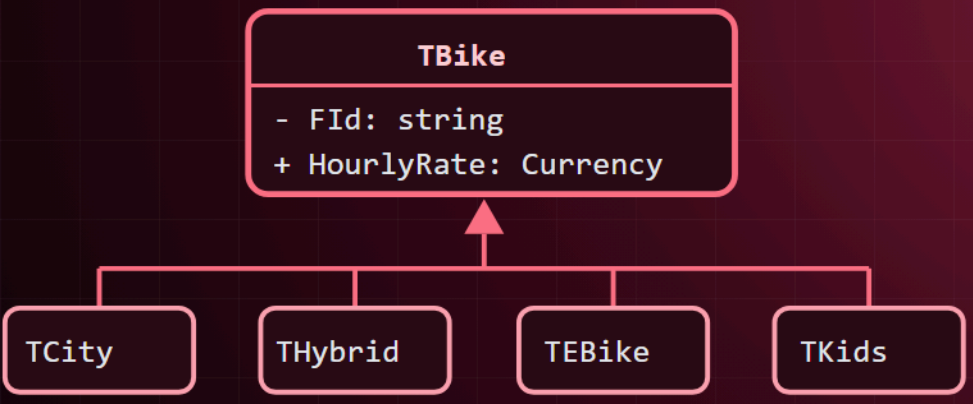



```
procedure Rent;
begin
  Bike := TBike.Create;
  try
    Bike.Rent(90);
  finally
    Bike.Free;
end;
```

OBJECT PASCAL · DELPHI

Delphi and Object Pascal from Zero

Build desktop, database, and cross-platform applications

Language, OOP, generics, VCL, FireDAC, and FireMonkey — with exercises and solutions in every chapter

OOP · VCL · FIREDAC · FIREMONKEY

THENEXBOOKS

Delphi and Object Pascal from Zero

Build Desktop, Database, and Cross-Platform Applications
Language Fundamentals, OOP, Generics, VCL, FireDAC, FireMonkey, and a Complete Project

First Edition

THENEXBOOKS

Delphi and Object Pascal from Zero
Build Desktop, Database, and Cross-Platform Applications

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples and exercise solutions may be used freely for learning and in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Examples target recent versions of Delphi, with notes for Free Pascal and Lazarus; menus, editions, and license terms change over time, so always consult the official documentation.

Delphi, RAD Studio, VCL, FireMonkey, and FireDAC are trademarks or registered trademarks of Embarcadero Technologies, Inc. Windows is a trademark of Microsoft Corporation. Other product names are trademarks of their respective owners. Their use in this book is for identification only; this book is independent and is not affiliated with or endorsed by any of these companies.

Tidewell Bike Rentals, Port Alden, and all people, customers, and data in the examples are fictional. Any resemblance to real businesses or people is coincidental.

Preface

Object Pascal is a language designed to be read. Its keywords are words rather than symbols, its types are strict, and its compiler catches many mistakes before your program ever runs. Delphi, the development environment built around it, adds a visual designer that lets you place buttons and grids on a form, wire them to code, and produce a fast native application in minutes. That combination has kept countless business, engineering, and desktop applications running for decades — and it remains a productive way to build software today.

This book starts from zero. We begin with variables, decisions, and loops, move through procedures, records, units, and exceptions, and then spend real time on object-oriented programming, because classes, interfaces, and object lifetime are where most Delphi code lives and where most bugs hide. Later parts cover modern language features such as generics and anonymous methods, and then the practical work of building applications: forms and components, databases with FireDAC and SQLite, cross-platform user interfaces, web APIs, and background threads.

Every chapter ends with graded exercises and worked solutions. By the end of the book you will have built a complete desk application for a small bike-rental shop — with pricing rules, a database, tests, and a responsive user interface — and you will be ready to start new Delphi projects or take over existing ones with confidence.

