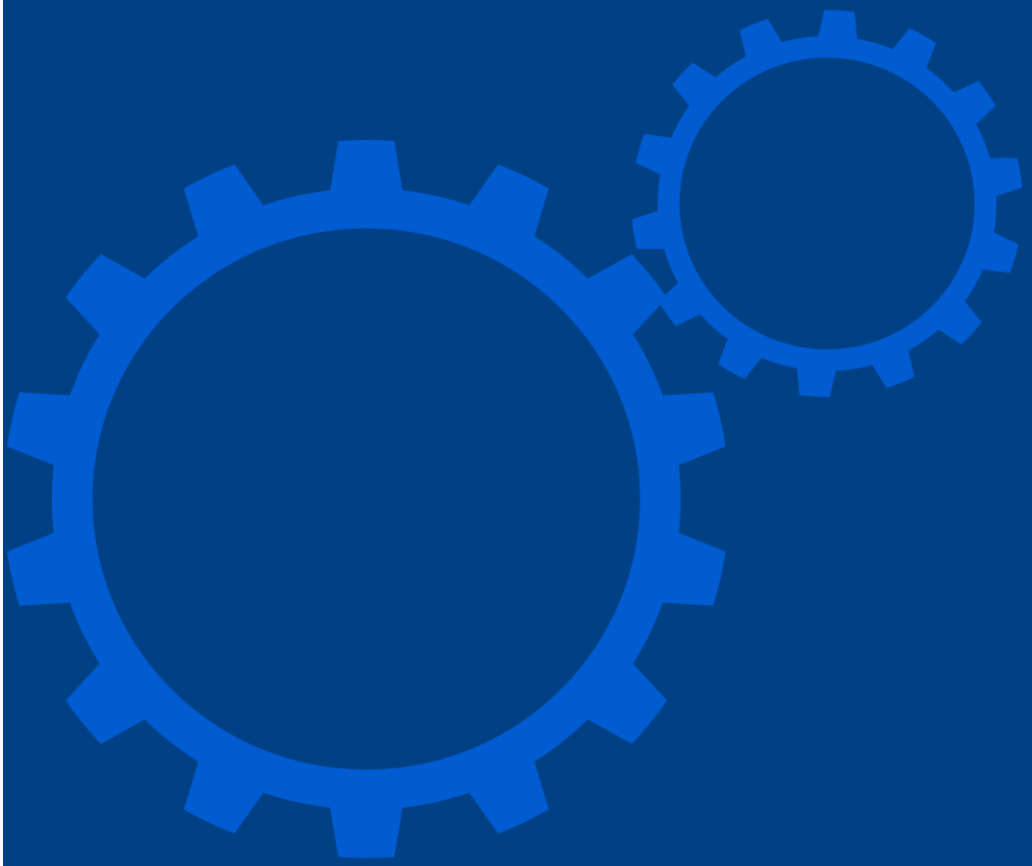

Object Oriented Programing

Principle and Theory

Afzaal Ahmad Zeeshan

OOPs

Principle and Theory

This free book is provided by courtesy of C# Corner and Mindcracker Network and its authors. Feel free to share this book with your friends and co-workers. Please do not reproduce, republish, edit or copy this book.

Afzaal Ahmad Zeeshan

Table of Contents

Preface

This book is focused on basic concepts of object-oriented programming for beginners, intermediates and college students. I ("Afzaal Ahmad Zeeshan") am writing this for the sake of my class fellows who are going to have "Object-oriented programming" in their upcoming semester for MCS. Although this would be considered a short guide for students, it would cover the topics of object-oriented programming.

This book does not talk about one specific programming language like C++, C# or Java. Instead this is a theoretical explanation of Object-oriented programming paradigm used in computer programming to build applications and software. This book would contain a design language to demonstrate the concepts of object-oriented paradigm. These would resemble to those of C++ or C#. Typically, Java is used for object-oriented programming, but I cannot find any reason to "not" use C++ or C# for the teaching purposes when they also support the same concepts and same paradigm in a much better and simpler way.

There are many excellent tutorials and books out there on the Internet that teach Object-oriented programming and the authors have put a great effort in writing those books. The difference that this book makes is that it does not cover any specific language. Many books are titled as, "Object-oriented programming using C++" or some books are titled as, "Object-oriented programming using Java"! This book doesn't confuse you with any single language instead the focus is put forth to only explaining the core concept of Object-oriented paradigm instead of teaching you how to write programs in C++ or Java. If you are interested in learning a programming language take my tip, to learn the programming language itself and not the paradigm it is working in. For example, if you want to learn C++, learn C++ not Object-oriented programming using C++ because object-oriented programming can also be done in Java, and in C# and in Swift and others. This book would target the following topics:

1. What are programming paradigms?
2. How object-oriented programming got influenced?
3. What is object-oriented programming?
4. What are the features of object-oriented programming language?
5. Then, I will talk about the different features that are supported by object-oriented programming languages.
6. When should you use object-oriented programming?
7. A few references, good resources and a few articles of mine that cover the same topics.

My favorite language is C# (no doubt!) so I will be using Visual Studio to generate the class diagrams that I have also mentioned below to be used instead of the source code. Visual Studio is a very powerful tool for teams of any size and project of any complexity level. Class diagram support in Visual Studio is very powerful and I would encourage you to try Visual Studio code designer features.

I would love to hear your feedback on this, you may also share the errors that you find and let me correct them.

Email: justin17862@gmail.com

Computer programming

Ever since computers have been developed and developers have been building efficient systems for helping and guiding the humans through their fast solving skills. Developers have always been trying to find a good way of writing the software! Initially there were only vacuum tubes and a computer to get programmed needed a special amount of those tubes connected and so on. Then we got ourselves updated to transistors, ICs and microprocessors. Long time ago there was only one programming language, Machine language. As name states it was a very tough language to learn and each machine had a separate language set known as, “instruction set”. This was really very tough task for computer programmers to write similar programs on multiple machines and each machine consisted of separate instruction set commands and so on.

A solution that they came up with was to create a language that could be translated to each machine architecture. So they created the Assembly language. Now the thing is that Assembly language resembles machine language is that every machine has its own Assembly language. Then what makes it special? That would be the question arising in your minds right now, well. The thing is that Assembly language consists of small mnemonics and English like short hex-based codes that can allow the machine language code generation. The benefit is that programmers can write code in simple commands instead of writing the long binary statements for op-codes and operands. This enables the developers to write the logic in a simpler way and in many ways Assembly language is simpler and easier to learn and use as compared to machine language. Nonetheless, Assembly language should be translated e.g. assembled, back to machine language before a processor can execute it. For example, the following Assembly language code:

mov al, 61h

Would be translated as:

10110000 01100001

This is a 2-byte command to save the value of 61 hexadecimal to the al register. This demonstrate the purpose of using Assembly language as it is clear as to what each command would do. In binary, you cannot tell what each of this command is meant to do and a simple error in 0 and 1 would cause a great change in the command itself.

