = rateOfInterest/numberOfPeriodsPerYear;
                    for (var j = 0; j < numberOfPeriodsPerYear; j++)
                    {
                        initialAmount += initialAmount*periodRate;
                    }
                }
                return Math.Round((decimal)initialAmount, 2);
            }
        }
    }
```

```
using System;

namespace Refactoring
{
    public class Driver
    {
        private int PointsOnLicense { get; set; }
        private string LicenseNumber { get; set; }
        private DateTime LicenseExpireDate { get; set; }
        private DateTime DateOfBirth { get; set; }
        public string AddressLine1 { get; set; }
        public string AddressLine2 { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Zip { get; set; }
        public Driver(DateTime dateOfBirth, int pointsOnLicense, string
          licenseNumber, DateTime licenseExpireDate)
         {
             PointsOnLicense = pointsOnLicense;
             DateOfBirth = dateOfBirth;
             LicenseNumber = licenseNumber;
             LicenseExpireDate = licenseExpireDate;
         }
         public int GetPointsOnLicense()
         {
             return PointsOnLicense;
         }

         public bool IsLicenseValid()
         {
             return PointsOnLicense < 5;
         }
         public string GenerateLicenseReport()
         {
             return String.Format("Your license number is {0} and you have {1}
               points in your license. Your license expires on {2}", LicenseNumber,
                 PointsOnLicense, LicenseExpireDate.ToString("d"));
         }

         public int GetAge()
         {
             var today = DateTime.Today;
             var age = today.Year - DateOfBirth.Year;
             if (DateOfBirth > today.AddYears(-age)) age--;
             return age;
         }

         public string FormattedAddress()
         {
             var formattedZip = Zip;
             if (Zip != null && Zip.Length > 5)
             {
                 formattedZip= Zip.Substring(0, 5);
             }
             var fullAddress = String.Format("{0} {1} {2} {3} {4}", AddressLine1, AddressLine2, City, State, formattedZip).Trim();
             var outAddress = String.Empty;
             if (String.IsNullOrWhiteSpace(fullAddress)) return outAddress;
             outAddress = AddressLine1;
             if (!String.IsNullOrWhiteSpace(AddressLine2)) outAddress += "\n" + AddressLine2;
             if (!String.IsNullOrWhiteSpace(City) || !String.IsNullOrWhiteSpace(State))
             {
                 outAddress += "\n";
                 if (!String.IsNullOrWhiteSpace(City)) outAddress += City;
                 if (!String.IsNullOrWhiteSpace(City) && !String.IsNullOrWhiteSpace(State)) outAddress += ", ";
                 if (!String.IsNullOrWhiteSpace(State)) outAddress += State;
             }
             if (!String.IsNullOrWhiteSpace(formattedZip)) outAddress += "\n" + formattedZip;
             return outAddress;
         }
     }
 }

```

```
namespace Refactoring
{
    public class InsuranceQuote
    {
        private Driver Driver { get; set; }
        public InsuranceQuote(Driver driver)
        {
            Driver = driver;
        }
        public RiskFactor CalculateDriverRiskFactor()
        {
            if (Driver.GetPointsOnLicense() > 3 || Driver.GetAge() < 25)
                return RiskFactor.High;

            if (Driver.GetPointsOnLicense() > 0)
                return RiskFactor.Moderate;

            return RiskFactor.Low;
        }
        public double CalculateInsurancePremium(double insuranceValue)
        {
            var riskFactor = CalculateDriverRiskFactor();
            //Switch Statements - Try to add case -  make extension method class 
              along with enum
            switch (riskFactor)
            {
                case RiskFactor.Low:
                    return insuranceValue * 0.02;
                case RiskFactor.Moderate:
                    return insuranceValue * 0.04;
                case RiskFactor.High:
                    return insuranceValue * 0.06;
            }
            return insuranceValue;
        }
    }

    public enum RiskFactor
    {
        Low,
        Moderate,
        High
    }
}
```

```
namespace Refactoring
{
    public class Statement
    {
        private Account Account { get; set; }
        public Statement(Account account)
        {
            Account = account;
        }
        public decimal GetTotalCreditBalance()
        {
            var totalCreditBalance = 0m;
            var totalTransactions = Account.GetTransactionCount();
            for (var i = 0; i < totalTransactions; i++)
            {
                var transaction = Account.GetTransactionAt(i);
                if (transaction is CreditTransaction)
                {
                    totalCreditBalance += transaction.Amount;
                }
            }
            return totalCreditBalance;
        }

        public decimal GetTotalDebitBalance()
        {
            var totalDebitBalance = 0m;
            var totalTransactions = Account.GetTransactionCount();
            for (var i = 0; i < totalTransactions; i++)
            {
                var transaction = Account.GetTransactionAt(i);
                if (transaction is DebitTransaction)
                {
                    totalDebitBalance += transaction.Amount;
                }
            }
            return totalDebitBalance;
        }
    }
}
```

```
using System;

namespace Refactoring
{
    public class Transaction
    {
        public bool IsDebit { get; private set; }
        public decimal Amount { get; private set; }

        protected Transaction(bool isDebit, decimal amount)
        {
            IsDebit = isDebit;
            Amount = amount;
        }
        public void ScheduleTransaction(DateTime futureDate)
        {
            throw new NotImplementedException();
        }
    }

    public abstract class InvestmentTransaction : Transaction
    {
        protected InvestmentTransaction(bool isDebit, decimal amount) : base
          (isDebit, amount)
        {
        }
        public string InvestmentFundName { get; set; }

        public string GetSummary()
        {
            return String.Format("This is an investment transaction for ${0} in
              fund {1}", Amount, InvestmentFundName);
        }
    }

    public class LongTermInvestmentTransaction : InvestmentTransaction
    {
        public string InvestmentPeriod { get; set; }
        public LongTermInvestmentTransaction(bool isDebit, decimal amount) : base
          (isDebit, amount)
        {
        }
    }
}
```

```
namespace Refactoring
{
    public abstract class Vehicle
    {
        protected Vehicle()
        {
            Wheel = new Wheel();
        }
        public Wheel Wheel { get; set; }
        public virtual string Drive()
        {
            return "I am driving a vehicle";
        }
        public virtual string Move()
        {
            return Wheel.Move();
        }
        public virtual string Stop()
        {
            return Wheel.Stop();
        }
        public abstract int GetNumberOfWheels();
    }

    public class Bycicle : Vehicle
    {
        public Bycicle(string bycicleModel)
        {
            BycicleModel = bycicleModel;
        }

        public string BycicleModel { get; set; }
        public override string Drive()
        {
            return "I am driving a bike";
        }
        public override int GetNumberOfWheels()
        {
            return 2;
        }
    }
}
```

```
using System;
namespace Refactoring
{
    public class Wheel
    {
        public Wheel()
        {
            Tire = new Tire();
        }
        public Tire Tire { get; set; }
        public string Move()
        {
            return Tire.Move();
        }
        public string Stop()
        {
            return Tire.Stop();
        }
    }
    public class Tire
    {
        public string Move()
        {
            return "I am a moving tire";
        }
        public string Stop()
        {
            return "I am a stopping tire";
        }
    }
    public class FortuneWheel : Vehicle
    {
        public override string Drive()
        {
            throw new NotImplementedException();
        }

        public virtual string Move()
        {
            return Wheel.Move();
        }
        public virtual string Stop()
        {
            return Wheel.Stop();
        }

        public override int GetNumberOfWheels()
        {
            return 1;
        }
    }
}
```