THE WORLD OF SMALLTALK

FROM PHILOSOPHY TO PRACTICE

A COMPLETE GUIDE TO THE LANGUAGE THAT INVENTED OBJECT-ORIENTED PROGRAMMING

- EVERYTHING IS AN OBJECT
- COMMUNICATION VIA MESSAGE PASSING
- INTERACTIVE. LIVE. PRODUCTIVE.

MATTHIAS FALKNER

The World of Smalltalk

From Philosophy to Practice: A Complete Guide to the Language That Invented Object-Oriented Programming

Steve Publications

This book is available at https://leanpub.com/theworldofsmalltalk

This version was published on 2026-08-12

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

From Philosophy to Practice A Complete Guide to the Language That Invented Object-Oriented Programming

About This Book

Smalltalk is not just a programming language. It is a complete computing environment whose design philosophy where everything is an object, communication via message passing and interactive live development that offers unique power and clarity that continues to influence modern software engineering. This book takes you from zero Smalltalk experience to advanced practitioner level, covering the language's history, its distinctive concepts, practical programming techniques and modern ecosystem tools. Whether you are a curious developer exploring object-oriented roots or a professional seeking deeper mastery, this book functions as both a structured learning resource and a long-term reference for serious Smalltalk development.

Introduction

In 1972, inside the hushed halls of Xerox PARC's Learning Research Group, a small team did something radical. They built an entire programming environment where every single value was an object, where every computation happened through message passing between those objects and where the tools for writing code lived in the same universe as the code itself. There were no files to compile, no separate build steps, no wall between editor and runtime. The system was alive, editable and responsive in ways that developers today still find astonishing.

That environment was Smalltalk. And it changed everything.

If you have programmed in Java, Python, Ruby, JavaScript, or any modern language with object-oriented features, you are standing on ground that Smalltalk prepared. The graphical user interface paradigm of windows, icons, menus and pointing devices emerged from the same lab. Model-View-Controller architecture originated there. Unit testing frameworks trace their lineage back to Smalltalk's SUnit. Even the concept of an integrated development environment as a cohesive workspace rather than a collection of disjointed tools comes directly from Smalltalk.

Yet for all its influence, Smalltalk itself remains something of a mystery to most programmers. You encounter traces of it everywhere Ruby's elegant object model, Python's dynamic capabilities, JavaScript's prototype system but the original is rarely seen firsthand. This is a missed opportunity. Learning Smalltalk does not merely add another language to your toolkit. It changes how you think about programming itself.

This book exists to close that gap.

What You Will Learn

By the time you finish this book, you will understand:

- The philosophical foundations of Smalltalk and why its designers made the choices they did

- How to navigate and work productively in a live Smalltalk environment
- Every aspect of Smalltalk syntax and semantics, from literals to message cascades
- The complete object model including classes, metaclasses and inheritance mechanics
- How blocks and closures work as first-class citizens powering control flow and iteration
- Smalltalk's elegant collection hierarchy and the patterns it enables
- Reflection and metaprogramming techniques that let you inspect and modify running systems
- Concurrency models, persistence strategies and testing practices specific to Smalltalk
- The architecture of the virtual machine, garbage collection and performance optimization
- How to structure real applications, manage dependencies and deploy Smalltalk software

Who This Book Is For

This book assumes you have programmed before. You should be comfortable with basic programming concepts variables, conditionals, loops, functions or methods. If you have used Java, Python, Ruby, JavaScript, C#, or similar languages, you are in the right place. The book does not assume any prior Smalltalk knowledge whatsoever.

At the same time, experienced Smalltalk developers will find value here as a reference and systematic treatment of topics that are often left implicit in community knowledge. The detailed explanations of the object model, method dispatch, metaclasses and VM architecture should serve as useful documentation even for veterans.

How to Read This Book

Read it in order. Each chapter builds on concepts introduced earlier and the progression is deliberate. Early chapters establish foundational mental models that later chapters depend on. You can certainly skip ahead to topics

of immediate interest, but you will get the most out of the book by following the sequence from start to finish.

Code examples throughout are written primarily for Pharo, the most actively developed modern Smalltalk implementation. Where behavior differs between implementations such as Squeak or GNU Smalltalk, I note those differences explicitly. All examples are complete and runnable you can type them into a workspace and see them work immediately.

A Note on Implementations

Smalltalk is not a single product but a family of related systems sharing common ancestry. The language core is remarkably stable across implementations, but the class libraries, tools and ecosystems vary considerably. This book uses Pharo as its primary reference implementation because it is open source, actively maintained, has excellent documentation and represents where modern Smalltalk development is headed. When I describe a feature as "Smalltalk behavior" without qualification, I mean behavior common to most major implementations. When something is specific to Pharo or another dialect, I say so explicitly.