Assembly language then introduced the world of computers with another technique of writing the programs. In this technique, programmers were abstracted from the underlying system architecture and were only shown a few words that they can use to write their full featured programs. These programs would then execute on the processor regardless of their architecture or their instruction sets. That is when languages like C, C++ and Java come into action. C is also sometimes referred to as, “Assembly language with a better syntax.” C++ and Java are then language with a different programming paradigm.

The language that Assembly language introduced us are called, “high-level programming languages”. These languages are very close to English statements and they are always compiled before they can be executed on a computer. One thing to consider at this moment is that these programming languages are of different flavors. Different methods of programming are introduced by programmers of different taste. Each added their own interests, their own naming conventions and styles of structure to the programming language they were asked to write the programs in. This can be listed as the following:

1. Programmers created Assembly language; who did, when did and where did is unclear. Assembly language was created to write dynamic machine language code for many architectures. This language has many flavors and many syntax, so it is unclear as to who started it initially.

2. Dennis Ritchie (along with Ken Thompson and the team) developed a much better version of Assembly language, known as C. C was developed because the team had to write the same programs for multiple platforms, so instead they wrote a language that can be compiled differently for different machines keeping the same output! C is a language that is much easier to write, much more portable and is a compiled language. Which means that the language needs to be compiled before it can be used on a machine. This uses the same concept as Assembly language which is translated to machine language. C is compiled to an architecture’s Assembly language and then it is translated to binary op-codes and data for the processor to work on.

3. Bjarne Stroustrup was much more an Object-oriented programmer. He worked in an OOP environment. So when he was asked to come and program in C environment. He didn’t feel at home. So instead he added the flavors of object-oriented programming to C, making it C++. The only difference that C and C++ has is the addition of object-oriented programming features like objects, classes, inheritance etc.

4. While that was not the end, many other programming languages arose, many different paradigms arose, many different techniques for software development arose which changed the programming language usage itself. We will only focus on these points and will talk about the object-oriented programming concepts only.

The purpose of this book is to describe what Object-oriented programming is, why it should be used or when it should be used in your application development processes.

Object-oriented programming

In the previous section you read that computer programming itself is just a bunch of algorithm based binary commands in the form of op-codes and operands, using which your processor is able to provide you with information and resources. Programming languages are usually categorized under the term “paradigm”. These paradigm define how a language looks and feels like, what it does and for what purposes it should be used. The common paradigms are:

1. Non-structured
    - Typically, Assembly language programs or machine language instructions are considered in this paradigm because they are straight-forward instructions and are not structured into blocks.
2. Structured
    - A counterpart or inverse of the non-structured programming paradigm.
3. Functional
4. Object-oriented (the one this book is all about!)
5. Data-driven
6. Event-driven
7. Metaprogramming

These are not all of them! Instead, new paradigms are created when a new developers start to write an application in a different paradigm. That is because paradigms are not required to be followed. You can create your own separate paradigm, create the rules and laws and then implement it in your own application development lifecycle. That is what these old people had been doing ever since they started programming. Object-oriented programming paradigm also came into existence through this kind of philosophy. Many programming languages came into existence following this paradigm, notables are C++, Java and C#.

The paradigm of object-oriented programming focuses entirely on the objects. This paradigm considers “everything” as an “object”. So the definition of the object may be:

> *In object-oriented programming, an object is a variable, class instance, function or a data structure that can hold some information in the form of a field and can perform anything as a function. The objects are also able to depict inheritance from each other.*

This book will talk about this definition and would try to explain the common sections and philosophical aspects of this paradigm without going in depth about any programming language at all. I will not talk about any specific programming language, instead I will be sharing the theoretical explanation of object-oriented programming. To explain the development procedures, I will use another design language. You can find the basic explanation of the language at the end of this book.

What are objects?

Although we have been talking about this paradigm and a definition was still provided. But the questions still is compelling and somewhat unclear as to what an object actually is. That was just the definition, in real programs you will not see anything “object”. Instead, everything will be an object. But technically, the objects are usually classes. All of the object-oriented programming languages have a special type declaration, class. Classes are available in almost every programming language that supports object-oriented paradigm for programming. The concept is just to make a separate type for each of the object. Then these object types are called, classes. Nothing special or rocket science going on here.

A class holds the information about an object’s data structure and procedures. The class is just the abstract structure of the object upon which we want to do some processing. The class can then introduce the rest of the philosophical aspects of object-oriented programming like inheritance and polymorphism, etc. Defining a class is similar in most of the programming language. Just for the sake of an example, the following is the example of creating the classes in mostly used object-oriented programming languages:

```
// C++ code is like this for a class

class Example {

private:

  data field;


public:

  some function() { }

};


// C# and Java have similar structure for a base class.

class Example {

  private data field;

  public some function () { }

}
```

But, I will be using the following depiction of a class for this book.

