```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4
 5  namespace Refactoring
 6  {
 7      public class Account
 8      {
 9          public Account(string accountHolderName, int accountNumber)
10          {
11              TransactionList = new List<Transaction>();
12              AccountHolderName = accountHolderName;
13              AccountNumber = accountNumber;
14          }
15          private int AccountNumber { get; set; }
16          private string AccountHolderName { get; set; }
17          private List<Transaction> TransactionList { get; set; }
18          private decimal Balance { get; set; }
19          private DateTime? LastTransactionDate { get; set; }
20          public decimal MaxCreditAmount { get; set; }
21          public DateTime BillingCycleStartDate { get; set; }
22          public int BillingCycleDays { get; set; }
23
24          public DateTime GetNextBillingCycleStart()
25          {
26              var currentDate = DateTime.Now.Date;
27              var iteratingDate = BillingCycleStartDate.Date;
28              while (iteratingDate <= currentDate)
29              {
30                  iteratingDate = iteratingDate.AddDays(BillingCycleDays);
31              }
32              return iteratingDate;
33          }
34
35          public Transaction GetLastTransaction()
36          {
37              return TransactionList.LastOrDefault();
38          }
39
40          public Transaction GetTransactionAt(int index)
41          {
42              return TransactionList.ElementAtOrDefault(index);
43          }
44          public int GetTransactionCount()
45          {
46              return TransactionList.Count;
47          }
48
49          public void Credit(decimal amount, string recipient)
50          {
51              Balance += amount;
52              var creditTransaction = new CreditTransaction(false, amount);
```

```
53              creditTransaction.SetRecipient(recipient);
54              creditTransaction.SetSender(AccountHolderName);
55              TransactionList.Add(creditTransaction);
56              LastTransactionDate = DateTime.Now;
57          }
58
59          public void Debit(decimal amount, string recipient)
60          {
61              Balance -= amount;
62              var debitTransaction = new DebitTransaction(true, amount);
63              debitTransaction.SetRecipient(recipient);
64              debitTransaction.SetSender(AccountHolderName);
65              TransactionList.Add(debitTransaction);
66              LastTransactionDate = DateTime.Now;
67          }
68          public string SummaryCreditChargedMonthly(decimal totalAmount, string
              recipient, int numberOfMonths ,decimal maxCreditAmount, double
              rateOfInterest, int numberOfYears)
69          {
70              var baseMonthlyTotal = totalAmount/numberOfMonths;
71              Balance += baseMonthlyTotal;
72              var creditTransaction = new CreditTransaction(false,
                  baseMonthlyTotal);
73              creditTransaction.SetRecipient(recipient);
74              creditTransaction.SetSender(AccountHolderName);
75              TransactionList.Add(creditTransaction);
76              if (Balance > maxCreditAmount)
77              {
78                  Balance -= baseMonthlyTotal;
79                  TransactionList.RemoveAt(TransactionList.Count-1);
80                  return "Your credit transaction was initially rejected because
                      you reached your max balance";
81              }
82              var nextCreditTransactionValue = new CreditTransaction(false,
                  baseMonthlyTotal).CalculateInterest(rateOfInterest, numberOfYears,
                  "Month");
83              Balance += nextCreditTransactionValue;
84              var nextCreditTransaction = new CreditTransaction(false,
                  nextCreditTransactionValue);
85              nextCreditTransaction.SetRecipient(recipient);
86              nextCreditTransaction.SetSender(AccountHolderName);
87              TransactionList.Add(nextCreditTransaction);
88              if (Balance > maxCreditAmount)
89              {
90                  Balance -= baseMonthlyTotal;
91                  TransactionList.RemoveAt(TransactionList.Count - 1);
92                  return "Your credit transaction was completely rejected because
                      you reached your max balance";
93              }
94              return "Your transaction was accepted";
95          }
96          public decimal GetBalance()
```

```
 97          {
 98              return Balance;
 99          }
100          public DateTime? GetLastTransactionDate()
101          {
102              return LastTransactionDate;
103          }
104      }
105  }
106
```

```
1  using System;
2
3  namespace Refactoring
4  {
5      public class Car : Vehicle
6      {
7          public string CarBrand { get; set; }
8          public int NumberOfWheels { get; set; }
9          public Car(string carBrand)
10         {
11             CarBrand = carBrand;
12             NumberOfWheels = 4;
13         }
14
15         private Driver Driver { get; set; }
16         public void SetDriver(CarDriver driver)
17         {
18             Driver = driver;
19         }
20
21         public override string Drive()
22         {
23             return "I am driving a car";
24         }
