**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра информационной безопасности**

**ОТЧЕТ**

**по лабораторной работе №6**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема:** **Использование методов**

| Студент гр. 3363 |  | Овсейчик Н. И., Минко Д. А.  Гончаренко О.Д. |
| --- | --- | --- |
| Преподаватель |  | Новакова Н. Е. |

**Цель работы**

Цель лабораторной работы — освоить механизмы инкапсуляции в объектно-ориентированном программировании путем преобразования структуры в класс, работы с модификаторами доступа, создания и использования методов для управления объектами, а также применения статических методов для генерации данных. В процессе выполнения необходимо выполнить инкапсуляцию данных банковского счета, реализовать методы для ввода и вывода информации, а также добавить функциональность пополнения и снятия средств со счета.

**ХОД РАБОТЫ**

1. Создание классов.

В Visual Studio 2022 был открыт новый проект с типом "Console Application" и названием "FileDetails". В файле *BankAccount.cs* была изучена исходная программа, в которой класс BankAccount имел тип struct. Программа была скомпилирована и запущена, после чего пользователю было предложено ввести номер счета и баланс для двух разных счетов (рис. 1).

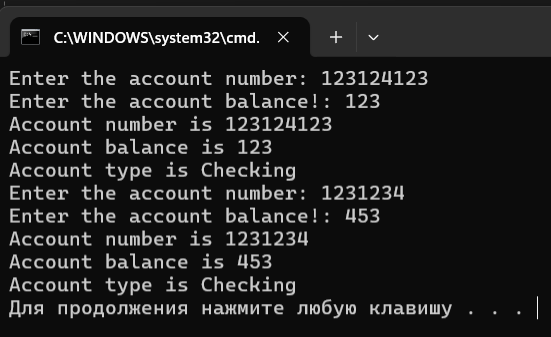


Рисунок 1 – Первоначальная скомпилированная программа

В программе *BankAccount.cs* структура была преобразована в класс, после чего была выполнена компиляция, которая привела к ошибке. В файле *CreateAccount.cs* было открыто определение класса CreateAccount, в котором метод NewBankAccount использовал переменную created для создания нового объекта BankAccount. Поскольку BankAccount теперь является ссылочным типом, объявление переменной created было изменено на создание объекта с использованием ключевого слова new. Программа была успешно скомпилирована, ошибки были откорректированы, и проверено, что данные вводятся корректно (рис. 2).

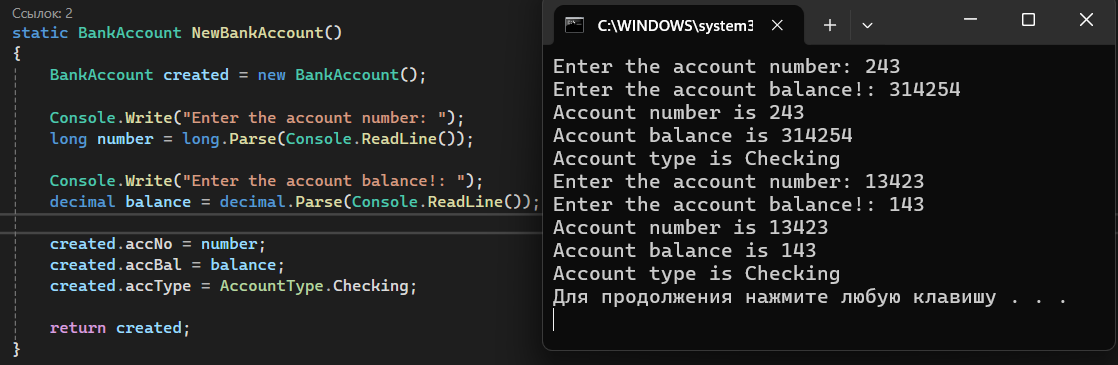


Рисунок 2 – Преобразование структуры в класс и создание его экземпляра

В классе BankAccount была выполнена инкапсуляция: все члены класса, имеющие модификатор public, были изменены на private. После компиляции программы возникла ошибка. Был написан нестатический метод Populate, который принимает два параметра: номер счета и баланс. В теле метода параметры были назначены соответствующим полям accNo и accBal, а также полю accType было присвоено значение AccountType.Checking. В файле *BankAccount.cs* были закомментированы назначения переменной created в методе NewBankAccount, после чего добавлено выражение, вызывающее метод Populate с передачей аргументов.

