OOP using JAVA

INTRODUCTION

Why Java Programming Language?

- Simple
- Safe
- Platform-independent ("write once, run anywhere")
- Rich library
- Designed for the internet

Little History of Java Programming Language

- Java
  - Based on C and C++
  - Originally developed in early 1991 for intelligent consumer electronic devices
    - Market did not develop, project in danger of being cancelled
  - Internet exploded in 1993, saved project
    - Used Java to create web pages with dynamic content
  - Java formally announced in 1995
  - Now used to create web pages with interactive content, enhance web servers, applications for consumer devices (pagers, cell phones)...

All about Java Programming Language

- Java programs
  - Consist of pieces called classes
  - Classes contain methods, which perform tasks
- Class libraries
  - Also known as Java API (Applications Programming Interface)
  - Rich collection of predefined classes, which you can use
- Two parts to learning Java
  - Learning the language itself, so you can create your own classes
  - Learning how to use the existing classes in the libraries

Basics of a Typical Java Environment

- Java Systems
  - Consist of environment, language, Java Applications Programming Interface (API), Class libraries
- Java programs have five phases
  - Edit
    - Use an editor to type Java program
    - **`vi`** or **`emacs`**, notepad, Jbuilder, Visual J++
    - **`.java`** extension
  - Compile
    - Translates program into bytecodes, understood by Java interpreter
    - **`javac`** command: **`javac myProgram.java`**
    - Creates **`.class`** file, containing bytecodes (**`myProgram.class`**)

Basics of a Typical Java Environment

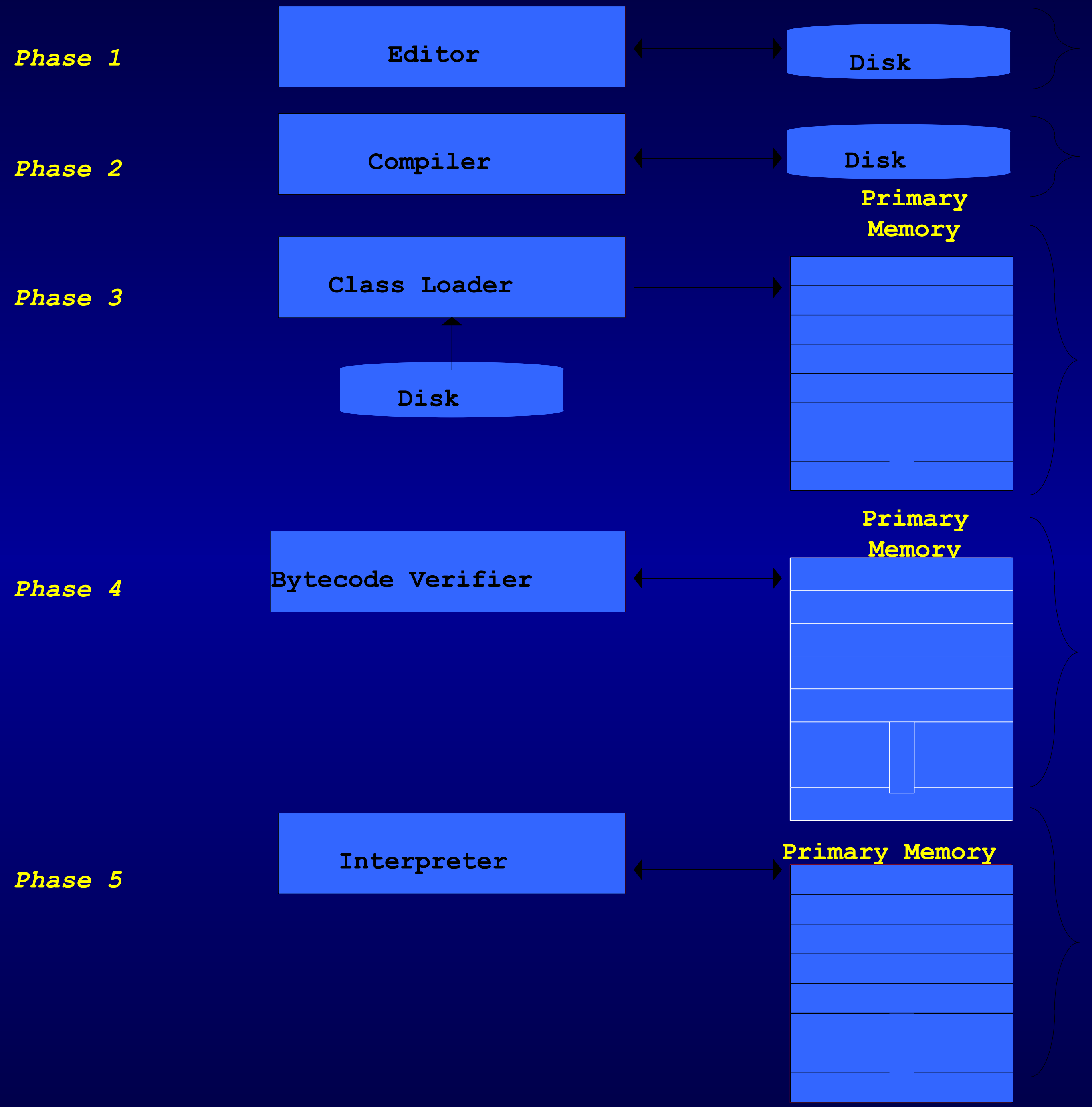
- Java programs have five phases
  - Loading
    - Class loader transfers `.class` file into memory
      - Applications - run on user's machine
      - Applets - loaded into Web browser, temporary
    - Classes loaded and executed by interpreter with `java` command