25
26         public override int GetNumberOfWheels()
27         {
28             return 4;
29         }
30         public string VerifyOwnership()
31         {
32             var result = "This car has no owner";
33             if (Driver != null)
34             {
35                 result = "This car has an owner";
36                 if (!String.IsNullOrWhiteSpace(Driver.FormattedAddress()))
37                 {
38                     result += "\nThe owner's address is:
                        \n"+Driver.FormattedAddress();
39                 }
40             }
41             return result;
42         }
43
44     }
45 }
```

```
1  using System;
2  namespace Refactoring
3  {
4      public class CarDriver : Driver
5      {
6          public CarDriver(DateTime dateOfBirth, int pointsOnLicense, string
             licenseNumber, DateTime licenseExpireDate, string carBrand) : base
             (dateOfBirth, pointsOnLicense, licenseNumber, licenseExpireDate)
7          {
8              CarBrand = carBrand;
9              Car = new Car(CarBrand);
10         }
11         public Car Car { get; set; }
12         private string CarBrand { get; set; }
13         public string GetCarBrand()
14         {
15             return CarBrand;
16         }
17         public string Drive()
18         {
19             return Car.Drive();
20         }
21         public string BuySpareWheel()
22         {
23             while (Car.NumberOfWheels <= 4) Car.NumberOfWheels ++;
24             return String.Format("My car now has {0} number of wheels",
                 Car.NumberOfWheels);
25         }
26     }
27     public class BycicleDriver : Driver
28     {
29         public BycicleDriver(DateTime dateOfBirth, int pointsOnLicense, string
             licenseNumber, DateTime licenseExpireDate, string bycicleModel) : base
             (dateOfBirth, pointsOnLicense, licenseNumber, licenseExpireDate)
30         {
31             BycicleModel = bycicleModel;
32             Bycicle = new Bycicle(BycicleModel);
33         }
34
35         public Bycicle Bycicle { get; set; }
36         private string BycicleModel { get; set; }
37         public string GetBycicleModel()
38         {
39             return BycicleModel;
40         }
41         public string Drive()
42         {
43             return Bycicle.Drive();
44         }
45     }
46 }
```

```
1  using System;
2  namespace Refactoring
3  {
4      public class CreditTransaction : Transaction
5      {
6          public CreditTransaction(bool isDebit, decimal amount) : base(isDebit,
             amount){}
7          private string Recipient { get; set; }
8          private string Sender { get; set; }
9          public void SetRecipient(string recipient)
10         {
11             Recipient = recipient;
12         }
13         public string GetRecipient() { return Recipient; }
14         public void SetSender(string sender)
15         {
16             Sender = sender;
17         }
18         public string GetSender()
19         {
20             return Sender;
21         }
22         public string GetSummary()
23         {
24             return String.Format("This is a credit transaction for ${0} from {1}
                 to {2}", Amount, Sender, Recipient);
25         }
26         public decimal CalculateInterest(double rateOfInterest, int numberOfYears,
              string interestPeriod)
27         {
28             double numberOfPeriodsPerYear = 0;
29             switch (interestPeriod)
30             {
31                 case "Day":
32                     numberOfPeriodsPerYear = 365;
33                     break;
34                 case "Month":
35                     numberOfPeriodsPerYear = 12;
36                     break;
37                 case "Semester":
38                     numberOfPeriodsPerYear = 2;
39                     break;
40                 case "Year":
41                     numberOfPeriodsPerYear = 1;
42                     break;
43             }
44             return Math.Round((decimal)((double) Amount*Math.Pow(1 +
                 rateOfInterest/numberOfPeriodsPerYear,
                 numberOfPeriodsPerYear*numberOfYears)), 2);
45         }
46     }
47 }
```

```
1  using System.Collections.Generic;
2
3  namespace Refactoring
4  {
5      public class Customer
6      {
7          public Customer()
8          {
9              PersonalAccounts = new List<Account>();
10         }
11         public string FirstName { get; set; }
12         public string LastName { get; set; }
13         public string Title { get; set; }
14         public List<Account> PersonalAccounts { get; set; }
15     }
16 }
17
```

```
1  using System;
2
3  namespace Refactoring
4  {
5      public class DebitTransaction : Transaction
6      {
7          public DebitTransaction(bool isDebit, decimal amount) : base(isDebit,
             amount)
8          {
9          }
10         private string Recipient { get; set; }
11         private string Sender { get; set; }
12         public void SetRecipient(string recipient)
13         {
14             Recipient = recipient;
15         }
16         public string GetRecipient()
17         {
18             return Recipient;
19         }
20         public void SetSender(string sender)
21         {
22             Sender = sender;
23         }
24         public string GetSender()
25         {