Who This Book Is For

- **Beginners** who want a clear, structured first programming language.
- **Developers from other languages** who need to learn Delphi for a job or a project.
- **Maintainers of existing Delphi applications** who want to understand and modernize the code they inherited.
- **Free Pascal and Lazarus users** looking for a practical guide to modern Object Pascal.

What You Will Learn

- How to install Delphi or Lazarus and build console and windowed programs.
- Object Pascal fundamentals: types, control flow, routines, arrays, records, sets, units, and exceptions.
- Object-oriented programming: classes, properties, inheritance, polymorphism, interfaces, and object lifetime.
- Modern features: generics, collections, anonymous methods, advanced records, helpers, and JSON.
- Building applications with the VCL, FireDAC and SQLite, FireMonkey, REST APIs, and threads.
- Testing with DUnitX, debugging, deployment, and maintaining legacy Delphi code.

How This Book Is Organized

Part I — Getting Started (Chapters 1–5) covers setup, program structure, types, expressions, and control flow. **Part II — Core Language** (Chapters 6–10) covers routines, structured types, units, memory basics, and exceptions. **Part III — Object-Oriented Programming** (Chapters 11–15) covers classes, properties, inheritance, interfaces, and object lifetime. **Part IV — Modern Object Pascal** (Chapters 16–20) covers generics, anonymous methods, advanced records, strings and files, and testing. **Part V — Building Applications** (Chapters 21–25) covers the VCL, databases, FireMonkey, and REST. **Part VI — Advanced Topics and Case Study** (Chapters 26–28) covers threads, deployment, and the complete project. The appendices provide a cheat sheet, templates, a code review checklist, and a glossary.

The Running Example: Tidewell Bike Rentals

Throughout the book you write the software for **Tidewell Bike Rentals**, a fictional bike-rental shop in the seaside town of Port Alden. Its fleet, rates, pricing rules, customers, and rentals give every chapter a practical job, and in the final chapter the pieces come together as a complete desk application.

Conventions

Code appears in `monospace`. Examples use Delphi syntax and unit names; where Free Pascal differs, a note explains how. File captions such as `Tidewell.Pricing.pas` show where code belongs. Exercises are labeled **[Easy]**, **[Medium]**, or **[Challenge]**.

> **TIP**
>
> A recommended practice, or a "Try It" prompt to pause and experiment.

> **NOTE**
>
> Background information, or a difference between Delphi and Free Pascal.

> **WARNING**
>
> A common mistake that causes bugs, leaks, or crashes.