При следующей компиляции возникли ошибки, связанные с попытками метода Write обращаться к полям, объявленным как private. В классе BankAccount были добавлены три публичных метода: Number, Balance и Type, возвращающие значения полей соответствующих типов. Затем в методе Write в классе CreateAccount были заменены прямые обращения к полям на вызовы новых методов. После исправлений программа была успешно скомпилирована, ошибки были устранены, и данные были введены корректно. Полученный листинг упражнения был сохранен для отчета.

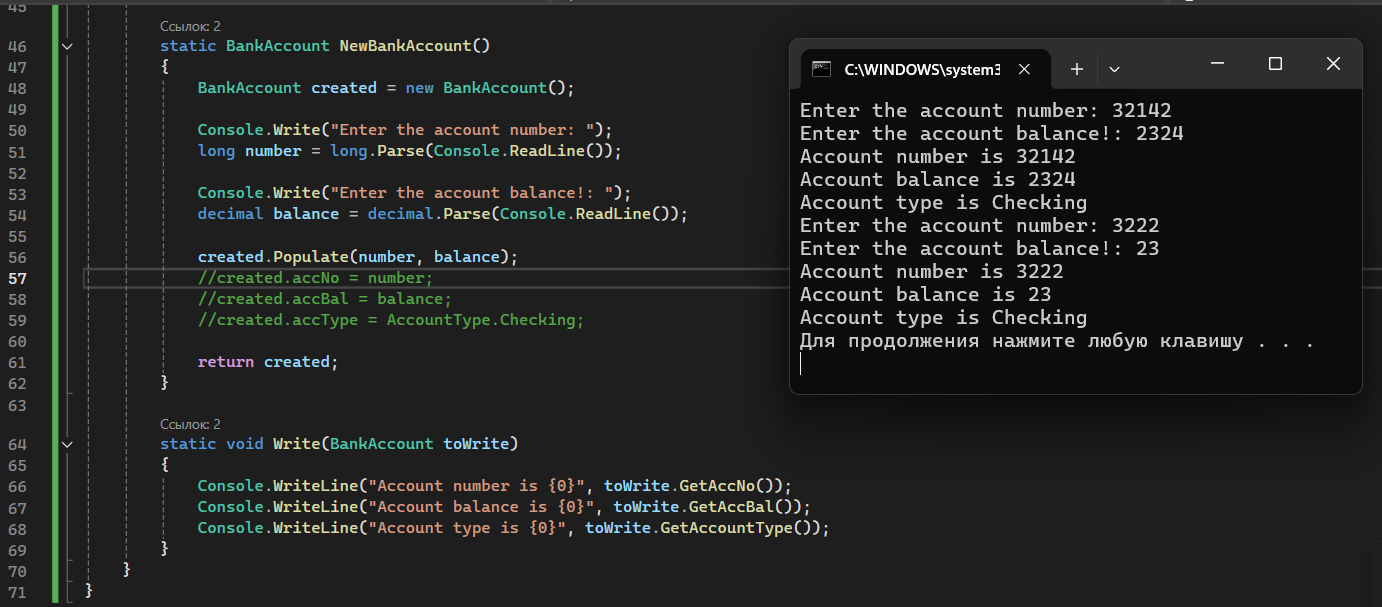


Рисунок 3 – Инкапсуляция класса BankAccount и создание методов

1. Использование методов со ссылочными параметрами.

В классе BankAccount было добавлено приватное статическое поле nextAccNo типа long. Также был написан статический метод NextNumber, который не принимает параметров и возвращает значение поля nextAccNo, увеличенное на 1.

В файле *CreateAccount.cs* были закомментированы строки, которые запрашивали у пользователя ввод номера счета. Переменная number была инициализирована как результат работы метода NextNumber(). После этих изменений программа была успешно скомпилирована, ошибки были исправлены, и проверено, что данные вводятся корректно.

