



C o m m u n i t y   E x p e r i e n c e   D i s t i l l e d

# Learning Object-Oriented Programming

Explore and crack the OOP code in Python, JavaScript, and C#

Gastón C. Hillar

**[PACKT]**  
PUBLISHING

# Learning Object-Oriented Programming

Explore and crack the OOP code in Python, JavaScript, and C#

**Gastón C. Hillar**



BIRMINGHAM - MUMBAI

# Learning Object-Oriented Programming

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: July 2015

Production reference: 1100715

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-78528-963-7

[www.packtpub.com](http://www.packtpub.com)

# Credits

**Author**

Gastón C. Hillar

**Project Coordinator**

Nikhil Nair

**Reviewers**

Róman Joost

Hugo Solis

**Proofreader**

Safis Editing

**Commissioning Editor**

Sarah Croufton

**Indexer**

Monica Ajmera Mehta

**Acquisition Editor**

Nadeem Bagban

**Graphics**

Disha Haria

**Content Development Editor**

Divij Kotian

**Production Coordinator**

Arvindkumar Gupta

**Technical Editor**

Parag Topre

**Cover Work**

Arvindkumar Gupta

**Copy Editor**

Relin Hedly

# About the Author

**Gastón C. Hillar** is an Italian and has been working with computers since he was 8 years old. In the early 80s, he began programming with the legendary Texas TI-99/4A and Commodore 64 home computers. Gaston has a bachelor's degree in computer science and graduated with honors. He also holds an MBA, in which he graduated with an outstanding thesis. At present, Gaston is an independent IT consultant and a freelance author who is always looking for new adventures around the world.

He has been a senior contributing editor at Dr. Dobb's and has written more than a hundred articles on software development topics. Gaston was also a former Microsoft MVP in technical computing. He has received the prestigious Intel® Black Belt Software Developer award seven times.

He is a guest blogger at Intel® Software Network (<http://software.intel.com>). You can reach him at [gastonhillar@hotmail.com](mailto:gastonhillar@hotmail.com) and follow him on Twitter at <http://twitter.com/gastonhillar>. Gastón's blog is <http://csharpmulticore.blogspot.com>.

He lives with his wife, Vanesa, and his two sons, Kevin and Brandon.

# Acknowledgments

At the time of writing this book, I was fortunate to work with an excellent team at Packt Publishing. Their contributions vastly improved the presentation of this book. James Jones gave me a brilliant idea that led me to jump into the exciting project of teaching object-oriented programming in three popular, yet heterogeneous, programming languages. Divij Kotian helped me realize my vision for this book and provided many sensible suggestions regarding the text, format, and flow of the book, which is quite noteworthy. I would like to thank my technical reviewers and proofreaders for their thorough reviews and insightful comments. I was able to incorporate some of the knowledge and wisdom that they have gained in their many years in the software development industry. This book was possible because of their valuable feedback.

The entire process of writing a book requires a huge number of lonely hours. I couldn't have written an entire book without dedicating some time to play soccer with my sons, Kevin and Brandon, and my nephew, Nicolas. Of course, I never won a match.

# About the Reviewers

**Róman Joost** first learned about open source software in 1997. He has contributed to multiple open source projects in his professional career. Roman is currently working at Red Hat in Brisbane, Australia. In his leisure time, he enjoys photography, spending time with his family, and digital painting with GIMP.

**Hugo Solis** is an assistant professor in the physics department at the University of Costa Rica. His current research interests include computational cosmology, complexity, and the influence of hydrogen on material properties. Hugo has vast experience in languages, including C, C++, and Python for scientific programming and visualization. He is a member of the Free Software Foundation and has contributed code to some free software projects. Hugo has contributed to *Mastering Object-oriented Python* and *Kivy: Interactive Applications in Python* as a technical reviewer and is the author of *Kivy Cookbook*, Packt Publishing. He is currently in charge of the IFT, a Costa Rican scientific nonprofit organization for the multidisciplinary practice of physics (<http://iftucr.org>).

---

I'd like to thank my beloved mother, Katty Sanchez, for her support and vanguard thoughts.

---

# www.PacktPub.com

## Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit [www.PacktPub.com](http://www.PacktPub.com).

