Object-Oriented Programming In TypeScript

A Beginner's Guide

Adegoke Akintoye

Object-Oriented Programming In TypeScript

A Beginner's Guide

Adegoke Akintoye

First edition. April 8th, 2024.

Juri Books (email: call.juri@outlook.com tel: +2349012885870)

Disclaimer

While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Other Books By The Author:

- Mastering JavaScript Array Methods: A Beginner's Guide to Simplifying Array Manipulation
- Mastering JavaScript String Methods: A Beginner's Guide to Simplifying String Manipulation
- Mastering Coding Test: 50 Problems with Solutions
- Mastering Design Patterns in TypeScript: An Approachable Guide
- Object-Oriented Programming In TypeScript: A Beginner's Guide
- Mastering TypeScript: A Beginner’s Guide
- JavaScript: The Ultimate Guide to Interview Questions
- Integrating HTMX with Laravel: An Approachable Guide
- Object-Oriented Programming in PHP:An Approachable Guide
- Functional Programming in TypeScript: An Approachable Guide
- Laravel Guide
- Mastering API Development with Laravel
- HTMX Guide
- Lumen Illuminated
- **Easy, Fast, and Practical PWA**

Table of Contents

Other Books By The Author:....5
- Preface....10
  - Introduction to TypeScript....10
  - Why Object-Oriented Programming (OOP)?....10
  - Setting Up Your TypeScript Environment....10

Chapter 1: Grasping the Basics of TypeScript....11
- Variables, Types, and Interfaces....11
  - Understanding Variables and Types....11
  - Interfaces: Defining Object Shapes....11
- Functions and Generics....12
  - Functions: Performing Tasks....12
  - Generics: Flexible and Reusable Code....12
- Modules and Namespaces....13
  - Modules: Organizing Code....13
  - Namespaces: Grouping Related Code....13

Chapter 2: Diving Into Object-Oriented Programming with TypeScript....14
- What is Object-Oriented Programming?....14
  - The Four Pillars of OOP....15

Chapter 3: Mastering Encapsulation in TypeScript....18
- Understanding Encapsulation....18
  - Why Encapsulation?....18
- Implementing Encapsulation in TypeScript....18
  - A Simple Example....18
  - Getters and Setters....20
- Conclusion....20

Chapter 4: Unveiling Abstraction in TypeScript....21
- The Essence of Abstraction....21
  - Why Abstraction?....21
  - Interfaces vs. Abstract Classes: When to Use Which?....24
- Interface, Abstract Class, and Contract:....24
  - Interfaces: The Blueprint for Structure....24
  - Abstract Classes: The Partial Blueprint....24
  - When to Use Each?....25
  - Contracts: The Core Agreement....25
  - Contract Practical Application: Payment System Example....26
- Conclusion....27

Chapter 5: Harnessing the Power of Inheritance in TypeScript....27
- Understanding Inheritance....27
  - Why Inheritance?....27
- Implementing Inheritance in TypeScript....28
  - Extending Classes....28
  - Overriding Methods....29
  - The "is-a" Relationship....29
- Conclusion....30

Chapter 6: Exploring Polymorphism in TypeScript....30
- Understanding Polymorphism....30
  - Why Polymorphism?....30

Implementing Polymorphism in TypeScript....31
Polymorphism with Interfaces....31
Polymorphism with Abstract Classes....32
Method Overloading and Overriding....32
Polymorphism through Generics:....33
Conclusion....34
Chapter 7: Advanced Object-Oriented Concepts in TypeScript....34
Composition over Inheritance....34
Example: Using Composition....34
Static Members and Methods....35
Example: Using Static Members....35
Understanding the "this" Keyword....36
Example: Using "this" Keyword....36
Advanced Types and Their Use Cases in OOP....36
Example: Using Enums....36
Example: Using Generics....36
Example: Using Union Types....37
Conclusion....37
Chapter 8: Design Patterns in TypeScript....38
Understanding Design Patterns....38
Why Design Patterns?....38
Implementing Design Patterns in TypeScript....39
Singleton Pattern....39
Factory Pattern....40
Observer Pattern....41
Conclusion....43
Chapter 9: First Project with TypeScript: A Simple Task Management....43
Project Overview: Building a Simple Task Management Application....43
Setting Up the Project Environment....43
Designing the Application Architecture....44
Defining the Task Model....44
Implementing the Task Controller....44
Creating the View....45
Implementing OOP Principles in the Project....45
Utilizing Encapsulation for Data Protection....45
Applying Abstraction for a Clean API....45
Leveraging Inheritance for Code Reuse....45
Employing Polymorphism for Flexibility....46
Conclusion....46
Chapter 10: Second Project with TypeScript:Crafting a Simple Blogging Platform with TypeScript….46
Project Setup and Environment Configuration....46
Initializing the Project....47
Installing TypeScript....47
Designing the Application Architecture....47
Defining the Blog Post Model....47
Implementing the Post Controller....48
Simulating the View and Interaction....49
Running the Application....49
Conclusion....49