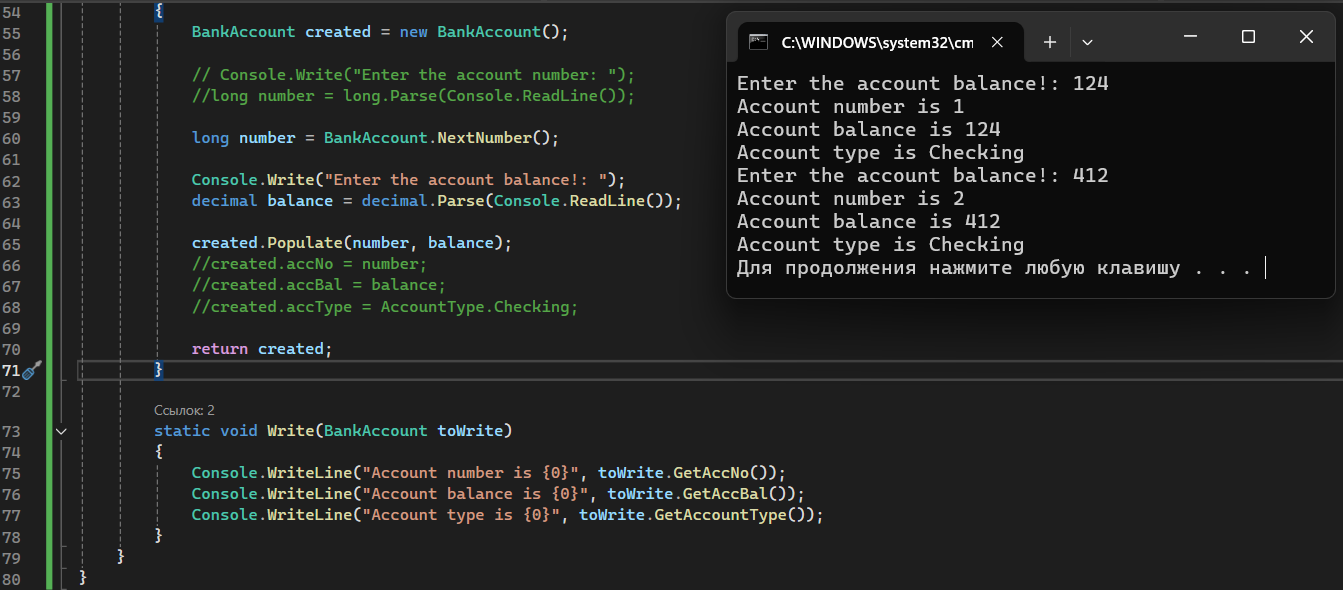


Рисунок 4 – Добавление статического поля и метода для генерации номеров

В классе BankAccount значение поля nextAccNo было явно инициализировано равным 123. После этого программа была скомпилирована, и все ошибки были исправлены. Проверено, что данные вводятся корректно, и подтверждено, что два созданных счета имеют номера 123 и 124.

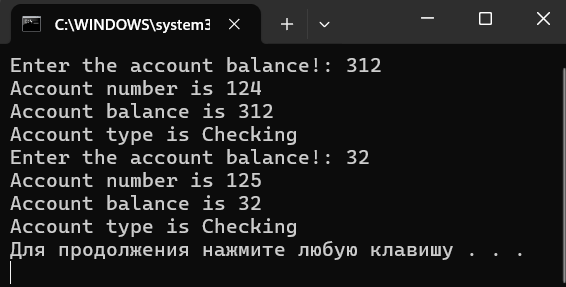


Рисунок 5 – Инициализация номера счета и проверка

В классе BankAccount была выполнена дальнейшая инкапсуляция. Метод Populate был изменен, оставив только один параметр — decimal balance. Внутри метода поле accNo было назначено с помощью статического метода NextNumber, который был изменен на private. В методе NewBankAccount было закомментировано объявление и инициализация переменной number.