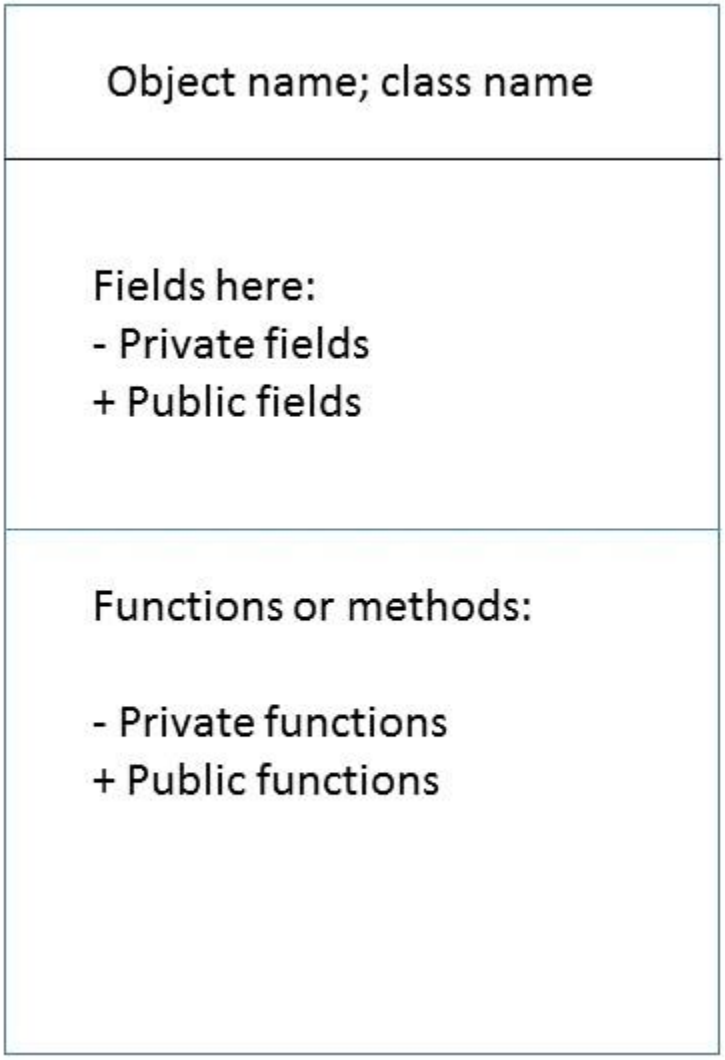

***Figure**: Class diagram of an imaginary object.*

Although this image just talks about the fields and function. It does not talk about inheritance at all. In real projects, inheritance plays a very vital role! Inheritance opens the gateways toward framework building. Most common frameworks like .NET or Java's virtual machine are all based on inheritance concepts where the developers provide a base class (typically called System) and the derived objects are used to define the business logic of the application. So from demonstration it is clear that an object (or a class) has:

1. A name that identifies that object in the system. Following the rules, it should be different for each of the object.

2. Data structure.

    - Typically the data structures are confused with something “rocket science”. In real data structure is just the fields or properties that an object holds. For example, the data structure for an object “Person” would be something like, “{Name, Age, Occupation}” etc.

3. Functions that it can perform.

    - Not to confuse yourself with methods or member functions. They are all same thing and their names are different, only!

The average definition of an object requires these to be present but they are not obligatory. You can create a simple class that holds only data, and that is what is usually called “data structure” or a “model” in many other frameworks like Model-view-controller pattern for programming. They are much broader concepts in themselves and need a separate book to target themselves so I would consider them off-topic as per the scope of this book.

Object-oriented programming “inherits” the use of variables (fields) and procedures (functions) from non-structured and structured programming. The difference is that in object-oriented programming these variables and procedures are tied to the object itself and are not exposed to the general public, unless required.

What is an object instance?

Before I move any further I may provide a definition for the term object-instance. This all backs up to the native English terminology, where instance means occurrence. “Object instance” in our program would mean the “object occurrence”. Computer programming involves CPU, RAM and hard disk, so object occurrence would mean its presence in RAM. Every programming language supports object creation through constructors. Constructors are used to create object instances. It is also common to call a class instance an object. In most languages, an object is an instance of a class, because their types are also a class and thus everything happens to be an object in their programs.

The concept arises in the terms of “instance variables” and “static variables”. At a higher level an instance is just an occurrence (or use) of an object. At a lower level it is different. They both have their own pros and cons and they differ in many things.

Instance variables	Static variables
Instance variables require to have an instance of the object before they can be accessed in the program.	Static variables do not need an instance of the object to be access.
Their lifetime is same as the lifetime of that object instance.	Their lifetime starts when the program starts. They remain in the RAM even when the object is removed (but the program exists).
Instance variables (and functions) can be called only from instance scope.	Static variables (and functions) can be called from static and instance scopes.

Table: *Difference of an instance variable and static variable.*

These were roughly made assumptions and differences between these two, there are more and (yes,) they are off-topic as per this book.

Example of a class:

I have already talked about a type of a class, “Person”. The person can hold information about his name, age (or any other data variables that we may want to specify) and also holds a constructor and a few functions that it can execute to perform a special type of task. So our object had the following structure:

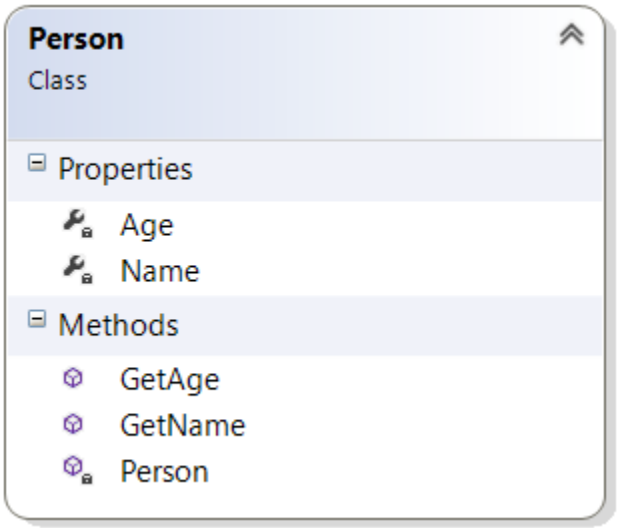

Figure*: Class diagram of a person object.*

From image it is clear that this class holds information about:

1. Age of the person
2. Name of the person

Also that it can perform the following functions:

1. GetAge
    a. As name states, it should return the age of that person.
2. GetName
    a. Same as above.
3. Person
    a. This is a constructor for the class. It is used to allocate the space for this object in RAM. You can define your own logic for this or leave it to the compiler to write the implications.

This is object #1 in the program. An application uses typical more than 5 objects working together inside a project and providing you with resources and additional information. But that is not all, usually objects are created to share their resources. Sharing the resources can come through dynamic message passing, inheritance or any other technique that can be used at that time. Once your project is working, you will not feel any difference between objects because on a higher level of execution they are all synchronized to work together in a singularity to provide a functionality.

Constructor and destructor of a class

There is another very important term and concept that needs to be discussed before we close the topic of class and objects themselves. The objects, when are being initialized (or *instantiated*), need memory allocation in RAM. Every compiler, when it generates the code, generates in a such a way that the RAM is allocated in the best possible way! Each variable is allocated in the memory and the address is linked to its name. Although compiler can do that all on itself, but sometimes programmer may want to do that himself. The purpose of doing it all by himself is that he can make sure that the program is ready from his perspective. Compiler may be doing its job but programmer may not be expecting such results. In such cases, there are two special functions provided. A constructor and a destructor. As their names tell, they are used to specify the logic of construction and destruction of an object in the program. They execute when a variable is either created in RAM or when it is being deleted from the RAM.

Constructor	Destructor
A constructor is called when a new object is being created.	A destructor is called when an object is being deleted from the memory.
Constructors should hold the logic for object creation. Like memory-allocation.	Destructors should be used to hold the clean-up logic. Like memory-freeing process.
Constructor functions have similar name as their class names.	Destructors also have similar names, but typically include a tilde (~) sign as a prefix.
It is called by the underlying operating system or the framework.	It is also called by the underlying framework or operating system.

You can call it when you want to create a new object.	You can call it when you want to delete an object.