Why This Matters Now

You might be wondering why you should learn a language created in the 1970s when there are dozens of modern alternatives. The answer is not that Smalltalk will replace your current language in production tomorrow. The answer is deeper.

Smalltalk forces you to confront fundamental questions about what programming languages and environments could be. It demonstrates that compilation barriers, file-based organization and separation between development tools and runtime are design choices, not necessities. Its purity of concept one model applied recursively until everything works uniformly offers clarity that hybrid, compromise-heavy languages cannot match.

Understanding Smalltalk makes you a better programmer in any language. You see through the accidents of syntax to the essential mechanics of computation. You appreciate design trade-offs more clearly. And occasionally, when

facing a particularly thorny problem, you find yourself thinking "how would Smalltalk handle this?" and the answer often illuminates a path forward even in languages far removed from Smalltalk's lineage.

Let us begin with the story of how all of this came to be.

Chapter 1: The Birth of a Revolution

The Xerox PARC Years (1970–1980)

In the middle of 1970, Xerox opened the door of its brand new research center in Palo Alto, California. The Xerox Palo Alto Research Center PARC, as it became known was established to explore future technologies for a computer subsidiary that Xerox had recently acquired. Among the groups assembled there was the Learning Research Group (LRG), led by Alan Kay, whose mandate was unusually ambitious: figure out how computers could transform education and human thinking itself.

Kay's vision began with a concept he called the Dynabook a portable, personal computer for children that would serve as a "thinking amplifier." This was 1970. Personal computers did not exist. The idea of a laptop was science fiction. Yet Kay insisted that whatever system they built to support this vision needed a new kind of programming language and environment, one that children could use directly and that embodied principles of clarity, consistency and power.

The team at PARC moved with remarkable speed. In just three months in 1972, they built the Xerox Alto, the first personal computer with a graphical user interface, bitmap display, mouse input and networking capability. But the hardware was only half the equation. The real revolution would be the software environment that ran on it.

Smalltalk emerged from this crucible as the programming language and integrated development environment for the Alto. It was not designed by committee or through incremental evolution of existing systems. It was conceived from first principles, driven by a coherent philosophy about how computation should work and how humans should interact with computers.

The core team included Alan Kay as visionary and intellectual leader, Dan Ingalls as primary implementer and virtual machine architect, Adele Goldberg who focused on educational applications and documentation, Ted Kaehler who developed the OOZE virtual memory system, Diana Merry who contributed to the graphics subsystem and Scott Wallace among others. This was a small,

deeply committed group working on something they genuinely believed would change the world.

Alan Kay's Vision: Objects as Living Things

To understand Smalltalk, you must understand Alan Kay's mental model. Kay held degrees in molecular biology and mathematics before entering computer science and his biological background profoundly shaped how he thought about software design.

Kay imagined large software systems growing like living bodies, composed of millions of cells. Each cell is a complete computational unit that protects its own internal state and communicates with neighboring cells only through messages. No cell reaches into another cell's internals to grab or modify data directly. Messages are the sole means of interaction. This design scales naturally biological organisms coordinate trillions of cells using exactly this principle, achieving complexity and robustness that traditional top-down software engineering struggles to match.

Kay articulated his vision in a famous formulation: "Object-oriented programming to me means only messaging, local retention and protection and hiding of state-process and extreme late-binding of all things." Notice what he emphasizes: messaging comes first, not objects or classes. State is local and protected. Binding the connection between a message and the code that handles it happens as late as possible, at runtime rather than compile time.

Kay has since expressed regret about coining the term "object-oriented," because it led many people to focus on what he considers the lesser aspect of his vision:

"I'm sorry that I long ago coined the term 'objects' for this topic because it gets many people to focus on the lesser idea. The big idea is messaging." [1]

This distinction matters. In mainstream object-oriented languages like Java or C++, objects often become little more than data structures with attached methods bundles of state and behavior that expose their internals through getter and setter methods. That approach, while useful, misses Kay's deeper insight about autonomous units communicating through well-defined interfaces. Smalltalk takes the messaging idea more seriously than almost any other language.

The influence on Smalltalk came from several sources. Simula 67, created by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center, introduced the concepts of classes and objects and served as a direct technical ancestor. Lisp contributed ideas about dynamic typing, garbage collection and interactive development environments. Logo influenced the educational philosophy and some syntactic choices. Ivan Sutherland's Sketchpad demonstrated the power of direct manipulation and graphical interaction. Smalltalk synthesized these influences into something new and coherent.