Chapter 11: Integrating TypeScript with Modern Web Development....50
Setting Up a TypeScript Project with Webpack....50
Step 1: Initializing the Project....50
Step 2: Installing Dependencies....50
Step 3: Configuring TypeScript....50
Step 4: Configuring Webpack....51
Creating a Simple Web Application....51
Step 1: Defining the Task Model....51
Step 2: Implementing the Task Service....52
Step 3: Building the User Interface....52
Step 4: Running the Application....53
Conclusion....53
Chapter 12: Mastering TypeScript for Dynamic Web Applications....53
Understanding Event Handling in TypeScript....54
Example: Handling Click Events....54
Asynchronous Programming with Promises....54
Example: Fetching Data with Promises....54
Using async/await for Asynchronous Code....55
Example: Rewriting the Fetch with async/await....55
Interacting with APIs....55
Example: Typing API Responses....55
Conclusion....56
Chapter 13: Deploying TypeScript Applications....56
Preparing for Deployment....56
Compiling TypeScript to JavaScript....56
Minifying and Bundling....57
Environment Variables....57
Deploying to a Cloud Platform....57
Setting Up Netlify....58
Deploying Your Application....58
Monitoring and Maintenance....58
Conclusion....58
Chapter 14: Testing and Debugging TypeScript Applications....59
Setting Up a Testing Environment....59
Installing Jest....59
Writing Unit Tests....59
Example: Testing a Function....59
Running Tests....60
Debugging TypeScript....60
Debugging in Visual Studio Code....60
Integration Testing....61
Example: Testing API Endpoints....61
Conclusion....62
Chapter 15: Best Practices in TypeScript OOP....62
Code Organization and Structure....62
Use Modules Wisely....62
Leverage Access Modifiers....62
Prefer Composition Over Inheritance....63
Performance Considerations....63

Minimize Class Size....63
Use Lazy Loading....63
Readability and Maintainability....63
Implement Interfaces for Complex Types....63
Document Your Code....63
Write Unit Tests....64
Conclusion....64
Glossary of Key Terms in TypeScript and Object-Oriented Programming (OOP)....64
A....64
B....65
C....65
D....65
E....65
F....65
G....66
I....66
J....66
M....66
O....66
P....66
S....67
T....67
U....67
Appendix: Supplementary Resources and Tools for TypeScript Development....67
Learning Resources....68
Official TypeScript Documentation....68
TypeScript Exercises....68
Development Tools....68
Visual Studio Code....68
TypeScript Playground....68
Libraries and Frameworks....68
TSLint (Deprecated) and ESLint....68
DefinitelyTyped....69
Testing Tools....69
Jest....69
Mocha and Chai....69
Build Tools....69
Webpack....69
Rollup....69
Deployment and Continuous Integration....69
Netlify....69
GitHub Actions....70
Conclusion....70

Preface

Introduction to TypeScript

TypeScript, a superset of JavaScript, offers static typing and object-oriented programming features, making it a powerful tool for building large-scale applications. This book aims to introduce you to the world of object-oriented programming (OOP) in TypeScript, guiding you from the basics to more advanced concepts.

Why Object-Oriented Programming (OOP)?

OOP is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). OOP models real-world entities as software objects, which have both data and behavior. This approach to programming facilitates more modular, scalable, and maintainable code.

Setting Up Your TypeScript Environment

Before diving into the concepts, we'll guide you through setting up your TypeScript development environment, ensuring you have the tools needed to follow along with the examples and projects in this book.

This book is designed to be an easy-to-follow guide for beginners, with each chapter building upon the last, culminating in a project chapter that utilizes all the knowledge acquired. Through clear explanations, practical examples, and a hands-on project, you'll gain a solid understanding of object-oriented programming in TypeScript, ready to apply these concepts to your own projects.

Yor feedback will be highly appreciated. You can reach me here:
call.juri@outlook.com

Chapter 1: Grasping the Basics of TypeScript

Before we delve into the object-oriented part, it's essential to get comfortable with the basics of TypeScript. This chapter will walk you through the foundational elements, such as variables, types, interfaces, functions, generics, modules, and namespaces. We aim to clarify each concept and reinforce your understanding with practical code examples.

Variables, Types, and Interfaces

Understanding Variables and Types

In TypeScript, variables are like containers or boxes that store data values. TypeScript enhances JavaScript by adding types, allowing you to specify what kind of data can be stored in these containers.