Table: *Differences of a constructor and a destructor.*

Data abstraction; Encapsulation in Object-oriented programming

This term has a great use in data logic layer in many frameworks. The concept is as simple as, hiding your data from external world! The external world in object-oriented programming means external objects and classes. So you would need to hide your data fields from external world, the purpose is to make sure your inputs and outputs are all validated and as expected. A mechanism to check the input is applied on both sides, input and output. This makes it easier to make the data consistent and to protect the data sources from any external exploits like injections in SQL. Attacks can alter the data set's values and even enter an invalidated data to your data source which may destroy the schematics of your databases and any external sources attached.

Data abstraction in classes is a very fundamental concept as you cannot rely on users to be optimistic toward your application. Of every 10 user, 2 are potential programmers! I am not saying every hacker (or programmer, let's say) is a cracker, but you just simply cannot leave that to a chance, would you? Some just like to exploit the logic to test their skills or to show you how to write good code. But once you are writing commercial software you cannot risk your applications and their source code or security. It is your job to make sure things are going as they are expected to go and the paths that you define are all safe. Usually, tests are conducted before a project is published to the public. Data abstraction or (as the official term in object-oriented programming names it,) "Encapsulation" is the process of hiding your data fields from external world and allowing them to use a specially designed interface to communicate with your objects and to access their data or to modify their data. Most of the programming languages have them as functions called, "getters and setters". Getters and setters as their name states are functions that can provide you with the data and the functions that allow you to modify the data.

Getter	Setter
Allow external world objects to get the data from variable.	Allow external world objects to modify the data of the variable.
You can modify, alter or change the representation of the data (like in the case of date and time representation) before it is provided to the caller.	You can verify and validate the data before it is stored in the data source. This can also allow you to change the format of the data.
You can check for user permissions before providing the data back to the caller. This can add extra layer of security to your application.	You can check for authorization of current session before you enter the data to the data source.

Table: *Differences of getters and setters.*

So typically, encapsulation allows us to secure our data sources and data sets from external world. This allows us to provide the data only to the sources who we trust or are authorized to access the sensitive data. If request maker is not authorized to access the data we can simply ignore the request or ask them to get authorized for such actions.

How do we implement encapsulation?

Technically, we just hide the items in our data structures from the external world. But first, let us understand a simple example of a library system where we can allow users to get a book, to return a book and so on. We also use a parameter to know which book is being returned and who is returning it in case we have multiple items for each book on our shelf.

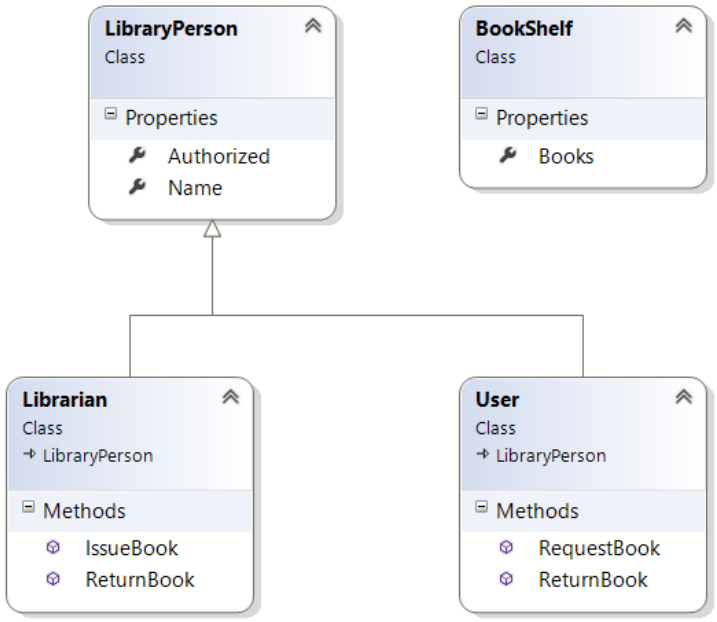

Figure: *Library system with one librarian a person and a shelf of books.*

Usually these are the entities created in a system for library. Now if you pay close attention our data source is indeed the "BookShelf". In that object, you can see that any external object is allowed to access the books; which is a list of books. An ordinary user would have to go through the "formal" ways of getting and returning a book but if you run into someone who likes to get the books but doesn't typically return a book. In such cases, the data source is not validated and protected from such cases. In most cases, user may just simple imitate to have returned the book; which he hasn't.

In such cases, encapsulation would help us to make our data source better and secure. So what we do is that we only make our data set private. Private, in programming languages means that the variable, function or an entire object is not accessible to the external world. But our own class is capable of using it in any way. So our new structure looks like the following:

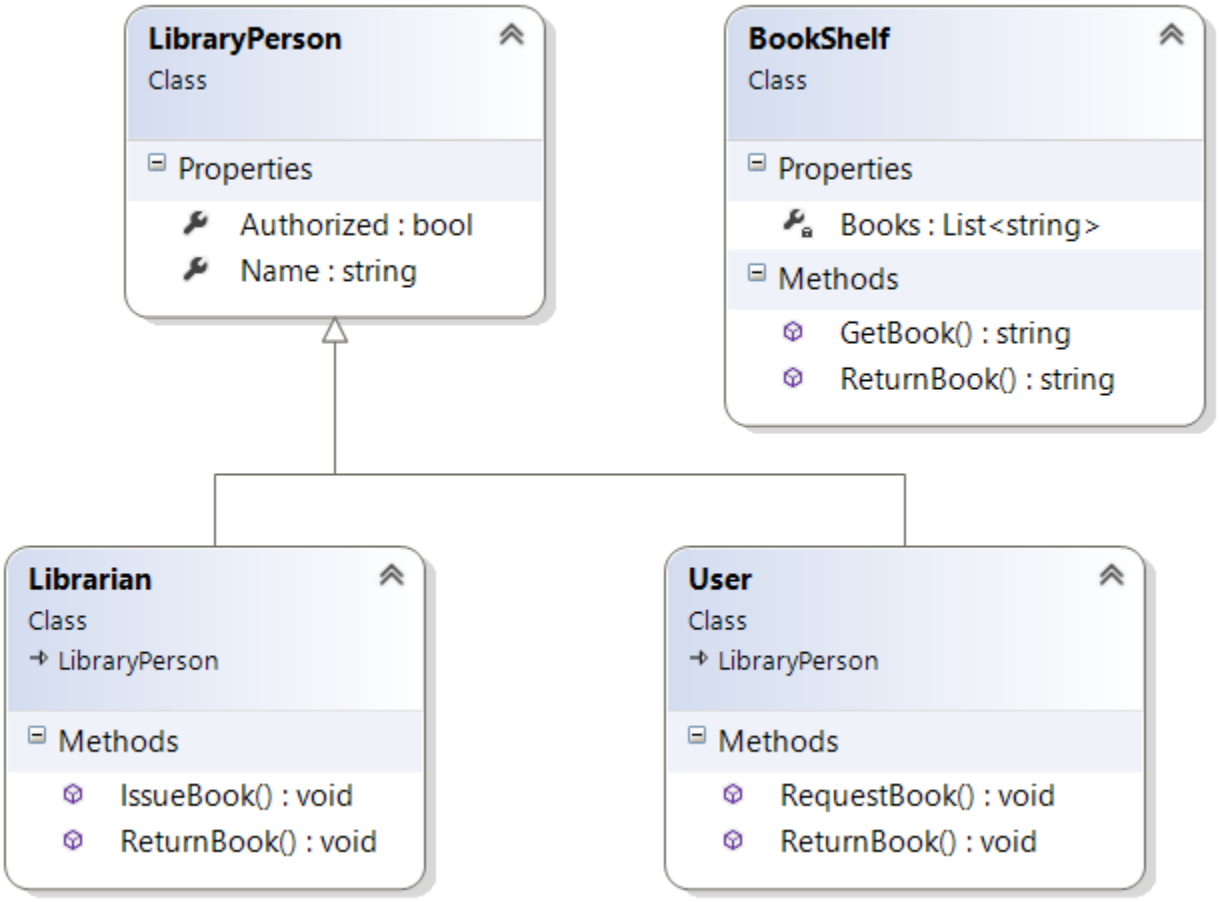

Figure: *BookShelf being modified.*

The difference is that now the shelf is having encapsulation. The books shelf is now private, which means that it is only accessible in the shelf itself. Any external object cannot use the shelf. However we have defined an interface for the external world that they can use and either get a book or return a book to the shelf.

This way, we can protect our data. There are many access levels, typically the following are used:

1. Public
    a. Accessible to any object, external or internal.
2. Private
    a. Accessible to the current class only.
3. Protected
    a. Accessible to the current class and inherited classes only. It is useful when you are trying to make full use of inheritance.

There are many more access levels but typically they are combination of these. Each programming language and framework may introduce their own and they should be confused with. Encapsulation has many other uses and examples too, but they would make this complex.

Exercise:

1. What is the different between data abstraction and encapsulation?

2. Suppose that you are not using encapsulation, is your application performing to its fullest?

3. Take an example from your regular routine (let it be your school, office or bus station), digitalize it and while designing their codes, make assumptions of the system with and without encapsulation. Also demonstrate which one is better.

Like father like son; Inheritance

This one is the most popular and most widely used concept of object-oriented programming. There is no difficult terminology in this one at least, and even if there was, the prefix may have removed it by now. Object-oriented programming provides us with a feature in which we can relate objects as parent or child objects. Most programming languages use the term, “super” and “sub”, some prefer to use “base” and “derived” for these relations. Not to be confused with any of that sugary coating, they are all same and they use the same concept of inheritance to create a relation among different objects. This concept goes something like this:

> *Inheritance in object-oriented programming allows to create and maintain a hierarchal type-to-type relationship among different classes.*

The term hierarchal is used because you would have a parent-to-child and then child (as parent) to other children relationship and it would go something as a tree list, showing a relationship between multiple classes where some are parent and many are children.

The concept can be summarized in a very simple and ordinary regular way! Your father, take him as an object. He has some properties like his name and graduation, he also performs some actions, like he eats, he sleeps and much more. You, the derived ones, get this from him! You eat, you sleep and you do have a name and (well, assumingly!) you do study. You have derived these properties and functions from your father and you are going to pass them on to your own children later. That is how inheritance in object-oriented programming works.

In real-world example, you can see that a professor’s son would be good at studies and a son of a shoe-mender would likely be a shoe-mender. I am seriously hoping to be proven wrong as in the case of a shoe-mender!

However, let’s talk about it in a diagram way. The following diagram for example, take it into consideration and have a glance at common factors.