Smalltalk-72, -74 and -80: A Brief Evolution

Smalltalk did not appear in its final form overnight. It evolved through several major versions, each refining the language and environment while staying true to core principles. Understanding this evolution helps explain design decisions that might seem puzzling without historical context.

Smalltalk-72 was the first real implementation, created by Dan Ingalls running on the Alto. Kay has described how remarkably compact it was the original interpreter fit on a single page of code, written almost as a bet that such a language could be implemented so concisely. Smalltalk-72 introduced overlapping windows, an interactive workspace and the basic object/message model. However, its syntax and execution semantics were quite different from modern Smalltalk. Symbols were byte-coded, message sends and returns used symmetric conventions and there was no class inheritance hierarchy in the Simula sense. The system was elegant but limited in performance and scope.

Smalltalk-74 refined the language by reducing the number of primitive objects and operations, moving more functionality into the class library implemented in Smalltalk itself. This version introduced the OOZE (Object-Oriented Zapped Environment) virtual memory system developed by Ted Kaehler, which allowed the system to manage more objects than would fit in physical memory by swapping them between primary memory and disk. The BitBLT graphics routine, also developed at PARC, enabled fast bitmap operations essential for a responsive graphical interface. Smalltalk-74 was powerful enough to support serious development work but still felt experimental.

Smalltalk-76 represented a major architectural shift. To achieve better performance, the team adopted a Simula-like class inheritance model and froze certain execution semantics that had previously been more flexible. This

version included a development environment with most of the familiar tools: a class browser for navigating code, an object inspector for examining live objects, a debugger and workspace environments for interactive experimentation. The syntax began resembling modern Smalltalk, with selector names using colons to mark keyword arguments. Importantly, almost the entire system including the development tools themselves was now implemented in Smalltalk rather than in lower-level languages. This "bootstrapped" quality became one of Smalltalk's defining characteristics.

Smalltalk-80 was the mature, stable release that defined the language for decades to come. Version 2, released in 1983 as a general availability product, included the metaclass system that solidified the "everything is an object" principle even for classes themselves. The class library was comprehensive and well-designed, providing collections, streams, graphics primitives, user interface components using Model-View-Controller architecture and development tools. Smalltalk-80's influence spread rapidly after Steve Jobs visited PARC in 1979, saw the system and brought its ideas back to Apple, eventually influencing the Macintosh interface design.

The ANSI standard for Smalltalk (INCITS 319-1998), ratified in 1998, codified the language specification based on decades of implementation experience across multiple vendors. While modern implementations like Pharo have evolved beyond strict ANSI compliance in places, the standard remains a useful reference for understanding the core language.

The Influence on Modern Computing

Smalltalk's influence on computing is vast and often underappreciated because its ideas became so thoroughly absorbed into mainstream practice that we no longer recognize them as innovations. Let us trace some of the most significant impacts.

Graphical User Interfaces: The Alto running Smalltalk was the first computer system with a bitmap display, overlapping windows, icon-based file management, mouse-driven interaction and menu systems. These concepts now utterly ordinary were revolutionary in the 1970s. When Steve Jobs saw the Alto at PARC, he reportedly said "Now you must see it!" to his colleagues, recognizing immediately that this was the future of personal computing. The Macintosh, Windows and every modern desktop operating system owe an enormous debt to work done at PARC.

Object-Oriented Programming: While Simula introduced objects and classes, Smalltalk made object-oriented programming practical, coherent and attractive as a complete programming paradigm rather than an optional feature bolted onto a procedural language. Virtually every major object-oriented language that followed Objective-C, Java, Python, Ruby, C#, Swift, Dart, Scala drew inspiration from Smalltalk's design. Even languages that do not look like Smalltalk syntactically borrowed its ideas about dynamic typing, message passing, inheritance and polymorphism.

Ruby's creator Yukihiro Matsumoto has explicitly cited Smalltalk as a major influence on Ruby's design, particularly the idea that every piece of data is part of the object system and the emphasis on making programming enjoyable. Python's object model, while more restrained, shares Smalltalk's commitment to dynamic typing and runtime flexibility. Java adopted Smalltalk's class-based inheritance and single inheritance discipline while adding static typing for enterprise-scale development.

Integrated Development Environments: Before Smalltalk, programming typically involved editing text files in a separate editor, running a compiler from the command line, fixing errors by returning to the editor and repeating. Smalltalk unified all these activities into a single, consistent environment where code editing, compilation, debugging and testing happened seamlessly within the same system that ran your programs. This concept of an IDE as a cohesive workspace rather than a collection of tools became the standard expectation for modern development environments.

