Sistemas Distribuídos

About Java and Programming Methodologies

Eurico Pedrosa

António Rui Borges

Universidade de Aveiro - DETI

2025-02-18

Summary

- A brief history and key features of Java.
- Methods to describe and solve programming problems.
- Different programming approaches, including procedural, modular, object-oriented, concurrent, and distributed programming.

Java Overview

Historical Background

- Java was originally named **Oak** and created in the 1990s by **James Gosling** and a team at **Sun Microsystems**.
- It was first designed for **embedded systems** in consumer electronics.
- Java became widely popular for **Internet applications**, starting with **applets** (small programs running in browsers) and later **web services**.
- It is a common choice for **distributed applications** due to its built-in **network communication support**.

Key Objectives

Java was designed to:

- **Be independent of hardware and operating systems** → Runs on a **Java Virtual Machine (JVM)**.
- **Be robust and reduce programming errors** → Eliminates **multiple inheritance** and **operator overloading** to avoid complexity.
- **Ensure security** → **No pointers** (to prevent unauthorized memory access) and built-in **garbage collection**.

Main Features

- **Object-oriented programming** → Uses **classes, inheritance, and interfaces**.
- **Built-in support for concurrency** → **Threads and synchronization mechanisms**.
- **Distributed computing support** → Communication using **Sockets** (message passing) and **Remote Method Invocation (RMI)** (shared memory approach).

Main Features

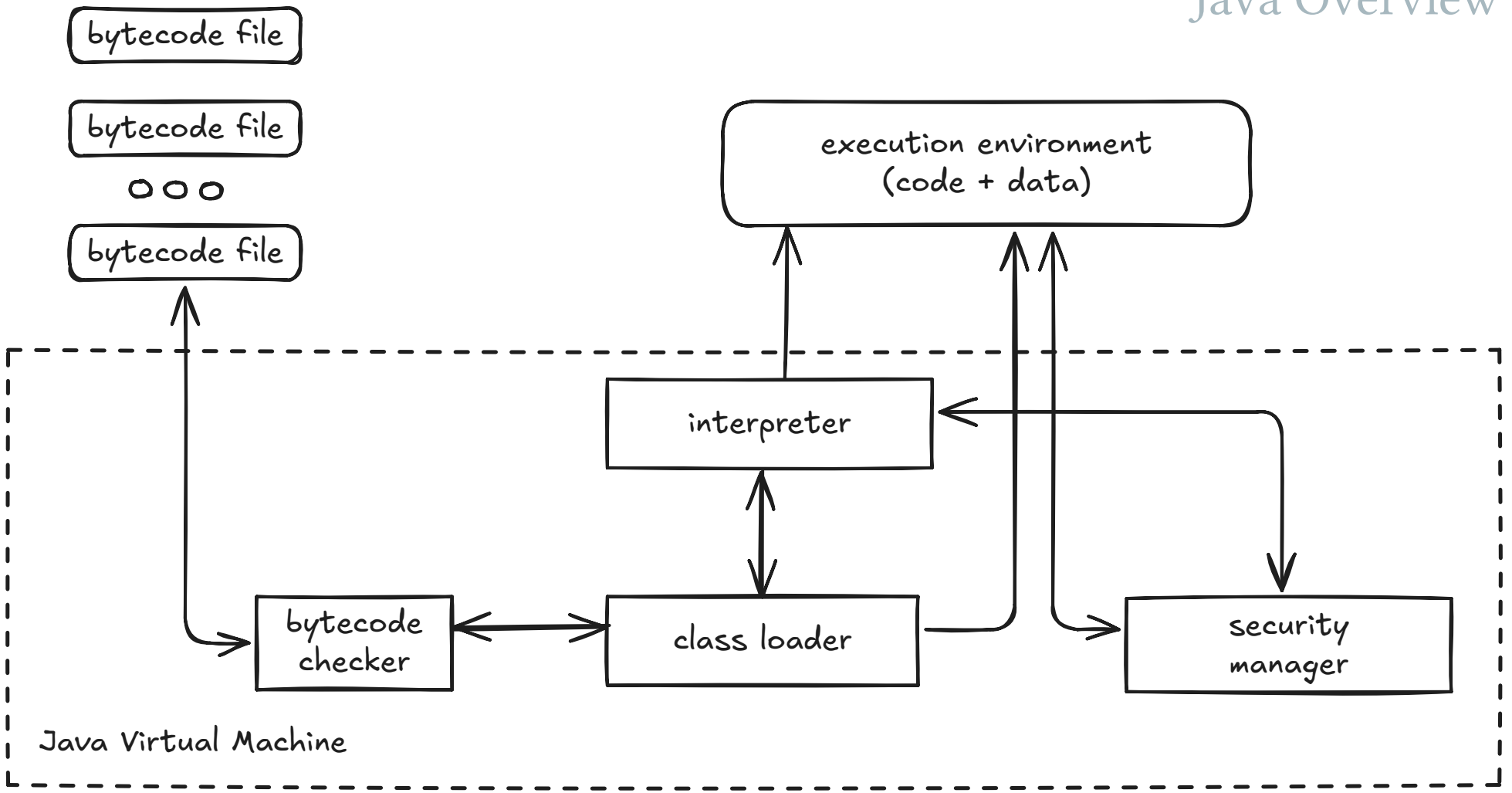
- **Internationalization support** → Uses **Unicode** instead of ASCII for global language support.
- **Automatic documentation generation** → Uses `javadoc` to create structured documentation from source code.
- **Platform independence** → Runs on a JVM, which has a **security model** to protect against harmful code.

