CHAPTER 01: INTRODUCTION TO COMPUTING AND APPLICATION DEVELOPMENT

Software consists of programs, which are a set of instructions.

**System software:** It is a software designed to provide a platform to another software.

(e.g., Operating systems, compilers, interpreters and assemblers)

**Application software:** It is a software that performs a specific task.

(e.g., word documents, spreadsheets)

**Compilers vs. Interpreters:**

c. Only convert a high-level language after removing all the syntax errors.

i. Translate one statement at a time, and execute it right after.

c. Fast.

i. Slow.

**Software development process:**

1. Analyze
2. Design
3. Code
4. Implement
5. Test and debug

1 – Without a clear understanding of the problem, there is no way the team will solve it. In this step of the process, the team is obligated to rise specifications.

1.1. Input?

1.2. Values?

1.3. Domain (range of values)

1.4. There will be interaction with the user? Is it going to enter data?

1.5. What remains constant?

2 – Several methods are used at this phase. For example:

2.1. The top-down design or step-down refinement approach uses the divide-and-conquer paradigm, which consists of breaking the problem from the overview perspective into small subtasks until it is so simple, you can solve it right away.

2.2. Process-oriented approach, which consists of determining the data flow throughout the program.

2.3. Object-oriented approach, which consists of determining entities that represent data and behaviours.

2.4. The algorithm is a straightforward step-by-step process for solving a problem.

3 – Chose a high-level programming language and write the source code.

4 – Compile the code. (Flow for C# program compilation)

4.1. Source code.

4.2. Compile.

4.3. Generates the **MSIL** (**M**icrosoft **I**ntermediate **L**anguage).

4.4. **CLR** (.NET **C**ommon **L**anguage **R**untime) use the **JIT compiler** (**J**ust-**I**n-**T**ime) with the loader.

4.5. Readable machine code.

5 – Use of **TDD** (**T**est **D**riven **D**evelopment) and/or **BDD** (**B**ehavior **D**riven **D**evelopment).

**Important:** .Net is a software environment that provides services in a layer between the operating and application systems. In a nutshell, it is a framework that provides an environment for developing and running code.

**Structured Procedural Programming:**

* Process-oriented.
* Follow the five steps of **SOFTWARE DEVELOPMENT PROCESS**.
* Tools such as flowchart and pseudocode.
* Top-down design.
* High cost with maintenance.

**Object-Oriented Programming:**

* Define entities that encapsulates attributes (data) and (methods) behaviours.
* Objects and classes. (objects are instances of a class)
* **UML** (**U**nified **M**odeling **L**anguage) is used as a tool.
* Inheritance.
* Polymorphism.

**Elements of a C# Program:**

* Using directives
  + Permit the use of classes inside a namespace without qualifying them.
* Namespaces
  + Provide a scope to group semantically related types.
* Classes
  + Building block of the OO paradigm. Every program needs at least one class.
* Main()
  + Entry point for an application
  + Every application needs to have ONE Main() method

CHAPTER 02: DATATYPES AND EXPRESSIONS

**Data Representation:**

The computer represents data using the binary system because it understands 0’s and 1’s, a discrete signal.

The 0 and 1 representation are called a **bit,** and they represent states on and off of that discrete signal.

The different combinations of states can represent a character, a colour. And for that, computers use an 8-bits architecture, also known as a **byte**. One byte is equal to 8-bits and can represent symbols, for example.

**Memory Locations for Data:**

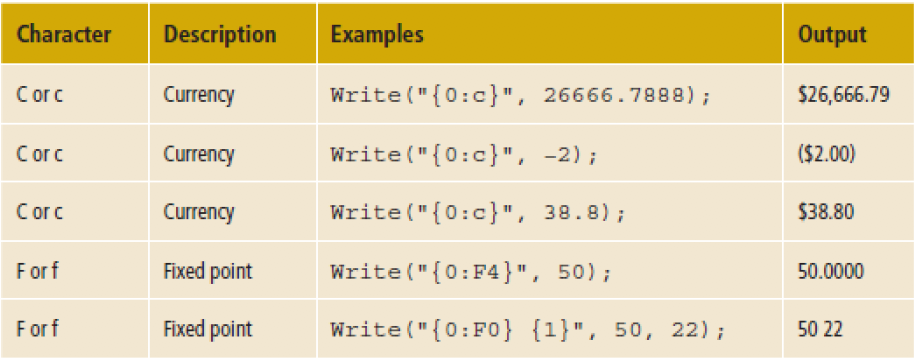
* Identifier
  + (a-z and A-Z) + (0-9) and underscore
  + Can’t initiate with a number
  + Can’t have any other special character besides the underscore and the at symbol (@)
* Pascal case
  + ThisIsPascalCase
  + Namespaces, Classes, Methods and properties identifies
* Camel case
  + thisIsCamelCase
  + variables and objects