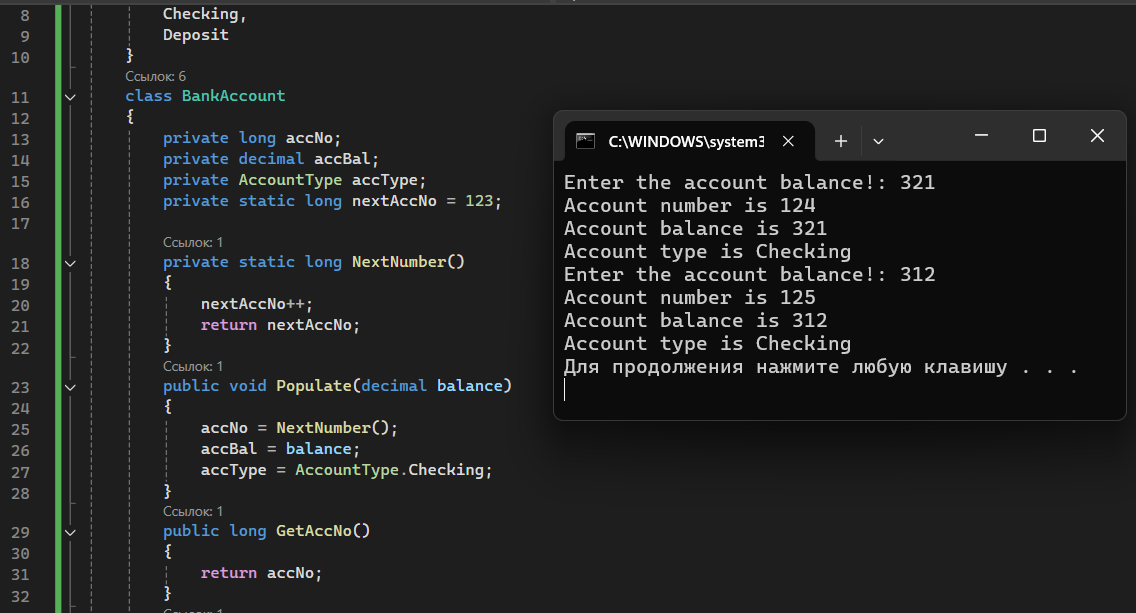


Рисунок 6 – Дальнейшая инкапсуляция и модификация метода Populate

1. Добавление методов Withdraw и Deposit.

В класс BankAccount был добавлен метод Deposit, который возвращает значение типа decimal и принимает параметр amount типа decimal. Этот параметр добавляется к балансу счета, хранящемуся в переменной accBal.

Также в класс CreateAccount был добавлен метод TestDeposit, принимающий параметр типа BankAccount. В этом методе реализована подсказка пользователю ввести сумму для депозита. Введенное значение преобразуется в десятичное и присваивается переменной amount, после чего вызывается метод Deposit с этим значением.

В метод Main были добавлены вызовы TestDeposit для параметров berts и freds, а также реализован метод Write для отображения информации о счетах после внесения депозита.

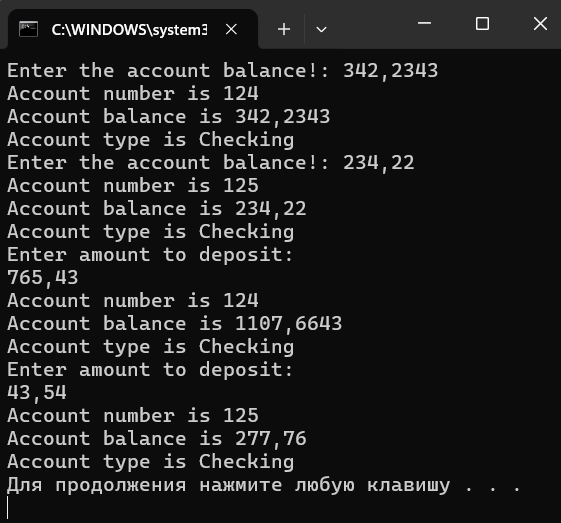


Рисунок 7 – Добавление метода Deposit и тестирования депозита

В класс BankAccount был добавлен метод Withdraw, который возвращает значение типа bool и принимает параметр amount типа decimal. Этот метод реализует логику снятия средств со счета.

В метод Main был добавлен вызов метода TestWithdraw, который использует метод Write для отображения информации о счетах после попытки снятия средств.

После внесения всех изменений программа была успешно скомпилирована, ошибки были исправлены, и проверено, что данные вводятся корректно.