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

## Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

## Free access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.





*To my sons, Kevin and Brandon, and my wife, Vanesa*



# Table of Contents

<b>Preface</b>	<b>v</b>
<b>Chapter 1: Objects Everywhere</b>	<b>1</b>
Recognizing objects from nouns	1
Generating blueprints for objects	4
Recognizing attributes/fields	5
Recognizing actions from verbs – methods	6
Organizing the blueprints – classes	9
Object-oriented approaches in Python, JavaScript, and C#	11
Summary	12
<b>Chapter 2: Classes and Instances</b>	<b>13</b>
Understanding classes and instances	13
Understanding constructors and destructors	14
Declaring classes in Python	16
Customizing constructors in Python	17
Customizing destructors in Python	19
Creating instances of classes in Python	21
Declaring classes in C#	23
Customizing constructors in C#	23
Customizing destructors in C#	26
Creating instances of classes in C#	27
Understanding that functions are objects in JavaScript	29
Working with constructor functions in JavaScript	30
Creating instances in JavaScript	34
Summary	35
<b>Chapter 3: Encapsulation of Data</b>	<b>37</b>
Understanding the different members of a class	37
Protecting and hiding data	39
Working with properties	40

<b>Understanding the difference between mutability and immutability</b>	<b>41</b>
<b>Encapsulating data in Python</b>	<b>43</b>
Adding attributes to a class	43
Hiding data in Python using prefixes	45
Using property getters and setters in Python	46
Using methods to add behaviors to classes in Python	50
<b>Encapsulating data in C#</b>	<b>53</b>
Adding fields to a class	53
Using access modifiers	55
Using property getters and setters in C#	57
Working with auto-implemented properties	63
Using methods to add behaviors to classes in C#	64
<b>Encapsulating data in JavaScript</b>	<b>66</b>
Adding properties to a constructor function	67
Hiding data in JavaScript with local variables	68
Using property getters and setters in JavaScript	69
Using methods to add behaviors to constructor functions	71
<b>Summary</b>	<b>74</b>
<b>Chapter 4: Inheritance and Specialization</b>	<b>75</b>
<b>Using classes to abstract behavior</b>	<b>75</b>
<b>Understanding inheritance</b>	<b>78</b>
<b>Understanding method overloading and overriding</b>	<b>81</b>
<b>Understanding operator overloading</b>	<b>81</b>
<b>Taking advantage of polymorphism</b>	<b>82</b>
<b>Working with simple inheritance in Python</b>	<b>82</b>
Creating classes that specialize behavior in Python	82
Using simple inheritance in Python	83
Overriding methods in Python	85
Overloading operators in Python	88
Understanding polymorphism in Python	89
<b>Working with simple inheritance in C#</b>	<b>91</b>
Creating classes that specialize behavior in C#	91
Using simple inheritance in C#	93
Overloading and overriding methods in C#	95
Overloading operators in C#	101
Understanding polymorphism in C#	102
<b>Working with the prototype-based inheritance in JavaScript</b>	<b>104</b>
Creating objects that specialize behavior in JavaScript	104
Using the prototype-based inheritance in JavaScript	105
Overriding methods in JavaScript	106

---

Overloading operators in JavaScript	109
Understanding polymorphism in JavaScript	110
<b>Summary</b>	<b>111</b>
<b>Chapter 5: Interfaces, Multiple Inheritance, and Composition</b>	<b>113</b>
<b>Understanding the requirement to work with multiple base classes</b>	<b>113</b>
<b>Working with multiple inheritance in Python</b>	<b>115</b>
Declaring base classes for multiple inheritance	115
Declaring classes that override methods	117
Declaring a class with multiple base classes	119
Working with instances of classes that use multiple inheritance	124
Working with abstract base classes	127
<b>Interfaces and multiple inheritance in C#</b>	<b>128</b>
Declaring interfaces	129
Declaring classes that implement interfaces	131
Working with multiple inheritance	134
Working with methods that receive interfaces as arguments	140
<b>Working with composition in JavaScript</b>	<b>143</b>
Declaring base constructor functions for composition	143
Declaring constructor functions that use composition	145
Working with an object composed of many objects	147
Working with instances composed of many objects	154
<b>Summary</b>	<b>158</b>
<b>Chapter 6: Duck Typing and Generics</b>	<b>159</b>
<b>Understanding parametric polymorphism and duck typing</b>	<b>159</b>
<b>Working with duck typing in Python</b>	<b>160</b>
Declaring a base class that defines the generic behavior	161
Declaring subclasses for duck typing	163
Declaring a class that works with duck typing	163
Using a generic class for multiple types	165
Working with duck typing in mind	167
<b>Working with generics in C#</b>	<b>170</b>
Declaring an interface to be used as a constraint	170
Declaring an abstract base class that implements two interfaces	171
Declaring subclasses of an abstract base class	174
Declaring a class that works with a constrained generic type	175
Using a generic class for multiple types	178
Declaring a class that works with two constrained generic types	181
Using a generic class with two generic type parameters	184
<b>Working with duck typing in JavaScript</b>	<b>185</b>
Declaring a constructor function that defines the generic behavior	185

