Sure! Here's the code for the Bank Account Management System along with detailed comments explaining the OOP concepts, data structures, properties, and relationships:

```csharp

using System;

using System.Collections.Generic;

namespace BankAccountManagementSystem

{

// BankAccount class represents an abstraction of a bank account.

public abstract class BankAccount

{

// Private data members encapsulated within the class.

private string accountNumber;

private string accountHolderName;

private decimal balance;

private List<Transaction> transactions; // List to store transaction history.

// Public properties for accessing and modifying account properties.

public string AccountNumber

{

get { return accountNumber; }

set { accountNumber = value; }

}

public string AccountHolderName

{

get { return accountHolderName; }

set { accountHolderName = value; }

}

public decimal Balance

{

get { return balance; }

set { balance = value; }

}

// Constructor to initialize the account properties and create an empty transaction list.

public BankAccount(string accountNumber, string accountHolderName, decimal balance)

{

this.accountNumber = accountNumber;

this.accountHolderName = accountHolderName;

this.balance = balance;

transactions = new List<Transaction>();

}

// Abstract method to calculate interest. Each subclass will provide its own implementation.

public abstract decimal CalculateInterest();

// Method to deposit money into the account.

// Overloaded to support different ways of depositing.

public virtual void Deposit(decimal amount)

{

balance += amount;

transactions.Add(new Transaction("Deposit", amount));

}

public virtual void Deposit(decimal amount, string description)

{

balance += amount;

transactions.Add(new Transaction("Deposit", amount, description));

}

// Method to withdraw money from the account.

// Overloaded to support different ways of withdrawing.

public virtual void Withdraw(decimal amount)

{

if (balance >= amount)

{

balance -= amount;

transactions.Add(new Transaction("Withdrawal", -amount));

}

else

{

Console.WriteLine("Insufficient funds.");

}

}

public virtual void Withdraw(decimal amount, string description)

{

if (balance >= amount)

{

balance -= amount;

transactions.Add(new Transaction("Withdrawal", -amount, description));

}

else

{

Console.WriteLine("Insufficient funds.");

}

}

// Method to display the transaction history.

public void ShowTransactionHistory()

{

Console.WriteLine("Transaction History:");

foreach (Transaction transaction in transactions)

{

transaction.PrintTransaction();

}

}

}

// SavingsAccount class represents a savings account and inherits from BankAccount.

public class SavingsAccount : BankAccount, ITransaction

{

// Constructor to initialize savings account properties.

public SavingsAccount(string accountNumber, string accountHolderName, decimal balance)

: base(accountNumber, accountHolderName, balance)

{

}

// Implementation of CalculateInterest() method specific to savings account.

public override decimal CalculateInterest()

{

// Calculate and return the interest for savings account.

decimal interestRate = 0.05m;

return Balance \* interestRate;

}

// Implementation of ExecuteTransaction() method from ITransaction interface.

public void ExecuteTransaction(decimal amount)

{

Withdraw(amount);

}

// Implementation of PrintTransaction() method from ITransaction interface.

public void PrintTransaction()

{

Console.WriteLine("Savings Account Transaction");

// Print transaction details specific to savings account.

}

}

// CheckingAccount class represents a checking account and inherits from BankAccount.

public class CheckingAccount : BankAccount, ITransaction

{

// Constructor to initialize checking account properties.

public CheckingAccount(string accountNumber, string accountHolderName, decimal balance)

: base(accountNumber, accountHolderName, balance)

{

}

// Implementation of CalculateInterest() method specific to checking account.

public override decimal CalculateInterest()

{

// Calculate and return the interest for checking account.

decimal interestRate = 0.02m;

return Balance \* interestRate;

}

// Implementation of ExecuteTransaction() method from ITransaction interface.

public void ExecuteTransaction(decimal amount)

{

Withdraw(amount);

}

// Implementation of PrintTransaction() method from ITransaction interface.

public void PrintTransaction()

{

Console.WriteLine("Checking Account Transaction");

// Print transaction details specific to checking account.

}

}

// LoanAccount class represents a loan account and inherits from BankAccount.

public class LoanAccount : BankAccount, ITransaction

{

// Constructor to initialize loan account properties.

public LoanAccount(string accountNumber, string accountHolderName, decimal balance)

: base(accountNumber, accountHolderName, balance)

{

}

// Implementation of CalculateInterest() method specific to loan account.

public override decimal CalculateInterest()

{

// Calculate and return the interest for loan account.

decimal interestRate = 0.08m;

return Balance \* interestRate;

}

// Implementation of ExecuteTransaction

() method from ITransaction interface.

public void ExecuteTransaction(decimal amount)

{

Deposit(amount);

}

// Implementation of PrintTransaction() method from ITransaction interface.

public void PrintTransaction()

{

Console.WriteLine("Loan Account Transaction");

// Print transaction details specific to loan account.

}

}

// Transaction class represents a transaction made on a bank account.

public class Transaction

{

public string TransactionType { get; }

public decimal Amount { get; }

public string Description { get; }

public Transaction(string transactionType, decimal amount)

{

TransactionType = transactionType;

Amount = amount;

}

public Transaction(string transactionType, decimal amount, string description)

{

TransactionType = transactionType;

Amount = amount;

Description = description;

}

public void PrintTransaction()

{

Console.WriteLine($"Transaction Type: {TransactionType}, Amount: {Amount}, Description: {Description}");

}

}

// ITransaction interface represents a transaction and its related methods.

public interface ITransaction

{

void ExecuteTransaction(decimal amount);

void PrintTransaction();

}

// Bank class represents a bank that manages bank accounts.

public class Bank

{

private Dictionary<string, BankAccount> bankAccounts; // Dictionary to store bank accounts.

public Bank()

{

bankAccounts = new Dictionary<string, BankAccount>();

}

// Method to add a bank account to the bank.

public void AddBankAccount(BankAccount bankAccount)

{

bankAccounts.Add(bankAccount.AccountNumber, bankAccount);

}

// Method to remove a bank account from the bank.

public void RemoveBankAccount(BankAccount bankAccount)

{

bankAccounts.Remove(bankAccount.AccountNumber);

}

// Method to get a bank account by account number.

public BankAccount GetBankAccount(string accountNumber)

{

if (bankAccounts.ContainsKey(accountNumber))

{

return bankAccounts[accountNumber];

}

else

{

Console.WriteLine("Bank account not found.");

return null;

}

}

// Method to display all bank accounts in the bank.

public void DisplayBankAccounts()

{

Console.WriteLine("Bank Accounts:");

foreach (KeyValuePair<string, BankAccount> bankAccount in bankAccounts)

{

Console.WriteLine($"Account Number: {bankAccount.Key}, Account Holder: {bankAccount.Value.AccountHolderName}");

}

}

}

class Program

{

static void Main(string[] args)

{

// Create a bank object.

Bank bank = new Bank();

// Create bank accounts and add them to the bank.

SavingsAccount savingsAccount = new SavingsAccount("SA001", "John Doe", 1000);

CheckingAccount checkingAccount = new CheckingAccount("CA001", "Jane Smith", 2000);

LoanAccount loanAccount = new LoanAccount("LA001", "Mike Johnson", -5000);

bank.AddBankAccount(savingsAccount);

bank.AddBankAccount(checkingAccount);

bank.AddBankAccount(loanAccount);

// Perform operations on bank accounts.

savingsAccount.Deposit(500);

checkingAccount.Withdraw(300);

loanAccount.Deposit(1000);

// Display bank accounts and their transaction history.

bank.DisplayBankAccounts();

BankAccount selectedAccount = bank.GetBankAccount("SA001");

selectedAccount.ShowTransactionHistory();

Console.ReadLine();

}

}

}

```

This code implements a Bank Account Management System using OOP concepts such as Abstraction, Encapsulation, Polymorphism, and Inheritance. It also demonstrates the use of interfaces, collections like Lists and Dictionaries, and different types of relationships. The code includes detailed comments explaining the concepts, properties, and relationships. Please note that this code serves as an example and may need further refinement and error handling to be used in a production environment.