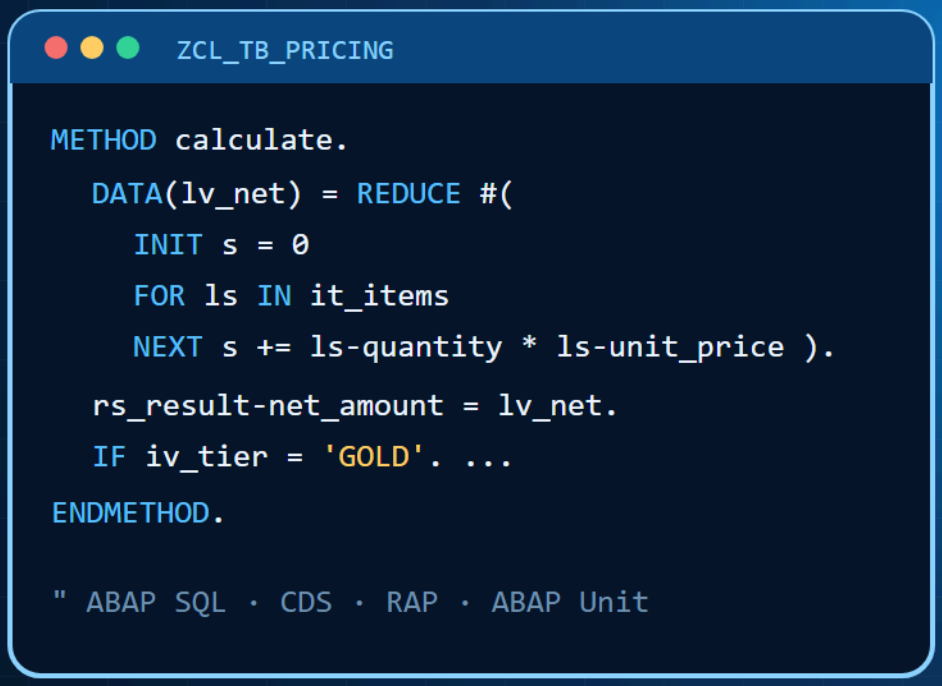


ABAP SQL

CDS views

RAP

ABAP Unit

```
› ABAP Unit: ltc_pricing 3 passed
› SELECT FROM zi_tb_product FIELDS ...
› Fiori preview: ZUI_TB_ORDER_O4
```

A COMPLETE, PRACTICAL GUIDE

ABAP Programming from Zero

Learn modern ABAP and build real business logic, data models, and Fiori apps

S/4HANA · ABAP Cloud · internal tables · OO ABAP
ABAP Unit · ABAP SQL · CDS · RAP · OData · Fiori

LANGUAGE · DATA · RAP · ABAP CLOUD

THENEXBOOKS

ABAP Programming from Zero

A Practical Guide to Modern ABAP, ABAP SQL, CDS, and RAP for SAP S/4HANA and ABAP Cloud
From your first class to tested business logic, data models, and a Fiori app

First Edition

THENEXBOOKS

ABAP Programming from Zero
A Practical Guide to Modern ABAP, ABAP SQL, CDS, and RAP for SAP S/4HANA and ABAP Cloud

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 may be used freely 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. SAP products, releases, tools, and trial offerings change; always consult SAP's official documentation for the release and edition you use.

SAP, ABAP, SAP S/4HANA, SAP HANA, SAP Fiori, SAP BTP, and other SAP products and services mentioned herein are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. Eclipse is a trademark of the Eclipse Foundation, Inc. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by SAP SE.

Tarnbrook Outdoor Gear, the business used as the running example in this book, is fictional, as are its people, customers, partners, products, and data. Domains such as example.com and hollowcrest-freight.example are reserved for examples.

Preface

ABAP is the language that runs the business logic of SAP systems: the orders, deliveries, invoices, stock movements, and reports that companies rely on every day. When a business needs SAP to do something specific to it — a pricing rule, a validation, an interface to a partner, a new app for its sales team — someone writes ABAP. That someone can be you.

ABAP has changed a great deal. The language you may have seen in old screenshots — long lists of `MOVE` statements, header lines, and `FORM` routines — has grown into a modern, expression-oriented, object-oriented language with inline declarations, constructor expressions, a powerful SQL dialect, and first-class unit testing. Around it, SAP built Core Data Services for data modeling and the ABAP RESTful Application Programming Model (RAP) for building transactional services and SAP Fiori apps. With ABAP Cloud, the same skills apply both in SAP S/4HANA and on SAP BTP.