---

Working with the prototype chain and duck typing	187
Declaring methods that work with duck typing	188
Using generic methods for multiple objects	190
Working with duck typing in mind	192
<b>Summary</b>	<b>194</b>
<b>Chapter 7: Organization of Object-Oriented Code</b>	<b>195</b>
<b>Thinking about the best ways to organize code</b>	<b>195</b>
<b>Organizing object-oriented code in Python</b>	<b>198</b>
Working with source files organized in folders	198
Importing modules	200
Working with module hierarchies	204
<b>Organizing object-oriented code in C#</b>	<b>207</b>
Working with folders	207
Using namespaces	211
Working with namespace hierarchies in C#	220
<b>Organizing object-oriented code in JavaScript</b>	<b>223</b>
Working with objects to organize code	224
Declaring constructor functions within objects	226
Working with nested objects that organize code	229
<b>Summary</b>	<b>230</b>
<b>Chapter 8: Taking Full Advantage of Object-Oriented Programming</b>	<b>231</b>
<b>Putting together all the pieces of the object-oriented puzzle</b>	<b>231</b>
<b>Refactoring existing code in Python</b>	<b>234</b>
<b>Refactoring existing code in C#</b>	<b>241</b>
<b>Refactoring existing code in JavaScript</b>	<b>248</b>
<b>Summary</b>	<b>252</b>
<b>Index</b>	<b>253</b>

---

# Preface

Object-oriented programming, also known as OOP, is a required skill in absolutely any modern software developer job. It makes a lot of sense because object-oriented programming allows you to maximize code reuse and minimize the maintenance costs. However, learning object-oriented programming is challenging because it includes too many abstract concepts that require real-life examples to make it easy to understand. In addition, object-oriented code that doesn't follow best practices can easily become a maintenance nightmare.

Nowadays, you need to work with more than one programming language at the same time to develop applications. For example, a modern Internet of Things project may require the Python code running on a board and a combination of C#, JavaScript, and HTML code to develop both the web and mobile apps that allow users to control the Internet of Things device. Thus, learning object-oriented programming for a single programming language is usually not enough.

This book allows you to develop high-quality reusable object-oriented code in Python, JavaScript, and C#. You will learn the object-oriented programming principles and how they are or will be used in each of the three covered programming languages. You will also learn how to capture objects from real-world elements and create object-oriented code that represents them. This book will help you understand the different approaches of Python, JavaScript, and C# toward object-oriented code. You will maximize code reuse in the three programming languages and reduce maintenance costs. Your code will become easy to understand and it will work with representations of real-life elements.



## What this book covers

*Chapter 1, Objects Everywhere*, covers the principles of object-oriented paradigms and some of the differences in the approaches toward object-oriented code in each of the three covered programming languages: Python, JavaScript, and C#. You will understand how real-world objects can become part of fundamental elements in the code.

*Chapter 2, Classes and Instances*, tells you how to generate blueprints in order to create objects. You will understand the difference between classes, prototypes, and instances in object-oriented programming.

*Chapter 3, Encapsulation of Data*, teaches you how to organize data in the blueprints that generate objects. You will understand the different members of a class, learn the difference between mutability and immutability, and customize methods and fields to protect them against undesired access.