Java Virtual Machine (JVM)

The JVM is an execution environment that allows Java programs to run independently of the underlying hardware and operating system.

- Java applications run on any system with a **JVM** making it ideal for distributed computing.
- **Security** → Implements a three-layered security model to protect against untrusted code:
  - **Bytecode Checker** → Verifies Java bytecode to prevent unsafe operations.
  - **Class Loader** → Dynamically loads required classes for execution.
  - **Security Manager** → Controls access to system resources (files, network, external processes).

Java Virtual Machine (JVM)



Example Java Program

A simple Java program that prints a greeting:

```
public class Hello {
    public static void main (String [] args) {
        System.out.println ("Hello " + args[0] + ", how are
you?");
    }
}
```

Execution:

```
javac Hello.java
java Hello Pedro
```

Output: `Hello Pedro, how are you?`

Problem Solving Approaches

Problem Solving Approaches

Hierarchical Decomposition

- Break a problem into smaller steps.
- Use structured **data types and operations**.
- Establish **clear relationships between data and functions**.

Interactive Autonomous Structures

- Define **clear interaction rules** between components.
- **Separate interface and implementation** for modularity.

Interactive Autonomous Entities

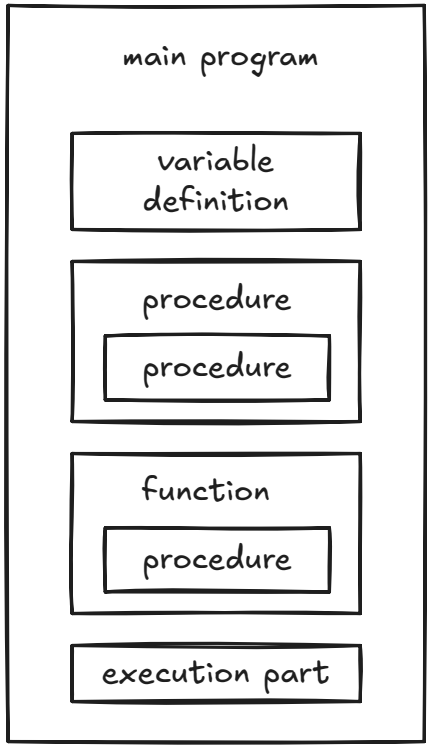
- Specify **communication models**.
- Define **synchronization mechanisms** to manage interactions.