This book teaches that modern ABAP from the ground up. We learn it by building real software for a small company: a pricing engine, an order service with clean error handling and unit tests, database tables and CDS data models, and finally a complete sales order app built with RAP and SAP Fiori elements. Along the way we also show you the classic ABAP you will still meet in existing systems, so you can read it, maintain it, and modernize it with confidence.

Who This Book Is For

- **Beginners** who want to start a career as an SAP developer and need a clear path from the first program to real applications.
- **Developers from other languages** such as Java, C#, Python, or JavaScript who are joining an SAP project and want to become productive in ABAP quickly.
- **SAP functional consultants and key users** who want to understand, read, and write the code behind their processes.
- **ABAP developers with a classic background** who want a structured tour of modern syntax, ABAP Unit, CDS, RAP, and ABAP Cloud.

What You Will Learn

- What SAP, S/4HANA, and ABAP Cloud are, and how to set up ABAP Development Tools (ADT) in Eclipse and write your first program.
- The language from the ground up: data types, inline declarations, expressions, string templates, control flow, and internal tables.
- Modern internal table techniques: table expressions, `VALUE`, `FOR`, `REDUCE`, `FILTER`, and `CORRESPONDING`.
- The ABAP Dictionary: domains, data elements, structures, and database tables.

- Object-oriented ABAP: classes, interfaces, inheritance, polymorphism, class-based exceptions, and clean design with dependency injection.
- Unit testing with ABAP Unit and test doubles; strings, dates, times, and regular expressions.
- ABAP SQL for reading and changing data, the logical unit of work, locks, and performance rules.
- Core Data Services view entities, associations, annotations, calculations, aggregations, and access control.
- Calling HTTP APIs and working with JSON and XML.
- The ABAP RESTful Application Programming Model: business objects, behavior definitions, EML, validations, determinations, actions, draft, and OData services with SAP Fiori elements.
- Classic reports, selection screens, and ALV; debugging, ATC, performance analysis, and short dumps; clean core, transports, and abapGit.

How This Book Is Organized

Part I — Getting Started (Chapters 1–5) introduces SAP and ABAP, sets up your tools, and covers types, expressions, strings, and control flow. **Part II — Core Language** (Chapters 6–10) covers internal tables, the ABAP Dictionary, modularization, and classes. **Part III — Robust ABAP** (Chapters 11–15) covers inheritance and interfaces, exceptions, clean design, unit testing, and working with text, dates, and times. **Part IV — Data and Services** (Chapters 16–20) covers ABAP SQL, Core Data Services, and calling external APIs. **Part V — The ABAP RESTful Application Programming Model** (Chapters 21–25) builds transactional business objects and OData services, then covers the classic ABAP you will meet and the tools for debugging and analysis. **Part VI — Projects and Next Steps** (Chapters 26–28) assembles the complete order service and Fiori app and prepares you for ABAP Cloud, clean core, and the rest of your career. The appendices provide a syntax cheat sheet, an old-to-new syntax guide, a field guide to short dumps and common pitfalls, and a glossary of ABAP and SAP terms.

Each chapter ends with practice exercises, selected solutions, and a **Key Takeaways** box for quick review.

The Running Example: Tarnbrook Outdoor Gear

Throughout the book we write software for **Tarnbrook Outdoor Gear**, a fictional wholesale distributor that sells tents, sleeping bags, backpacks, and accessories to outdoor retail shops. Tarnbrook has just moved to SAP S/4HANA. Elin leads the IT team, Marco runs sales operations and keeps asking what the system can do for his customers, and Jun, a junior developer, is learning ABAP alongside you. It starts with a single price calculation and grows into an order service with tests, a CDS data model, and a sales order app built with RAP and SAP Fiori elements.

Versions and Conventions

The book teaches modern ABAP as it is used in SAP S/4HANA and in the SAP BTP ABAP environment (ABAP Cloud). Where a feature needs a particular release, we say so, so that readers on older on-premise systems know what they can use. SAP's tools, releases, and trial offerings change over time; where a detail may differ in your system, a Note points you to SAP's documentation rather than guessing. Because ABAP runs only inside an SAP system, output shown in this book is what you should see, or something very close to it; small differences in formatting between releases are normal.

> **TIP**
>
> A recommended practice or a technique that makes your work easier.

> **NOTE**
>
> Background information, release differences, or a useful aside.

> **WARNING**
>
> A common mistake or something that can cause bugs, data loss, or security problems.

Code appears in `monospace`. ABAP keywords are written in uppercase and names in lowercase, the convention most ADT users choose in the pretty printer. Long lines are wrapped to fit the page. Captions above code blocks, such as `ZCL_TB_PRICING (global class)`, tell you which repository object the code belongs to.

