Bartosz Gawryszewski WCY20IJ1S1

**Praca dodatkowa – dodanie funkcjonalności do aplikacji bankowości**

*Zadania:*

1. Uniemożliwienie dodawania wpłat/wypłat o wartości 0
2. Dodanie możliwości wpłaty pieniędzy
3. Ograniczenie ilości pieniędzy na koncie CheckingAccount – w przypadku wpłaty przekraczającej limit środki mają trafić na SavingsAccount
4. Zablokować możliwość otwarcia wielu okien do wykonania transakcji

## Zad 1.

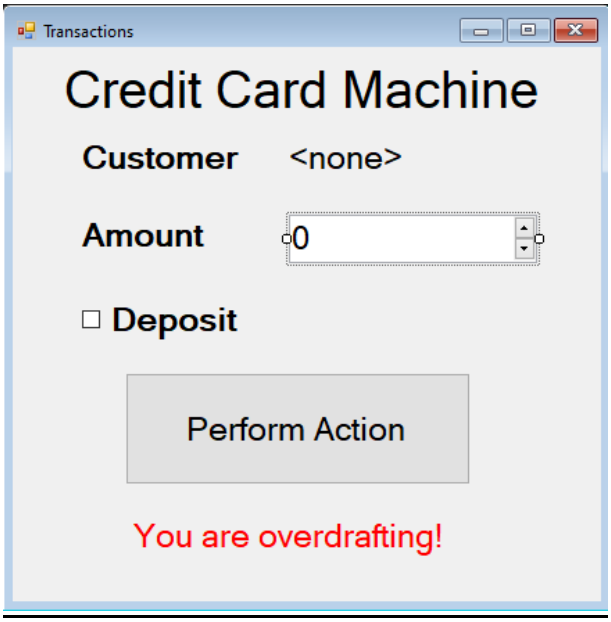
Aby uniemożliwić użytkownikowi wpłatę/wypłatę pieniędzy o wartości 0, w metodzie obsługującej zdarzenie wciśnięcia przycisku makePurchaseButton sprawdzana jest wartość przechowywana w komponencie amountValue (NumericUpDown) należącego do okna Transactions za pomocą instrukcji warunkowej if. Funkcje odpowiadające za wpłatę/wypłatę pieniędzy z kont użytkownika wykonają się tylko w przypadku, gdy wartość ta będzie większa od 0.

```
1 odwołanie
private void makePurchaseButton_Click(object sender, EventArgs e)
{
    if (!isDeposit.Checked)
    {
        //wyplata
        if (amountValue.Value > 0)
        {
            bool paymentResult = _customer.CheckingAccount.MakePayment("Credit Card Purchase", amountValue.Value, _customer.Savin

        }
    }
    else
    {
        //wplata
        if (amountValue.Value > 0)
        {
            decimal[] amounts = _customer.CheckingAccount.CheckDeposit(amountValue.Value);
            if (amounts.Length == 0)
            {
                bool paymentResult = _customer.CheckingAccount.AddDeposit("Added Deposit", amountValue.Value);
            }
            else
            {
                if (amounts[0]>0)
                {
                    bool paymentResultC = _customer.CheckingAccount.AddDeposit("Added Deposit", amounts[0]);
                }
                bool paymentResultS = _customer.SavingsAccount.AddDeposit("Added Deposit", amounts[1]);
            }
        }
    }
    amountValue.Value = 0;
}
```

Zad 2.

Aby umożliwić użytkownikowi możliwość wpłaty pieniędzy w oknie Transactions dodano komponent CheckBox o nazwie isDeposit odpowiedzialny za rozróżnianie akcji wybranej przez użytkownika. W metodzie obsługującej zdarzenie kliknięcia przycisku makePurchaseButton sprawdzane jest przy pomocy instrukcji warunkowej if, czy zaznaczono komponent CheckBox „Deposit”. Jeśli nie, wpisana przez użytkownika kwota zostaje traktowana jako wypłata – wykonywana jest metoda MakePayment klasy Account, a w przeciwnym przypadku jako wpłata – wykonywana jest metoda AddDeposit klasy Account.

```
1 odwołanie
private void makePurchaseButton_Click(object sender, EventArgs e)
{
    if (!isDeposit.Checked)
    {
        //wyplata
        if (amountValue.Value > 0)
        {
            bool paymentResult = _customer.CheckingAccount.MakePayment("Credit Card Purchase", amountValue.Value, _customer.Savin

        }
    }
    else
    {
        //wplata
        if (amountValue.Value > 0)
        {
            decimal[] amounts = _customer.CheckingAccount.CheckDeposit(amountValue.Value);
            if (amounts.Length == 0)
            {
                bool paymentResult = _customer.CheckingAccount.AddDeposit("Added Deposit", amountValue.Value);
            }
            else
            {
                if (amounts[0]>0)
                {
                    bool paymentResultC = _customer.CheckingAccount.AddDeposit("Added Deposit", amounts[0]);
                }
                bool paymentResultS = _customer.SavingsAccount.AddDeposit("Added Deposit", amounts[1]);
            }
        }
    }
    amountValue.Value = 0;
}
```



Zad 3.