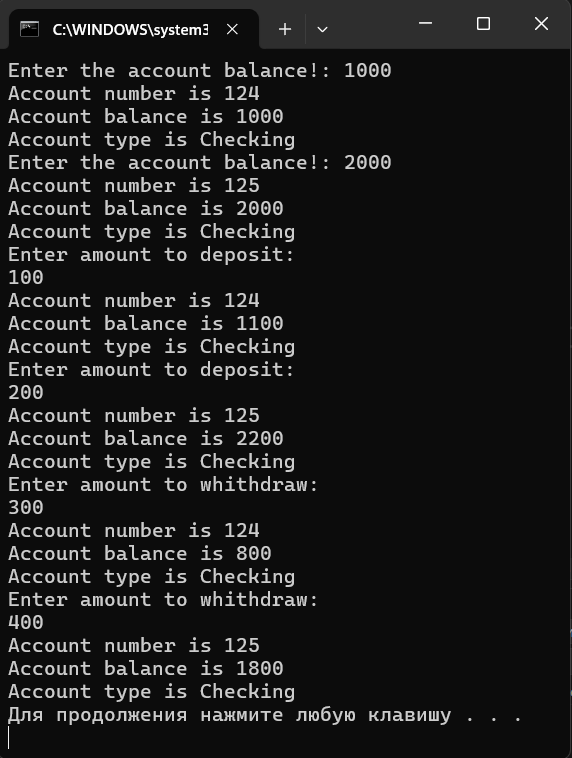


Рисунок 8 – Добавление метода Withdraw и тестирование

**ВЫВОД**

По результатам выполнения работы было установлено, что преобразование структуры в класс и применение инкапсуляции обеспечивают контроль доступа к данным через методы. Реализация методов для работы с полями объекта, а также использование статических методов для генерации номеров счетов позволяют автоматизировать процесс создания объектов и упрощают управление данными. Методы пополнения и снятия средств работают корректно, обеспечивая выполнение операций с балансом.

**ИСХОДНЫЙ КОД**

Начальный код:

using System;

namespace FileDetails

{

enum AccountType

{

Checking,

Deposit

}

struct BankAccount

{

public long accNo;

public decimal accBal;

public AccountType accType;

}

class CreateAccount

{

static void Main()

{

BankAccount berts = NewBankAccount();

Write(berts);

BankAccount freds = NewBankAccount();

Write(freds);

}

static BankAccount NewBankAccount()

{

BankAccount created;

Console.Write("Enter the account number : ");

long number = long.Parse(Console.ReadLine());

Console.Write("Enter the account balance! : ");

decimal balance = decimal.Parse(Console.ReadLine());

created.accNo = number;

created.accBal = balance;

created.accType = AccountType.Checking;

return created;

}

static void Write(BankAccount toWrite)

{

Console.WriteLine("Account number is {0}", toWrite.accNo);

Console.WriteLine("Account balance is {0}", toWrite.accBal);

Console.WriteLine("Account type is {0}", toWrite.accType.ToString());

}

}

}

Упражнение 1:

using System;

namespace FileDetails

{

enum AccountType

{

Checking,

Deposit

}

class BankAccount

{

private long accNo;

private decimal accBal;

private AccountType accType;

public void Populate(long number, decimal balance)

{

accNo = number;

accBal = balance;

accType = AccountType.Checking;

}

public long GetAccNo()

{

return accNo;

}

public decimal GetAccBal()

{

return accBal;

}

public AccountType GetAccountType()

{

return accType;

}

}

class CreateAccount

{

static void Main()

{

BankAccount berts = NewBankAccount();

Write(berts);

BankAccount freds = NewBankAccount();

Write(freds);

}

static BankAccount NewBankAccount()

{

BankAccount created = new BankAccount();

Console.Write("Enter the account number: ");

long number = long.Parse(Console.ReadLine());

Console.Write("Enter the account balance!: ");

decimal balance = decimal.Parse(Console.ReadLine());

created.Populate(number, balance);

//created.accNo = number;

//created.accBal = balance;

//created.accType = AccountType.Checking;

return created;

}

static void Write(BankAccount toWrite)

{

Console.WriteLine("Account number is {0}", toWrite.GetAccNo());

Console.WriteLine("Account balance is {0}", toWrite.GetAccBal());

Console.WriteLine("Account type is {0}", toWrite.GetAccountType());

}

}

}

Упражнение 2 (6):

using System;

using System.Data.SqlTypes;

namespace FileDetails

{

enum AccountType

{

Checking,

Deposit

}

class BankAccount

{

private long accNo;

private decimal accBal;

private AccountType accType;

private static long nextAccNo;

public static long NextNumber()

{

nextAccNo++;

return nextAccNo;

}

public void Populate(long number, decimal balance)

{

accNo = number;

accBal = balance;

accType = AccountType.Checking;

}

public long GetAccNo()

{

return accNo;

}

public decimal GetAccBal()

{

return accBal;

}

public AccountType GetAccountType()

{

return accType;

}

}

class CreateAccount

{

static void Main()

{

BankAccount berts = NewBankAccount();

Write(berts);

BankAccount freds = NewBankAccount();

Write(freds);

}

static BankAccount NewBankAccount()

