**Homework 11**

**Part I**

Design an inheritance hierarchy to include classes for Student, GraduateStudent, and UnderGraduate and show it in the form of UML diagram. Name your own members for each class. For example, GraduateStudent may include a data member for the type of undergraduate degree awarded, such as B.A. or B.S., and the location of the institution that awarded the degree. UnderGraduate may include classification (freshman, sophomore).

Implement your design in C# and write a driver program to test it.

**Screenshot**

**A screen shot of a computer

Description automatically generated with medium confidence**

**Code (Main)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork\_11

{

internal class Program

{

static void Main(string[] args)

{

UnderGraduate Test1 = new UnderGraduate("Daniele", "Ricciardelli", 25, 3.8, "Computer Engineering", "Sophomore");

GraduateStudent Test2 = new GraduateStudent("Emilio", "Ricciardelli", 62, 2.8, "BS", "Cal Poly Pomona" );

// Student calls

Console.WriteLine(Test1.Name);

Console.WriteLine(Test1.Last);

Console.WriteLine(Test1.Age);

Console.WriteLine(Test1.GPA);

Console.WriteLine();

Console.WriteLine(Test2.Name);

Console.WriteLine(Test2.Last);

Console.WriteLine(Test2.Age);

Console.WriteLine(Test2.GPA);

Console.WriteLine();

Console.WriteLine();

// Under call

Console.WriteLine(Test1.DegreeName);

Console.WriteLine(Test1.DegreeYear);

Console.WriteLine();

//Grad call

Console.WriteLine(Test2.DegreeType);

Console.WriteLine(Test2.DegreeLocation);

Console.WriteLine();

}

}

}

**Code (Student)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork\_11

{

public class Student

{

// Data Members

protected string name;

protected string last;

protected int age;

protected double gpa;

// Default C.

public Student()

{

this.name = "";

this.last = "";

this.age = 0;

this.gpa = 0.00;

}

// Overload C.

public Student( string n, string l, int a, double g)

{

this.name = n;

this.last = l;

this.age = a;

this.gpa = g;

}

// Get n Set

public string Name

{

get { return name; }

set { name = value; }

}

public string Last

{

get { return last; }

set { last = value; }

}

public int Age

{

get { return age; }

set { age = value; }

}

public double GPA

{

get { return gpa; }

set { gpa = value; }

}

~Student() { }

// No need to make more destructors than this as all the other classes make reference to Student.

}

}

**Code (Undergraduate)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork\_11

{

public class UnderGraduate : Student

{

// Data Members

private string degreeName;

private string degreeYear;

// Inheritance from Student

public UnderGraduate() : base() { }

// Inheritance + Overload C.

public UnderGraduate(string n, string l, int a, double g, string dn, string dy) : base (n, l, a, g)

{

this.degreeName = dn;

this.degreeYear = dy;

}

// Get n Set

public string DegreeName

{

get { return degreeName; }

set { degreeName = value; }

}

public string DegreeYear

{

get { return degreeYear; }

set { degreeYear = value; }

}

}

}

**Code (GraduateStudent)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork\_11

{

public class GraduateStudent : Student

{

// Data Members

private string degreeType;

private string degreeLocation;

// Inheritance from Student

public GraduateStudent() : base() { }

// Inheritance + Overload C.

public GraduateStudent(string n, string l, int a, double g, string dt, string dl) : base (n, l, a, g)

{

this.degreeType = dt;

this.degreeLocation = dl;

}

// Get n Set

public string DegreeType

{

get { return degreeType; }

set { degreeType = value; }

}

public string DegreeLocation

{

get { return degreeLocation; }

set { degreeLocation = value; }

}

}

}

**Notes:**

* Instructions didn’t specify user input and therefore I assumed no validation was required (nor instructed).
* No instructions for ToString, decided to take a “shortcut” and simply include Console.Write in order to proof correct implementation of the inheritance/classes.
* Called protected for base class (student) in order to access it when called – made private members that were part of the base (undergrad, and grad).
* Separate prints/writes in order to differentiate classes calls.

|  |
| --- |
| Student |
|  |
| -name: string |
| -last: string |
| -age: int |
| -gpa: double |
|  |
| +Student() |
| +Student(string, string, int, double) |
| +GetName(): string |
| +SetName(string) |
| +GetLast(): string |
| +SetLast(string) |
| +GetAge(): int |
| +SetAge(int) |
| +GetGPA(): double |
| +SetGPA(double) |
| ~Student() |

