Non Object-Oriented Code Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

class NonObjectOrientedCodeExample
{
    static void Main(string[] args)
    {
        object[] person = new object[3];
        person[0] = "Bob"; // First name
        person[1] = "Smith"; // Last name
        person[2] = new DateTime(1990, 6, 20); // Date of birth 20/06/1990

        object[] account = new object[4];
        account[0] = 062620; // Bank Code: 06 | State code: 3 | Branch code: 262
        account[1] = 12341234; // Account number
        account[2] = 500.00; // Balance
        account[3] = person; // Account holder

        PrintBankDetails(account);
        WithdrawMoney(account, 250.00);
        WithdrawMoney(account, 750.00);
        PrintBankDetails(account);
        WithdrawMoneyVersion2(account, 750.00);
        PrintBankDetails(account);
    }

    static void PrintBankDetails(object[] account)
    {
        Console.WriteLine("Bank Details");
        Console.WriteLine("BSB           : {0}", account[0]);
        Console.WriteLine("Account Number: {0}", account[1]);
        Console.WriteLine("Balance       : {0:C}", account[2]);

        // Need to cast because complier doesn't know that a "person" is at position 3.
        object[] person = (object[]) account[3];

        Console.WriteLine("Account Owner : {0} {1}", person[0], person[1]);
        Console.WriteLine();
    }

    static void WithdrawMoney(object[] account, double amount)
    {
        // Again need to cast because complier doesn't know that a "balance" is at position 2.
        double balance = (double) account[2];

        // Balance shouldn't become negative.
        if(amount <= balance)
        {
            account[2] = balance - amount;
        }
    }

    static void WithdrawMoneyVersion2(object[] account, double amount)
    {
        // Again need to cast because complier doesn't know that a "balance" is at position 2.
        double balance = (double) account[2];

        // Who wrote this code? ...I thought balance was not meant to become negative?
        account[2] = balance - amount;
    }
}
```

Notes:

- Offset numbers were coded into the program - offsets can change when attributes are added or removed from the account and person entities.
- Several casts were required because the complier was not aware of what was happening - thus less compile time checks are performed resulting in potential errors deferred to runtime.
- Any portion of code could modify the data in an unexpected way - this was demonstrated in the WithdrawMoneyVersion2(...) method.
- This planted "bug" of modifying the balance to be negative did **not** result in the program crashing at that point in time - and in this example the "bug" was not detected until the account details were printed.
- Thus incorrect code, which can be in any file, could potential corrupt correct code leaving you unsure why the problem has occurred and where to look to fix it - even in a high level language such as C#.

Object-Oriented Code Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

class Account
{
    public Account(int bsb, int accountNumber, double balance, Person accountHolder)
    {
        BSB = bsb;
        AccountNumber = accountNumber;
        Balance = balance;
        AccountHolder = accountHolder;
    }

    public int BSB { get; private set; }
    public int AccountNumber { get; private set; }
    public double Balance { get; private set; }
    public Person AccountHolder { get; private set; }

    public void WithdrawMoney(double amount)
    {
        // Balance shouldn't become negative.
        if(amount <= Balance)
        {
            Balance -= amount;
        }
    }
}

class Person
{
    public Person(string firstName, string lastName, DateTime dateOfBirth)
    {
        FirstName = firstName;
        LastName = lastName;
        DateOfBirth = dateOfBirth;
    }

    public string FirstName { get; private set; }
    public string LastName  { get; private set; }
    public DateTime DateOfBirth { get; private set; }
}

class ObjectOrientedCodeExample
{
    static void Main(string[] args)
    {
        Person person = new Person("Bob", "Smith", new DateTime(1990, 6, 20));

        Account account = new Account(062620, 12341234, 500.00, person);

        PrintBankDetails(account);
        account.WithdrawMoney(250.00);
        account.WithdrawMoney(750.00);
        PrintBankDetails(account);
        account.WithdrawMoney(750.00);
        PrintBankDetails(account);
    }

    static void PrintBankDetails(Account account)
    {
        Console.WriteLine("Bank Details");
        Console.WriteLine("BSB            : {0}", account.BSB);
        Console.WriteLine("Account Number: {0}", account.AccountNumber);
        Console.WriteLine("Balance        : {0:C}", account.Balance);

        // Don't need to cast now - complier is more informed of what is happening.
        Person person = account.AccountHolder;

        Console.WriteLine("Account Owner : {0} {1}", person.FirstName, person.LastName);
        Console.WriteLine();
    }
}
```

Notes:

- The Account and Person details have been flushed out as a class - the class code describes these entities.
- The complier is more informed of what is happening - the need for offset values is replaced with meaningful names and the names are also bound to a datatype that the complier is aware of. This reduces the number of potential runtime errors.
- Due to the private setter of the properties within the Account and Person classes, only code within those classes can modify the data they hold - thus the balance cannot be accidently set to a negative value anywhere else in the program.