      `java Welcome`
    - HTML documents can refer to Java Applets, which are loaded into web browsers. To load,

      `appletviewer Welcome.html`
      - `appletviewer` is a minimal browser, can only interpret applets

Basics of a Typical Java Environment

- Java programs have five phases Verify
    - Bytecode verifier makes sure bytecodes are valid and do not violate security
    - Java must be secure - Java programs transferred over networks, possible to damage files (viruses)
  - Execute
    - Computer (controlled by CPU) interprets program one bytecode at a time
    - Performs actions specified in program
  - Program may not work on first try
    - Make changes in edit phase and repeat



Program is created in the editor and stored on disk.

Compiler creates bytecodes and stores them on disk.

Class loader puts bytecodes in memory.

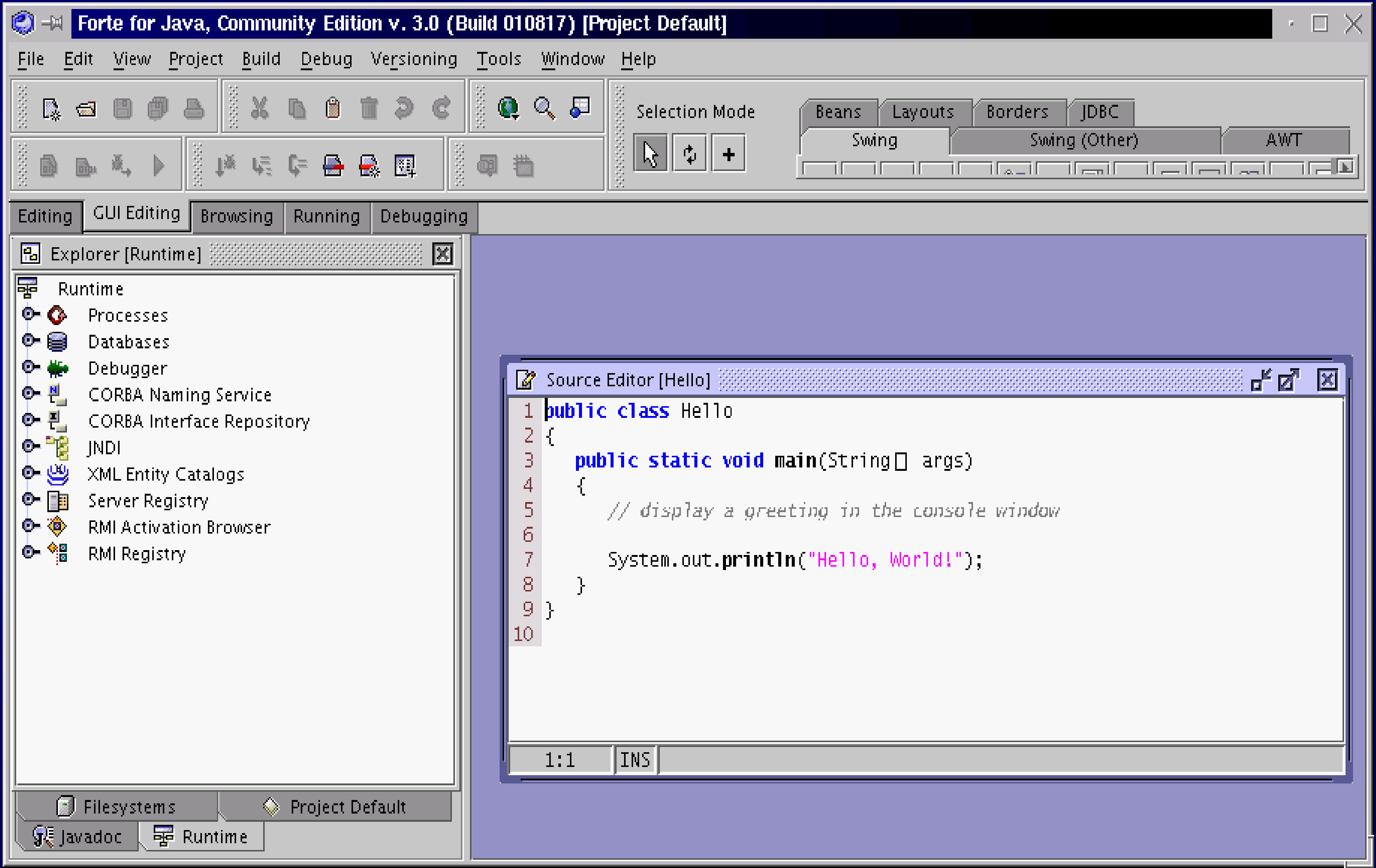
Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.

Interpreter reads bytecodes and translates them into a language that the computer can understand, possibly storing data values as the program executes.

1.14 General Notes about Java and This Book

- Just-in-time compiler
  - Midway between compiling and interpreting
    - As interpreter runs, compiles code and executes it
    - Not as efficient as full compilers
      - Being developed for Java
  - Integrated Development Environment (IDE)
    - Tools to support software development
    - Several Java IDE's are as powerful as C / C++ IDE's

An Integrated Development Environment


Forte for Java, Community Edition v. 3.0 (Build 010817) [Project Default]
File Edit View Project Build Debug Versioning Tools Window Help
Selection Mode
Beans Layouts Borders JDBC
Swing Swing (Other) AWT
Editing GUI Editing Browsing Running Debugging
Explorer [Runtime]
Runtime
Processes
Databases
Debugger
CORBA Naming Service
CORBA Interface Repository
JNDI
XML Entity Catalogs
Server Registry
RMI Activation Browser
RMI Registry
Filesystems Project Default
Javadoc Runtime
Source Editor [Hello]
1:1 INS

```java
public class Hello
{
   public static void main(String[] args)
   {
      // display a greeting in the console window

      System.out.println("Hello, World!");
   }
}
```

File Hello.java

```java
public class Hello
{
    public static void main(String[] args)
    {
    // display a greeting in the console window
          System.out.println("Hello, World!");
    }
}
```

A simple program

- public class *ClassName*
- public static void main(String[] args)
- // comment
- Method call *object*.*methodName*(*parameters*)
- System class
- System.out object
- println method

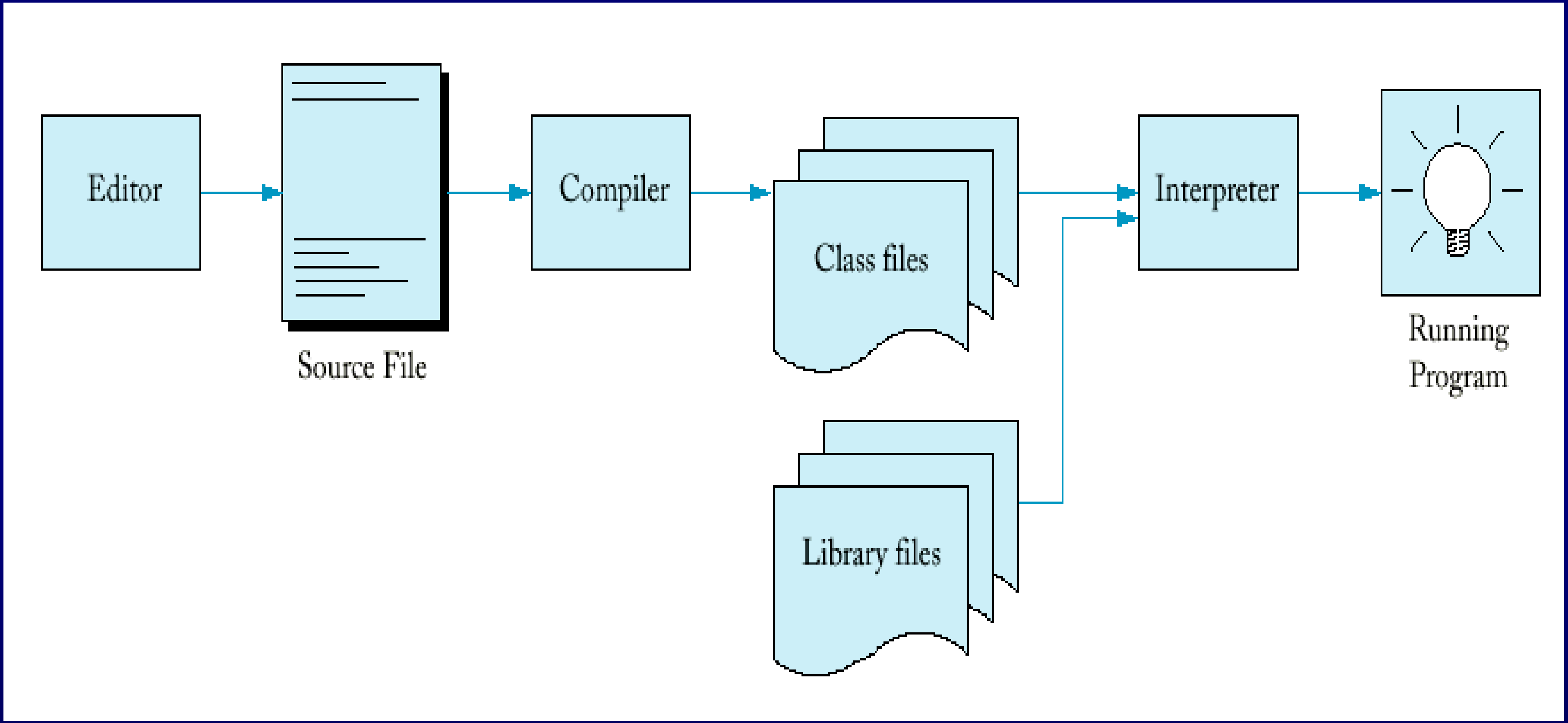
Syntax 1.1: Method Call

- `object.methodName(parameters)`
  - **Example:**
- `System.out.println("Hello,Dave!");`
  - **Purpose:**
- To invoke a method of an object and supply any additional parameters

Compiling and Running

- Type program into text editor
- Save
- Open command shell
- Compile into byte codes
  ```
  javac Hello.java
  ```
- Execute byte codes
  ```
  java Hello
  ```

From Source Code to Running Program



Errors

- Syntax errors

```
System.ouch.print("...");
System.out.print("Hello);
```

- Detected by the compiler
- Logic errors

```
System.out.print("Helo");
```

- Detected (hopefully) through testing