*Chapter 4, Inheritance and Specialization*, explores how to create a hierarchy of blueprints that generate objects. We will take advantage of inheritance and many related features to specialize behavior.

*Chapter 5, Interfaces, Multiple Inheritance, and Composition*, works with more complex scenarios in which we have to use instances that belong to more than one blueprint. We will use the different features included in each of the three covered programming languages to code an application that requires the combination of multiple blueprints in a single instance.

*Chapter 6, Duck Typing and Generics*, covers how to maximize code reuse by writing code capable of working with objects of different types. In this chapter, you will learn parametric polymorphism, generics, and duck typing.

*Chapter 7, Organization of Object-Oriented Code*, provides information on how to write code for a complex application that requires dozens of classes, interfaces, and constructor functions according to the programming language that you use. It will help you understand the importance of organizing object-oriented code and think about the best solution to organize object-oriented code.

*Chapter 8, Taking Full Advantage of Object-Oriented Programming*, talks about how to refactor existing code to take advantage of all the object-oriented programming techniques that you learned so far. The difference between writing object-oriented code from scratch and refactoring existing code is explained in this chapter. It will also help you prepare object-oriented code for future requirements.

## What you need for this book

You will need a computer with at least an Intel Core i3 CPU or equivalent with 4 GB RAM, running on Windows 7 or a higher version, Mac OS X Mountain Lion or a higher version, or any Linux version that is capable of running Python 3.4, and a browser with JavaScript support.

You will need Python 3.4.3 installed on your computer. You can work with your favorite editor or use any Python IDE that is compatible with the mentioned Python version.

In order to work with the C# examples, you will need Visual Studio 2015 or 2013. You can use the free Express editions to run all the examples. If you aren't working on Windows, you can use Xamarin Studio 5.5 or higher.

In order to work with the JavaScript examples, you will need web browsers such as Chrome 40.x or higher, Firefox 37.x or higher, Safari 8.x or higher, Internet Explorer 10 or higher that provides a JavaScript console.

## Who this book is for

If you're a Python, JavaScript, or C# developer and want to learn the basics of object-oriented programming with real-world examples, this book is for you.

## Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:

"We can use a `rectangle` class as a blueprint to generate the four different `rectangle` instances."

A block of code is set as follows:

```
function calculateArea(width, height) {  
    return new Rectangle(width, height).calculateArea();  
}  
  
calculateArea(143, 187);
```


When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:


```
function Mammal() {}  
Mammal.prototype = new Animal();  
Mammal.prototype.constructor = Mammal;  
Mammal.prototype.isPregnant = false;  
Mammal.prototype.pairsOfEyes = 1;
```