Example:

```
let greeting: string = "Welcome to TypeScript!";
console.log(greeting); // Output: Welcome to TypeScript!
```

In this example, `greeting` is a variable that is designated to hold a `string` type of data, as indicated by `: string`. TypeScript supports several basic types such as `number`, `boolean`, `null`, `undefined`, and `string`, along with more complex types like arrays and objects.

Interfaces: Defining Object Shapes

An interface in TypeScript is a powerful way to define the structure of an object. It specifies what properties the object should have and the types of those properties.

Example:

```
interface User {
  name: string;
  age: number;
}

let user: User = { name: "Jane Doe", age: 28 };
console.log(user.name); // Output: Jane Doe
```

Here, the `User` interface specifies that any object matching it should have a `name` and an `age`, both of specific types. The variable `user` is of type `User` and adheres to this structure.

Functions and Generics

Functions: Performing Tasks

Functions in TypeScript, much like in JavaScript, are used to perform tasks or calculate and return values. TypeScript allows you to specify types for function parameters and the function's return value.

Example:

```
function welcome(name: string): string {
  return `Welcome, ${name}!`;
}

console.log(welcome("Alex")); // Output: Welcome, Alex!
```

This function, `welcome`, accepts one parameter, `name`, of type `string` and is also expected to return a `string`.

Generics: Flexible and Reusable Code

Generics allow you to create flexible and reusable code components. With generics, you can write a function or a class that works with any data type, not just one.

Example:

```
function getArray<T>(items: T[]): T[] {
  return new Array().concat(items);
}

let numberArray = getArray<number>([1, 2, 3]);
let stringArray = getArray<string>(["hello", "world"]);
console.log(numberArray); // Output: [1, 2, 3]
console.log(stringArray); // Output: ["hello", "world"]
```

In this example, <T> is a type variable that stands for "any type". The `getArray` function can then be used to create arrays of any type, as demonstrated with both numbers and strings.

Modules and Namespaces

Modules: Organizing Code

Modules in TypeScript help organize code into separate files and namespaces, making it easier to maintain and reuse. A module might contain a part of your program, and you can export and import members (like variables, functions, classes) between modules.

Example:

```
// file: mathUtils.ts
export function add(x: number, y: number): number {
  return x + y;
}

// file: app.ts
import { add } from './mathUtils';

console.log(add(5, 3)); // Output: 8
```

Namespaces: Grouping Related Code

Namespaces are another TypeScript feature for organizing code. They allow you to group logically related code under a named umbrella without the need for separate files.

Example:

```
namespace MathOperations {
  export function subtract(x: number, y: number): number {
    return x - y;
  }
}

console.log(MathOperations.subtract(10, 5)); // Output: 5
```

This chapter has laid the groundwork by introducing you to the basic building blocks of TypeScript. Understanding these concepts is crucial as they form the foundation upon which we'll explore object-oriented programming in TypeScript. In the following chapters, we'll dive deeper into how these principles apply to object-oriented programming, enhancing your ability to write more structured and maintainable code.

Glossary of Key Terms in TypeScript and Object-Oriented Programming (OOP)

This glossary provides definitions for key terms related to TypeScript and Object-Oriented Programming (OOP) as discussed throughout the chapters. It's designed to help you quickly reference and understand the fundamental concepts essential for working with TypeScript and building structured, scalable applications using OOP principles.

A

- **Abstract Class**: A class that cannot be instantiated on its own and is designed to be extended by other classes. It can contain both complete (concrete) methods and incomplete (abstract) methods that must be implemented by subclasses.
- **API (Application Programming Interface)**: A set of rules, protocols, and tools for building software applications. In the context of web development, it often refers to web services or web APIs that can be accessed over the HTTP protocol.

B

- **Bundling**: The process of combining multiple JavaScript files into a single file to reduce the number of server requests and optimize web application performance.

C

- **Class**: A blueprint for creating objects that encapsulate data and functionality related to that data. Classes can contain properties (data) and methods (functions).
- **Composition**: An OOP design principle where a class is composed of one or more objects of other classes, rather than inheriting from them, to achieve code reuse and flexibility.

- **Constructor**: A special method within a class that is called when a new instance of the class is created. It is often used to initialize class properties.

D

- **Debugging**: The process of identifying, tracing, and correcting errors or bugs in software code.

E

- **Encapsulation**: An OOP principle that involves bundling data (attributes) and methods (functions) that operate on the data into a single unit (class) and restricting access to some of the object's components.
- **Enum (Enumeration)**: A special "class" that represents a group of constants (unchangeable variables).

F

- **Function Overloading**: The ability to create multiple functions with the same name but different implementations. TypeScript achieves a similar effect through interfaces and type annotations.

G

- **Generics**: A feature that allows classes, interfaces, and functions to operate on parameters of various types while maintaining type safety.