Live Programming: Smalltalk's ability to modify running code adding new methods, changing existing behavior, examining live objects while the system continued to operate was unprecedented. This "live programming" capability fundamentally changed the development experience, enabling rapid experimentation and immediate feedback that batch-oriented languages could not match. Modern tools like REPLs in Lisp and Clojure, Jupyter notebooks for Python and hot-reload features in web frameworks all echo Smalltalk's approach, though typically in more limited forms.

Unit Testing: Kent Beck developed SUnit, the first unit testing framework, in Smalltalk while working on Extreme Programming practices. SUnit's design test cases as subclasses of TestCase, assertion methods like assert:, setUp and tearDown lifecycle hooks became the template for JUnit, NUnit, pytest and essentially every xUnit-family testing framework used today. The very concept of automated unit testing as a standard development practice traces directly

to Smalltalk culture.

Design Patterns: The Gang of Four book "Design Patterns: Elements of Reusable Object-Oriented Software," which codified patterns like Observer, Strategy, Factory and Singleton, was written by developers working primarily in Smalltalk. Many of these patterns emerged naturally from experience building large Smalltalk systems and represent solutions to recurring design problems that the language's features made both necessary and tractable.

Why Learn Smalltalk Today?

Given all this influence, you might wonder: if Smalltalk's ideas live on in modern languages, why learn Smalltalk itself rather than just learning Ruby or Python or whatever language is currently popular?

There are several compelling reasons.

First, **purity of concept**. Modern mainstream languages are hybrids they mix paradigms, accumulate features over decades and make compromises for compatibility and market considerations. Java has objects but also static methods and primitive types that are not objects. Python has classes but also module-level functions and a distinction between value types and reference types. C# layers properties, events, delegates and attributes on top of basic object-oriented mechanics. Smalltalk, by contrast, applies its core model uniformly and consistently. Everything is an object. Every computation is message passing. There are no special cases, no escape hatches to procedural code, no separate namespaces for functions versus types. This purity makes it easier to understand the essential mechanics without distraction.

Second, **the environment matters**. Learning Smalltalk means learning not just a language syntax but a complete development philosophy. The interactive tools browser, inspector, debugger, workspace are not afterthoughts bolted onto the language. They are first-class components designed with the same care and consistency as the language itself. Experiencing this integrated approach changes how you think about what a programming environment can be.

Third, **historical understanding**. You cannot fully understand modern object-oriented languages without understanding where their ideas came from. Reading Smalltalk code especially the original class library is like reading

foundational texts in any discipline. It reveals design decisions and trade-offs that are obscured when you only encounter their descendants.

Fourth, **practical utility**. Smalltalk is not a museum piece. Pharo and other modern implementations are actively developed, with sophisticated web frameworks (Seaside), data analysis tools (Moose, Roassal), database integration (Glorp) and deployment options for production systems. Some financial institutions and enterprises still run critical systems in commercial Smalltalk implementations like Cincom VisualWorks or GemStone/S. Learning Smalltalk can open doors to maintaining and developing these systems.

Fifth, **intellectual enrichment**. Even if you never write production code in Smalltalk again after learning it, the experience will make you a better programmer. You will see through syntactic accidents to essential structures. You will appreciate design consistency more deeply. And when you encounter problems that resist conventional approaches, having Smalltalk in your mental toolkit may suggest solutions you would not have considered otherwise.

The journey ahead covers all of this in detail. We start with getting a working environment installed and running, then systematically explore every aspect of the language and ecosystem. By the end, you will not just know how to write Smalltalk code you will understand why Smalltalk was designed the way it was, what benefits that design provides, what trade-offs it entails and how its ideas continue to shape software engineering today.

Chapter Summary: Smalltalk emerged from Xerox PARC in the 1970s as a radical vision of computing: objects communicating through messages in an integrated, interactive environment. Its evolution through versions -72, -74, -76 and -80 refined this vision into a practical system whose influence extends to virtually every aspect of modern software development from GUIs and OOP languages to IDEs and unit testing frameworks. Learning Smalltalk today offers both practical skills and deeper understanding of fundamental programming concepts.

Chapter 2: Your First Steps in Smalltalk

Choosing an Implementation (Pharo, Squeak, GNU Smalltalk)

Before writing any code, you need to choose which Smalltalk implementation to use. This choice matters less for learning the core language all major implementations share essentially the same syntax and semantics but it affects your development experience, available libraries, tooling quality and community support.

Pharo is the most actively developed modern Smalltalk and the one this book uses as its primary reference. Pharo began as a fork of Squeak version 3.9 in March 2008, created by developers who wanted a leaner, more professional-focused environment while preserving Smalltalk's core qualities. As of this writing, the latest stable release is Pharo 13 (released May 2025). Pharo runs on the Cog virtual machine with just-in-time compilation and the Spur memory manager for efficient 64-bit operation. It has strong community support, excellent documentation including the "Pharo by Example" book and a rich ecosystem of frameworks for web development (Seaside), data analysis (Moose), visualization (Roassal) and more. If you are new to Smalltalk and want the most modern experience with the largest active community, Pharo is the recommended choice.