Any command-line input or output is written as follows:

```
Rectangle {width: 293, height: 117}  
Rectangle {width: 293, height: 137}
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "The following line prints "**System.Object**" as a result in the **Immediate Window** in the IDE."

[  Warnings or important notes appear in a box like this. ]

[  Tips and tricks appear like this. ]

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

## Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

## **eBooks, discount offers, and more**

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [customercare@packtpub.com](mailto:customercare@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

## **Questions**

If you have a problem with any aspect of this book, you can contact us at [questions@packtpub.com](mailto:questions@packtpub.com), and we will do our best to address the problem.

# 1

## Objects Everywhere

Objects are everywhere, and therefore, it is very important to recognize elements, known as objects, from real-world situations. It is also important to understand how they can easily be translated into object-oriented code. In this chapter, you will learn the principles of object-oriented paradigms and some of the differences in the approaches towards object-oriented code in each of the three programming languages: Python, JavaScript, and C#. In this chapter, we will:

- Understand how real-world objects can become a part of fundamental elements in the code
- Recognize objects from nouns
- Generate blueprints for objects and understand classes
- Recognize attributes to generate fields
- Recognize actions from verbs to generate methods
- Work with UML diagrams and translate them into object-oriented code
- Organize blueprints to generate different classes
- Identify the object-oriented approaches in Python, JavaScript, and C#

### Recognizing objects from nouns

Let's imagine, we have to develop a new simple application, and we receive a description with the requirements. The application must allow users to calculate the areas and perimeters of squares, rectangles, circles, and ellipses.

It is indeed a very simple application, and you can start writing code in Python, JavaScript, and C#. You can create four functions that calculate the areas of the shapes mentioned earlier. Moreover, you can create four additional functions that calculate the perimeters for them. For example, the following seven functions would do the job:

- `calculateSquareArea`: This receives the parameters of the square and returns the value of the calculated area for the shape
- `calculateRectangleArea`: This receives the parameters of the rectangle and returns the value of the calculated area for the shape
- `calculateCircleArea`: This receives the parameters of the circle and returns the value of the calculated area for the shape
- `calculateEllipseArea`: This receives the parameters of the ellipse and returns the value of the calculated area for the shape
- `calculateSquarePerimeter`: This receives the parameters of the square and returns the value of the calculated perimeter for the shape
- `calculateRectanglePerimeter`: This receives the parameters of the rectangle and returns the value of the calculated perimeter for the shape
- `calculateCirclePerimeter`: This receives the parameters of the circle and returns the value of the calculated perimeter for the shape

However, let's forget a bit about programming languages and functions. Let's recognize the real-world objects from the application's requirements. It is necessary to calculate the areas and perimeters of four elements, that is, four nouns in the requirements that represent real-life objects:

- Square
- Rectangle
- Circle
- Ellipse

We can design our application by following an object-oriented paradigm. Instead of creating a set of functions that perform the required tasks, we can create software objects that represent the state and behavior of a square, rectangle, circle, and an ellipse. This way, the different objects mimic the real-world shapes. We can work with the objects to specify the different attributes required to calculate their areas and their perimeters.