Contents

Preface

PART I — GETTING STARTED

1 What Is ABAP? SAP, S/4HANA, ABAP Cloud, and Where ABAP Runs

2 Setting Up: ABAP Development Tools, Your System, and Your First Program

3 Data Types, Variables, Constants, and Inline Declarations

4 Operators, Expressions, Strings, and String Templates

5 Control Flow: IF, CASE, Loops, COND, and SWITCH

PART II — CORE LANGUAGE

6 Internal Tables I: Table Kinds, Keys, and Classic Operations

7 Internal Tables II: Table Expressions, VALUE, FOR, REDUCE, FILTER, and CORRESPONDING

8 Structures and the ABAP Dictionary: Domains, Data Elements, and Tables

9 Modularization: Methods, Function Modules, and Legacy Subroutines

10 Object-Oriented ABAP I: Classes, Objects, Visibility, and Constructors

PART III — ROBUST ABAP

11 Object-Oriented ABAP II: Inheritance, Interfaces, and Polymorphism

12 Exceptions and Messages: TRY, CATCH, CLEANUP, and Exception Classes

13 Clean ABAP: Naming, Design Patterns, and Dependency Injection

14 Unit Testing with ABAP Unit and Test Doubles

15 Strings, Dates, Times, Regular Expressions, and Conversions

PART IV — DATA AND SERVICES

16 ABAP SQL I: SELECT, Joins, Aggregates, and Host Expressions

17 ABAP SQL II: Changing Data, the LUW, Locks, and Performance Rules

18 Core Data Services I: View Entities, Associations, and Annotations

19 Core Data Services II: Parameters, Calculations, Aggregations, and Access Control

20 Calling APIs and Working with JSON and XML

PART V — THE ABAP RESTFUL APPLICATION PROGRAMMING MODEL

21 RAP Fundamentals: Business Objects, Behavior Definitions, and EML

22 RAP Behavior: Validations, Determinations, Actions, and Draft

23 Exposing a RAP Business Object: Projections, OData Services, and Fiori Elements

24 Classic ABAP You Will Meet: Reports, Selection Screens, ALV, and Dynpros

25 Debugging, ATC, Performance Analysis, and Troubleshooting Short Dumps

PART VI — PROJECTS AND NEXT STEPS

26 Project: The Tarnbrook Order Service with Tests

27 Project: A Tarnbrook Sales Order App with RAP and Fiori Elements

28 ABAP Cloud, Clean Core, Transports and abapGit, and Where to Go Next

APPENDICES

A Modern ABAP Syntax Cheat Sheet

B Old-to-New Syntax Translation Guide

C Short Dumps, Syntax Errors, and Common Pitfalls Field Guide

D Glossary of ABAP and SAP Terms

PART I

Getting Started

What SAP and ABAP are and where ABAP runs today, how to set up ABAP Development Tools and write your first class, and the fundamentals every ABAP program is built from: data types, inline declarations, expressions, string templates, and control flow.

1 · What Is ABAP?

2 · Setting Up and Your First Program

3 · Data Types and Declarations

4 · Operators, Strings, and Templates

5 · Control Flow

CHAPTER 1 · PART I — GETTING STARTED

What Is ABAP? SAP, S/4HANA, ABAP Cloud, and Where ABAP Runs

Before you write a single line of code, it helps to know what world you are stepping into. This chapter explains what SAP systems do, why ABAP exists, where your programs actually run, and how modern ABAP Cloud differs from the classic ABAP you will still meet in many companies — and it introduces the company whose software we will build together throughout the book.

Why This Chapter Exists

Most programming books can start with "install the compiler and print Hello World." ABAP is different. ABAP is not a general-purpose language that you download and run on your laptop. It lives inside an SAP system, it stores its source code in a database, it is moved between systems with a dedicated transport mechanism, and the same piece of code may behave differently depending on which *client* you are logged on to. If nobody explains these things, the first week of ABAP feels like a maze of unfamiliar words.

So we will take one chapter to build a mental map. Treat it as orientation, not something to memorize. In Chapter 2 you will set up your tools and run your first program.

SAP, ERP, and S/4HANA in Plain Words

SAP is a German software company, and its best-known product is an **ERP system**. ERP stands for *Enterprise Resource Planning*, which is a formal name for a simple idea: one integrated system that runs the core business processes of a company. Sales orders, purchasing, warehouse stock, accounting, invoices, payroll, production planning — in an ERP system these are not separate programs with separate databases. They share one set of data, so when a salesperson creates an order, the warehouse sees the reservation, finance sees the expected revenue, and the planners see the demand.