**UML**

|  |
| --- |
| GraduateStudent |
|  |
| -degreeType |
| -degreeLocation |
|  |
| +GraduateStudent() |
| +GraduateStudent(string, string, |
| int, double, string, string) |
| +GetDegreeType(): string |
| +SetDegreeType(string) |
| +GetDegreeLocation(): string |
| +SetDegreeLocation(string) |

|  |
| --- |
| UnderGraduate |
|  |
| -degreeName |
| -degreeYear |
|  |
| +UnderGraduate() |
| +UnderGraduate(string, string, |
| int, double, string, string) |
| +GetDegreeName(): string |
| +SetDegreeName(string) |
| +GetDegreeYear(): string |
| +SetDegreeYear(string) |

**Part II**

Create a base class BankAccount. Decide what characteristics are common for checking and saving accounts and include these characteristics in the base class. Define derived classes for checking and savings. In your design, do not allow the banking base account to be instantiated --only the checking and saving classes. Use a Program class to test your design.

**Screenshot**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Code (main)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork\_11.\_2

{

internal class Program

{

static void Main(string[] args)

{

Checking MyAccount = new Checking("Daniele Ricciardelli", 011022, 06442, 15000, "VIP");

Console.WriteLine(MyAccount);

MyAccount.Deposit(5000);

Console.WriteLine("");

Console.WriteLine(MyAccount);

MyAccount.Withdraw(10000);

Console.WriteLine("");

Console.WriteLine(MyAccount);

Console.WriteLine("");

Saving MyAccount2 = new Saving("Daniele Ricciardelli", 011022, 06442, 15000, 0.2);

Console.WriteLine(MyAccount2);

MyAccount2.Deposit(5000);

Console.WriteLine("");

Console.WriteLine(MyAccount2);

MyAccount2.Withdraw(10000);

Console.WriteLine("");

Console.WriteLine(MyAccount2);

MyAccount2.Interest();

Console.WriteLine("");

Console.WriteLine(MyAccount2);

}

}

}

**Code(bankaccount)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork\_11.\_2

{

public abstract class BankAccount

{

// Data Members

public string fullName { get; private set; }

public int accountNumber { get; private set; }

public int routingNumber { get; private set; }

public double balance { get; protected set; }

// Default C.

public BankAccount()

{

fullName = "";

accountNumber = 123456;

routingNumber = 01234;

balance = 0.00;

}

//Overload C.

public BankAccount(string fn, int an, int rn, double b)

{

fullName = fn;

accountNumber = an;

routingNumber = rn;

balance = b;

}

// Methods

public abstract void Deposit (double amount);

public abstract void Withdraw(double amount);

public override string ToString()

{

return

$"The account under {fullName} has a remainding balance of {balance}.\n\n" +

"To make a deposit in the future, remember to provide these two to the sender:\n" +

$"Account number: {accountNumber}\nRoutingNumber: {routingNumber}";

}

~BankAccount() { }

}

}

**Code(checking)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork\_11.\_2

{

public class Checking : BankAccount

{

// Data Member

public string CheckingType { get; protected set; }

// Overload C.

public Checking(string fn, int an, int rn, double b, string DoC) : base (fn, an, rn, b)

{

CheckingType = DoC;

}

// Methods

public override void Deposit (double amount)

{

balance += amount;

}

public override void Withdraw (double amount)

{

if (amount <= balance)

{

balance -= amount;

}

else

{

Console.WriteLine("Not enough money :(");

}

}

}

}

**Code(saving)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork\_11.\_2

{

public class Saving : BankAccount

{

// Data Member

public double interest { get; private set; }

// Overload C.

public Saving(string fn, int an, int rn, double b, double i) : base (fn, an, rn, b)

{

interest = i;

}

// Methods

public override void Deposit(double amount)

{

balance += amount;

}

public override void Withdraw(double amount)

{

if (amount <= balance)

{

balance -= amount;

}

else

{

Console.WriteLine("Not enough money :(");

}

}

public void Interest()

{

balance += balance \* interest;

}

}

}

**UML**

|  |
| --- |
| (abstract) BankAccount |
|  |
| -fullname: string |
| -accountNumber: int |
| -accountRouting: int |
| -balance: double |
|  |
| +Deposit(double):void |
| +WithDraw(double): void |
| +ToString(): string |
| ~BankAccount() |

|  |
| --- |
| Savings |
|  |
| -interest: double |
|  |
| +Deposit(double):void |
| +WithDraw(double): void |

|  |
| --- |
| Checking |
|  |
| -checkingType |
|  |
| +Deposit(double):void |
| +WithDraw(double): void |
| +Interest():void |