{

BankAccount created = new BankAccount();

// Console.Write("Enter the account number: ");

//long number = long.Parse(Console.ReadLine());

long number = BankAccount.NextNumber();

Console.Write("Enter the account balance!: ");

decimal balance = decimal.Parse(Console.ReadLine());

created.Populate(number, balance);

//created.accNo = number;

//created.accBal = balance;

//created.accType = AccountType.Checking;

return created;

}

static void Write(BankAccount toWrite)

{

Console.WriteLine("Account number is {0}", toWrite.GetAccNo());

Console.WriteLine("Account balance is {0}", toWrite.GetAccBal());

Console.WriteLine("Account type is {0}", toWrite.GetAccountType());

}

}

}

Упражнение 2 (12):

using System;

using System.Data.SqlTypes;

namespace FileDetails

{

enum AccountType

{

Checking,

Deposit

}

class BankAccount

{

private long accNo;

private decimal accBal;

private AccountType accType;

private static long nextAccNo = 123;

private static long NextNumber()

{

nextAccNo++;

return nextAccNo;

}

public void Populate(decimal balance)

{

accNo = NextNumber();

accBal = balance;

accType = AccountType.Checking;

}

public long GetAccNo()

{

return accNo;

}

public decimal GetAccBal()

{

return accBal;

}

public AccountType GetAccountType()

{

return accType;

}

}

class CreateAccount

{

static void Main()

{

BankAccount berts = NewBankAccount();

Write(berts);

BankAccount freds = NewBankAccount();

Write(freds);

}

static BankAccount NewBankAccount()

{

BankAccount created = new BankAccount();

// Console.Write("Enter the account number: ");

//long number = long.Parse(Console.ReadLine());

//long number = BankAccount.NextNumber();

Console.Write("Enter the account balance!: ");

decimal balance = decimal.Parse(Console.ReadLine());

created.Populate(balance);

//created.accNo = number;

//created.accBal = balance;

//created.accType = AccountType.Checking;

return created;

}

static void Write(BankAccount toWrite)

{

Console.WriteLine("Account number is {0}", toWrite.GetAccNo());

Console.WriteLine("Account balance is {0}", toWrite.GetAccBal());

Console.WriteLine("Account type is {0}", toWrite.GetAccountType());

}

}

}

Упражнение 3:

using System;

using System.Diagnostics.Contracts;

namespace FileDetails

{

enum AccountType

{

Checking,

Deposit

}

class BankAccount

{

private long accNo;

private decimal accBal;

private AccountType accType;

private static long nextAccNo = 123;

private static long NextNumber()

{

nextAccNo++;

return nextAccNo;

}

public void Populate(decimal balance)

{

accNo = NextNumber();

accBal = balance;

accType = AccountType.Checking;

}

public long GetAccNo()

{

return accNo;

}

public decimal GetAccBal()

{

return accBal;

}

public AccountType GetAccountType()

{

return accType;

}

public decimal Deposit(decimal amount)

{

accBal += amount;

return accBal;

}

public bool Withdraw(decimal amount)

{

accBal -= amount;

return true;

}

}

class CreateAccount

{

static void Main()

{

BankAccount berts = NewBankAccount();

Write(berts);

BankAccount freds = NewBankAccount();

Write(freds);

TestDeposit(berts);

Write(berts);

TestDeposit(freds);

Write(freds);

TestWithdraw(berts);

Write(berts);

TestWithdraw(freds);

Write(freds);

}

static BankAccount NewBankAccount()

{

BankAccount created = new BankAccount();

Console.Write("Enter the account balance!: ");

decimal balance = decimal.Parse(Console.ReadLine());

created.Populate(balance);

return created;

}

static void Write(BankAccount toWrite)

{

Console.WriteLine("Account number is {0}", toWrite.GetAccNo());

Console.WriteLine("Account balance is {0}", toWrite.GetAccBal());

Console.WriteLine("Account type is {0}", toWrite.GetAccountType());

}

public static void TestDeposit(BankAccount acc)

{

Console.WriteLine("Enter amount to deposit: ");

string line = Console.ReadLine();

decimal amount = decimal.Parse(line);

acc.Deposit(amount);

}

public static void TestWithdraw(BankAccount acc)

{

Console.WriteLine("Enter amount to whithdraw: ");

string line = Console.ReadLine();

decimal amount = decimal.Parse(line);

acc.Withdraw(amount);

}

}

}