26             return Sender;
27         }
28
29         public string GetSummary()
30         {
31             return String.Format("This is a debit transaction for ${0} from {1} to
                {2}", Amount, Sender, Recipient);
32         }
33
34         public decimal CalculateInterest(double rateOfInterest, int numberOfYears,
              string interestPeriod)
35         {
36             double numberOfPeriodsPerYear = 0;
37             switch (interestPeriod)
38             {
39                 case "Day":
40                     numberOfPeriodsPerYear = 365;
41                     break;
42                 case "Month":
43                     numberOfPeriodsPerYear = 12;
44                     break;
45                 case "Semester":
46                     numberOfPeriodsPerYear = 2;
47                     break;
48                 case "Year":
49                     numberOfPeriodsPerYear = 1;
```

```
50                      break;
51              }
52              var initialAmount = (double)Amount;
53              for (var i = 0; i < numberOfYears; i++)
54              {
55                  var periodRate = rateOfInterest/numberOfPeriodsPerYear;
56                  for (var j = 0; j < numberOfPeriodsPerYear; j++)
57                  {
58                      initialAmount += initialAmount*periodRate;
59                  }
60              }
61              return Math.Round((decimal)initialAmount, 2);
62          }
63      }
64  }
```

```
1  using System;
2
3  namespace Refactoring
4  {
5      public class Driver
6      {
7         private int PointsOnLicense { get; set; }
8         private string LicenseNumber { get; set; }
9         private DateTime LicenseExpireDate { get; set; }
10        private DateTime DateOfBirth { get; set; }
11        public string AddressLine1 { get; set; }
12        public string AddressLine2 { get; set; }
13        public string City { get; set; }
14        public string State { get; set; }
15        public string Zip { get; set; }
16        public Driver(DateTime dateOfBirth, int pointsOnLicense, string
            licenseNumber, DateTime licenseExpireDate)
17         {
18             PointsOnLicense = pointsOnLicense;
19             DateOfBirth = dateOfBirth;
20             LicenseNumber = licenseNumber;
21             LicenseExpireDate = licenseExpireDate;
22         }
23         public int GetPointsOnLicense()
24         {
25             return PointsOnLicense;
26         }
27
28         public bool IsLicenseValid()
29         {
30             return PointsOnLicense < 5;
31         }
32         public string GenerateLicenseReport()
33         {
34             return String.Format("Your license number is {0} and you have {1}
                 points in your license. Your license expires on {2}", LicenseNumber,
35                   PointsOnLicense, LicenseExpireDate.ToString("d"));
36         }
37
38         public int GetAge()
39         {
40             var today = DateTime.Today;
41             var age = today.Year - DateOfBirth.Year;
42             if (DateOfBirth > today.AddYears(-age)) age--;
43             return age;
44         }
45
46         public string FormattedAddress()
47         {
48             var formattedZip = Zip;
49             if (Zip != null && Zip.Length > 5)
50             {
```

```
51                  formattedZip= Zip.Substring(0, 5);
52              }
53              var fullAddress = String.Format("{0} {1} {2} {3} {4}", AddressLine1,
                  AddressLine2, City, State, formattedZip).Trim();
54              var outAddress = String.Empty;
55              if (String.IsNullOrWhiteSpace(fullAddress)) return outAddress;
56              outAddress = AddressLine1;
57              if (!String.IsNullOrWhiteSpace(AddressLine2)) outAddress += "\n" +
                  AddressLine2;
58              if (!String.IsNullOrWhiteSpace(City) || !String.IsNullOrWhiteSpace
                  (State))
59              {
60                  outAddress += "\n";
61                  if (!String.IsNullOrWhiteSpace(City)) outAddress += City;
62                  if (!String.IsNullOrWhiteSpace(City) && !String.IsNullOrWhiteSpace
                      (State)) outAddress += ", ";
63                  if (!String.IsNullOrWhiteSpace(State)) outAddress += State;
64              }
65              if (!String.IsNullOrWhiteSpace(formattedZip)) outAddress += "\n" +
                  formattedZip;
66              return outAddress;
67          }
68      }
69  }
70
```

```
1  namespace Refactoring
2  {
3      public class InsuranceQuote
4      {
5          private Driver Driver { get; set; }
6          public InsuranceQuote(Driver driver)
7          {
8              Driver = driver;
9          }
10         public RiskFactor CalculateDriverRiskFactor()
11         {
12             if (Driver.GetPointsOnLicense() > 3 || Driver.GetAge() < 25)
13                 return RiskFactor.High;
14
15             if (Driver.GetPointsOnLicense() > 0)
16                 return RiskFactor.Moderate;
17
18             return RiskFactor.Low;
19         }
20         public double CalculateInsurancePremium(double insuranceValue)
21         {
22             var riskFactor = CalculateDriverRiskFactor();
23             //Switch Statements - Try to add case -  make extension method class
                 along with enum
24             switch (riskFactor)