Aby umożliwić użytkownikowi możliwość wpłaty pieniędzy w oknie Transactions dodano komponent CheckBox o nazwie isDeposit odpowiedzialny za rozróżnianie akcji wybranej przez użytkownika. W metodzie obsługującej zdarzenie kliknięcia przycisku makePurchaseButton sprawdzane jest przy pomocy instrukcji warunkowej if, czy zaznaczono komponent CheckBox „Deposit”. Jeśli nie, wpisana przez użytkownika kwota zostaje traktowana jako wypłata – wykonywana jest metoda MakePayment klasy Account, a w przeciwnym przypadku jako wpłata – wykonywana jest metoda AddDeposit klasy Account.

```
namespace DemoLibrary
{
    Odwołania: 2
    public class CheckingAccount : Account
    {
        Odwołania: 4
        public decimal Limit { get; set; }
        1 odwołanie
        public decimal[] CheckDeposit(decimal amount)
        {
            List<decimal> list = new List<decimal>();
            decimal value = Balance + amount;
            if (value > Limit)
            {
                decimal toSavings = value - Limit;
                decimal toChecking = amount - toSavings;
                list.Add(toChecking);
                list.Add(toSavings);
            }
            return list.ToArray();
        }
    }
}
```

```
1 odwołanie
private void LoadTestingData()
{
    customer.CustomerName = "Tim Corey";
    customer.CheckingAccount = new CheckingAccount();
    customer.SavingsAccount = new Account();

    customer.CheckingAccount.AccountName = "Tim's Checking Account";
    customer.SavingsAccount.AccountName = "Tim's Savings Account";

    // set checking account limit
    customer.CheckingAccount.Limit = 1000.0M;
    customer.CheckingAccount.AddDeposit("Initial Balance", 155.43M);
    customer.SavingsAccount.AddDeposit("Initial Balance", 98.45M);
}
```

```
1 odwołanie
private void WireUpForm()
{
    customerText.Text = customer.CustomerName;
    checkingTransactions.DataSource = customer.CheckingAccount.Transactions;
    savingsTransactions.DataSource = customer.SavingsAccount.Transactions;
    // display checking account limit
    checkingBalanceLimitValue.Text = string.Format("{0:C2}", customer.CheckingAccount.Limit);
    checkingBalanceValue.Text = string.Format("{0:C2}", customer.CheckingAccount.Balance);
    savingsBalanceValue.Text = string.Format("{0:C2}", customer.SavingsAccount.Balance);

    customer.CheckingAccount.TransactionApprovedEvent += CheckingAccount_TransactionApprovedEvent;
    customer.SavingsAccount.TransactionApprovedEvent += SavingsAccount_TransactionApprovedEvent;
    customer.CheckingAccount.OverdraftEvent += CheckingAccount_OverdraftEvent;
}
```

```
1 odwołanie
private void makePurchaseButton_Click(object sender, EventArgs e)
{
    if (!isDeposit.Checked)
    {
        //wyplata
        if (amountValue.Value > 0)
        {
            bool paymentResult = _customer.CheckingAccount.MakePayment("Credit Card Purchase", amountValue.Value, _customer.Savin

        }
    }
    else
    {
        //wplata
        if (amountValue.Value > 0)
        {
            decimal[] amounts = _customer.CheckingAccount.CheckDeposit(amountValue.Value);
            if (amounts.Length == 0)
            {
                bool paymentResult = _customer.CheckingAccount.AddDeposit("Added Deposit", amountValue.Value);
            }
            else
            {
                if (amounts[0]>0)
                {
                    bool paymentResultC = _customer.CheckingAccount.AddDeposit("Added Deposit", amounts[0]);
                }
                bool paymentResultS = _customer.SavingsAccount.AddDeposit("Added Deposit", amounts[1]);
            }
        }
    }
    amountValue.Value = 0;
}
```

Zad 4.

Aby uniemożliwić otwarcie wielu okienek Transactions po wielokrotnym wciśnięciu przycisku „Record Transactions” w metodzie obsługującym zdarzenie kliknięcia tego przycisku zmieniono sposób wyświetlania okienka z transactions.Show() na transactions.ShowDialog(). Metoda ShowDialog() blokuje wątek wywołujący, dopóki okienko to nie zostanie zamknięte.

```
1 odwołanie
private void recordTransactionsButton_Click(object sender, EventArgs e)
{
    Transactions transactions = new Transactions(customer);
    transactions.ShowDialog();
}
```