Squeak is the open-source implementation that descends directly from the original Smalltalk-80 system developed at Xerox PARC. It has a strong emphasis on education and creative exploration, including Etoys an environment designed for children to create interactive multimedia projects. Squeak's class library retains more of the original Smalltalk heritage and includes educational tools that Pharo streamlined away. The community is smaller than Pharo's but still active, particularly in educational contexts. If you are interested in the historical lineage of Smalltalk or working on educational projects, Squeak is worth considering.

GNU Smalltalk takes a different approach entirely. Rather than providing a graphical environment with image-based development like Pharo and Squeak,

GNU Smalltalk operates primarily from the command line with file-based source code, closer to how most developers work in languages like Python or Ruby. This makes it more familiar to developers uncomfortable with image-based systems and easier to integrate into standard build pipelines and version control workflows. However, it lacks the rich interactive tools that define the Smalltalk experience and has a smaller ecosystem of libraries. GNU Smalltalk is useful if you want to experiment with Smalltalk syntax and semantics without committing to the full environment model, or if you need command-line scripting capabilities.

For this book, all code examples are written for Pharo unless explicitly noted otherwise. The core language concepts objects, messages, classes, blocks, inheritance are identical across implementations. Differences appear mainly in class library details, tooling and deployment mechanisms, which I highlight where relevant.

Installing and Launching Your Environment

Getting Pharo installed is straightforward. The recommended approach is to use the Pharo Launcher, a small application that manages your Pharo images and automatically downloads the appropriate virtual machine for your platform.

Visit https://pharo.org/download and select the launcher package for your operating system (Windows, macOS, or Linux). On Linux systems, you can also install via package managers or use Zeroconf scripts running `curl -L https://get.pharo.org | bash` in a terminal will automatically download and set up everything needed.

The Pharo Launcher is itself a Smalltalk image. When you run it, you see a simple interface listing available images you can download. Select a stable Pharo release (the launcher typically shows options like “Pharo 13” or “Pharo Stable”) and the launcher will download both the virtual machine executable and the image file needed to run it. This may take a few minutes depending on your connection speed.

Once downloaded, launch the image from within the launcher. The first time you open a fresh Pharo image, you will see a desktop environment that looks somewhat retro reminiscent of early graphical systems but fully functional and surprisingly modern underneath. Do not be intimidated by

the appearance. This is a complete programming environment containing thousands of classes, millions of lines of code, all running in memory and ready for interaction.

Pharo does not require installation in the traditional sense. It runs standalone from wherever you extracted or downloaded it. The entire system virtual machine, class library, development tools and your code lives in a single image file (typically named something like `Pharo13.image`). This image-based approach is fundamental to Smalltalk and deserves careful explanation.

The Smalltalk Desktop: Overview of Tools

When Pharo launches, you see a desktop with several key tools available through menus or keyboard shortcuts. Understanding these tools is essential because they are how you interact with the system throughout your development work.

The Workspace is your interactive playground a place where you can type and evaluate Smalltalk expressions immediately, inspect results, experiment with ideas and prototype code before committing it to permanent methods. Think of it as a combination of a REPL (read-eval-print loop), scratchpad and documentation area. You open a workspace from the World menu (accessed by right-clicking on the desktop background) selecting "Open It" then "Workspace," or via keyboard shortcut.

The System Browser is where you navigate, read, write and organize the source code of classes and methods. Unlike file-based editors that work with text files on disk, the browser works directly with the live class definitions in memory. When you edit a method in the browser, it compiles immediately into the running system. You can open the browser from the World menu or via keyboard shortcut. The browser displays packages (groupings of related classes) on the left, classes in the middle pane and individual methods organized by category on the right.

The Object Inspector lets you examine any live object in the system its class, instance variables, their current values and the messages it understands. If an instance variable holds another object, clicking on it opens a nested inspector for that object. The inspector is invaluable for understanding how objects are structured during debugging and exploration. You can open an inspector on any expression result by selecting "Inspect It" from context menus.

The Debugger activates automatically when your code encounters an error an undelivered message, arithmetic overflow, or explicit halt. Rather than crashing or printing a stack trace to a terminal, the debugger opens a window showing the call stack, local variables at each level and the source code where execution stopped. Crucially, you can interact with objects in the debugger, evaluate expressions, modify code and resume execution turning error situations into learning opportunities rather than frustrating dead ends.