Contents

Preface

PART I — GETTING STARTED

1 Why Delphi and Object Pascal Today

2 Setting Up: Delphi, Lazarus and Free Pascal, and Your First Program

3 Program Structure, Variables, Types, and Constants

4 Operators, Expressions, Strings, and Console Input and Output

5 Control Flow: if, case, and Loops

PART II — CORE LANGUAGE

6 Procedures and Functions: Parameters, Overloading, and Default Values

7 Arrays, Records, Sets, and Enumerated Types

8 Units: Organizing Code into Modules

9 Dynamic Arrays, Pointers, and Memory Basics

10 Exceptions: try..except, try..finally, and raise

PART III — OBJECT-ORIENTED PROGRAMMING

11 Classes and Objects: Fields, Methods, Constructors, and Destructors

12 Properties, Visibility, and Encapsulation

13 Inheritance, Virtual Methods, and Polymorphism

14 Interfaces and Reference Counting

15 Object Lifetime: Ownership, Free, and Avoiding Memory Leaks

PART IV — MODERN OBJECT PASCAL

16 Generics and Generic Collections

17 Anonymous Methods and Closures

18 Records with Methods, Operator Overloading, and Helpers

19 Strings, Unicode, Files, Streams, and JSON

20 Testing and Debugging: DUnitX, the Debugger, and Clean Code

PART V — BUILDING APPLICATIONS

21 VCL Desktop Applications: Forms, Components, and Events

22 Layouts, Actions, Dialogs, and Multi-Form Applications

23 Databases with FireDAC and SQLite

24 Cross-Platform Applications with FireMonkey (and Lazarus LCL)

25 Calling REST APIs and Working with JSON Data

PART VI — ADVANCED TOPICS AND CASE STUDY

26 Multithreading: TThread, Tasks, and a Responsive User Interface

27 Deployment, Packages, and Maintaining Legacy Delphi Code

28 Case Study: Building the Tidewell Bike Rentals Application

APPENDICES

A Object Pascal Cheat Sheet

B Templates and Reference

C Code Review Checklist

D Glossary

PART I

Getting Started

Why Delphi, installing Delphi or Lazarus, program structure, types, expressions, strings, and control flow.

1 · Why Delphi and Object Pascal Today

2 · Setting Up

3 · Program Structure, Variables, Types, and Constants

4 · Operators, Expressions, Strings, and Console Input and Output

5 · Control Flow

CHAPTER 1 · PART I — GETTING STARTED

Why Delphi and Object Pascal Today

Before we write a single line of code, we look at what Object Pascal and Delphi are, where they came from, what they are good at, and which tools you can use to learn them. We also meet Tidewell Bike Rentals, the small business whose software we will build together throughout this book.

What This Chapter Covers

This is a short, friendly orientation chapter. You do not need to install anything yet; that happens in Chapter 2. By the end of this chapter you will be able to answer a few basic questions that every newcomer has:

- What is the difference between Pascal, Object Pascal, and Delphi?
- Why do people still choose Delphi for new and existing applications?
- How does Delphi relate to Free Pascal and Lazarus, the free and open-source alternatives?
- What are the VCL and FireMonkey, and when would you use each one?
- What kinds of programs are written in Delphi?
- How is this book organized, and how should you work through it?

If you already know some of these answers — perhaps you have inherited a Delphi application at work — feel free to skim. If you are completely new to programming, read everything; we will define each term the first time it appears.

From Pascal to Object Pascal to Delphi

Pascal: a language designed to be clear

Pascal is a programming language created by the Swiss computer scientist Niklaus Wirth around 1970. It was designed with two goals in mind: to be a good language for *teaching* programming, and to be efficient enough for *real* programs. To meet the first goal, Pascal reads almost like structured English. Blocks of code start with `begin` and finish with `end`. Variables must be declared before they are used, and each variable has a type that the compiler checks. Assignment uses `:=`, so it is never confused with the equality test `=`.

Those design choices made Pascal very popular in universities and, later, on personal computers. Many programmers of an earlier generation learned their craft with a Pascal compiler, and Pascal compilers became known for being fast and for producing efficient programs.

Object Pascal: Pascal plus objects

As software grew larger, programmers needed better ways to organize it. **Object-oriented programming** (OOP) was one answer: you group data together with the code that works on that data, in units called *classes*. Several companies extended Pascal with object-oriented features, and the resulting family of dialects is known as **Object Pascal**. Object Pascal keeps everything that made Pascal readable and adds classes, inheritance, properties, exceptions, interfaces, and — in recent versions — generics, anonymous methods, and much more. Part III and Part IV of this book cover these features in depth.

Delphi: Object Pascal plus a visual development environment

Delphi was first released by Borland in the mid-1990s. It combined a fast Object Pascal compiler with a visual designer: instead of writing code to position every button on a window, you could drop components onto a form, set their properties with the mouse, and write only the code that responds to events such as a button click. This style of work is called **RAD**, short for *Rapid Application Development*. Today Delphi is developed by Embarcadero Technologies and is sold as part of **RAD Studio**, which also includes a C++ product.

People use the word "Delphi" in two ways. Sometimes it means the product — the IDE (Integrated Development Environment), the compiler, and the libraries. Sometimes it means the language itself, as in "this code is written in Delphi." Strictly speaking, the language is Object Pascal, and Embarcadero's documentation calls it "the Delphi language." In this book we say **Object Pascal** when we talk about the language and **Delphi** when we talk about the product and its tools.