SAP's ERP products have gone through several generations. The older generation that many companies still run is usually called **SAP ECC** (ERP Central Component). The current generation is **SAP S/4HANA**. The "4" marks it as the fourth major product line after earlier systems, and "HANA" refers to **SAP HANA**, the in-memory database that S/4HANA runs on. S/4HANA simplified much of the old data model, because a fast in-memory database can calculate many totals on the fly instead of storing them in separate summary tables.

S/4HANA can run in a company's own data center or with a hosting provider (**on-premise** or "private" deployments), or as SAP's **public cloud** edition, operated by SAP. The deployment model affects which kinds of ABAP code you may write — a theme we return to later in this chapter.

Standard software and custom code

SAP delivers a huge amount of standard functionality, yet no two companies work exactly the same way: a distributor may need a special discount rule, a retailer a report nobody else wants. Companies configure the standard system first — SAP calls this **customizing** — and when configuration is not enough, they write custom code. That custom code is written in ABAP.

What ABAP Is For

ABAP is SAP's own programming language. It is designed for business applications: it has built-in support for database access, for decimal arithmetic with currencies, for dates and times, for tables of data held in memory, and for the transaction handling that business systems need. Typical tasks for an ABAP developer include:

- **Custom business logic** — rules the standard system does not have, such as "gold customers get a 10% discount."
- **Reports and analytics** — reading business data and presenting it, either as a classic list, a grid, or a data model consumed by analytical tools.
- **Interfaces** — exchanging data with other systems through web services, REST APIs, files, or SAP-specific protocols.
- **Extensions** — plugging custom behavior into standard processes at predefined extension points, without modifying SAP's own code.
- **Fiori apps** — building the back end of modern web applications. SAP Fiori is SAP's user-experience design for browser-based apps, and in modern ABAP the back end of such apps is built with the **ABAP RESTful Application Programming Model (RAP)**, which exposes your business objects as OData services.
- **Data conversion and migration** — loading and transforming data when a company moves to a new system.

If you come from Java, C#, Python, or JavaScript, you will find ABAP familiar in some ways (classes, interfaces, exceptions, unit tests) and unusual in others (SQL is part of the language, every statement ends with a period, and source code lives in a database rather than in files). Modern ABAP is also much more compact than its verbose reputation suggests.

A Short History of ABAP

You do not need a history lesson to write code, but a little history explains why ABAP code in the wild looks so varied. When you open a program written fifteen or thirty years ago, you will see a different style than in code written last month, and both run on the same system.

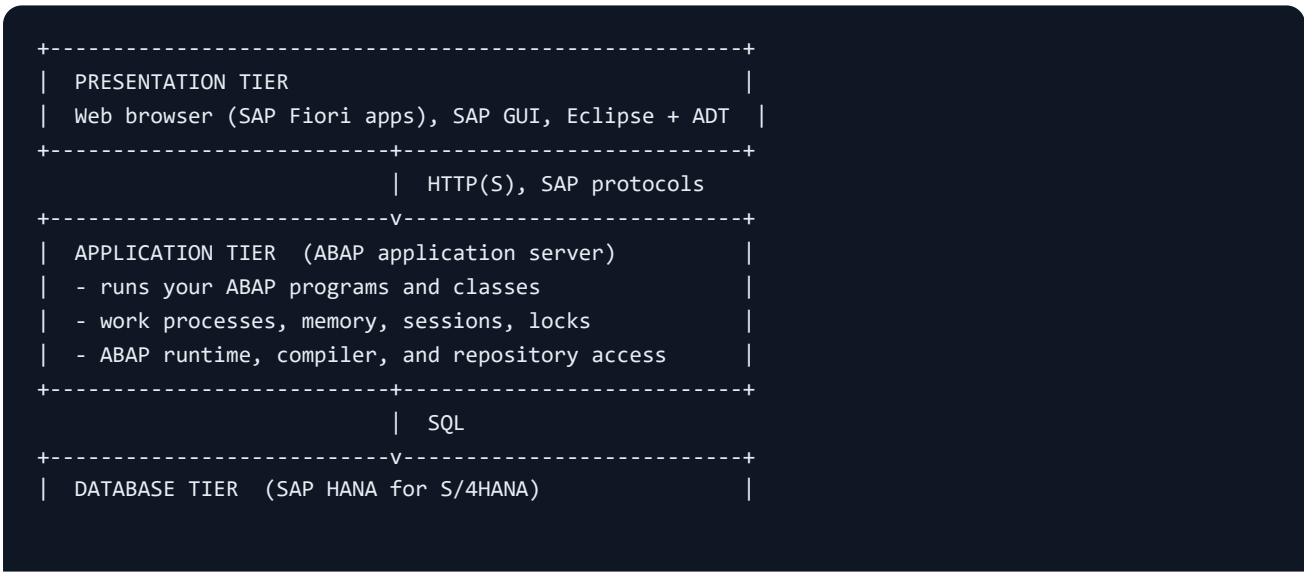
- **The 1980s:** ABAP began as a language for generating reports on SAP's mainframe-era systems. Its original German name roughly meant "general report preparation processor." Later the acronym was reinterpreted as *Advanced Business Application Programming*.
- **The 1990s:** With SAP's client-server system, SAP R/3, ABAP became the language in which most of SAP's own application was written. Procedural constructs from this era — reports, subroutines (`FORM`), function modules, and classic screens — are still everywhere.
- **The late 1990s:** SAP added object orientation under the name **ABAP Objects**: classes, interfaces, inheritance, events, and class-based exceptions. Today, classes are the standard way to structure new code.
- **Release 7.40 and later:** A wave of modern syntax arrived — inline declarations, constructor expressions such as `VALUE #( )` and `NEW #( )`, table expressions, and a modernized SQL syntax. These features made ABAP far more expressive, and this book uses them from the start.
- **The S/4HANA and cloud era:** Core Data Services (CDS) became the standard way to model data, RAP became the standard way to build transactional apps and services, and SAP introduced **ABAP Cloud**, a restricted, upgrade-safe flavor of ABAP for cloud-ready development.