The Transcript is a simple output window where code can send text messages using `Transcript show: 'text'; cr`. It serves as a basic logging and debugging output mechanism, similar to console.log in JavaScript or print in Python.

These tools are not separate applications communicating through files or network connections. They all run inside the same Smalltalk image, operating on the same live objects. This integration is what makes Smalltalk development feel so immediate and cohesive compared to file-based environments.

The Workspace: Your Interactive Playground

Let us start using the workspace immediately. Open one now if you have not already and let us try some basic expressions.

In the workspace, type the following and select it with your mouse (highlight it), then right-click and choose “Do It” or press Ctrl-D:

```
5 + 3
```

You should see the result `8` appear below the expression. What happened? The workspace sent the message `+` to the number object `5`, with `3` as the argument. The Integer class knows how to handle the `+` message by performing addition and returning the result. This is message passing in its simplest form we will explore it in much greater depth later.

Try a few more expressions, selecting and evaluating each one:

```
'Hello, World!'
```

This returns the string itself `'Hello, World!'`. In Smalltalk, every expression evaluates to a value, even if that value is just the literal you typed.

```
'Hello' , ' ' , 'World'
```

The comma operator (`,`) concatenates strings. This returns `'Hello World'`. Notice that `,` is itself a message send specifically a binary message sent from `'Hello'` to `' '` and then from the result to `'World'`.

```
(1 to: 10) collect: [:each | each * 2]
```

This creates a collection of numbers from 1 to 10, then transforms each element by multiplying it by 2. The result is `(an OrderedCollection[2 4 6 8 10 12 14 16 18 20])`. The bracket syntax `[:each | each * 2]` defines a block Smalltalk's version of an anonymous function or closure which we will study in detail in Chapter 8. For now, just observe how naturally this expression reads: "take numbers from 1 to 10 and for each one, compute each times 2."

You can also assign values to temporary variables within workspace evaluations:

```
| name greeting |
name := 'Smalltalk'.
greeting := 'Hello, ', name, '!'.
greeting
```

The vertical bars `| |` declare temporary variables. The assignment operator `:=` binds a value to a variable name. The last expression in the selection determines what value the workspace displays as the result in this case, `'Hello, Smalltalk!'`.

The workspace is also useful for exploring objects that already exist in the system. Try:

```
Date today
```

This sends the `today` message to the Date class, which returns today's date as a Date object. You can then send messages to that result:

```
(Date today) dayOfWeekName
```

This might return `'Monday'`, `'Tuesday'`, or whatever day it is when you evaluate it. The parentheses ensure that `today` is sent first and then `dayOfWeekName` is sent to the resulting Date object.

As you experiment in the workspace, use "Inspect It" (right-click on a result) to examine objects in detail. Inspect the result of `Date today` and you will see instance variables holding the year, month and day values. Click on any of those values to inspect them further. This exploratory approach type something, see what happens, inspect the result, try another message is fundamental to Smalltalk development.

Image-Based Development vs File-Based Development

Now we must address a concept that will feel unfamiliar if you have only programmed in file-based languages like Java, Python, or JavaScript: image-based development.

In conventional file-based environments, your program exists as text files on disk. To run it, you load those files into memory, compile them (if necessary) and execute the resulting code. When the program terminates, the in-memory state disappears. The next time you run it, everything starts fresh from the files on disk. Your development tools editor, compiler, debugger are separate programs that read and write these files but do not share memory with your running application.

Smalltalk works differently. A Smalltalk **image** is a complete snapshot of the system's memory at a point in time. This snapshot includes:

- All class definitions (the blueprints for creating objects)
- All methods (the behavior associated with each class)
- All global objects (system-level state, including development tools)
- Any other objects that exist in the system at save time

When you launch Pharo, you are not compiling source files and starting from scratch. You are restoring a previously saved memory snapshot essentially resuming execution from where you left off. If you had windows open, objects under inspection, or computations in progress when you saved the image, all of that state is restored when you load the image again.

The image file (typically with extension `.image`) is a binary representation of this memory snapshot. Alongside it, there is usually a companion `.changes` file that records modifications made since the last save new or changed methods, class definitions and other structural changes. When you save the image, these changes are incorporated into the main image file.

This approach has profound implications for how you work:

Immediate feedback: When you edit a method in the browser, it compiles instantly into the running system. You do not need to save files, run a build command, restart your application and navigate back to where you were. The change takes effect immediately and you can test it right away against live objects.

Live objects: Objects created during development persist as long as you keep the image running. You can create an object, modify its class's methods and see the new behavior affect that existing object without recreating it. This enables a style of exploratory programming where you develop code and data together incrementally.

