LEARNHUB / TECHNICAL COURSE LIBRARY

Java Programming

Course introduction

Learn Java from its runtime model and core syntax through object-oriented design, interfaces, and the exception-handling patterns used in real programs.



Level: Beginner · Category: Programming

Original, beginner-friendly study content designed to accompany the course syllabus and its linked video lesson.

Contents

- How to use this resource
- Learning map
- Chapter 01 · The Java platform and types
- Chapter 02 · Control flow and methods
- Chapter 03 · Classes, objects, and encapsulation
- Chapter 04 · Inheritance and interfaces
- Chapter 05 · Exceptions and safe failure
- Summary and next steps

Course purpose: Learn Java from its runtime model and core syntax through object-oriented design, interfaces, and the exception-handling patterns used in real programs.

How to use this resource

Read the chapter titles first, then study the explanations and examples. Pause at each practice prompt before reading its suggested answer. Write down questions that remain and revisit them during revision.

Learning map



Chapter 01 · The Java platform and types

The central idea

Java source is compiled to bytecode, which a Java Virtual Machine executes. A type defines which values a variable may hold and which operations are valid.

How to think about it

Use primitive types for simple values and reference types for objects. Declare variables with meaningful names, use operators intentionally, and understand that expressions have types and precedence. The compiler catches many type mismatches before execution.

First example

A fee estimator reads an integer day count and a decimal daily rate, then computes a total using compatible numeric types.

Try this

Declare fields for a book title, page count, and price; choose appropriate types and explain why.

A way to check your answer

> Use `String` for the title, `int` for a whole page count, and `double` or `BigDecimal` for price depending on precision needs. Currency systems should avoid careless binary floating-point rounding.

Chapter 02 · Control flow and methods

The central idea

Control flow determines which statements execute. A method packages behavior under a name and can accept parameters and return a value.

How to think about it

Use `if/else` for decisions and `for` or `while` for repetition. Methods should have focused responsibilities. Distinguish a return value from printing: returned results can be reused and tested by other code.

First example

A method `isWeekend(day)` returns whether a day code belongs to the weekend; a caller decides what message to show.

Try this

Design a method that finds the largest of three integers. Include how ties are handled.

A way to check your answer

Accept three `int` parameters, compare them with `Math.max` or explicit branches, and return the largest value. Equal inputs naturally produce the same maximum.

Chapter 03 · Classes, objects, and encapsulation

The central idea

A class defines fields and methods; an object is an instance of that class. Encapsulation keeps state private and exposes controlled operations.

How to think about it

Use constructors to create valid objects. Make fields private and provide methods that enforce rules rather than allowing arbitrary state changes. Separate class responsibilities so each type has a clear purpose.

First example

A `TemperatureReading` object stores a value and unit and converts through a method that checks the unit before calculation.

Try this

Model a library book with a private checkout flag and methods for borrowing and returning it.

A way to check your answer

A `Book` class can keep `checkedOut` private. `borrow()` should fail or return false if it is already checked out; `returnBook()` restores availability.

Chapter 04 · Inheritance and interfaces

The central idea

Inheritance creates a specialized class from a base class. An interface specifies behavior that multiple unrelated classes can promise to implement.

How to think about it

Use inheritance when a true “is-a” relationship shares stable behavior. Favor interfaces for capabilities and use composition when behavior can be assembled from separate parts. Override methods deliberately and use `@Override` to catch signature mistakes.

First example

A payment flow can accept a `Payable` interface implemented by card and invoice payment classes.

Try this

Choose whether `Square` should extend `Rectangle` or implement a `Drawable` interface, and justify the choice.

A way to check your answer

> A `Drawable` interface is useful when shapes share the ability to draw, regardless of their place in the class hierarchy. Use inheritance only if the subtype preserves the base type's behavioral expectations.

Chapter 05 · Exceptions and safe failure

The central idea

An exception signals that normal control flow cannot complete an operation as expected. Checked exceptions must be declared or handled by the caller.

How to think about it

Catch exceptions where the program can recover or add useful context. Do not catch broadly and silently. Use `try-with-resources` for objects such as files that must be closed, even when an error occurs.

First example

A configuration reader reports a missing file clearly while still ensuring the file handle closes after a successful read.

Try this

Plan how a numeric input routine should handle malformed text and an out-of-range value.

A way to check your answer

> Catch parsing failure to ask for valid input; validate the range separately and explain that rule. Avoid treating both cases as the same opaque error.

Summary and next steps

Java Programming brings together 5 connected topics. Review each definition, explain one example without looking, and then try the practice prompt again.

Key ideas to carry forward

The Java platform and types: Java source is compiled to bytecode, which a Java Virtual Machine executes. A type defines which values a variable may hold and which operations are valid.

Control flow and methods: Control flow determines which statements execute. A method packages behavior under a name and can accept parameters and return a value.

Classes, objects, and encapsulation: A class defines fields and methods; an object is an instance of that class. Encapsulation keeps state private and exposes controlled operations.

Inheritance and interfaces: Inheritance creates a specialized class from a base class. An interface specifies behavior that multiple unrelated classes can promise to implement.

Exceptions and safe failure: An exception signals that normal control flow cannot complete an operation as expected. Checked exceptions must be declared or handled by the caller.

A short review routine

1. Recall the main idea from each chapter before rereading. 2. Work one example by hand. 3. Explain a common mistake and how you would catch it. 4. Revisit a difficult topic after a short break.

Course outcomes

- Explain how Java source is compiled and run
- Use variables, types, operators, and control flow
- Model data with classes, objects, inheritance, and interfaces
- Handle exceptional conditions with explicit recovery paths

This course introduction is an original LearnHub learning aid. Use the referenced ideas for study, then consult the course video and the rest of the library for additional practice.