**Data Types:**

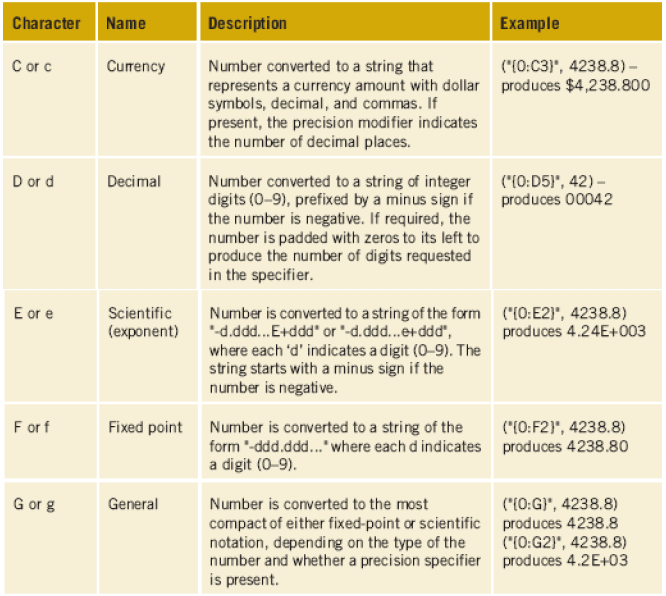
* Integral:
  + int a = 1;
* Floating point:
  + float a = 1.1e+3; // 1,100
  + float a = 1.1e-3; // 0.0011
  + float a = 1.1 or float a = 1.1F (f or F);
  + double a = 1.10D (d or D)
  + decimal a = 123456.12M (m or M)
* Boolean:
  + Does not accept numerical values, only true or false.
  + bool isFinished = true;
* String: string a = “a”;

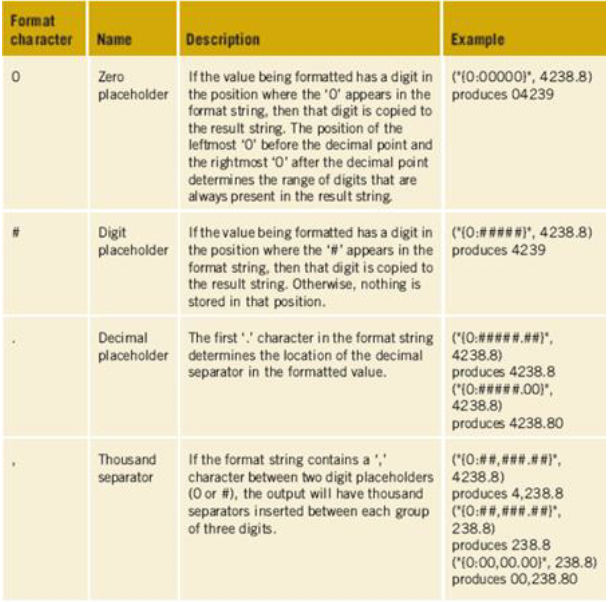
Each and every type is a class in C#. Every time you are instantiating a variable, you are instantiating a class.

**Formatting Output:**



**Numeric Format Specifiers:**





The order in which calculations are performed is called the order of operations. Parentheses can be added to an expression to alter the order.

Left associative means that the operations are performed from left to

right. This means that as you move from left to right in an expression,

the operation that is encountered first is executed first.

CHAPTER 03: METHODS AND BEHAVIOURS

[Modifiers] [Return Types] [Method Name]( [parameters] )

* Modifiers:
  + Static
  + Access (public, private, protected, internal, protected internal)
* Return type:
  + Void
  + Any other type like int, float, double, string…
* Method name:
  + Pascal case convention
* Parameters:
  + Camel case convention
  + Void (doesn’t need to be declared if you don’t have parameters)
  + Official input types, from data type.

**Predefined Methods:**

* ReadLine()
  + Returns string
  + Not overloaded
  + It will enter the characters until it “sees” that the enter key was pressed
* Write and WriteLine()
  + Returns void
  + Receive a string as an argument
  + overloaded

CHAPTER 04: METHODS AND BEHAVIOURS

**Classes:**

* Attributes (data types).
* Behaviours (methods).
* Objects are instances of a class.
* Templates, blueprints of an object.

**Constructor:**

* Special method inside a class.
* Do not include the keyword void or any other returning type.
* Default constructor: do not receive any parameters as input.
  + They initialize the data types (data members) to a default value.
* Use public access modifiers.
* Overloaded

**General methods:**

* Usually you don’t use the static keyword, only if you need it to be instantiated without and explicit object.

**Accessors:**

* Getters
* Returns the current value of the value inside the class.
* Provides a way to access private modifiers protecting the integrity of the data.
* You can “overload”, implementing what the getter can return.
* Method name is usually the Pascal case of the camel case name variable.

**Mutator:**

* Setters.
* Receive one parameter.
* Provides a way to change private modifiers protecting the integrity of the data.
* Method name is usually the Pascal case of the camel case name variable.
* Can be overloaded

**Property:**

* Can replace accessors and mutators.
* Get and set
* Auto implemented get and set

CHAPTER 05: MAKING DECISIONS

**Selection statements:**

* If statement
* Switch

**Iteration:**

* Looping
* While
* For
* Do-while

**TryParse():**

* Public static bool.
* If( int.TryParse(int var1, out var2) == false).
* Int var1 is the string value returned from ReadLine().
* Var2 is the result stored when the conversion occurs.
* Var2 holds the value zero if the conversion goes wrong