Programming Methodologies

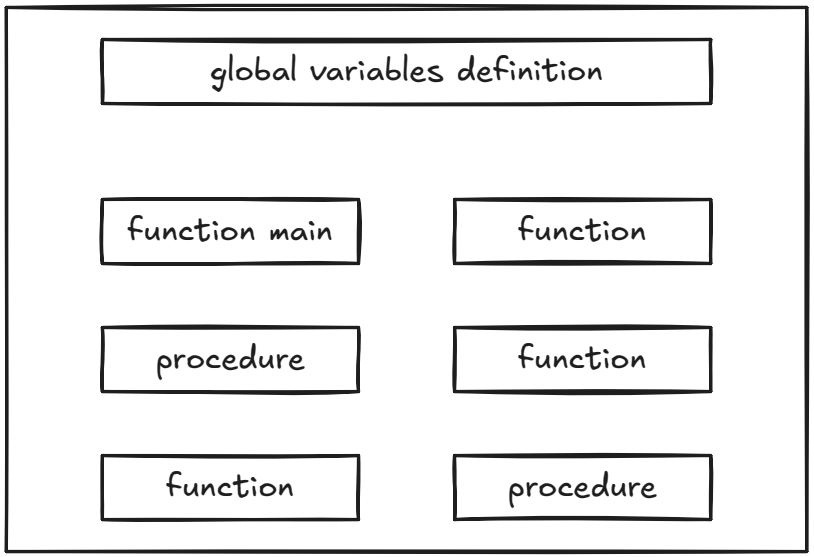
Procedural (Imperative) Programming

- Focuses on **step-by-step instructions** using functions and procedures.
- Uses **global variables** for data management.
- Communication happens through **parameter passing**.
- Examples: C, Pascal.