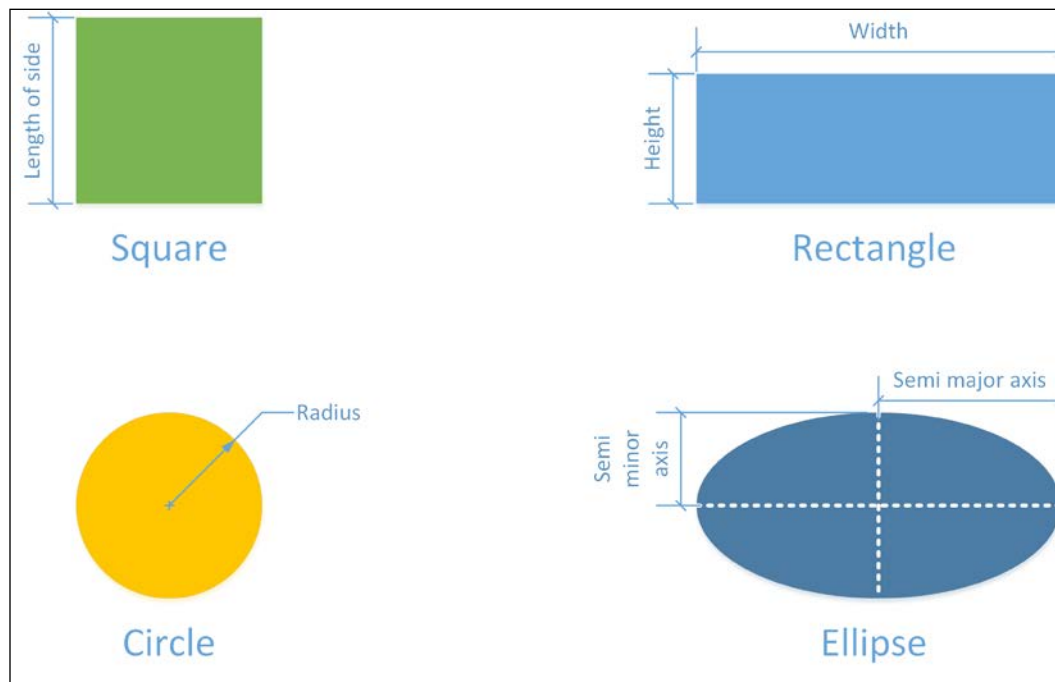
Now, let's move to the real world and think about the four shapes. Imagine that you have to draw the four shapes on paper and calculate both their areas and perimeters. What information do you require for each of the shapes? Think about this, and then, take a look at the following table that summarizes the data required for each shape:

Shape	Required data
Square	Length of side
Rectangle	Width and height
Circle	Radius (usually labeled as $r$ )
Ellipse	Semi-major axis (usually labeled as $a$ ) and semi-minor axis (usually labeled as $b$ )



The data required by each of the shapes is going to be encapsulated in each object. For example, the object that represents a rectangle encapsulates both the rectangle's width and height. *Data encapsulation* is one of the major pillars of object-oriented programming.

The following diagram shows the four shapes drawn and their elements:






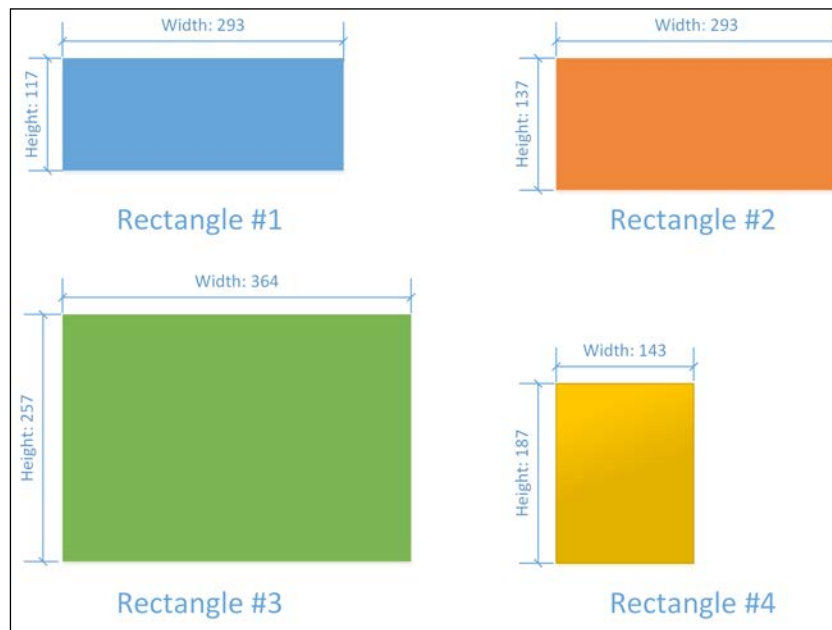
## Generating blueprints for objects

Imagine that you want to draw and calculate the areas of four different rectangles. You will end up with four rectangles drawn, with their different widths, heights, and calculated areas. It would be great to have a blueprint to simplify the process of drawing each rectangle with their different widths and heights.

In object-oriented programming, a class is a blueprint or a template definition from which the objects are created. Classes are models that define the state and behavior of an object. After defining a class that defines the state and behavior of a rectangle, we can use it to generate objects that represent the state and behavior of each real-world rectangle.

[  Objects are also known as instances. For example, we can say each rectangle object is an instance of the rectangle class. ]

The following image shows four rectangle instances drawn, with their widths and heights specified: Rectangle #1, Rectangle #2, Rectangle #3, and Rectangle #4. We can use a `rectangle` class as a blueprint to generate the four different `rectangle` instances. It is very important to understand the difference between a class and the objects or instances generated through its usage. Object-oriented programming allows us to discover the blueprint we used to generate a specific object. Thus, we are able to infer that each object is an instance of the `rectangle` class.




We recognized four completely different real-world objects from the application's requirements. We need classes to create the objects, and therefore, we require the following four classes:

- Square
- Rectangle
- Circle
- Ellipse

## Recognizing attributes/fields

We already know the information required for each of the shapes. Now, it is time to design the classes to include the necessary attributes that provide the required data to each instance. In other words, we have to make sure that each class has the necessary variables that encapsulate all the data required by the objects to perform all the tasks.

Let's start with the **Square** class. It is necessary to know the length of side for each instance of this class, that is, for each `square` object. Thus, we need an encapsulated variable that allows each instance of this class to specify the value of the length of side.

 The variables defined in a class to encapsulate data for each instance of the class are known as **attributes** or **fields**. Each instance has its own independent value for the attributes or fields defined in the class.

The `Square` class defines a floating point attribute named `LengthOfSide` whose initial value is equal to 0 for any new instance of the class. After you create an instance of the `Square` class, it is possible to change the value of the `LengthOfSide` attribute.

For example, imagine that you create two instances of the `Square` class. One of the instances is named **square1**, and the other is **square2**. The instance names allow you to access the encapsulated data for each object, and therefore, you can use them to change the values of the exposed attributes.