I

- **Inheritance**: An OOP principle where a class (subclass) inherits properties and methods from another class (superclass). It allows for code reuse and the creation of a hierarchical class structure.
- **Interface**: A TypeScript feature that defines the structure of objects. Interfaces are used to specify a contract within the code as well as contracts with code outside of the project.

J

- **Jest**: A delightful JavaScript Testing Framework with a focus on simplicity, often used for testing TypeScript applications.

M

- **Method Overriding**: A feature that allows a subclass to provide a specific implementation of a method that is already defined in its superclass.
- **Module**: A file within a TypeScript application that contains code (classes, functions, variables, etc.) that can be exported and used in other modules.

O

- **Object**: An instance of a class containing a set of key-value pairs. It represents a single entity that encapsulates data and functionality related to that data.
- **OOP (Object-Oriented Programming)**: A programming paradigm based on the concept of "objects," which can contain data in the form of fields (often known as attributes or properties) and code in the form of procedures (often known as methods).

P

- **Polymorphism**: An OOP principle that allows methods to do different things based on the object it is acting upon, enabling a single interface to represent different underlying forms (data types).

S

- **Singleton Pattern**: A design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system.
- **Source Maps**: Files that map from the transformed source to the original source, enabling the browser to reconstruct the original source and present the reconstructed original in the debugger.
- **Static Members**: Properties or methods of a class that are shared by all instances of the class. They can be accessed without creating an instance of the class.

T

- **Type Annotations**: TypeScript syntax used to explicitly specify the type of a variable, parameter, or return value.

- **TypeScript**: A strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.

U

- **Union Types**: A TypeScript feature that allows a variable to store values of two or more different types.

This glossary covers fundamental terms that are essential for understanding and working with TypeScript and OOP. It serves as a quick reference guide to help you navigate through the concepts discussed in the chapters.

Appendix: Supplementary Resources and Tools for TypeScript Development

This appendix provides a curated list of resources, tools, and libraries that can enhance your TypeScript development experience. Whether you're a beginner looking to learn TypeScript or an experienced developer seeking to optimize your workflow, these resources offer valuable insights and assistance.

Learning Resources

Official TypeScript Documentation

- **Website**: TypeScript Documentation
- **Description**: The official TypeScript documentation is an excellent starting point for developers of all levels. It covers everything from basic concepts to advanced features.

TypeScript Exercises

- **Website**: TypeScript Exercises
- **Description**: An interactive TypeScript tutorial with exercises to practice and improve your TypeScript skills.

Development Tools

Visual Studio Code

- **Website**: Visual Studio Code
- **Description**: A powerful, open-source code editor developed by Microsoft. It has excellent support for TypeScript out of the box, including IntelliSense, code navigation, and debugging.

TypeScript Playground

- **Website**: TypeScript Playground
- **Description**: An online editor that allows you to write TypeScript or JavaScript, see the compiled JavaScript, and run it directly in your browser.

Libraries and Frameworks

TSLint (Deprecated) and ESLint

- **Website**: ESLint TypeScript
- **Description**: While TSLint was historically used for linting TypeScript code, it has been deprecated in favor of ESLint with TypeScript support. ESLint helps maintain code quality and consistency.

DefinitelyTyped

- **Website**: DefinitelyTyped
- **Description**: A repository of high-quality TypeScript type definitions for popular JavaScript libraries that do not provide their own.

Testing Tools

Jest

- **Website**: Jest
- **Description**: A delightful JavaScript Testing Framework with a focus on simplicity, supporting TypeScript through the `ts-jest` package.

Mocha and Chai

- **Website**: Mocha, Chai
- **Description**: Mocha is a feature-rich JavaScript test framework running on Node.js, making asynchronous testing simple. Chai is an assertion library that pairs well with Mocha for testing TypeScript applications.

Build Tools

Webpack

- **Website**: Webpack
- **Description**: A static module bundler for modern JavaScript applications. When used with the `ts-loader`, it allows you to bundle TypeScript files.

Rollup

- **Website**: Rollup

- **Description**: An efficient module bundler for JavaScript that supports TypeScript through plugins, ideal for libraries and smaller projects.

Deployment and Continuous Integration

Netlify

- **Website**: Netlify
- **Description**: A platform for automating modern web projects, supporting continuous deployment from Git across all of your frontend projects, including those written in TypeScript.

GitHub Actions

- **Website**: GitHub Actions
- **Description**: Automate your workflow from idea to production directly from GitHub. You can set up CI/CD pipelines for TypeScript projects, running tests, and deploying applications.

Conclusion

The TypeScript ecosystem is rich and continuously evolving, with numerous resources and tools available to support developers. By leveraging these resources, you can enhance your productivity, ensure code quality, and keep your skills up-to-date. Whether you're learning TypeScript for the first time or looking to streamline your development process, the resources listed in this appendix will provide valuable support on your journey.