Procedural (Imperative) Programming



hierarchical organization of a source file – Pascal like

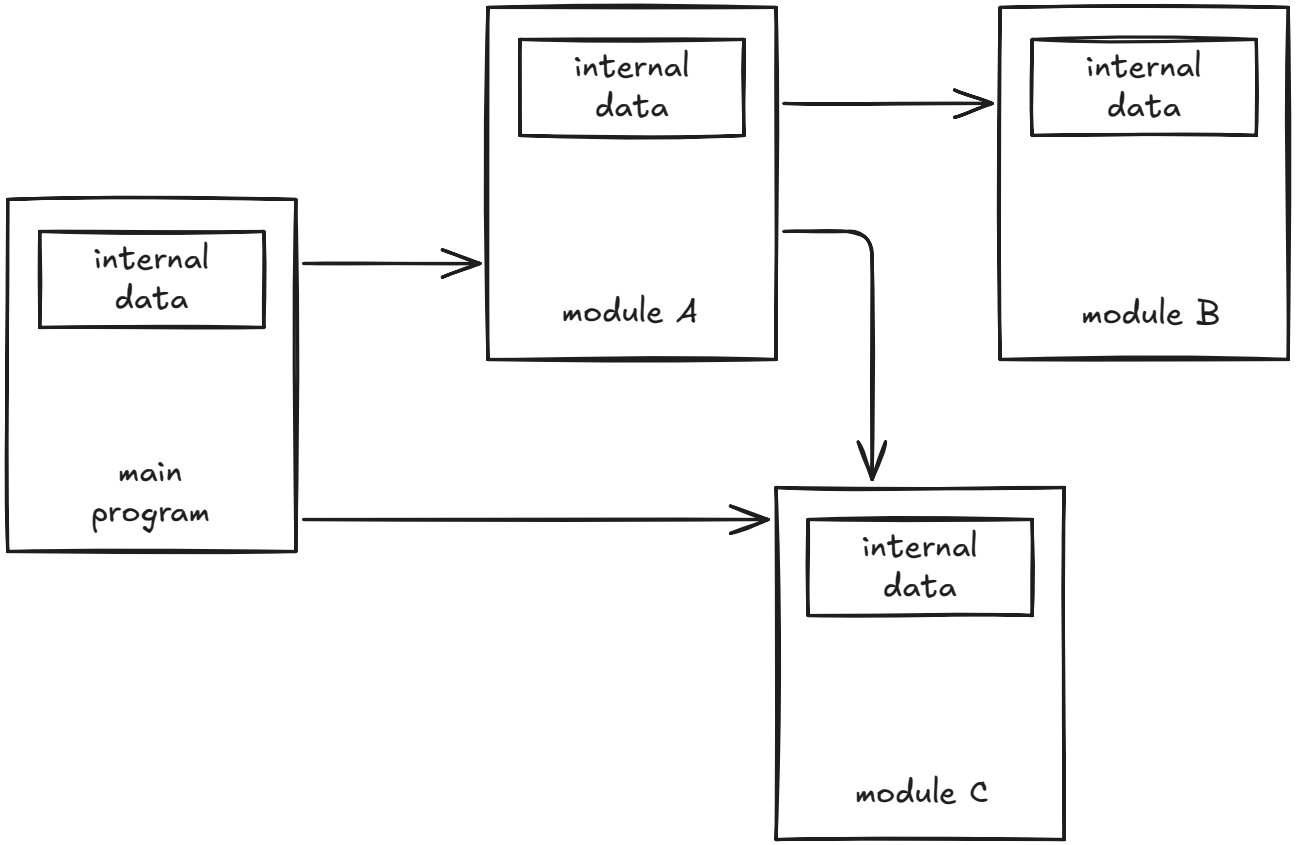


horizontal organization of a source file – C language like (sea of functions structure)

Modular Programming

- Code is divided into **independent modules**.
- Each module has its **own data space** and **access functions**.
- Improves **code reuse** and **maintainability**.
- Used for **libraries and well-structured applications**.

Modular Programming



Object-Oriented Programming (OOP)

- Programs are structured using **objects** (which combine data and behavior).
- Uses **classes** as **blueprints for objects**.
- Supports:
  - **Encapsulation** → Hides internal data.
  - **Inheritance** → Extends functionality from a parent class.
  - **Polymorphism** → Uses a common interface for different data types.

Object-Oriented Programming (OOP)

Example:

```
class Car {
    String model;
    void drive() {
        System.out.println("Driving the " + model);
    }
}
```

Concurrent Programming

- Runs **multiple processes at the same time**.
- Two main approaches:
  - **Event-driven** → Processes wait for an event (e.g., GUI applications).
  - **Peer-to-peer** → Independent processes work together (e.g., multi-threading).
- **Two communication models:**
  - **Shared variables** → Requires **synchronization (mutex, locks)**.
  - **Message passing** → Uses **channels for communication (sockets, IPC)**.

Concurrent Programming

Example using Java Threads:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running.");
    }
}

public class Example {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

Concurrent Programming

- A program following this methodology adheres to **modular** or **object-oriented programming** principles, structuring interactions across **multiple source files**.
- The key distinction is the presence of **multiple execution threads**, requiring a **clear separation** between **active entities** (intervening processes) and **passive entities** (explicit functional components).

Distributed Programming

- Expands concurrent programming across **multiple machines**.
- Requires:
  - **Efficient process distribution**.
  - **Handling failures in networks**.
- **Communication methods**:
  - **Sockets** (low-level, direct data exchange).
  - **Remote Method Invocation (RMI)** (calling methods remotely like local methods).
  - **Middleware** (e.g., Java EE, CORBA, gRPC).

Suggested Reading

- **Java Official Documentation**
  - from Oracle (Java Platform Standard Edition).