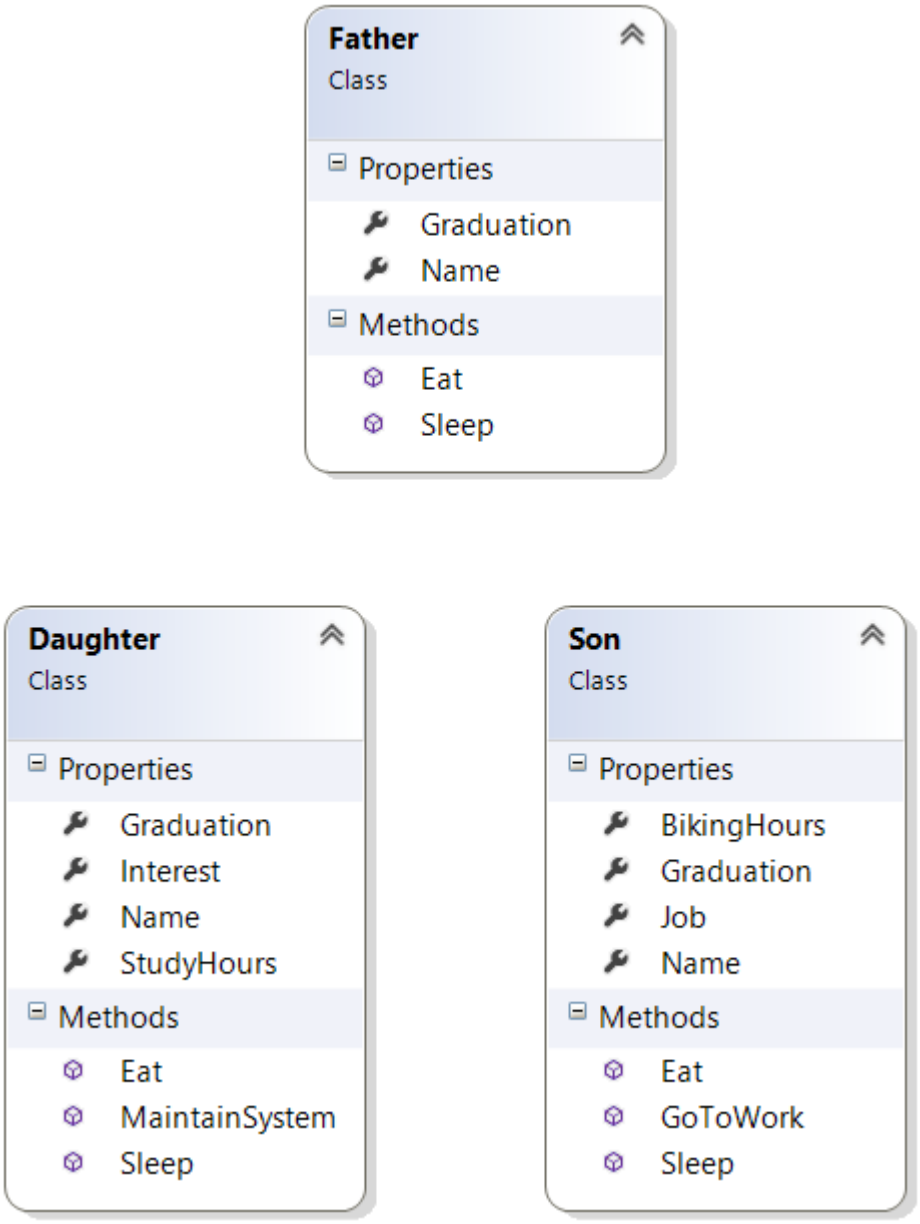

***Figure**: Three objects with no relation.*

In this diagram if you see, program is able to consume these three objects without any problem! But as most of you already know there is a problem with this type of programming. We are breaking the “DRY” rule of ours. DRY rule is an abbreviation for, “Don’t repeat yourself”. Since our objects perform many things similarly and are very closely related to each other. We can create a relationship between them and can sum up their code in such a way that their data logic and properties or functions are not repeated everywhere. So we can re-write the above code in such a way that daughter and son now **inherit** from father. By inheriting they can allow re-using the same code and we can start to follow the DRY rule.

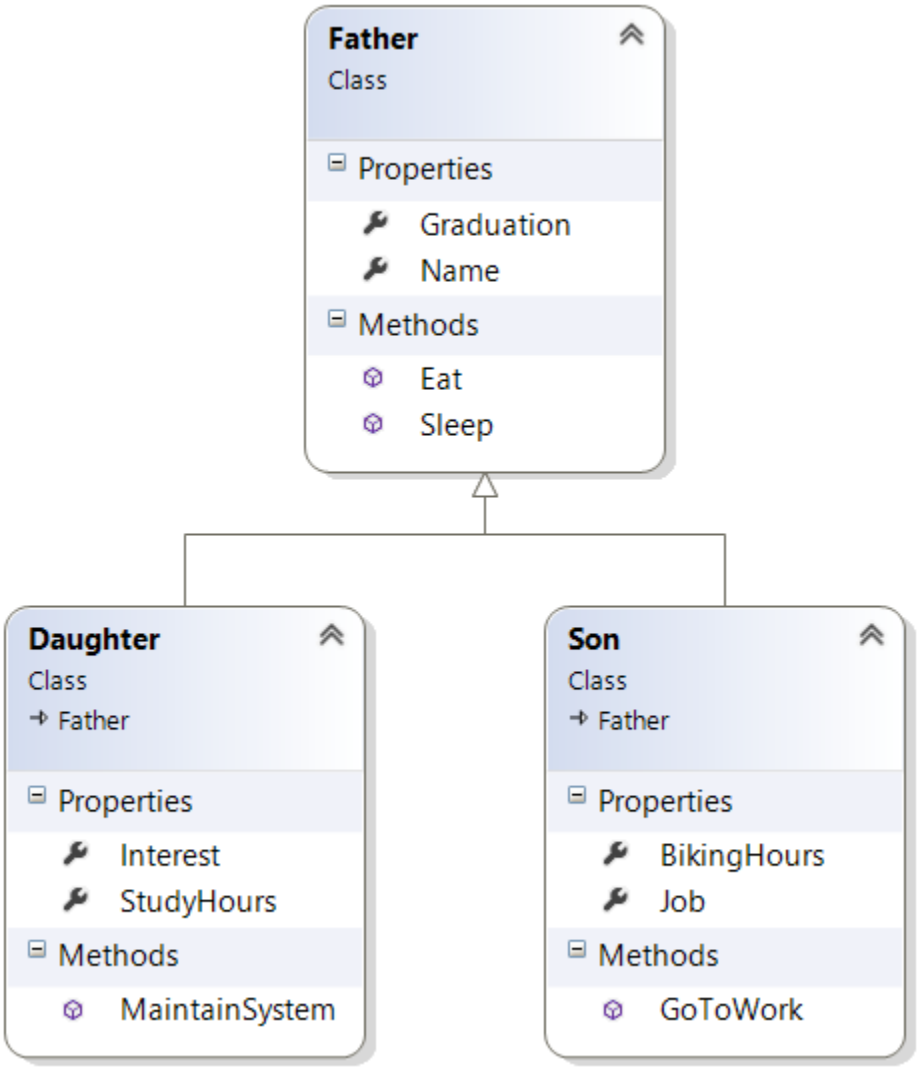

***Figure**: Father to daughter and son relation example.*

Although these diagrams were roughly made because I have not yet reached the status of a father so I have to assume that is what fathers usually do, so forgive me. Anyways, the diagram depicts that daughter and son are inheriting the properties and functions from “father”. Then they have their own purposes. Son does a job and daughter is currently studying. We can see that in both of these cases, our code now looks much shorter, concise and to the point. Also we only need to edit and modify our functions at a single location and not here and there.

Inheritance also plays a vital role in many frameworks like .NET framework. In .NET framework every object inherits from System.Object class. The purpose is to provide the basic framework functionality to that one single object! Each class can then have its own properties and functions, but the framework work execute them as if they were object class types.

Using interfaces and abstract classes

In the above section we have only talked about inheritance itself. Most programming languages and frameworks introduced the usage of interfaces or abstract classes. This is not a common concept because some languages have interfaces, some use abstract classes for this purpose. The common use of them is simply to define a basic structure of the object. They are not used in the program itself, because they cannot be instantiated.

Abstract classes

First of all, let me talk about the abstract classes and then I will move on to the interface sections. An abstract class is usually a concept of high-level language. In high-level languages you can define an abstract class that contains the data structures and the declaration of the methods. The class does not hold any definition. The purpose is there to allow the classes to have their own implementation of the functions and to have their own values in the data structure; *that can also be done in ordinary classes*!

One thing to note here is that abstract classes cannot be used in the programs, like an ordinary class can be. They cannot be instantiated. Instead their use is just as a super-class. Ordinary classes can then inherit from them and add their own implementation to it. A common example is given for the shape classes and their implementation by circle, triangle and rectangle. Just for the sake of this example, see the following depiction:

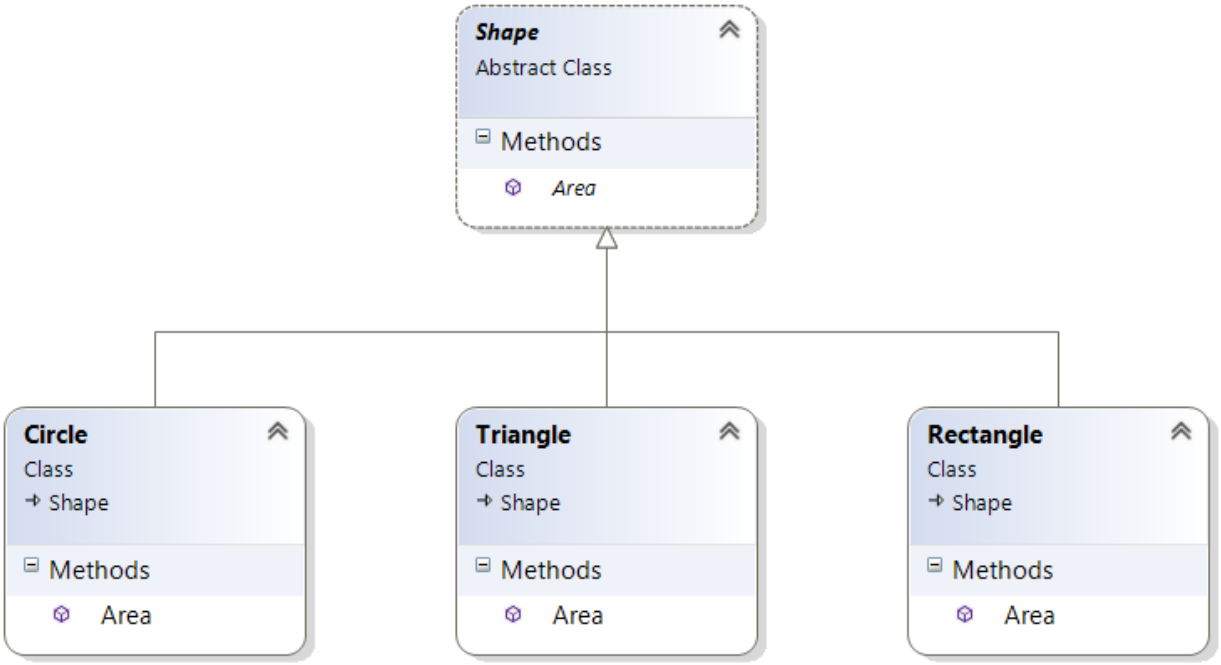

***Figure**: Abstract class being used by three classes for a base function using their own implementation.*

Now in this example if you can see there is one base class with the abstract function, without any implementation. The purpose is because circle, triangle and rectangle, all have a different

function and formula. That is why each of the object would have its own implementation of the function and so on. You are not going to use the Shape class in your code, of course, because you have a special shape type that works just like you want it to work, right?

Interfaces

Interfaces are very much similar to abstract classes. Their use is just to hold the properties and the methods. They are not directly inherited and the objects do not claim the inherited properties, instead they are allowed to add their own implementations to the functions. The following image may demonstrate the purpose of the interfaces in a program.

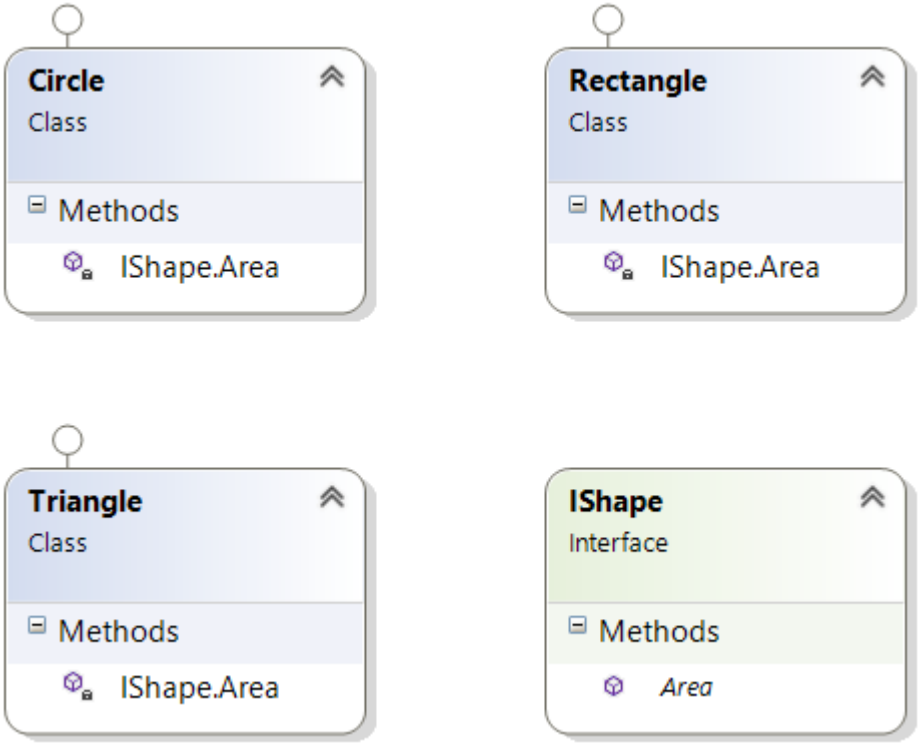

***Figure**: An interface being consumed by three different types.*

Although they are all not connected, yet they can use the method of the interface and apply their own implementation to the function. Abstract functions do the same thing, but in a different way. They allow the objects to directly inherit the functions and properties like if they were their own!

Interface	Abstract class
An interface is used as a base structure.	Being a class, it is a valid and better candidate for base-classes.
Interface can hold properties and functions with no implementation.	Abstract class can hold properties with values. Function should be marked abstract so that inheriting class can have its own implementation for that function.
Interfaces can be used a template for multiple purposes.	Abstract class is more like a structure with typical functions that may or may not be inherited.
Classes that inherit an interface must implement the functions.	Classes that inherit the abstract class must be abstract classes or should implement the functions.
All functions of interface are just declarations.	Abstract class functions can have definition body too, if they are not declared abstract.
Interface implementations depend on the derived class instance.	Abstract class is inherited and thus each object can override the default behavior for itself.
Interfaces have more security because they have no implementation at all!	Abstract classes can still have some implementations and definitions. But their implementations can be seen via methods like reflection.
One or more interfaces can be inherited.	In most programming languages, you cannot inherit multiple classes. In those cases, interfaces are used.

***Table**: Interfaces and abstract classes.*

In my own personal opinion, abstract classes are very much common to use instead of interfaces. Abstract classes are classes after all and they can easily get themselves fit into the tree of type hierarchy. Interfaces, not being a class, cannot fit into the type hierarchy as well as an abstract class can fit in the hierarchy.

Polymorphism in Object-oriented programming

After the concept of inheritance another big concept in object-oriented programming is polymorphism. Polymorphism is a combination of two words, meaning “occurrence in many forms”. But what does occur in many forms, what this concept covers is the main topic of this section.

Polymorphism as we have already discussed in the paragraph above means many forms of something. Technically, in object-oriented programming, “something” is meant to be a function. In object-oriented programming, many forms of functions in objects is called polymorphism. You may be wondering where functions come from. Well, the functions that come from inheritance procedures are considered to be the candidates. The functions of father, son and daughter don’t need to be working in a similar manner or returning the same value. You may have to override some functions and change their default behavior.