> **NOTE**
>
> We do not list exact release dates or version histories in this book. Versions and editions change over time, and what matters for learning is the language and the ideas. When a feature needs a particular version — for example, inline variable declarations need Delphi 10.3 or later — we say so explicitly. This book targets **Delphi 12 or later**.

A First Look at Object Pascal

Before we go further, here is a small but complete program, so you can see what Object Pascal looks like. Do not worry about understanding every line; Chapters 2 to 5 explain all of it. Just notice how much of it you can already read.

QuickQuote.dpr

```
program QuickQuote;

{$APPTYPE CONSOLE}

uses
  System.SysUtils;
```

```
const
  CityRate: Currency = 8.00;   // dollars per hour for a City bike

var
  Minutes: Integer;
  ChargedHours: Integer;
  Price: Currency;

begin
  Minutes := 135;                          // 2 hours 15 minutes
  ChargedHours := (Minutes + 59) div 60;   // round up to whole hours
  Price := ChargedHours * CityRate;
  WriteLn('Charged hours: ', ChargedHours);
  WriteLn('Price: $', FormatFloat('0.00', Price));
  ReadLn;
end.
```

On an English locale, this program prints:

```
Charged hours: 3
Price: $24.00
```

Even without any training, you can probably guess what happens here. The program declares a constant for the hourly rate, a few variables, and then a block between `begin` and `end.` that does the work: it rounds 135 minutes up to 3 whole hours and multiplies by the City rate of $8.00. The words `program`, `const`, `var`, `begin`, `end` are **reserved words** of the language. Everything else is a name that we chose or a name from the standard library.

> **TRY IT**
>
> Read the program above once more and try to predict the output if `Minutes` were 61 instead of 135. (Answer: 61 + 59 is 120, and 120 `div` 60 is 2, so the price would be $16.00.) You will be able to run it yourself at the end of Chapter 2.

Why Choose Delphi and Object Pascal?

There are many programming languages. Why learn this one? Here are the strengths that keep Delphi in use, both for new projects and for applications that have been running for many years.

Fast, native, compiled applications

Delphi compiles your source code directly to **native machine code** for the target platform. The result is a normal executable file — for example, `TidewellDesk.exe` on Windows. It does not need a separate runtime, a virtual machine, or an interpreter installed on the user's computer. In many cases you can copy a single `.exe` file to another Windows machine and it simply runs. Deployment gets more involved once you add databases or extra libraries, as Chapter 27 explains, but the starting point is refreshingly simple.

The Delphi compiler is also known for being quick. Building a small or medium project usually takes moments, which keeps the edit–compile–run cycle short and pleasant while you learn.

Rapid Application Development with a visual designer

The Delphi IDE includes a **form designer**. You place components such as buttons, edit boxes, grids, and menus onto a form, arrange them visually, and configure them in the **Object Inspector**, a panel that lists every property and event of the selected component. When you need behavior, you double-click a component and the IDE creates an empty event handler for you to fill in. This approach makes it fast to build data-entry screens, internal tools, and business applications — exactly the kind of software Tidewell Bike Rentals needs.

Strong, static typing

Object Pascal is **strongly and statically typed**. Every variable has a type that is known when the program is compiled, and the compiler refuses to mix types in unsafe ways. If you try to store a piece of text in an integer variable, the program does not compile. This may feel strict at first, but it catches a large class of mistakes before your program ever runs. For business software, where a wrong number on an invoice is a real problem, that strictness is a feature.

Readable, explicit syntax

Pascal was designed to be read by humans. Words like `begin`, `end`, `then`, `do`, `procedure`, and `function` make the structure of code obvious. Declarations are separated from statements, so you always know where to look for a variable's type. Code written this way is easier to review and easier to hand over to the next developer.

Long-lived business applications

Many Delphi applications have been maintained for a very long time. Delphi has a strong tradition of backward compatibility, which means that code written for older versions can often be opened and compiled in newer ones with modest changes. That is why many developers meet Delphi not by choice but by inheritance: they join a company and find an important application written in it. If that is your situation, this book is for you too. Chapter 27 is dedicated to maintaining and modernizing legacy Delphi code.

