Chapter 1
Introduction to Computers, the Internet and Java

Objectives

- Basic object-technology concepts
- A typical java program development environment
- Test-driving a java application

1.1 Introduction

- Java is widely used for implementing
  - internet-based applications
  - software for devices that communicate over a network
  - enterprise programming
  - mobile applications

Devices		
Access control systems	Airplane systems	ATMs
Automobiles	Blu-ray Disc™ players	Building controls
Cable boxes	Copiers	Credit cards
CT scanners	Desktop computers	e-Readers
Game consoles	GPS navigation systems	Home appliances
Home security systems	Internet-of-Things gateways	Light switches
Logic controllers	Lottery systems	Medical devices
Mobile phones	MRIs	Network switches
Optical sensors	Parking meters	Personal computers
Point-of-sale terminals	Printers	Robots
Routers	Servers	Smart cards
Smart meters	Smartpens	Smartphones
Tablets	Televisions	Thermostats
Transportation passes	TV set-top boxes	Vehicle diagnostic systems

Fig. 1.1 | Some devices that use Java.

1.1 Introduction (Cont.)

- **Java Editions: SE, EE and ME**
  - Java Standard Edition (Java SE)
    - is needed to develop desktop and server applications
    - Programming paradigms

      Procedural programming

      Object-oriented programming

      Generic programming

      Functional programming(with lambdas and streams) – Java SE 8

      Modularization of Java API classes – Java SE 9

      ※ Java How to Program, 10th and 11th editions are based on **Java SE 8 and Java SE 9**.

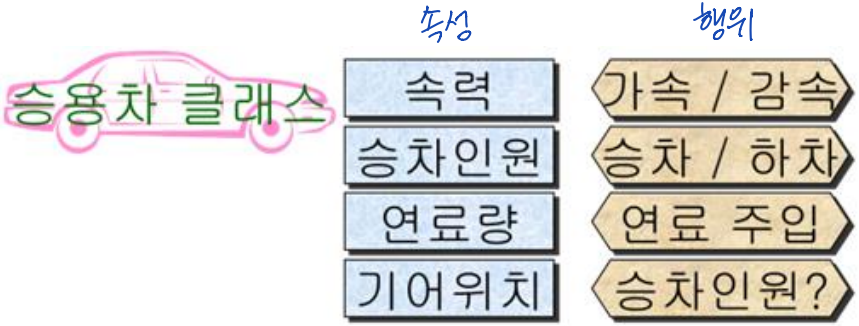
1.1 Introduction (Cont.)

- Java Enterprise Edition (Java EE)
  - geared toward developing large-scale, distributed networking applications and web-based applications.

- Java Micro Edition (Java ME)
  - a subset of Java SE
  - geared toward developing applications for *resource-constrained embedded devices* such as smartwatches, MP3 players, TV set-top box, smart meters and more.

1.5 Introduction to Object Technology

- **Class**
  - 같은 종류의 집단을 추상화(abstraction)하여 속성(attribute)과 행위(behavior)를 정의한 것
  - 사용자 정의 데이터형



1.5 Introduction to Object Technology (Cont.)

- **Object**
  - 클래스의 인스턴스
  - 고유 속성을 가지며 클래스에서 정의한 행위 수행 가능
  - 객체의 행위는 클래스의 행위에 대한 정의를 공유함으로써 메모리를 경제적으로 사용할 수 있음

1.5 Introduction to Object Technology (Cont.)

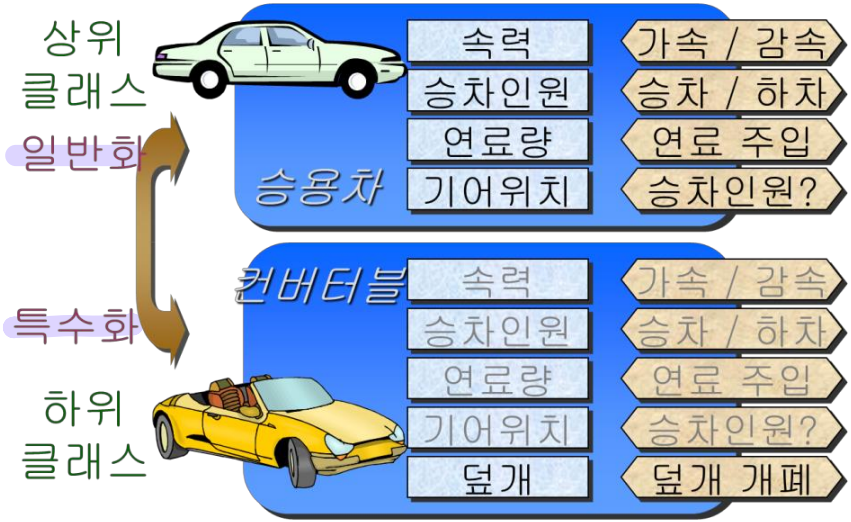
- **Data Abstraction**
  - 불필요한 정보는 숨기고 중요한 정보만을 표현하는 것
  - 자료추상화를 통해 정의된 자료형이 추상 자료형(ADT)
    - 자료 표현(속성)과 연산(행위)을 캡슐화(encapsulation)함
    - 접근 제어를 통해서 자료형의 정보를 은닉(information hiding)함

1.5 Introduction to Object Technology (Cont.)

- **Object-Oriented Programming**
  - 일반적으로 추상 자료형을 클래스,
  - 추상 자료형의 인스턴스를 객체,
  - 추상 자료형에서 정의된 연산을 메소드(method),
  - 메소드의 호출을 메시지(message)라고 한다.
    - 객체 사이의 정보전달은 메시지를 통해 일어난다.

1.5 Introduction to Object Technology (Cont.)

- **Inheritance** (상속)
  - 새로운 클래스가 기존 클래스의 속성과 행위를 이용할 수 있게 하는 것, 재사용(reuse)



1.8 Java

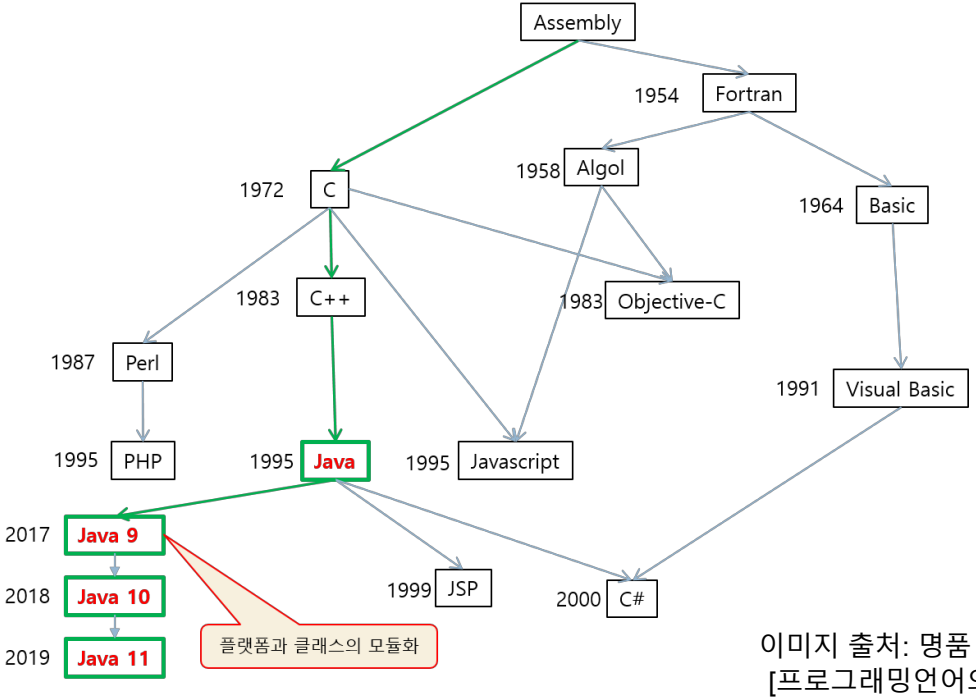
- Microprocessors have had a profound impact in intelligent *consumer-electronic devices*, including the recent explosion in the "*Internet of Things*."
  (IOT)

- 1991
  - *Sun Microsystems* funded an internal corporate research project, *green project* led by *James Gosling*, which resulted in *a C++-based object-oriented programming language* that Sun called *Java* in 1995.

  - Key goal of Java
    - "write once, run anywhere."

1.8 Java (Cont.)

- 2010
  - Sun Microsystems was acquired by Oracle.



이미지 출처: 명품 자바프로그래밍 [프로그래밍언어의 진화]

1.8 Java (Cont.)

- Java is now used
  - to develop *large-scale enterprise applications*
  - to enhance the functionality of *web servers*
  - to provide *applications for consumer devices*
  - to develop *robotics software*
  - to develop *Android* smartphone and tablet apps

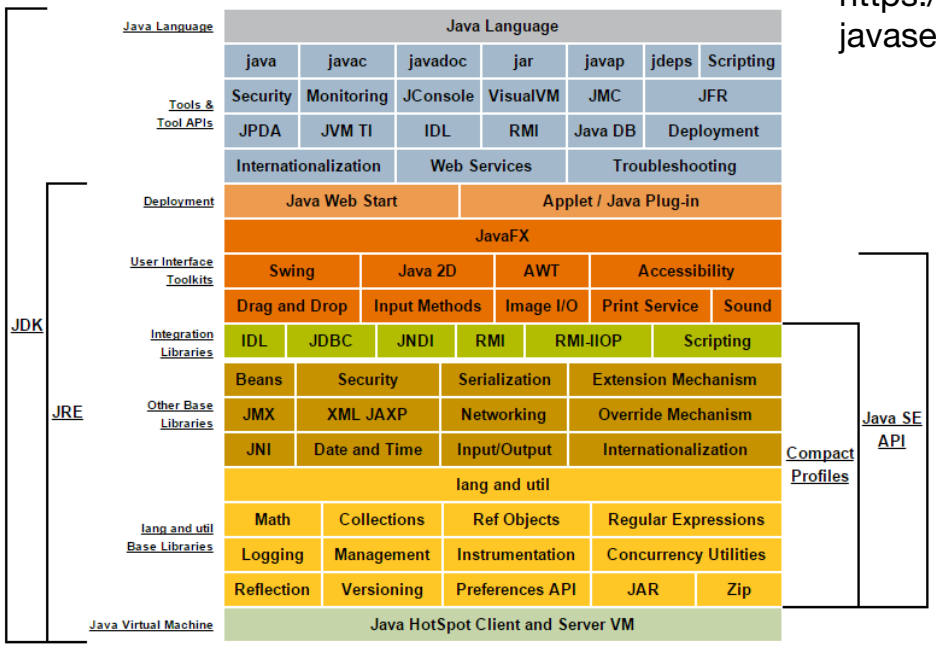
1.8 Java (Cont.)

- Java Class Libraries
  - rich collections of existing classes and methods
  - known as the Java APIs (Application Programming Interfaces).

The following conceptual diagram illustrates the components of Oracle's Java SE products:

Description of Java Conceptual Diagram

https://docs.oracle.com/javase/8/docs/



※ Java SE 8

1.9 A Typical Java Development Environment

- Normally there are *five phases*
  - edit
  - compile
  - load
  - verify
  - execute.

1.9 A Typical Java Development Environment (Cont.)

- *Phase 1*. Creating a Program
  - Linux editors : vi and emacs
  - Windows editor : Notepad
  - macOS editor : TextEdit

  - Freeware and shareware editors
    - Notepad++ (http://notepad-plus-plus.org)
    - EditPlus (http://www.editplus.com)
    - TextPad (http://www.textpad.com)
    - jEdit (http://www.jedit.org) and more

  - Integrated development environments (IDEs)
    - Eclipse (http://www.eclipse.org)
    - IntelliJ IDEA (http://www.jetbrains.com)
    - NetBeans (http://www.netbeans.org)

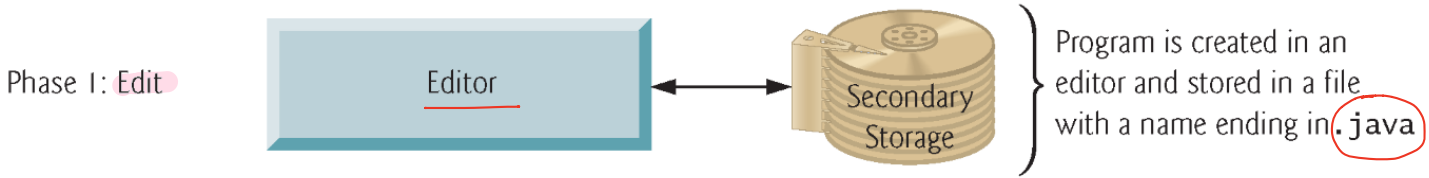


Fig. 1.6 | Typical Java development environment—editing phase.

1.9 A Typical Java Development Environment (Cont.)

- *Phase 2*. Compiling a Java Program into Bytecodes
  - Use the command `javac` to compile a program. .class
    - ex) `javac Welcome.java`

컴파일 → javac welcome.class

  - Java Virtual Machine (JVM)
    - is a part of the JDK and the foundation of the Java platform
    - executes bytecodes.
  - Virtual Machine (VM)
    - is a software application that simulates a computer
    - hides the underlying OS and HW from the programs that interact with it.
    - ex) JVM or MS's .NET

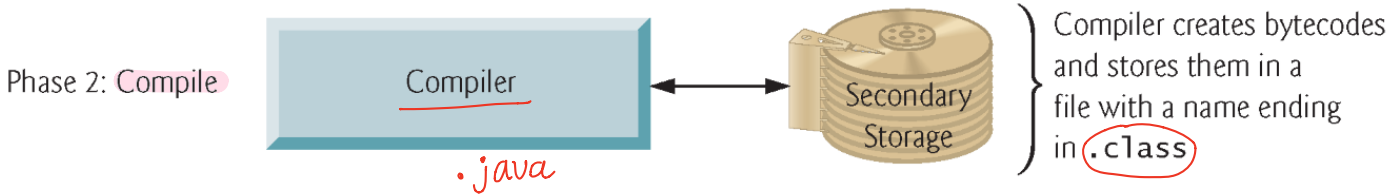


Fig. 1.7 | Typical Java development environment—compilation phase.

1.9 A Typical Java Development Environment (Cont.)

.class 파일

- Bytecode instructions are *platform independent*
- Bytecodes are *portable* (이식성의)
  - The same bytecode instructions can execute on any platform containing a *JVM that understands the version of Java in which the bytecode instructions were compiled*.

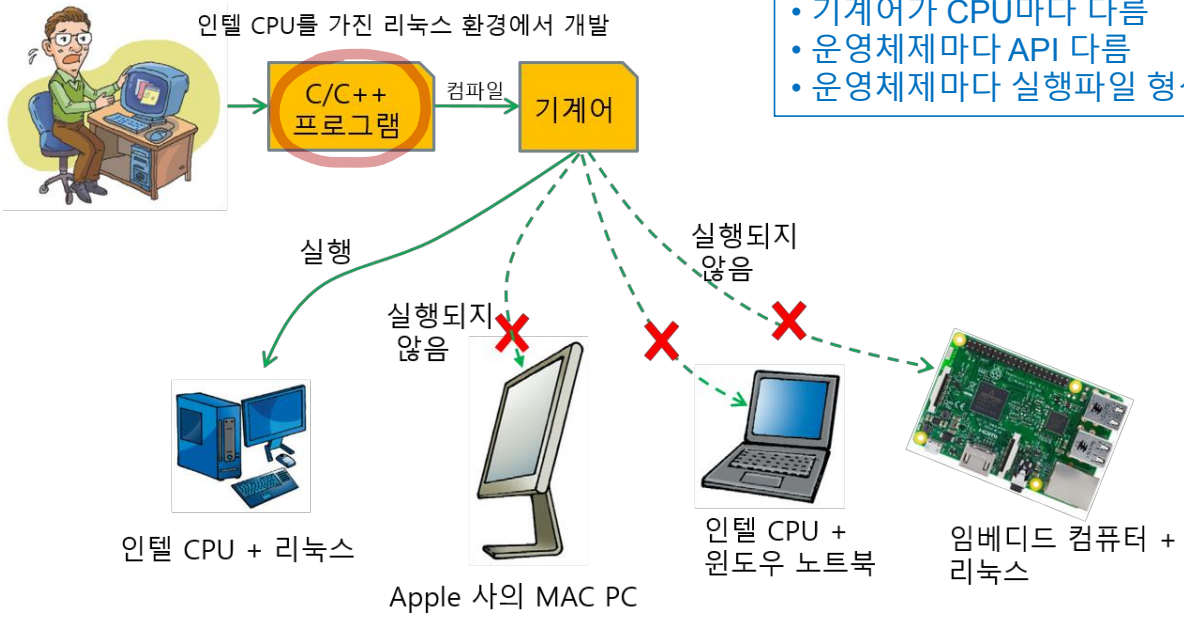
- The JVM is invoked by the `java` command.
  - ex) `java welcome`

1.9 A Typical Java Development Environment (Cont.)

- Platform-dependency (플랫폼 의존적)

플랫폼 = 하드웨어 플랫폼 + 운영체제 플랫폼

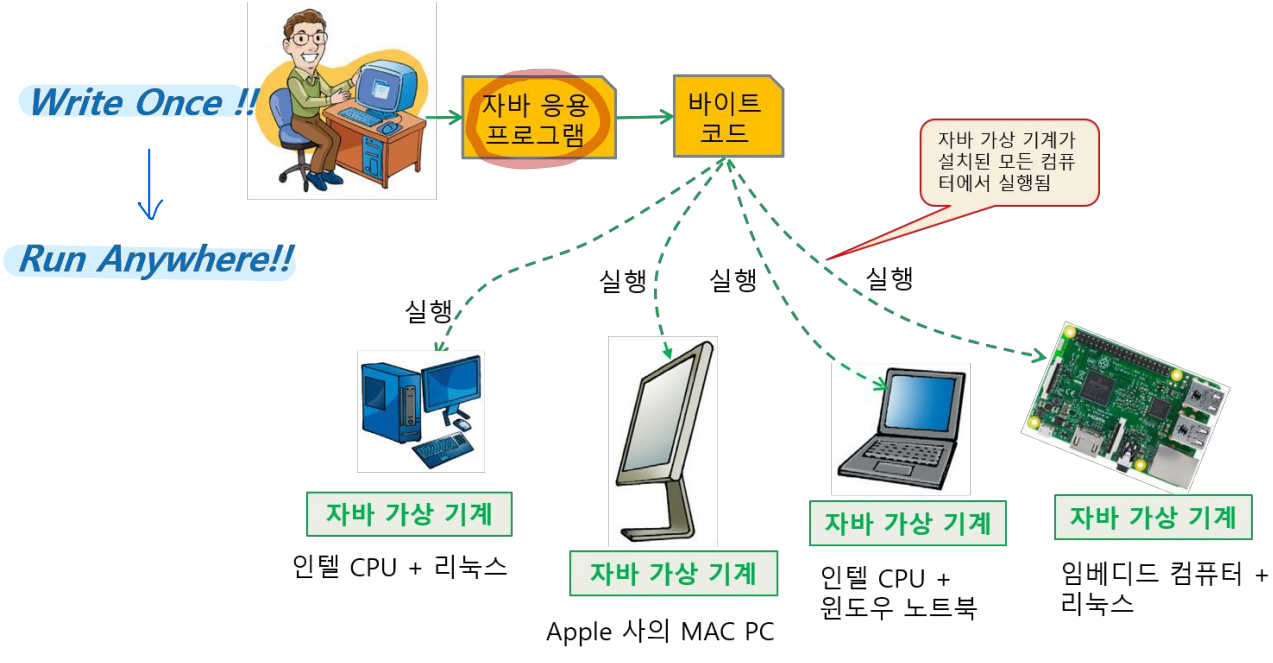
프로그램의 플랫폼 호환성 없는 이유
- 기계어가 CPU마다 다름
- 운영체제마다 API 다름
- 운영체제마다 실행파일 형식 다름



이미지 출처: 명품 자바프로그래밍

1.9 A Typical Java Development Environment (Cont.)

- Platform-independency (플랫폼 독립적)



※ 자바 가상 기계 자체는 플랫폼에 종속적

이미지 출처: 명품 자바프로그래밍

1.9 A Typical Java Development Environment (Cont.)

- *Phase 3*. Loading a Program into Memory
  - The JVM places the program in memory to execute it. — this is known as loading.
  - Class loader loads
    - the `.class` files containing the program's bytecodes
    - any of the `.class` files provided by Java that your program uses.
  - The `.class` files can be loaded from a disk on your system or over a network.

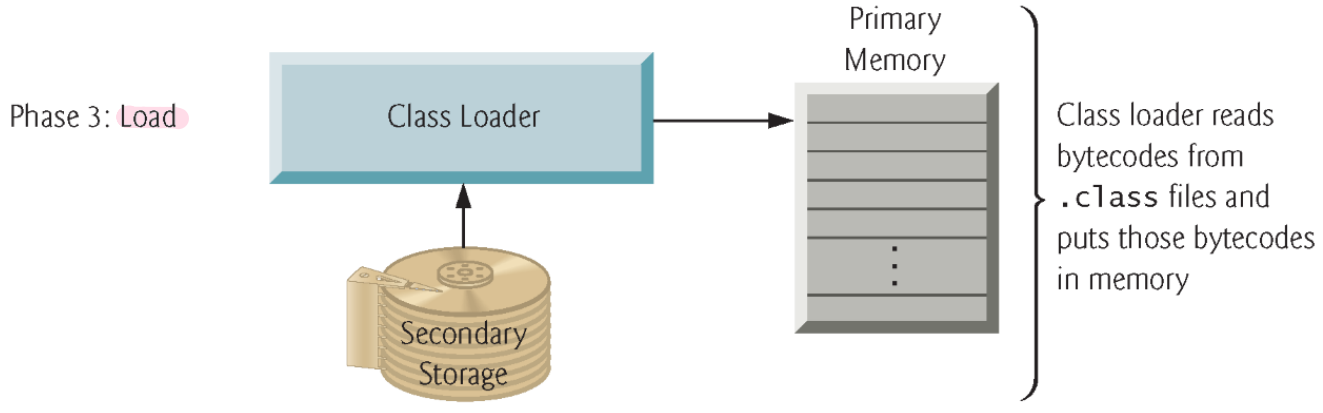


Fig. 1.8 | Typical Java development environment—loading phase.

1.9 A Typical Java Development Environment (Cont.)

- *Phase 4*. Bytecode Verification
  - As the classes are loaded, the bytecode verifier examines their bytecodes
  - Ensures that they're valid and do not violate Java's security restrictions.

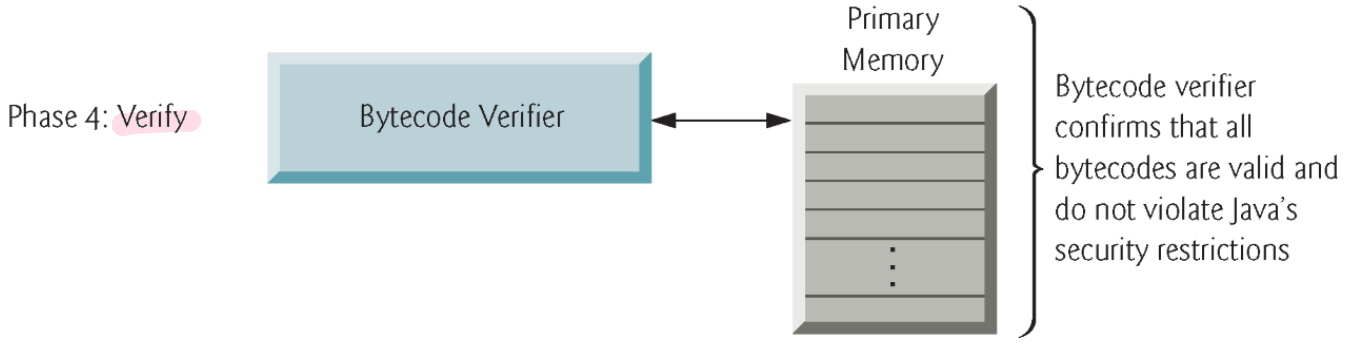


Fig. 1.9 | Typical Java development environment—verification phase.

1.9 A Typical Java Development Environment (Cont.)

- *Phase 5*. Execution
  - The JVM executes the program's bytecodes.
  - JVMs typically execute bytecodes using *a combination of* ①*interpretation* and so called ②*just-in-time(JIT) compilation*.
    - In this process, the JVM analyzes the bytecodes as they're interpreted, searching for *hot spots*—bytecodes that execute frequently.
    - For these parts, a just-in-time (JIT) compiler—such as Oracle's Java HotSpot™ compiler—translates the bytecodes into the underlying computer's machine language.
    - When the JVM encounters these compiled parts again, the faster machine-language code executes.

1.9 A Typical Java Development Environment (Cont.)

- Thus java programs go through ***two compilation phases***.
  - one in which source code is translated ***into bytecodes*** (for portability across JVMs on different computer platforms) and
  - a second in which, during execution, the bytecodes are translated ***into machine language*** for the computer on which the program executes.

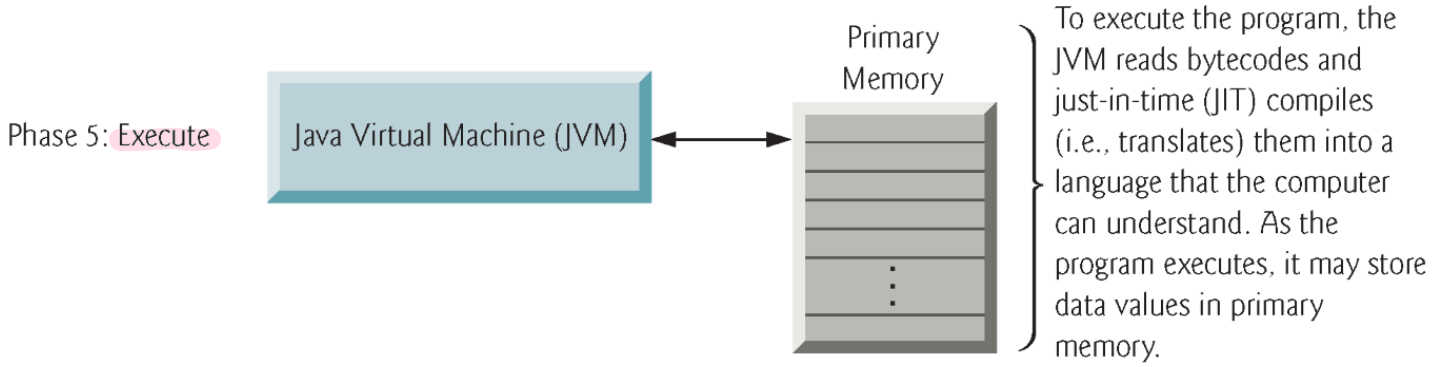


Fig. 1.10 | Typical Java development environment—execution phase.