Because SAP takes backward compatibility seriously, all of this still works side by side — which is why Chapter 24 covers the classic constructs you will maintain, even if you never write them.

Where ABAP Runs: The Three-Tier Architecture

An SAP system is organized in three layers, or **tiers**. Understanding them answers a common beginner question: "Where is my program actually executing?"

The three tiers of an SAP system



```
|  - business data: orders, products, customers         |
|  - the ABAP repository: your source code and metadata |
+-------------------------------------------------------+
```

The presentation tier

This is what users see: SAP Fiori apps in a web browser, or **SAP GUI**, a desktop client for classic SAP screens. Your main developer tool, **Eclipse with ABAP Development Tools (ADT)**, also sits here. The presentation tier does not run your ABAP code; it displays results and sends input.

The application tier

This is where ABAP runs. The **ABAP application server** (often called the ABAP platform, or historically the "AS ABAP") compiles your programs and executes them in **work processes** — server processes that each handle one request at a time. When a user clicks a button in a Fiori app, the request arrives at the application server, a work process runs the relevant ABAP code, and the result goes back to the browser. A larger system has several application server instances to share the load, and they all run the same code from the same repository.

The database tier

The database stores the business data — every order, product, and customer — and also the ABAP **repository**, which holds the source code and metadata of every program, class, and table definition. For S/4HANA and the SAP BTP ABAP environment, the database is SAP HANA; older systems may run on other databases, and ABAP SQL hides most of the differences.

> **NOTE**
>
> "Code pushdown" is a phrase you will hear often. It means moving data-intensive calculations from the application tier down into the database, where SAP HANA can process large data volumes quickly. In modern ABAP you do this mainly with ABAP SQL and CDS view entities, covered in Chapters 16–19. The rule of thumb: do not fetch a million rows to the application server just to add them up.

The ABAP Repository, Packages, Transports, and Clients

Four concepts confuse almost every newcomer, because they have no direct equivalent in most other environments. Let's take them one at a time.

The repository: code lives in the database

In Java or Python, your source code is a set of files in a folder, usually tracked with Git. In ABAP, every development object — a class, an interface, a table definition, a CDS view, a message class — is a **repository object** stored in the system's database. When you edit a class in Eclipse, you are editing a

copy held on the server. Saving stores an *inactive* version; **activating** it checks the code and makes it the version that everyone else in the system uses. There is no separate build step to run: the ABAP server compiles on demand.

Because the repository is shared, developers on the same system see each other's activated objects immediately, and an object you are editing is locked against parallel edits.

Packages: folders with rules

Every repository object belongs to a **package**. A package groups related objects, like a folder or a module, and it can also define rules about which other packages may use its objects. Customer packages start with `Z` or `Y` (or with a registered company namespace), so they never clash with SAP's own objects. In this book, all Tarnbrook objects go into the package `ZTARNBROOK`.

