|  |
| --- |
| **SIT232 Object-Oriented Development** |
| **Session 1 Solution** |

**Task 1.1**

There is no clear right or wrong answer to this question. The aim of the question is to start you thinking about objects in a problem statement, which is the first step in building an object-oriented solution. For any problem, there can be many correct object-oriented models used to solve the problem. The following answers represent only one example. It is also worth noting that the identification of objects, classes, attributes, and operations, will become much clearer as you gain experience developing software.

**a. Possible objects:**

* Web store;
* Computer;
* Software;
* Entertainment System;
* XBox System;
* PlayStation System;
* HiFi Product;
* Blu-Ray Player;
* PVR;
* Smart Phone;
* Tablet;
* Accessory;
* Cable;
* Purchase;
* Logistics Company;
* Postal Service;
* Courier;
* Supplier / Distributor / Wholesaler;
* Customer;
* Account Payable;
* Account Receivable;
* Clearing Account;
* Electronic Payment Gateway; and
* Bank.

Note that not all of these objects would necessarily appear in a real system, nor would the list of objects in a real system necessarily be restricted to this list of objects, i.e., there may be other objects not listed here.

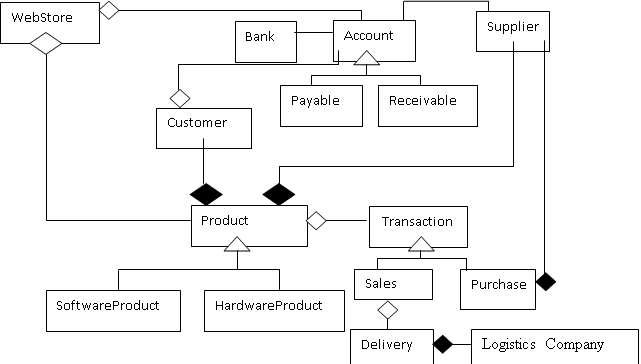
**b. Possible classes:**

The list of classes will depend the list of objects identified in (a), above. A class represents a classification, or category, of the objects in the system. Examples:

* Computer, Entertainment System, XBox System, PlayStation System, HiFi Product, Blu-Ray Player, PVR, Smart Phone, and Tablet all represent individual electronics products and usually have common attributes such as brand, model, warranty, and so on, and could be be covered by a single class 'ElectronicsProduct';
* Modern software can be sold as a separate product (such as a game) or as a subscription (such as Office 365) and so may have unique properties requiring a different class 'SoftwareProduct', or potentially could be combined with the previous class as a more general 'Product' in the system.
* Logistics Company, Postal Service, and Courier could be covered by a single class 'LogisticsCompany' (fundamentally the postal service and courier companies are the same, just with varying service levels).
* Account Payable, Account Receivable, and Clearing account could each be covered by a single class 'Account'

Note that further classes could be used to represent objects not listed here, e.g., printer, computer game, USB stick, etc. Additionally, class names do not need to strictly match the information above, but should generally be representative of the information they model.

**UML**



Logistics Company

Delivery

Purchase

Sales

HardwareProduct

SoftwareProduct

Transaction

Bank

Payable

Receivable

Account

Customer

Supplier

Product

WebStore

**c. Possible attributes and operations:**

The attributes and operations will depend on the classes identified in (b), above. Here are some examples:

'Product' class:

* Attributes: brand, model, release date, cost price, sales price, special sales price, sale start/end date, purchase history, supplier, stock level, etc.
* Operations: record sale, record return, record purchase of new stock, etc.

'LogisticsCompany' class:

* Attributes: provider, collection dates, servces, link to account payable (and potentialy account receivable if cash on delivery supported), etc.
* Operations: record delivery request, track delivery, print delivery label, etc.

'Account' class:

* Attributes: name, transaction history, balance/amount, due date, etc.
* Operations: record increase, record decrease, adjust balance, etc.

**Task 1.2**

The solution to this task can be found in the following file/s:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* File: FirstProgram.cs

\*\* Author/s: Justin Rough

\*\* Description:

\*\* A simple program used to introduce the basic structure

\*\* of a C# application.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using System;

namespace FirstProgram

{

class FirstProgram

{

// Main function, where the program’s execution begins

static void Main(string[] args)

{

Console.WriteLine("Welcome to OO Development!");

}

}

}

**Task 1.3**

For most data, there is no one truly correct data type. When considering the data itself, there is often one data type that is the most efficient and appropriate at storing data, but this can change depending on the application and the platform upon which the application is running.