Imagine that our object-oriented programming language uses a dot (.) to allow us to access the attributes of the instances. So, `square1.LengthOfSide` provides access to the length of side for the `Square` instance named `square1`, and `square2.LengthOfSide` does the same for the `Square` instance named `square2`.

You can assign the value 10 to `square1.LengthOfSide` and 20 to `square2.LengthOfSide`. This way, each `Square` instance is going to have a different value for the `LengthOfSide` attribute.

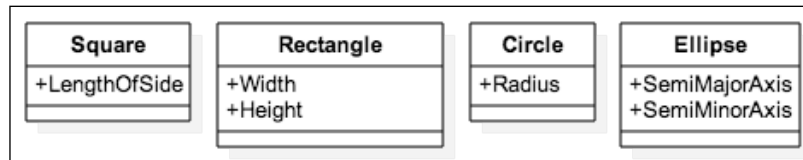
Now, let's move to the **Rectangle** class. We can define two floating-point attributes for this class: `Width` and `Height`. Their initial values are also going to be 0. Then, you can create two instances of the `Rectangle` class: `rectangle1` and **rectangle2**.

You can assign the value 10 to `rectangle1.Width` and 20 to `rectangle1.Height`. This way, `rectangle1` represents a 10 x 20 rectangle. You can assign the value 30 to `rectangle2.Width` and 50 to `rectangle2.Height` to make the second `Rectangle` instance, which represents a 30 x 50 rectangle.

The following table summarizes the floating-point attributes defined for each class:

Class name	Attributes list
Square	LengthOfSide
Rectangle	Width Height
Circle	Radius
Ellipse	SemiMajorAxis

The following image shows a **UML (Unified Modeling Language)** diagram with the four classes and their attributes:



## Recognizing actions from verbs – methods

So far, we have designed four classes and identified the necessary attributes for each of them. Now, it is time to add the necessary pieces of code that work with the previously defined attributes to perform all the tasks. In other words, we have to make sure that each class has the necessary encapsulated functions that process the attribute values specified in the objects to perform all the tasks.

Let's start with the **Square** class. The application's requirements specified that we have to calculate the areas and perimeters of squares. Thus, we need pieces of code that allow each instance of this class to use the **LengthOfSide** value to calculate the area and the perimeter.



The functions or subroutines defined in a class to encapsulate the behavior for each instance of the class are known as **methods**. Each instance can access the set of methods exposed by the class. The code specified in a method is able to work with the attributes specified in the class. When we execute a method, it will use the attributes of the specific instance. A good practice is to define the methods in a logical place, that is, in the place where the required data is kept.

The **Square** class defines the following two parameterless methods. Notice that we declare the code for both methods in the definition of the **Square** class:

- **CalculateArea**: This returns a floating-point value with the calculated area for the square. The method returns the square of the **LengthOfSide** attribute value ( $LengthOfSide^2$  or  $LengthOfSide \wedge 2$ ).
- **CalculatePerimeter**: This returns a floating-point value with the calculated perimeter for the square. The method returns the **LengthOfSide** attribute value multiplied by 4 ( $4 * LengthOfSide$ ).

Imagine that, our object-oriented programming language uses a dot (.) to allow us to execute methods of the instances. Remember that we had two instances of the **Square** class: **square1** with **LengthOfSide** equal to 10 and **square2** with **LengthOfSide** equal to 20. If we call **square1.CalculateArea**, it would return the result of  $10^2$ , which is 100. On the other hand, if we call **square2.CalculateArea**, it would return the result of  $20^2$ , which is 400. Each instance has a diverse value for the **LengthOfSide** attribute, and therefore, the results of executing the **CalculateArea** method are different.

If we call **square1.CalculatePerimeter**, it would return the result of  $4 * 10$ , which is 40. On the other hand, if we call **square2.CalculatePerimeter**, it would return the result of  $4 * 20$ , which is 80.