No compilation barrier: There is no separate compile step because methods are compiled individually as you edit them, directly into the running system. The workspace provides immediate evaluation of expressions. This tight feedback loop accelerates learning and experimentation dramatically.

Stateful development sessions: You can save your image in the middle of a debugging session, with breakpoints set, variables inspected and windows open. When you restart, you resume exactly where you left off. This is particularly valuable for investigating complex bugs or long-running processes.

However, image-based development also presents challenges that file-based developers initially find uncomfortable:

Version control integration: Traditional version control systems like Git work with text files, not binary memory snapshots. While tools exist to extract source code from images and manage it in Git (such as FileTree for Monticello packages), the workflow is less straightforward than committing text files directly. Modern solutions like Iceberg integrate Git more seamlessly into Pharo.

Reproducibility: Starting a project from a shared image means everyone works with the same baseline state. But if someone modifies their local image extensively without properly managing those changes, recreating that exact

state elsewhere can be difficult. Good practice involves regularly saving clean baseline images and using package management tools to track code changes separately.

Image size and startup time: As you add code and data to an image, it grows. Large images take longer to save and load. Production deployments often use stripped-down images containing only what is necessary, separate from development images with full tooling.

Neither approach is universally superior they represent different trade-offs. File-based systems excel at integration with standard toolchains, distributed collaboration via version control and clear separation between source and runtime. Image-based systems excel at interactive exploration, immediate feedback and cohesive developer experience. Understanding both perspectives helps you appreciate why Smalltalk works the way it does and how to work effectively within its model.

Running Your First Code: Hello World and Beyond

Let us write our first "proper" Smalltalk code not just workspace expressions but a defined class with methods. This demonstrates how the System Browser works and establishes patterns you will use throughout your Smalltalk journey.

Open the System Browser (World menu > Open It > System Browser, or use the keyboard shortcut). You will see several panes:

- The left pane shows packages logical groupings of related classes
- The middle pane shows classes within the selected package
- The right pane shows method categories and individual methods for the selected class
- At the bottom is an editor area where you write code

For our first example, let us create a simple class. In the browser, navigate to the "Kernel" package (or create a new package if your environment prompts you). With no class selected, look for a menu option to add a new class often accessible via right-click or an "Add Class" button.

Enter the following class definition:

```
Object subclass: #GreetingBot
    instanceVariableNames: 'name'
    classVariableNames: ''
    package: 'MyFirstPackage'
```

This declares a new class called `GreetingBot` that inherits from `Object` (the root of the class hierarchy in most Smalltalk systems). It has one instance variable called `name`, no class variables and belongs to a package called `MyFirstPackage`. The browser compiles this definition immediately.

Now let us add methods to our class. In the browser with `GreetingBot` selected, you will see method categories like "instance creation" and "accessing." Let us add an instance creation method that allows creating a GreetingBot with a name:

In the "instance creation" category (or create it if needed), write:

```
name: aName
    "Create a new GreetingBot with the given name."
    ^self new setName: aName
```

This is a class method defined in the class side of GreetingBot that creates a new instance and initializes its name. The `^` symbol indicates the return value. We will explore method structure in detail in Chapter 6.

Now add instance methods for accessing and using the name:

In the "accessing" category:

```
setName: aName
    "Set the bot's name."
    name := aName

name
    "Answer the bot's current name."
    ^name
```

And in a "printing" or "actions" category:

```
greet
    "Print a greeting message."
    Transcript show: 'Hello! I am ', name, '.'; cr

greet: someone
    "Greet someone by name."
    Transcript show: 'Hello, ', someone, '! I am ', name, '.'; cr
```

Now return to the workspace and try using your new class:

```
| bot |
bot := GreetingBot name: 'Alice'.
bot greet.
bot greet: 'Bob'.
bot name
```

You should see messages appear in the Transcript window: "Hello! I am Alice." followed by "Hello, Bob! I am Alice." The last expression returns `'Alice'` as the workspace result.

What just happened? You defined a class, added methods to it, created an instance and sent messages to that instance all within a single live session. The class definition and methods are now part of the running system. If you save your image and restart, they will still be there.

This simple example demonstrates several fundamental Smalltalk patterns:

- Classes inherit from existing classes (here, Object)
- Instance variables hold per-object state
- Methods define behavior associated with a class
- The caret ^ returns a value from a method
- Messages are sent to objects using dot notation (receiver message argument)

We will unpack each of these concepts thoroughly in the chapters ahead. For now, celebrate the fact that you have written and run your first Smalltalk class.