25             {
26                 case RiskFactor.Low:
27                     return insuranceValue * 0.02;
28                 case RiskFactor.Moderate:
29                     return insuranceValue * 0.04;
30                 case RiskFactor.High:
31                     return insuranceValue * 0.06;
32             }
33             return insuranceValue;
34         }
35     }
36
37     public enum RiskFactor
38     {
39         Low,
40         Moderate,
41         High
42     }
43 }
44
```

```
1  namespace Refactoring
2  {
3      public class Statement
4      {
5          private Account Account { get; set; }
6          public Statement(Account account)
7          {
8              Account = account;
9          }
10         public decimal GetTotalCreditBalance()
11         {
12             var totalCreditBalance = 0m;
13             var totalTransactions = Account.GetTransactionCount();
14             for (var i = 0; i < totalTransactions; i++)
15             {
16                 var transaction = Account.GetTransactionAt(i);
17                 if (transaction is CreditTransaction)
18                 {
19                     totalCreditBalance += transaction.Amount;
20                 }
21             }
22             return totalCreditBalance;
23         }
24
25         public decimal GetTotalDebitBalance()
26         {
27             var totalDebitBalance = 0m;
28             var totalTransactions = Account.GetTransactionCount();
29             for (var i = 0; i < totalTransactions; i++)
30             {
31                 var transaction = Account.GetTransactionAt(i);
32                 if (transaction is DebitTransaction)
33                 {
34                     totalDebitBalance += transaction.Amount;
35                 }
36             }
37             return totalDebitBalance;
38         }
39     }
40 }
41
```

```
1  using System;
2
3  namespace Refactoring
4  {
5      public class Transaction
6      {
7          public bool IsDebit { get; private set; }
8          public decimal Amount { get; private set; }
9
10         protected Transaction(bool isDebit, decimal amount)
11         {
12             IsDebit = isDebit;
13             Amount = amount;
14         }
15         public void ScheduleTransaction(DateTime futureDate)
16         {
17             throw new NotImplementedException();
18         }
19     }
20
21     public abstract class InvestmentTransaction : Transaction
22     {
23         protected InvestmentTransaction(bool isDebit, decimal amount) : base
             (isDebit, amount)
24         {
25         }
26         public string InvestmentFundName { get; set; }
27
28         public string GetSummary()
29         {
30             return String.Format("This is an investment transaction for ${0} in
                 fund {1}", Amount, InvestmentFundName);
31         }
32     }
33
34     public class LongTermInvestmentTransaction : InvestmentTransaction
35     {
36         public string InvestmentPeriod { get; set; }
37         public LongTermInvestmentTransaction(bool isDebit, decimal amount) : base
             (isDebit, amount)
38         {
39         }
40     }
41 }
```

```
1  namespace Refactoring
2  {
3      public abstract class Vehicle
4      {
5          protected Vehicle()
6          {
7              Wheel = new Wheel();
8          }
9          public Wheel Wheel { get; set; }
10         public virtual string Drive()
11         {
12             return "I am driving a vehicle";
13         }
14         public virtual string Move()
15         {
16             return Wheel.Move();
17         }
18         public virtual string Stop()
19         {
20             return Wheel.Stop();
21         }
22         public abstract int GetNumberOfWheels();
23     }
24
25     public class Bycicle : Vehicle
26     {
27         public Bycicle(string bycicleModel)
28         {
29             BycicleModel = bycicleModel;
30         }
31
32         public string BycicleModel { get; set; }
33         public override string Drive()
34         {
35             return "I am driving a bike";
36         }
37         public override int GetNumberOfWheels()
38         {
39             return 2;
40         }
41     }
42 }
```

```
1  using System;
2  namespace Refactoring
3  {
4      public class Wheel
5      {
6          public Wheel()
7          {
8              Tire = new Tire();
9          }
10         public Tire Tire { get; set; }
11         public string Move()
12         {
13             return Tire.Move();
14         }
15         public string Stop()
16         {
17             return Tire.Stop();
18         }
19     }
20     public class Tire
21     {
22         public string Move()
23         {
24             return "I am a moving tire";
25         }
26         public string Stop()
27         {
28             return "I am a stopping tire";
29         }
30     }
31     public class FortuneWheel : Vehicle
32     {
33         public override string Drive()
34         {
35             throw new NotImplementedException();
36         }
37
38         public virtual string Move()
39         {
40             return Wheel.Move();
41         }
42         public virtual string Stop()
43         {
44             return Wheel.Stop();
45         }
46
47         public override int GetNumberOfWheels()
48         {
49             return 1;
50         }
51     }
52 }
```