One language, from small tools to large systems

Object Pascal scales well. The same language that prints "Hello" in five lines also supports units (modules), classes, interfaces, generics, exceptions, multithreading, and database access. You can start with a tiny console program and grow it, step by step, into a complete application — which is precisely the path this book follows.

Delphi, Free Pascal, and Lazarus

Delphi is a commercial product. It is not the only way to write Object Pascal, though, and this book makes room for the main alternative.

- **Delphi** (part of RAD Studio, from Embarcadero) includes the IDE, compilers for several platforms, the VCL and FireMonkey libraries, FireDAC for databases, and many other components. A free **Community Edition** is offered for eligible individuals and small organizations; the license terms change from time to time, so always read the current terms before you use it.
- **Free Pascal** (often shortened to FPC) is a free, open-source Object Pascal compiler that runs on many operating systems. It has several language *modes*; its Delphi mode accepts much of the same syntax as Delphi.
- **Lazarus** is a free, open-source IDE built on top of Free Pascal. It has its own visual designer and its own component library, the **LCL** (Lazarus Component Library), which is deliberately similar in spirit to Delphi's VCL.

Topic	Delphi	Free Pascal / Lazarus
License	Commercial; Community Edition for eligible users	Free and open source
Compiler	Embarcadero compilers	FPC
GUI library	VCL (Windows), FireMonkey (cross-platform)	LCL (cross-platform)
Unit names	`System.SysUtils`, `Vcl.Forms`	`SysUtils`, `Forms`
Delphi syntax	Native	With `-Mdelphi` or `{$mode delphi}`

The core language you learn in Parts I and II — types, expressions, control flow, procedures, arrays, records, units — works almost identically in both. Differences grow as we reach newer language features and the libraries: some recent Delphi features, such as inline variable declarations, are not available in Free Pascal, and the GUI and database libraries are different products. Whenever an example needs changes for Free Pascal, we point that out in a Note box.

> **NOTE**
>
> Delphi uses *scoped* unit names such as `System.SysUtils` and `System.Classes`. Free Pascal's standard units use the short names `SysUtils` and `Classes`. If you compile a book example with Free Pascal and get an error about a unit that cannot be found, the unit name in the `uses` clause is the first thing to check.

Which one should you pick? If you want to follow the book exactly, including the VCL, FireMonkey, and FireDAC chapters, use Delphi. If you cannot use Delphi, or you prefer open-source tools, Free Pascal and Lazarus will carry you through the core language chapters with only small adjustments, and Chapter 24 shows the Lazarus LCL alongside FireMonkey. Many developers keep both installed.

VCL and FireMonkey: Two Ways to Build User Interfaces

Delphi ships with two frameworks for building graphical applications. A **framework** here means a large library of ready-made classes — forms, buttons, labels, lists, grids, dialogs — plus the machinery that connects them to the operating system.

VCL — the Visual Component Library

The **VCL** is Delphi's original user interface library, and it targets **Windows only**. VCL controls are built on top of native Windows controls, so a VCL application looks and behaves like a standard Windows program. The VCL is mature, fast, and very widely used for business desktop applications. Most existing Delphi applications you might inherit are VCL applications. In Delphi, VCL units have names that start with `Vcl.`, such as `Vcl.Forms` and `Vcl.StdCtrls`, and form layouts are stored in `.dfm` files.

FireMonkey — FMX

FireMonkey, usually abbreviated **FMX**, is Delphi's cross-platform framework. With FireMonkey, the same project can target Windows, macOS, iOS, and Android; Linux support depends on your edition and version. FireMonkey draws its own controls, which gives it a lot of styling freedom and lets one design adapt to different devices. FMX units start with `FMX.`, such as `FMX.Forms`, and form layouts are stored in `.fmx` files.

Which one should you use?