There is also a special package for local experiments, `$TMP`. Objects in `$TMP` are "local objects": they exist only in the current system and can never be transported. That makes `$TMP` a handy scratch pad on systems that allow it, but nothing in it can ever reach production.

Transports: how code travels to production

Companies do not write code directly in their production system. A typical **system landscape** has at least three systems:

```
DEV (development)  --->  QAS (quality assurance)  --->  PRD (production)
write and test           business users test           real work happens
```

Changes move from one system to the next inside **transport requests**. A transport request is a numbered container that records which repository objects you created or changed. When you create an object in a transportable package, the system asks you to assign it to a request. When the work is finished, the request is *released*, and the transport system imports it into the quality system and later into production. This gives companies a controlled, auditable path for every change. We look at the transport prompt in Chapter 2 and discuss transports, abapGit, and the cloud equivalents in Chapter 28.

Clients: several companies in one system

An SAP system can contain several **clients**. A client is a self-contained business unit inside one system, identified by a three-digit number such as `100`. You always log on to a specific client. Most business data — orders, customers, products — is **client-dependent**: a table holding such data has a client field as the first key field, and the ABAP runtime automatically restricts your database reads and writes to the client you are logged on to. A development system might have one client for development and another with test data, and they will not see each other's orders.

Repository objects, on the other hand, are **client-independent**. A class you activate exists in every client of that system. That is why our Tarnbrook table definitions start with a `client` key field — the data is per client — while our classes are simply available everywhere.

Concept	Plain-language meaning	Tarnbrook example
Repository object	A piece of code or metadata stored in the system	Class `zcl_tb_pricing`
Package	A folder with ownership and usage rules	`ZTARNBROOK`
Transport request	A numbered list of changes to ship	"Tarnbrook pricing rules"
Client	A separate data area inside one system	Client `100`
Activation	Check and publish the current version	Ctrl+F3 in ADT

Classic ABAP and ABAP Cloud

For most of its history, ABAP gave developers almost unlimited freedom. Custom code could read any SAP table, call any internal function module, and even modify SAP's own programs. That freedom made SAP systems very adaptable, but it came at a price: when SAP shipped an upgrade, the internal objects that custom code depended on might change, and upgrades turned into long, expensive projects of fixing broken custom code.

ABAP Cloud is SAP's answer to that problem. It is not a different language; it is a **development model** built on a restricted version of ABAP. Its key ideas are:

- **Released APIs only.** Your code may use only SAP objects that SAP has explicitly released for cloud development — classes, interfaces, CDS views, and other objects that SAP promises to keep stable across upgrades. Using an unreleased object is a syntax error, not just a style warning.
- **A restricted language version.** Each ABAP object has an **ABAP language version**. The version for cloud development, often called "ABAP for Cloud Development," forbids statements that only make sense in classic environments, such as the `WRITE` list output, classic screens, and direct access to unreleased database tables. The classic, unrestricted version is usually called "Standard ABAP."
- **Modern tools and models.** You develop in Eclipse with ADT, model data with CDS, and build transactional apps and services with RAP.
- **Clean core.** SAP's own code stays untouched. Customer extensions sit next to the standard, connected only through stable, released interfaces, so the core system can be upgraded without breaking them.

> **WARNING**
>
> Code that works in a classic on-premise system may not compile in ABAP Cloud. If you copy examples from old blog posts or forum threads, check whether they use classic-only statements (such as `WRITE`, selection screens, or unreleased function modules). This book labels classic-only constructs explicitly, and its main examples use ABAP Cloud-compatible code.

On-Premise Systems and the SAP BTP ABAP Environment

Where can ABAP code run today? There are two broad families of systems.

On-premise and private-cloud ABAP systems

These are S/4HANA systems (and older ECC systems) that a company operates itself or has hosted for it. They support both Standard ABAP and ABAP Cloud. Developers can use SAP GUI transactions as well as Eclipse with ADT. In recent S/4HANA releases, SAP encourages companies to write new custom code following ABAP Cloud rules even in these systems, so the code stays upgrade-safe. In practice, many on-premise systems contain a mix: old classic code, and new code written in the cloud style.

The SAP BTP ABAP environment

SAP Business Technology Platform (BTP) is SAP's cloud platform for building and running extensions and integrations. One of its services is the **SAP BTP ABAP environment** — a complete ABAP platform operated by SAP in the cloud. In this environment, only ABAP Cloud is allowed: no SAP GUI development, no classic reports, only released APIs, and Eclipse with ADT as the development tool. It is a natural place to learn modern ABAP because it simply does not let you fall back into old habits.