Chapter Summary: Getting started with Smalltalk means choosing an implementation (Pharo recommended for new learners), installing it via the Pharo Launcher or equivalent and learning to navigate the integrated development environment. The workspace provides immediate interactive evaluation, while the browser manages class definitions and methods. Most importantly, Smalltalk's image-based development model where the entire system state lives in a resumable memory snapshot rather than separate source files fundamentally shapes the development experience with benefits (immediate feedback, live objects) and trade-offs (version control integration, reproducibility) that differ from conventional file-based environments.

Chapter 3: Everything Is an Object

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Objects, Classes and Instances Explained

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Message Passing as Computation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Hierarchy: Object at the Root

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Literals Are Objects Too (Numbers, Strings, Characters)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Nil, True, False: Special Objects with Power

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 4: Syntax and Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Variables: Temporal, Instance, Class and Global

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Assignment and Binding Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Expression Evaluation Rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Comments and Documentation Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Whitespace, Line Breaks and Readability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Common Syntax Mistakes and How to Avoid Them

Chapter 5: Messages The Heart of Smalltalk

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Unary Messages: No Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Binary Messages: Operators as Messages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Keyword Messages: Named Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Message Cascades: Fluent Style

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Precedence and Evaluation Order

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Undeliverable Messages and Error Handling Basics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 6: Classes, Methods and Encapsulation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Defining a Class: Syntax and Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Instance Variables and Accessors (Getters/Setters)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Methods: Defining Behavior

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Self and Super: Identity and Delegation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Class Methods vs Instance Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Encapsulation Without Access Modifiers

Chapter 7: Inheritance and Polymorphism

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Inheritance Hierarchy in Detail

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Method Lookup and Dispatch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Overriding and Extending Behavior

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Polymorphism: Sending Messages Without Knowing Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Abstract Classes and Interfaces (Protocols)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Composition vs Inheritance in Smalltalk

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 8: Blocks and Closures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Block Syntax and Literals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Blocks as First-Class Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Lexical Scoping and Closure Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Return Behavior: Block Return vs Method Return

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Blocks in Control Structures (ifTrue:, whileTrue:)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Practical Patterns with Blocks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 9: Collections and Iteration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Collection Hierarchy Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

OrderedCollections and Arrays

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Sets and Bags

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Dictionaries and IdentityDictionaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Iteration with Blocks (do:, select:, reject:, inject:)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 10: Control Flow, Exceptions and Robustness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Conditional Execution (ifTrue:, ifFalse:, ifTrue:ifFalse:)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Loops Without Keywords (whileTrue:, to:do:)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Exception Handling Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Resuming vs Returning from Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Assertions and Invariants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Defensive Programming Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 11: The Smalltalk Environment in Depth

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The System Browser: Navigating Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Inspector: Examining Live Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Debugger: Interactive Error Recovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Workspace and Transcript

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Refactoring Tools and Live Modification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Why This Matters: The Smalltalk Way of Working

Chapter 12: Reflection, Metaprogramming and the Object Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Object Model: Classes Are Objects Too

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Metaclasses Explained

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Method Dictionaries and Lookup Chains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Runtime Introspection (species, class, respondsTo:)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Dynamic Code Modification (compile:, removeSelector:)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Build Your Own Domain-Specific Languages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 13: Testing and Quality Assurance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Built-In Testing Framework (TestCase, assert:)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Writing Your First Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Test Organization and Naming Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Mocking and Stubbing Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Continuous Integration with Smalltalk

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Refactoring Safely with Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 14: Persistence and Storage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Image-Based Persistence: The Default Approach

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Object Serialization (FileStream, BinaryOrTextStream)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Database Integration with SQL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Glorp: An ORM for Smalltalk

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Designing for Persistence from the Start

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 15: Concurrency and Parallelism

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Processes and Scheduling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Creating and Managing Threads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Synchronization: Semaphores, Mutexes, CriticalSections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Shared State and Thread Safety

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Message Passing for Concurrency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Performance Considerations in Concurrent Code

Chapter 16: Performance, Optimization and the Virtual Machine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Smalltalk Virtual Machine Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Compilation Model: Source to Bytecode

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Garbage Collection Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Primitives and Native Code Interfacing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Foreign Function Interface (FFI)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Profiling and Performance Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 17: Application Architecture and Design Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Packages and Code Organization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Model-View-Controller in Smalltalk

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Dependency Management and Monticello

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Design Patterns in Smalltalk Style

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Building Reusable Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Large-Scale Application Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Chapter 18: Deployment, Ecosystem and the Future

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Deploying Smalltalk Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Web Development (Seaside, Zinc)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Headless and Server-Side Smalltalk

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

The Modern Ecosystem Landscape

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Learning Resources and Community

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Smalltalk's Enduring Legacy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

Conclusion: The Smalltalk Mindset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theworldofsmalltalk.