If you need…	Consider
A Windows desktop application with a native look	VCL
To maintain an existing Windows application	Very likely VCL
One code base for desktop and mobile	FireMonkey
Custom styling across platforms	FireMonkey

The good news is that the Object Pascal language is the same in both, and the ideas — forms, components, properties, events — carry over. In this book we start with the VCL (Chapters 21 and 22) because it is the most common choice for desktop business software, and then build a cross-platform version with FireMonkey in Chapter 24. The core logic of Tidewell Bike Rentals — the pricing rules, the fleet, the rentals — will live in plain Object Pascal units that do not depend on either framework, so we can reuse it everywhere.

> **TIP**
>
> Keep your business logic out of your forms. A pricing calculation that lives in its own unit can be tested on its own, reused in a console tool, and shared between a VCL and a FireMonkey application. We follow this rule throughout the book, and Chapter 8 shows how to set up units to make it easy.

Where Delphi Is Used

We will not quote market-share numbers or surveys here; instead, here are the kinds of software that Delphi is commonly used to build, based on what the tools are designed for:

- **Desktop business applications**: point-of-sale systems, inventory and order management, accounting tools, booking systems, and line-of-business software for a specific industry.
- **Database front ends**: screens for entering, searching, and reporting on data stored in databases such as SQLite, InterBase, PostgreSQL, SQL Server, Oracle, or MySQL, often through the FireDAC library.
- **Internal tools and utilities**: small programs that automate a task inside a company, often written quickly by one developer.
- **Technical and engineering software** where native performance and direct access to the operating system matter.
- **Cross-platform and mobile apps** built with FireMonkey.
- **Servers and services**: Delphi can also produce console programs, Windows services, and back-end applications that expose web APIs.

Because Delphi applications tend to live for years, there is a steady need for developers who can read, maintain, and extend existing code. Learning Object Pascal well — not just the visual designer — is what lets you do that confidently.

Meet Tidewell Bike Rentals

Learning a language is easier with a real problem to solve. Throughout this book we write software for **Tidewell Bike Rentals**, a fictional bike-rental shop in the fictional seaside town of **Port Alden**. The shop is at **18 Harbor Street** and is open every day from 8:00 to 19:00.

The people

- **Nadia Brooks** owns the shop. She wants correct prices, fewer paper forms, and a clear view of which bikes are out.
- **Theo Lang** is the mechanic. He needs to mark bikes as being in maintenance and back again.
- **Ivy Chen** works at the front desk. She rents bikes to customers and takes them back, so she needs a fast, simple screen.

- **You** write the shop's software.

The fleet and the rates

Tidewell rents four types of bike. Each type has an hourly rate:

Bike type	Rate per hour	Notes
City	$8.00	
Hybrid	$10.00	
E-bike	$18.00	$50.00 deposit, refunded on return
Kids	$5.00	

The e-bike deposit is collected when the bike goes out and refunded when it comes back; it is *not* part of the rental price. The current fleet has seven bikes, each with an ID and a name:

ID	Type	Name
B101	City	Harbor Blue
B102	City	Sandpiper
B201	Hybrid	Ridgeway
B202	Hybrid	Tern
B301	E-bike	Volt One
B302	E-bike	Volt Two
B401	Kids	Minnow

Each bike is in one of three states at any time: **Available**, **Rented**, or **Maintenance**. The shop also has customers, some of whom are members: Ana Ruiz (C001, member), Ben Okoro (C002, not a member), and Chloe Martin (C003, member).

The pricing rules

Nadia's pricing rules are simple to say but contain just enough detail to make good programming practice:

1. **Charged hours** are the rental minutes rounded *up* to whole hours, with a minimum of 1 hour.
2. **Daily cap**: a customer pays for at most 6 hours per rental day. A rental longer than 24 hours charges each started 24-hour day separately, and each day is capped at 6 hours.
3. **Members** get 10% off the capped price.