* The day of the month, e.g., 5
  + Almost any numeric type could be used to store this data, but note that the range of valid values is 1-31, so an unsigned data type might often be preferable, such as the byte data type, which has the range 0-255.
* The salary an employee , e.g., $68,000
  + Many numeric types could also be used to store this data starting with the int or uint data type (depending on whether negative values are required or not –most of job advertising salary does not have cent details). If the currency value was to also include a decimal value (for cents) however, the decimal data type should be used in preference to the float or double data types. It is important to note that float and double types are good for performing mathematical calculations, however they are not necessarily accurate at storing the decimals. It is not unusual for example to find a value of 1.1 stored in a float variable to unexpectedly become 1.09999999999... which is not very useful for most currency related applications. The decimal type was introduced to provide precise representation of decimal values (such as currency values) at the cost of performance in mathematics calculations.
* The price of an item for sale, e.g., $24.95
  + As per the explanation for salary, the float, double and decimal type are the most appropriate.
* The value of PI (3.1415926535897932384...)
  + The value of PI would ordinarily be used for (many) mathematics calculations and in general should be stored in a double data type. Importantly, the value of PI is already pre-defined in the Microsoft.Net environment as Math.PI
* The grade received for a unit (e.g., C for credit)
  + Your immediate thoughts might be that the char data type might be the most appropriate data type for storing a value of 'C'. However, when programming you must be careful to select data types that fulfil all requirements, not just those that are immediately obvious. If you consider the grades that you can be awarded at Deakin, they include HD, D, C, P, and N. Note that the first result listed here, HD, has two characters, which would not fit into a char data type. Thus, string may be a more appropriate data type here.

**Task 1.4**

The solution to this task can be found in the following file/s:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* File: InputOutput.cs

\*\* Author/s: Justin Rough

\*\* Description:

\*\* A program that prompts the user to enter in contact

\*\* information (family name, given name, title, country of

\*\* birth, and spoken languages) and then displays the

\*\* information in a formatted report.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using System;

namespace InputOutput

{

class InputOutput

{

static void Main(string[] args)

{ // could use loop to check the input but leave them for next week tasks

Console.Write("What is your family name? ");

string familyName = Console.ReadLine();

Console.Write("What is your given name? ");

string givenName = Console.ReadLine();

Console.Write("What is your title (Mr, Mrs, Miss, Ms, etc.)? ");

string title = Console.ReadLine();

Console.Write("What is your country of birth? ");

string country = Console.ReadLine();

Console.Write("What languages do you speak? ");

string languages = Console.ReadLine();

/\*

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" Field \tValue");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" Name:\t{0} {1} {2}", title, givenName, familyName);

Console.WriteLine(" Born In:\t{0}", country);

Console.WriteLine(" Speaks:\t{0}", languages);

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

\*/

Console.WriteLine("{0}", new string('\*', 50));

Console.WriteLine("{0,10}{1,6}{2}", "Field", " ", "Value");

Console.WriteLine("{0,10}{1,6}{2} {3} {4}", "Name", " ", title, givenName, familyName);

Console.WriteLine("{0,10}{1,6}{2}", "Born In", " ", country);

Console.WriteLine("{0,10}{1,6}{2}", "Speaks", " ", languages);

Console.WriteLine("{0}", new string('\*', 50));

}

}

}

**Task 1.5**

The corrections are as follows:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* File: ThreeNumbers.cs

\*\* Author/s: Justin Rough

\*\* Description:

\*\* A simple program that calculates the sum of three

\*\* numbers and what percentage/factor of the sum the

\*\* three numbers contribute. This program is provided

\*\* with many syntax errors as an exercise in correcting

\*\* such errors.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using System;

public class ThreeNumbers

{

public static void Main()

{

string input;

// could use loop to check the input but leave them for next week tasks

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

input = Console.ReadLine();

int num1 = Convert.ToInt32(input);

Console.Write("Enter the second number: "); // missing closing string here

input = Console.ReadLine();

int num2 = Convert.ToInt32(input);

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

//input = Console.ReadLine(); // shorter this out

int num3 = Convert.ToInt32(Console.ReadLine()); // logical error get num3 NOT num2

//Console.WriteLine();

int sum = num1 + num2 + num3;

Console.WriteLine("**\n**The sum of the numbers is {0}", sum);

Console.WriteLine();

float factor = Convert.ToSingle(num1) / sum \* 100f;

Console.WriteLine("The number {0} represents {1,3:f}% of the sum", num1, factor);

// syntax error: factor already defined – reuse factor to store new calculation result

factor = Convert.ToSingle(num2) / sum \* 100f;

// run-time error: :f after the ,3

Console.WriteLine("The number {0} represents {1,3:f}% of the sum", num2, factor);

// syntax error: factor already defined – reuse factor to store new calculation result

factor = Convert.ToSingle(num3) / sum \* 100f;

// No f% notation in string control format

Console.WriteLine("The number {0} represents {1,3:f}% of the sum", num3, factor);

}

}

**Term Definitions**

***abstraction***

Abstraction allows us to focus on those aspects of the problem that are important/relevant to the solution while ignoring those aspects that are irrelevant.