One thing to note here is that:

1. Polymorphism is overriding of the default behavior to provide a different implementation
2. Polymorphism does not mean implementing the interfaces.
3. Implementing the abstract classes is also not polymorphism, because abstract class does not have a default behavior for abstract functions.

So that leaves polymorphism to the functions who do have a definition, but you are left to override their behavior. Some programming languages like Java or C# use the keyword “virtual” to support this functionality.

In my opinion, overriding and overloading are two separate parts of polymorphism. They are different in many ways and should be considered and thought of differently. Below I will talk about a few things about overloading and overriding.

Overloading

Overloading means to overload something with extra jobs to be performed. It is not the lines of code that you include, but the different list of parameters that you can include and make different functions. It is a rule of programming, that you can use one identifier for one object or function! You cannot use the same identifier twice. However, using overloading you are allowed to have multiple functions because you are passing different list of parameters to it. That may sound a bit difficult to understand, but have a look at the following diagram.

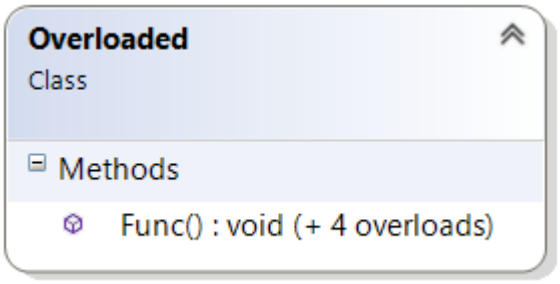

***Figure**: A function, with 4 overloaded functions (having different parameters)*

The function that is depicted here has 4 overloads. It has one default function and 4 other overloads. They have a different signature of function. A signature of a function is defined to have the following:

1. An identifier.
2. A parameter list.

The return type is usually not considered in overloading of a function. The good use of overloading is that you do not have to create a function with lengthy names. For example, if you were to create a function named, "func()" and then you had to create a function that accepts an integer value then you would have to create a function named, "funcwithintparam(int)". This would cause your code to be less ambiguous because you can create another function with a parameter named, "func(int)". Pretty simple, right? This is the same mechanism that I used in the code which is depicted above. The func can accept many parameters and can work on any number of parameters that you pass. This way you can also manage how your application processes the data of the users and their requests. You can define a different business logic layer in the functions for different parameters and it would also help you minimize the chances of errors in the code if there are occurrences of the code that may be expecting some data from the user and they have not provided any.

Operator overloading

This is another use of overloading in programming. In programming languages where you can use operators, such as + - * / etc. You can overload them to perform a different task. They work in a similar manner, but they return a user-defined value. The value depends on the function you write in it. A very typical and common example is given using the complex numbers type. Where you add the values as:

$$z_a + z_b = \big(Re(z_a) + Re(z_b)\big) + i(Im(z_a) + Im(z_b))$$

Typically, you can overload the "+" operator in your program and use this logic to evaluate the value of the complex numbers.

Your class can have these overloads as extra functions.

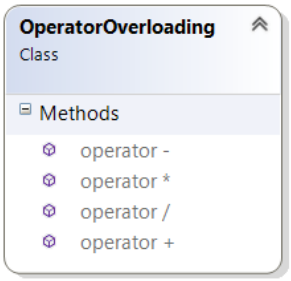

***Figure**: Four overloaded functions for operators.*

Overriding

Now this is something else and you may want to focus up! In my experience I have found that many developers find this very much confusing and they don't get the concept of them. They also find themselves confusing around with these two, "overloading and overriding". But they are no rocket science at all and I don't understand the fuss to not capture the concept. Anyways, the function overriding is the concept of polymorphism; *the one discussed above*. It is used to override the default behavior of the functions and to provide you own implementation of the functions.

It is used regularly by programmers to override the default behavior of the classes and then introduce their own implementations of the functions. As the name states, "overriding", it is used when you are "overriding" the default behavior of the functions to introduce your own concept. We have already talked about the shape implementation. We defined a function in shape as "area" and then we used "overriding" to "override" the default behavior and implement the mathematical formulae to find the areas for each one of the derived shape. Each one had its own function that is why we needed to override the default function; whatsoever it may be.

Overriding	Overloading
It is used to override the underlying default behavior of the functions.	It is used to create multiple functions with different parameters lists.
It is usually provided by many languages.	Some languages do not support operator overloading. But almost all languages support overloading of functions.
It is a very common concept in polymorphism.	Overloading is a common feature of programming languages.
Overriding is usually prohibited because he base functions are still present. New functions just create an extra function.	Overloading is a method of packaging the functions under same identifier.

***Table**: Overloading and overriding.*

You may learn the purpose of overloading and overriding as you continue to learn object-oriented programming using a programming language. They are not as hard concepts as they might look from this perspective of a novice developer.

Bonus

Object-oriented programming paradigm is a very concise and very short paradigm architecture. It has only few concepts and focuses primarily on them only:

1. Objects and classes.
    - Objects have functions and data structure to hold the values.
    - Classes are the templates.
    - Objects are instances of classes.
2. Encapsulation
    - Data sources must be hidden and private.
    - Getters and setters must be implemented for validation and verification of data.
3. Inheritance.
    - The relationship between different classes in the form of a tree hierarchy.
4. Polymorphism
    - Function overloading
    - Operator overloading
    - Overriding

How to create the classes?

A very technical and expert vision is needed to find and create a class. What? Of course not! If you are familiar with the definition of the object, you can create a class! An object is something that… You know.

While reading through the statements of your project. You can define the number of classes by reading the objects at stake. You can create a class for anything that has a data source and performs a function. Remember, you can also create a class that do not perform an action, they are called "**data structures**".

You can also introduce a bit of encapsulation to secure your data sources from invalid data entries. You can apply the validation rules in the getters and setters. Encapsulation must always be implemented if you have a property in your class! Public data fields expose your data source to the external world and may cause invalidity and inconsistency in your data sources.

Inheritance and polymorphism is a technical thing and really does require you to have experience before you can apply it. Inheritance should be applied to object after having a look at their:

1. "is-a"
2. "has-a"
3. "does-a"

This way, you can create classes that are base classes to the derived ones. You can then design the base classes to be abstract or interfaces, depending on your own need! These relations can be managed to make your data and projects even better after managing your inheritance threads.

References:

I have written other resources for developers too, you may enjoy reading them while learning object-oriented programming. By this date (11th January 2016), I have written the following blogs and articles that cover the most of the object-oriented programming and a few other concepts that I feel you need to know. I am attaching them as below, you can use them for your purposes.

1. Still don't get the difference of "overloading and overriding"?

You can also download the source from GitHub: https://github.com/afzaal-ahmad-zeeshan/Theoretical-OOP-Book