Here are four worked examples that we will use as test cases again and again:

Rental	Charged hours	Price
City, 2 h 15 min	3	$24.00
Hybrid, 90 min	2	$20.00
E-bike, 7 h	6 (capped)	$108.00; member $97.20
Kids, 40 min	1	$5.00

Check the e-bike example yourself: 7 hours is more than the 6-hour cap, so the customer pays 6 × $18.00 = $108.00. A member gets 10% off, so $108.00 – $10.80 = $97.20.

What we will build

Over the course of the book, the Tidewell software grows from a few lines into a real application:

- In Part I we write console programs that print price quotes.
- In Part II we organize the code into units such as `Tidewell.Types.pas` and `Tidewell.Pricing.pas`, with a `CalcRentalPrice` function at its heart.
- In Part III we model the shop with classes: `TBike`, `TCustomer`, `TRental`, `TFleet`, and `TRentalService`.
- In Part IV we use generics, anonymous methods, JSON, and unit tests.
- In Part V we build `TidewellDesk`, a VCL desktop application, store data in an SQLite database called `tidewell.db` with FireDAC, and create a cross-platform version.
- In Part VI we keep the user interface responsive with threads, prepare for deployment, and assemble everything in a final case study.

Here is a sketch of the main screen Ivy will use at the front desk by the end of the book:

```
+-------------------------------------------------------------+
| Tidewell Bike Rentals - Front Desk                          |
+-------------------------------------------------------------+
| Bike: [B201 Ridgeway (Hybrid)   v]  Customer: [C001 Ana  v] |
| Minutes: [ 90 ]                     [ Quote ]  [ Rent ]     |
|-------------------------------------------------------------|
| ID    Name          Type     Status                         |
| B101  Harbor Blue   City     Available                      |
| B202  Tern          Hybrid   Maintenance                    |
| B302  Volt Two      E-bike   Rented                         |
+-------------------------------------------------------------+
```

How to Use This Book

Who this book is for

This book is written for three groups of readers. **Beginners** with little or no programming experience can start at page one; we explain every keyword the first time we use it and show what every program prints. **Developers coming from other languages** such as C#, Java, Python, or JavaScript will find the early chapters quick and can focus on what is different about Object Pascal: its declaration sections, its strong typing, and its manual memory management for objects. **Maintainers** who have inherited a Delphi application will find the language chapters useful for reading unfamiliar code, and Chapter 27 for modernizing it safely.

How the book is organized

Part	Chapters	You will learn
I — Getting Started	1–5	Tools, program structure, types, expressions, control flow
II — Core Language	6–10	Routines, data structures, units, memory, exceptions
III — Object-Oriented Programming	11–15	Classes, properties, inheritance, interfaces, lifetime
IV — Modern Object Pascal	16–20	Generics, closures, helpers, files, JSON, testing
V — Building Applications	21–25	VCL, databases, FireMonkey, REST
VI — Advanced Topics	26–28	Threads, deployment, legacy code, case study

The appendices contain an Object Pascal cheat sheet, templates for programs, units, classes, and forms, a code review checklist, and a glossary.

Conventions used in this book

- Code appears in a `monospaced font`. A file name above a listing, such as `Tidewell.Pricing.pas`, tells you which file the code belongs in.
- Output from programs is shown in a separate block right after the program.
- **Tip** boxes give practical advice. **Note** boxes add background or point out Free Pascal differences. **Warning** boxes highlight common mistakes. **Try It** boxes invite you to experiment.
- We follow common Delphi naming conventions: types start with `T` (`TBike`), interfaces with `I`, exceptions with `E`, fields with `F`, and parameters with `A` (`AHours`). We indent with two spaces.
- Money uses the `Currency` type, which Chapter 3 explains.

How to study

Programming is learned by doing. Each chapter ends with Practice Exercises labeled **[Easy]**, **[Medium]**, or **[Challenge]**, followed by worked solutions to a selection of them. We recommend this routine:

1. Read a section, then type the code yourself instead of copying and pasting. Typing builds memory, and the mistakes you make teach you what compiler errors look like.
2. Run every program and compare the output with the book.
3. Change something — a number, a name, a condition — and predict the result before you run it again.
4. Try the exercises before reading the solutions. If you are stuck for more than twenty minutes, read the solution, close the book, and write it again from memory.