***associativity***

When two or more operators have equal precedence, associativity determines the order of evaluation for those operators. Operators can be either left associative or right associative.

***arithmetic expression***

An equation used for performing mathematical formulas using the rules of arithmetic, i.e., an expression consisting of addition, subtraction, multiplication, division, and/or remainder after division.

***attributes***

A small piece of data describing an object in some way, e.g., the state of a light switch (on/off), the ID number of a student (900123456), or the length of a queue.

***binary operator***

An operator that requires two operands, e.g., a + b

***class***

A grouping of similar objects into a single classification. A class also refers to the means by which we are able to define objects in an object-oriented application, where by the class forms a template for the creation of an object.

***code block***

Several statements grouped together and bounded by braces ({ }), used to collect those statements into a single logical grouping. Code blocks are used for define/bound the statements for a function, and for control structures (if, while, for, etc.).

***code readability***

How easy a section of programming code is to read and understand. This is particularly important given that the vast majority of time spent with code will be for maintenance, i.e., extending or correcting the code.

***complexity***

Large problems involve a lot of complexity, where it can be very difficult to break this functionality into a clear design for coding the application. Software development methodologies, such as object-oriented development, provide us with an approach to master this complexity, to decompose such complex problems into more manageable/solvable tasks.

***compound expression***

The combination of two or more operators into a single expression, e.g., a + b \* c

***data type***

All data is stored by a computer in binary. The functionality defined by us as programmers will interpret this binary data in different ways, thus we need to indicate the nature of the data and how it should be interpreted to the compiler, i.e., is the data an integral type (int, long, etc.), string, and so on.

***data type conversion***

The ability to change the way that data is represented/interpreted by a computer, e.g., all data retrieved from the user is a string type, but we may need to change that to a numeric data type such as an int or float.

***encapsulation***

Encapsulation, also known as information hiding or data hiding, maintains a separation between the external view of an object (its interface) and the implementation of that interface.

***escape sequence***

A sequence of characters/text that has special meaning to an input/output routine, e.g., \n for new line, \t for (horizontal) tab, and so on.

***expression***

Used in program code wherever a value can be specified, an expression is some combination of values, variables, operators, function calls, and so on, potentially combined together using arithmetic, to specify/calculate some value.

***formatted output***

Output that is sent to the use on the console in a structured manner. The structure used can be simplistic, such as using a tab, a field width, or a special formatting such as currency formatting, or can be combined together into more complex output such as a table/report.

***Hungarian notation***

One style of naming variables which was very popular in the late 1980s and early 1990s, whereby a short prefix is added to a variable name to indicate its data type, e.g., instead of a string variable FirstName, you can have strFirstName.

***indent style***

Correctly indentation of code can substantially improve the readability of that code. For example, each code block for a control structure (if, for, while, etc.) should be indented/tabbed in to show more clearly what code is part of/being used by those control structures. There are a number of indent styles used.

***interface (OO concept)***

The abstraction that is presented by an object, i.e., what attributes and operations are presented through which we can interact with that object.

***left associative***

Operators that are evaluated starting with the left-most operator, before evaluating the operators to the right.

***literal***

Refers to the specification of a single value in program code, e.g., in the statement int x = 5;, the literal is 5.

***object***

Objects are the fundamental building blocks of object-oriented applications, where an object-oriented application consists of several objects that communicate together to achieve the functionality of the program. Objects are defined by classes, thus it can also be said that an object is an instance of a class.

***object-oriented development***

The dominant software development methodology used today to decompose a complex application into smaller problems that we either know the solution to, or can solve in a relatively short time.

***operations***

Defines the behaviour (actions and reactions) of an object, e.g., turning on the light, enrolling a student, or adding another element to the queue.

***operand***

A value/variable/function call that is used as an input/parameter to an operator.

***operator***

A special character sequence used in program code to tell the compiler that a particular operation is required to be performed on one or more operands.

***precedence***

The priority or order to which operators are to be evaluated in a compound expression.

***reusability***

One of the key advantages of object-oriented development, a fully developed class can be reused many times in the same or different application with no modification.

***right associative***

Operators that are evaluated starting with the right-most operator, before evaluating the operators to the left.

***simple data type***

A trivial element of data that specifies only one value, e.g., integers (5), characters ('c'), doubles (1.2345) are all considered simple data types. A data type that specifies more than one value is known as a complex data type.

***statement***

The smallest unit of instruction in a program, that tells the computer to perform some function as part of a larger program/application.

***syntax error***

Where the rules of the programming language, i.e., its syntax, have been broken, a syntax error is produced by the compiler indicating the location and a description of how the rules were broken.

***ternary operator***

An operator that requires three operands, e.g., a > b ? a : b

***unary operator***

An operator that requires only one operands, e.g., -5

***variable***

A named memory location where data can be stored.

***variable initialisation***

When a variable is declared, it can also be provided with an initial/starting value.