Aspect	Classic on-premise	ABAP Cloud
SAP objects you may use	Anything, released or not	Released APIs only
Language version	Standard ABAP	ABAP for Cloud Development
Main tool	SAP GUI or ADT	ADT only
User interface	Classic screens, lists, Fiori	Fiori (via RAP)
Upgrade safety	Your responsibility	Designed in

This book's main path is ABAP Cloud. Everything in the core chapters works in the SAP BTP ABAP environment and in recent S/4HANA systems. When we show a classic construct, we say so, and we explain how to do the same thing the modern way.

Who Works with ABAP?

SAP projects involve many roles, and it helps to know who you will be talking to.

- **ABAP developers** write and maintain custom code: business logic, reports, interfaces, extensions, and the back ends of Fiori apps. That is the role this book prepares you for.
- **Functional consultants** specialize in a business area — sales, finance, procurement, warehouse management — and configure the standard system. They translate business requirements into configuration and, when configuration is not enough, into specifications for developers. Many functional consultants learn ABAP so they can read code, debug problems, and talk with developers on equal terms.
- **Basis administrators** run the systems: installation, users, transports, performance, and upgrades. ("Basis" is the traditional SAP name for the technical platform.)

The boundaries are fluid. A developer who understands the business process writes better code, and a consultant who can read ABAP solves problems faster.

Meet Tarnbrook Outdoor Gear

This book follows one company from start to finish. **Tarnbrook Outdoor Gear** is a fictional wholesale distributor of outdoor equipment. It buys tents, sleeping bags, backpacks, and accessories and sells them to retail shops. Tarnbrook has just moved to SAP S/4HANA, and the standard system covers most of its needs — but not all. It needs custom ABAP development, and you are joining the team that will build it.

Three people will appear throughout the book:

- **Elin** is the head of IT and the development lead. She sets the technical direction, insists on clean, tested code, and reviews what the team builds.
- **Marco** works in sales operations. He knows Tarnbrook's customers and pricing rules by heart and asks the business questions that drive our requirements: "Which orders qualify for free shipping?" "Why is this order blocked?"
- **Jun** is a junior developer who is learning ABAP alongside you. Jun makes the mistakes beginners make — forgotten periods, unknown variables, misplaced keys — so that you can learn from them without the embarrassment.

Tarnbrook's catalog is small enough to keep in your head:

Product	Name	Category	Price (USD)	Stock (EA)
`TB-1001`	Ridge 2 Tent	TENTS	249.00	40
`TB-1002`	Alpine Down Sleeping Bag	SLEEPING	189.50	25
`TB-1003`	Trail 40L Backpack	PACKS	129.00	60

`TB-1004`	Summit Trekking Poles	ACCESSORY	59.90	120
`TB-1005`	Canyon Water Filter	ACCESSORY	39.95	0

Its customers are retail shops: **Peakline Outfitters** (customer `C100`, a `GOLD` customer in the US), **Bluehollow Camping Co.** (`C200`, `STANDARD`, US), and **Fjellsti Sport** (`C300`, `STANDARD`, Norway). Prices are in US dollars (USD), and quantities are counted in `EA` — "each," SAP's usual unit for individual pieces.

Marco has explained the pricing rules to Elin: the net amount of an order is the sum of quantity times unit price; gold customers get a 10% discount; shipping is free when the discounted amount reaches 1,000.00 USD and costs 25.00 USD otherwise. For example, Peakline Outfitters' order `500001` — four tents and two pairs of trekking poles — has a net amount of 1,115.80, a discount of 111.58, and a total of 1,004.22 with free shipping. You will implement exactly this logic in the class `zcl_tb_pricing` and prove it with unit tests. Throughout the book we call these products, customers, and orders Tarnbrook's *canonical* data: the reference values that every example and test uses.

What we will build

1. Small console programs that teach the language with Tarnbrook's products and orders.
2. Database tables `ztb_product`, `ztb_customer`, `ztb_order`, and `ztb_order_item`.
3. A pricing class, `zcl_tb_pricing`, with ABAP Unit tests that pin down Marco's rules.
4. An order service, `zcl_tb_order_service`, with exception classes and a replaceable product repository.
5. CDS view entities and a RAP business object for sales orders, with validations, actions, and draft.
6. An OData service and a SAP Fiori elements app in which Marco's team can create and release orders.

All of these objects live in the package `ZTARNBROOK` and follow the same naming convention: the customer prefix `Z`, then `TB` for Tarnbrook — `ZCL_TB_` for classes, `ZIF_TB_` for interfaces, `ZCX_TB_` for exception classes, `ZTB_` for tables, and `ZI_TB_`, `ZR_TB_`, and `ZC_TB_` for CDS entities.