> **WARNING**
> Do not skip Part I and Part II to jump straight into the visual designer. It is possible to build a form by clicking and pasting snippets, but without the core language you will quickly get stuck on bugs you cannot explain, especially memory leaks and type errors. The visual tools are much more powerful once you understand the code they generate.

Using AI assistants

You may be tempted to ask a chat assistant such as `<chat-model>` to write Delphi code for you. That can be useful for ideas, but treat generated code like code from a stranger: read it, compile it, test it, and make sure you understand every line. Generated Object Pascal sometimes mixes Delphi and Free Pascal syntax, uses unit names that do not exist, or forgets to free objects. The skills in this book are exactly what you need to spot those problems.

> **TRY IT**
> Take a sheet of paper and work out the price for each of these rentals using Nadia's rules: a Hybrid for 61 minutes, a City bike for 45 minutes, and an E-bike for 2 h 30 min for a member. You should get $20.00, $8.00, and $48.60. You will write code for exactly this kind of calculation in Chapter 4.

Practice Exercises

1. **[Easy]** In one sentence each, explain the difference between Pascal, Object Pascal, and Delphi.
2. **[Easy]** Name three strengths of Delphi described in this chapter and, for each, give one reason it matters to Nadia Brooks as a business owner.
3. **[Easy]** Using the pricing rules, calculate the price of a Kids bike rented for 1 h 50 min and a City bike rented for exactly 3 hours.
4. **[Easy]** Which user interface framework — VCL or FireMonkey — would you choose for a Windows-only front-desk application? Which would you choose if Nadia also wanted an Android tablet version? Explain why.
5. **[Medium]** Look at `QuickQuote.dpr`. List every reserved word you can find, every name the program declares itself, and every name that comes from the standard library.

6. **[Medium]** Calculate the price of an E-bike rented for 5 h 1 min by a non-member and by a member. Then calculate the deposit the customer hands over at the start. What total amount changes hands at the start and at the end?
7. **[Medium]** In `QuickQuote.dpr`, which single line would you change to quote a Hybrid bike, and what would you change it to? What would the program print for 135 minutes?
8. **[Medium]** Explain in your own words why the expression `(Minutes + 59) div 60` rounds up to whole hours. Test it by hand with 60, 61, and 119 minutes.
9. **[Challenge]** Using the daily-cap rule, work out the charged hours and the price for a Hybrid rented for 25 hours (1,500 minutes). Show each rental day separately.
10. **[Challenge]** Write a short plan (a numbered list in plain English, not code) for how a program could calculate the price for any rental, including the multi-day rule and the member discount. This plan is called *pseudocode*; you will turn it into real code in later chapters.

Selected Solutions

Exercise 3

A Kids bike for 1 h 50 min is 110 minutes, which rounds up to 2 hours: 2 × $5.00 = **$10.00**. A City bike for exactly 3 hours is 180 minutes; no rounding is needed, so the charge is 3 × $8.00 = **$24.00**. Notice that "exactly 3 hours" and "2 hours 15 minutes" cost the same, because both are charged as 3 hours.

Exercise 7 (Medium)

Only the constant changes. Rename it as well, so the name stays honest:

```
const
  HybridRate: Currency = 10.00;   // dollars per hour for a Hybrid bike
...
  Price := ChargedHours * HybridRate;
```

Strictly, that is two lines — the declaration and the line that uses it — which is exactly why good names matter: when a name describes its content, changing the content means changing the name too. For 135 minutes the program prints:

```
Charged hours: 3
Price: $30.00
```

Exercise 8 (Medium)

`div` is integer division: it divides and throws away any remainder. Adding 59 before dividing pushes any value that is not an exact multiple of 60 over the next multiple, while an exact multiple stays below the next one: