In Module 7, you had learnt about constructors, destructors, access modifiers and inheritance. Object-oriented programming comes with several features and you will explore them in each new module.

In this module, you will see how the real strength of object-oriented programming is demonstrated using encapsulation and polymorphism.

Encapsulation ensures the prevention of the code or data from accidental corruption due to errors. In a class, methods that operate on the data in the same class are defined, so the data is always prone to accidental modification from outside methods.

In order to understand the concept of polymorphism, it is essential to learn about method overloading and overriding first. Method overloading is compile time polymorphism and method overriding is run time polymorphism. If a class has several methods with the same name but different parameters (signature), this is called method overloading. However, in method overriding, a derived class has the same method as declared in the base class. Polymorphism, in C#, is the ability of objects of different types to provide a unique interface for different implementations of methods.

In this module, you will learn about the following topics:

Encapsulation

Method overloading and overriding

Polymorphism.

**Encapsulation :**

[C# - Encapsulation (tutorialspoint.com)](https://www.tutorialspoint.com/csharp/csharp_encapsulation.htm)

**Method overloading and overriding :**

[Overloading and Overriding in C# | Working | Examples | Advantages (educba.com)](https://www.educba.com/overloading-and-overriding-in-c/)

**Polymorphism :**

[C# Static and Dynamic Polymorphism - Studytonight](https://www.studytonight.com/post/csharp-polymorphism)

[OOP Concepts in C# - Polymorphism, Interfaces and Inheritance | codeeasy.io](https://codeeasy.io/lesson/polymorphism)

**Exercise :**

**3.4   How private are private instance variables?**

|  |  |
| --- | --- |
|  | The purpose of this exercise is to find out how private *private instance variables* are in C#.  Given the [BankAccount](https://homes.cs.aau.dk/~normark/oop-csharp/html/notes/source-programs/classes/user/normark/oop-csharp-1/sources/c-sharp/bank-account/bank-account-0a/bank-account.txt) class. Now modify this class such that each bank account has a backup account. For the backup account you will need a new (private) instance variable of type BankAccount. Modify the Withdraw method, such that if there is not enough money available in the current account, then withdraw the money from the backup account. **As an experiment, access the balance of the backup account directly, in the following way:**  backupAccount.balance -= ...  Is it possible to modify the private state of one BankAccount from another BankAccount? Discuss and explain your findings. Are you surprised? |

**Solution**

|  |  |
| --- | --- |
|  | It is possible for one BankAccount object to modify the balance of another BankAccount object. The version of class BankAccount shown below demonstrates this. Please notice the line  backupAccount.balance -= amount - balance;  in the method withdraw.  Is this reasonable? Yes. Our observations pertain to objects in the running program - the dynamic situation. Our attention to firewalls (icebergs and representation independence) is oriented towards the source program - the static situation. In relation to programmers, who write the BankAccount class and related classes, it is important to enforce the discipline that the data representation of foreign classes is invisible. Without such a policy the idea of *representation independence* will be shipwrecked. This is a software engineering concern.  It causes no software engineering problems if one BankAccount object can modify the data of another BankAccount object in the way sketched above.  Here follows my programmed solution to this exercises: |

Solution :

using System;

public class BankAccount {

private double interestRate;

private string owner;

private double balance;

private BankAccount backupAccount;

public BankAccount(string owner, double interestRate,

BankAccount backupAccount) {

this.interestRate = interestRate;

this.owner = owner;

this.balance = 0.0;

this.backupAccount = backupAccount;

}

public BankAccount(string owner, double interestRate):

this(owner, interestRate, null){

}

public BankAccount(string owner): this(owner, 0.0) {

}

public double Balance () {

return balance;

}

public void withdraw (double amount) {

if (balance >= amount)

balance -= amount;

else if (balance < amount && backupAccount != null){

backupAccount.balance -= amount - balance;

balance = 0;

}

else throw new Exception("Help!");

}

public void deposit (double amount) {

balance += amount;

}

public void addInterests() {

balance = balance + balance \* interestRate;

}

public override string ToString() {

return owner + "'s account holds " +

+ balance + " kroner";

}

}