A Roadmap of the Book

The book is organized in six parts, each building on the previous one.

- **Part I (Chapters 1–5):** the SAP world, your tools, data types, expressions, strings, and control flow.
- **Part II (Chapters 6–10):** internal tables, the ABAP Dictionary, modularization, and your first classes.
- **Part III (Chapters 11–15):** interfaces, exceptions, Clean ABAP, unit testing, dates, and text.
- **Part IV (Chapters 16–20):** ABAP SQL, locks, Core Data Services, and calling APIs.

- **Part V (Chapters 21–25):** RAP, Fiori elements, classic ABAP, and debugging tools.
- **Part VI (Chapters 26–28):** two Tarnbrook projects, clean core, transports, and next steps.

Each chapter ends with practice exercises and selected solutions. Type the code yourself rather than copying it: ABAP's syntax check is strict, and learning to read its messages is a skill in itself.

A First Look at Modern ABAP

Let's finish the orientation by looking at real code. You will create and run this class yourself in Chapter 2; for now, just read it and see how much you can guess.

ZCL_TB_DEMO (global class) — a first look

```
CLASS zcl_tb_demo DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC.

  PUBLIC SECTION.
    INTERFACES if_oo_adt_classrun.
ENDCLASS.

CLASS zcl_tb_demo IMPLEMENTATION.
  METHOD if_oo_adt_classrun~main.
    " A product from the Tarnbrook catalog
    DATA(lv_product) = `Ridge 2 Tent`.
    DATA lv_price TYPE p LENGTH 8 DECIMALS 2 VALUE '249.00'.

    out->write( `Welcome to Tarnbrook Outdoor Gear` ).
    out->write( |{ lv_product } costs { lv_price } USD| ).
  ENDMETHOD.
ENDCLASS.
```

Here is what each part does:

- `CLASS ... DEFINITION` declares a class and its public interface. `PUBLIC` makes it a global class that other programs can use, `FINAL` means no other class may inherit from it, and `CREATE PUBLIC` allows anyone to create instances.
- `INTERFACES if_oo_adt_classrun.` says the class implements a released SAP interface. Any class that implements it can be run directly from Eclipse, and its output appears in the console. This is the modern replacement for the classic "Hello World" report.
- `CLASS ... IMPLEMENTATION` contains the method code; in `if_oo_adt_classrun~main`, the tilde joins interface and method name.
- `DATA(lv_product) = `Ridge 2 Tent`.` is an **inline declaration**: it declares a variable and assigns a value in one statement. The backquotes make the literal a `string`, so the variable's type is `string`.

- `DATA lv_price TYPE p LENGTH 8 DECIMALS 2 VALUE '249.00'.` declares a **packed number** — ABAP's decimal type, ideal for money because it does not suffer from binary rounding errors.
- `out->write( ... )` calls the method `write` of the object `out`, which the framework passes in. Note the spaces after the opening parenthesis and before the closing one: ABAP requires them.
- `|{ lv_product } costs { lv_price } USD|` is a **string template**. Text between the vertical bars is literal, and expressions in braces are inserted.

When you run this class, you should see something like:

```
Welcome to Tarnbrook Outdoor Gear
Ridge 2 Tent costs 249.00 USD
```

Notice what is *not* there: no imports and no build file. Also notice the period at the end of every statement — forgetting it is every beginner's most common syntax error, Jun's included.

> **TIP**
> ABAP keywords are not case-sensitive: `data`, `Data`, and `DATA` all work. Most teams let the editor's pretty printer convert keywords to uppercase and identifiers to lowercase, as in this book, so that keywords stand out. Chapter 2 shows you how to configure and run the pretty printer.

The same idea in classic ABAP

For comparison, here is how a "Hello World" looked for decades. This is a classic executable program — a **report** — that writes to a classic list screen in SAP GUI. It is **classic-only**: it does not compile in ABAP Cloud, because the `REPORT` program type and the `WRITE` statement are not part of ABAP for Cloud Development.

`ZTB_HELLO (classic report — on-premise only)`

```
REPORT ztb_hello.

WRITE 'Hello from classic ABAP'.
```

You will see thousands of programs like this in older systems, and Chapter 24 explains how to read them. But for new development, the class-based approach is the standard: it works everywhere, it can be unit-tested, and it is ready for ABAP Cloud.

Common Misunderstandings

- **"I can install ABAP on my laptop."** You install the *tools* locally and connect to an ABAP system elsewhere. Chapter 2 shows the options.