

CHAPTER 1

INTRODUCTION TO OBJECT ORIENTED PROGRAMMING

CONTENTS

- What is Programming?
- What is Programming Paradigm?
- Class
- Objects
- Class Members
- Inheritance
- Abstraction
- Polymorphism
- Encapsulation
- Phases of java Programs

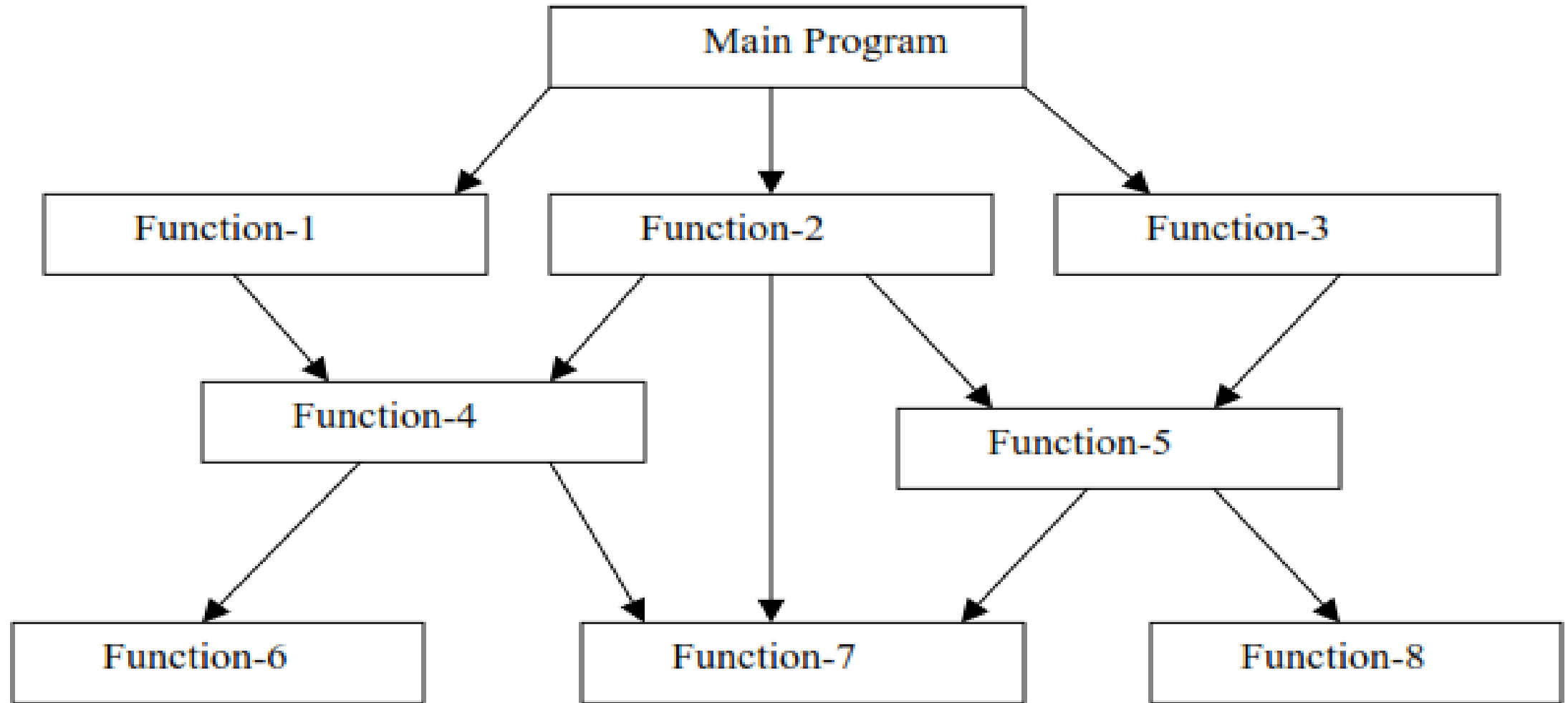
WHAT IS PROGRAMMING?

PROGRAMMING PARADIGMS

- It is a **fundamental style or approaches of programming languages.**
- The **paradigm of the programming language determines how you will solve a problem.**
- Programming Languages fit one or more paradigms
- A classification of programming languages based on their features (but most popular languages support multiple paradigms)
- Generally, there are **two types of programming paradigms:**
 - Structured(Process-Oriented) Programming Paradigms
 - Object Oriented Programming Paradigms (OOP)

STRUCTURED(PROCESS-ORIENTED) PROGRAMMING PARADIGMS

- In this approach, the **problem is viewed as the sequence of things to be done** such as reading, calculating and printing.
- Languages such as Cobol, Fortran, C, and C++ are examples of structured programming language.
- The **primary focus** of this paradigm is on **functions**.
- In structured programming paradigm, programs are written around code "what is happening?"
- The program is **organized-code acting on data** and it is **suitable only for small programs or problems**.
- The technique of **hierarchical decomposition** has been used to specify the tasks to be completed for solving a



▪ ***Fig 1:1 : A typical structure for procedural programming***

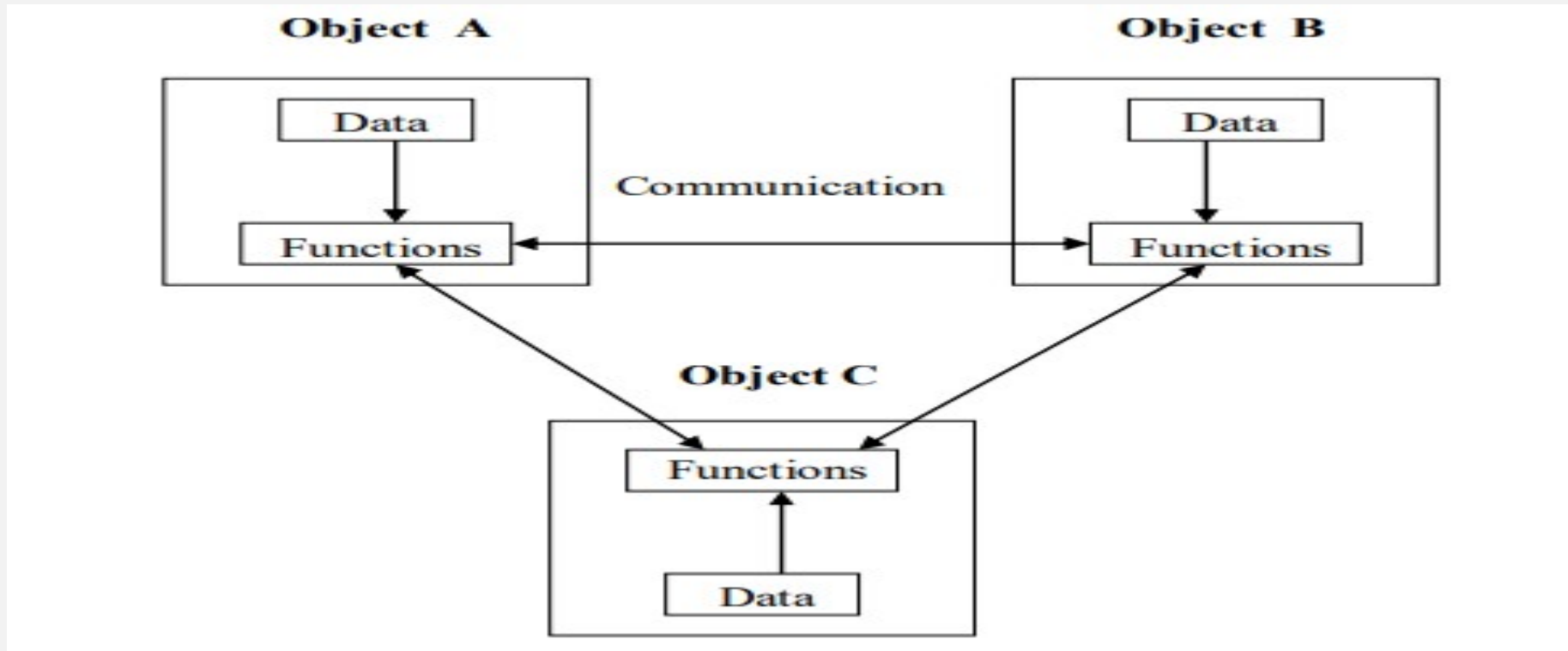
CONT....

- **Procedure/Function oriented programming** basically **consists of writing a list of instructions** for the computer to follow, and **organizing these instructions into groups known as functions**.
- We normally use flowcharts to organize these actions and represent the flow of control from one action to another.
- In a **multi-function program**, many important **data items are placed as global** so that they may be **accessed by all functions**. Each function may have its own local data.
- **Global data** are more **vulnerable to accidental change by a function**.
- In a **large program** it is very **difficult to identify what data is used by which function**.

- This provides an **opportunity for bugs to creep in.**
- Another serious drawback with the procedural approach is that **it does not model real world problems very well.**
- This is because **functions are action-oriented and do not really corresponding to the element of the problem.**
- Some characteristics exhibited by procedure-oriented programming are:
 - Emphasis is on doing things (algorithms)
 - Large programs are **divided into smaller programs known as functions**
 - Most of the functions **share global data.**
 - Data move openly around the system from function to function.
 - Employs top-down approach in program design.

OBJECT ORIENTED PROGRAMMING PARADIGMS

- **Problems** with structured programming approach **appear as programs grow larger and more complex.**
- To manage increasing complexity, the second approach, called **object-oriented programming, was conceived.**
- **OOP organizes a program around its data** (that is, objects) and a set of well-defined interfaces to that data.
- **OOP** allows us to **decompose a problem into a number of entities called objects** and then **build data and methods (functions) around these entities.**
- Structured programming is a subset of object-oriented programming.
- Therefore, OOP can help in developing **much larger and complex programs than structured programming.**



- Fig: Shows the **organization** of **data and function in object-oriented programs**.
- The **data of an object** can be **accessed only** by the **function (methods) associated with that object**.
 - However, **function (method) of one object can access the function (method) of other objects**

- Some of the features of object oriented programming are:
 - Emphasis is on data rather than procedure.
 - Programs are divided into what are known as objects.
 - Data structures are designed such that they characterize the objects.
 - Functions (methods) that operate on the data of an object are tied together in the data structure.
 - Data is hidden and cannot be accessed directly by external function.
 - Objects may communicate with each other through function.
 - New data and functions can be easily added whenever necessary.
 - Follows bottom up approach in program design.

STRUCTURED PROGRAMMING VERSUS OBJECT ORIENTED PROGRAMMING

Structured Programming	Object Oriented Programming
Both functions and data is considered separate because data structures are not well organized within the program	Data and methods are tied together since variables and methods are well organized under classes
Programs are divided into small self contained functions	Programs are divided into small entities called objects
Functions or procedures are the basic units of manipulation in structured programming	Objects are the basic units of manipulation in object oriented programming
Structured Programming	Object Oriented Programming

STRUCTURED PROGRAMMING VERSUS OBJECT ORIENTED PROGRAMMING

- | | |
|--|---|
| <ul style="list-style-type: none">▪ A program is focused on manipulation of data. | <ul style="list-style-type: none">▪ A program is focused on the manipulation of both data (classes and states of objects) and on methods. |
| <ul style="list-style-type: none">▪ Structured programming is less secure as there is no way of data hiding | <ul style="list-style-type: none">▪ Data is hidden within the objects and its manipulation can be strictly controlled in OOP |
| <ul style="list-style-type: none">▪ Structured Programming misses reuses of classes, codes and inheritance | <ul style="list-style-type: none">▪ OOP supports reuses of classes and their parts of the code and supports the inheritance of state and behavior |

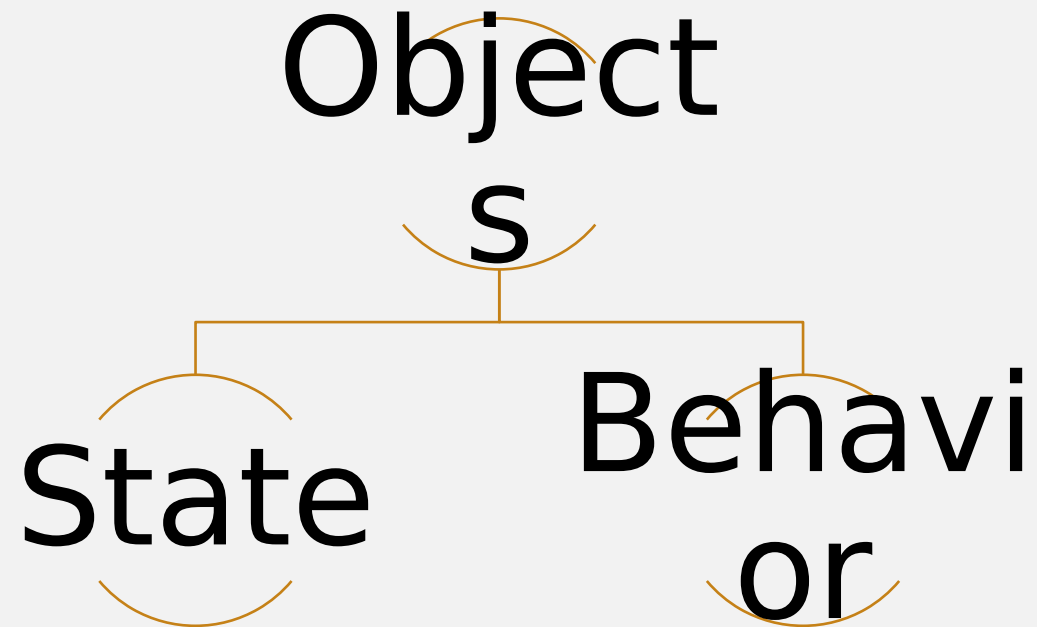
CLASS

- A class can be defined as a **template/ blue print that describes the behaviors/states that object of its type support.**
- A group of objects that share common properties for data part and some program part are collectively called as class.
- For example, a class that represents a bank account might contain one method to *deposit* money to an account, another to *withdraw* money from an account and a third to *inquire* what the account's current balance is.



A class that represent a House might contain one method to open door ,another to close the door. And states like color, Height, width.

- Objects have states and behaviors.
- Example. A dog has states- color, name, as well as behaviors – wagging, barking, eating, An object is an instance of a class.



VARIABLES

- A class can contain any of the following variable types
- **Local Variables :-** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables:-** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables :-** Class variables are variables declared with in a class, outside any method, with the static keyword.

METHODS AND CONSTRUCTORS

➤ Methods

- ❑ A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

➤ Constructors

- ❑ Every class has a constructor. If we do not explicitly write a constructor for a class the Java compiler builds a default constructor for that class.
- ❑ Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class.
- ❑ A class can have more than one constructor.

PILLARS OF OBJECT ORIENTED PROGRAMMING

- There are four major pillars of Object Oriented Programming .

These are:-

Encapsulation

Inheritance

Abstraction

Polymorphism

ENCAPSULATION

- Encapsulation is the **technique of making the fields in a class private** and **providing access to the fields via public methods**.
- If a field is declared private, it cannot be accessed by anyone outside the class, thereby **hiding the fields** within the class.
- For this reason, **encapsulation** is also referred to as **data hiding**.
- Encapsulation can be described as a **protective barrier that prevents the code and data being randomly accessed** by other code defined outside the class.
- Access to the data and code is tightly controlled by an interface.

INHERITANCE

- It can be defined as the **process where one class acquires the properties (methods and fields) of another.**
- With the use of inheritance, the information is made manageable in a hierarchical order.
- The class which inherits the properties of other is known as subclass (**derived class, child class**)
- The class whose properties are inherited is known as superclass (**base class, parent class**).
- **extends** is the keyword used to inherit the properties of a class.

ABSTRACTION

- As per dictionary, abstraction is the quality of **dealing with ideas rather than events**.
- In object-oriented programming, abstraction is a process of **hiding the implementation details** from the user, and **only the functionality will be provided** to the user. In other words, the user will have the information on **what the object does instead of how it does it**.
- Showing only the essential parts and hiding the implementation details.
- All other functionality of the class still exists, and its fields, methods, and constructors are all accessed in the same manner.
- Use the **abstract** keyword to declare a class abstract.
- The keyword appears in the class declaration somewhere **before the class keyword**.
- These classes cannot be used to instantiate objects, because, as we'll soon see, abstract classes are *incomplete*.

POLYMORPHISM

- Polymorphism is the ability of an object to take on **many forms**.
- Polymorphism means "many forms", and it occurs when we have **many classes that are related to each other** by inheritance.
- The most common use of polymorphism in OOP occurs **when a parent class reference is used to refer to a child class object**.
- **Inheritance** lets us inherit attributes and methods from another class. **Polymorphism** uses those **methods to perform different tasks**. This **allows us to perform a single action in different ways**.
- For example, think of a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

HISTORY OF JAVA

- Developed by Sun Microsystems and released in 1995.
- Initiated by **James Gosling and his team in 1991** as a project called **"Oak"**. It was later **renamed to "Java" in 1995** as a core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).
- Multiple configurations were built to suit various types of platforms. For example, J2EE was built for Enterprise Applications, and J2ME was built for Mobile Applications.
- Sun Microsystems has **renamed the new J2 versions** as Java SE (Standard Edition), Java EE (Enterprise Edition), and Java ME (Micro Edition) respectively.
- Java is guaranteed to be **Write Once, Run Anywhere (WORA)**. This means that **code written on one platform can be run on any platform** without modification, as long as the platform has a compatible Java Virtual Machine (JVM) installed.
- **On 13 November 2006**, Sun Microsystems released much of Java as **free and open source software** under the terms of the GNU General Public License (GPL).

FEATURES OF JAVA

- Object Oriented programming language : Which means that **everything in Java is an object**. This makes it easier to model complex systems and reuse code, as objects can be created, manipulated, and reused across different parts of a program.
- **Platform independent**: When Java code is compiled, it is **not compiled into platform-specific machine code**. Instead, it is compiled into platform-independent bytecode, which can be executed on any platform. This makes Java highly portable, as a single Java program can be written and run on multiple platforms, such as Windows, Mac, or Linux, without any changes to the code.
- **Simple**: If you understand the basic concept of OOP Java would be easy

FEATURES OF JAVA

- **Robust:** Java is designed to be a robust language that eliminates many common programming errors by emphasizing **compile-time error checking** and **runtime checking**. This makes it less likely for programs to crash or encounter runtime errors. Additionally, Java has built-in **garbage collection**, which **automatically frees up memory that is no longer being used** by a program. This helps prevent memory leaks and other types of memory-related errors.
- **Multithreaded:** Java's multithreading feature allows developers to **write programs that can perform multiple tasks simultaneously**. This can help **improve performance** and **responsiveness** in certain types of applications, such as web servers or video games.

JAVA FACTS

- Java is a popular programming language used for developing a wide range of applications, from enterprise systems to consumer devices.
- Sun Microsystems, the original developer of Java, was acquired by **Oracle in 2009.**
- In 2010, Oracle announced that Java was running on 97% of enterprise desktops, three billion handsets, and 80 million television devices.
- As of 2021, there are over 9 million Java developers worldwide.
- While the popularity of programming languages may vary, Java remains a widely used language in the software development industry.

DEFINITIONS OF JAVA APPLICATIONS, JAVA APPLET

➤ Java Applications

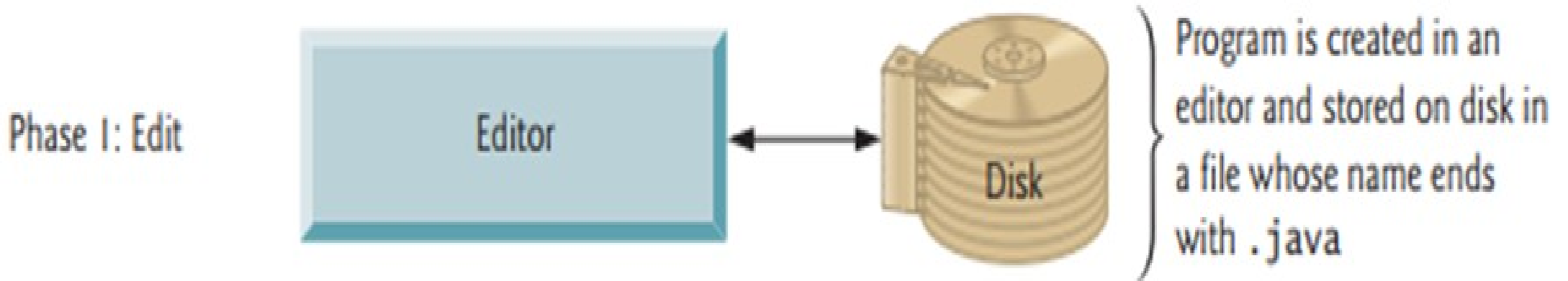
- A Java program that **runs stand alone in a client or server**. The Java Virtual Machine (**JVM**) **interprets the instructions**, like any programming language running in its native environment.
- A Java **application** is a computer program that executes when you use the **java command** to launch the Java Virtual Machine (JVM).

➤ Java Applet

- An applet is a **small Internet based program** written in Java
- A **programming language for the web**, which can be downloaded by any computer.
- The applet is also **able to run in HTML**. The applet is usually **embedded in an HTML page** on a Web site and **can be executed from within a browser**.

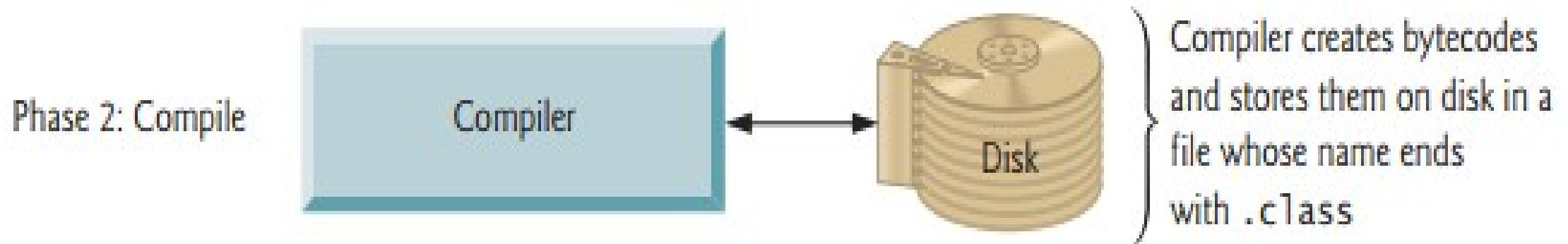
PHASES OF CREATING AND EXECUTING A JAVA APPLICATION

- Java programs normally go through five phases—**edit**, **compile**, **load**, **verify** and **execute**. We discuss these phases in the context of the Java SE Development Kit (JDK).
- Phase 1 **Edit**(*Writing the code*)
 - This involves using a **text editor or IDE**(integrated development environment) to write Java code.
 - The code is saved in a text file with a **.java** extension.



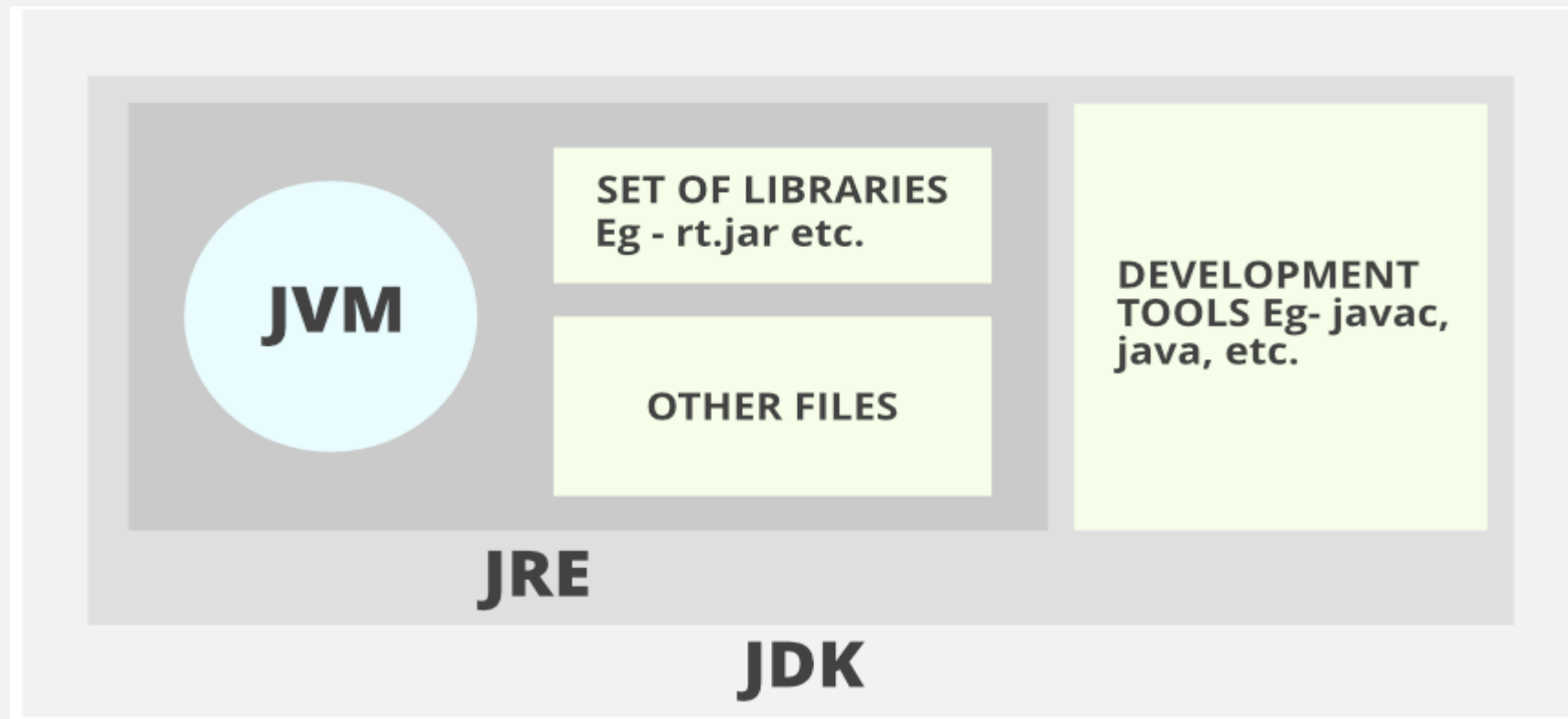
PHASE 2: COMPILING A JAVA PROGRAM INTO BYTECODES

- Once the code is written, it needs to be **compiled into bytecode** that can be executed by the Java Virtual Machine (JVM).
- The compiler **translates the source code into bytecode**, which is saved in a file with a **.class** extension.
- The bytecode is platform-independent, meaning it can be run on any device with a JVM installed.
- You use the command **javac** (the **Java compiler**) to **compile** a program.



JAVA VIRTUAL MACHINE (JVM)

- The Java **source code is compiled by the Java compiler into bytecodes** that define the tasks to be performed during the execution phase (Phase 5).
- These bytecodes are executed by the **Java Virtual Machine (JVM)**, which is an integral part of the JDK and serves as the foundation of the Java platform.



JAVA VIRTUAL MACHINE (JVM)

- A **virtual machine (VM)** is a software program that emulates a computer and provides a platform for executing applications.
- The JVM, specifically, **simulates the operation of a computer**, while hiding the underlying operating system and hardware from the programs that interact with it.
- **Bytecodes**, unlike machine language, **are not dependent on specific computer hardware** and **are platform-independent**. This means that the same set of bytecodes can be executed on any platform that has a JVM that understands the version of Java.
- As a result, **Java's bytecodes are portable**, and you can execute them without having to recompile the source code.

- **To invoke the JVM**, you can use the **java** command followed by the name of the Java application you want to execute.
- For example, if you want to run a Java application named "**Welcome**," you would type:

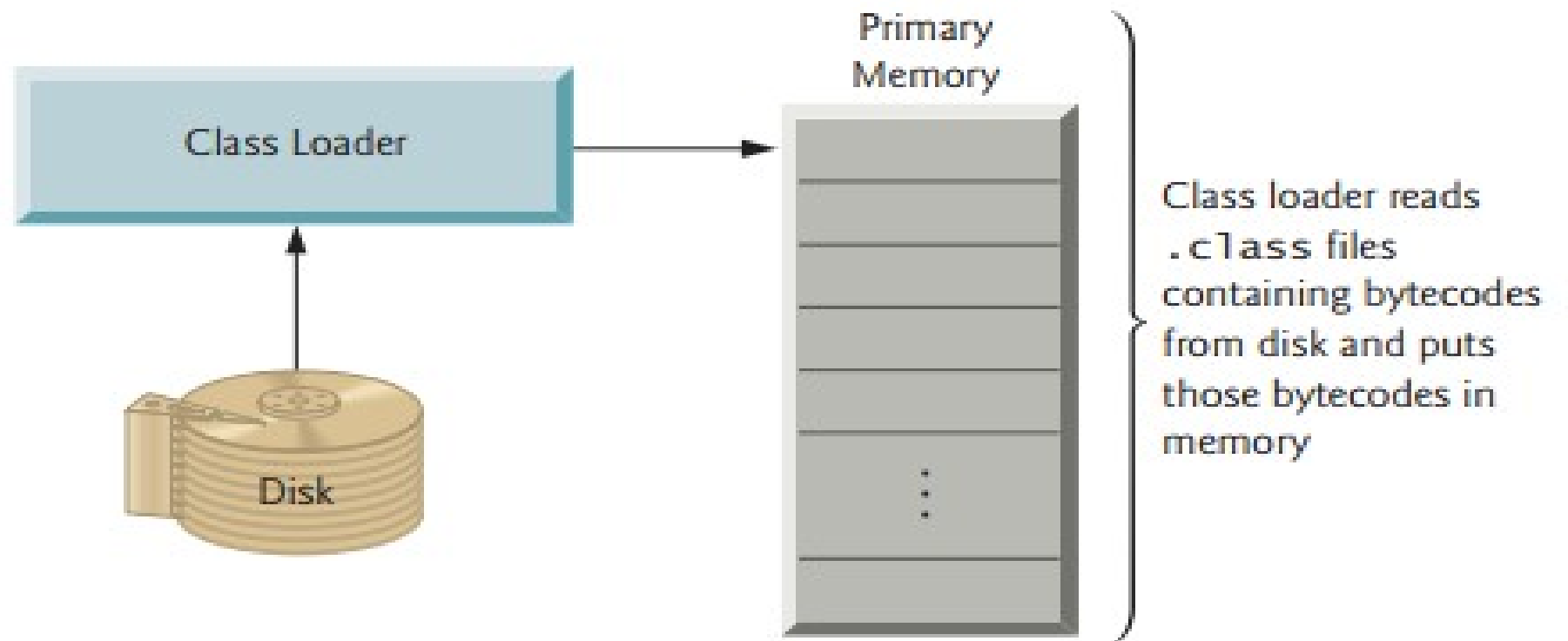
> java Welcome

- in a command window to invoke the JVM, which would then initiate the steps necessary to execute the application. This begins Phase 3.

PHASE 3: LOADING A PROGRAM INTO MEMORY

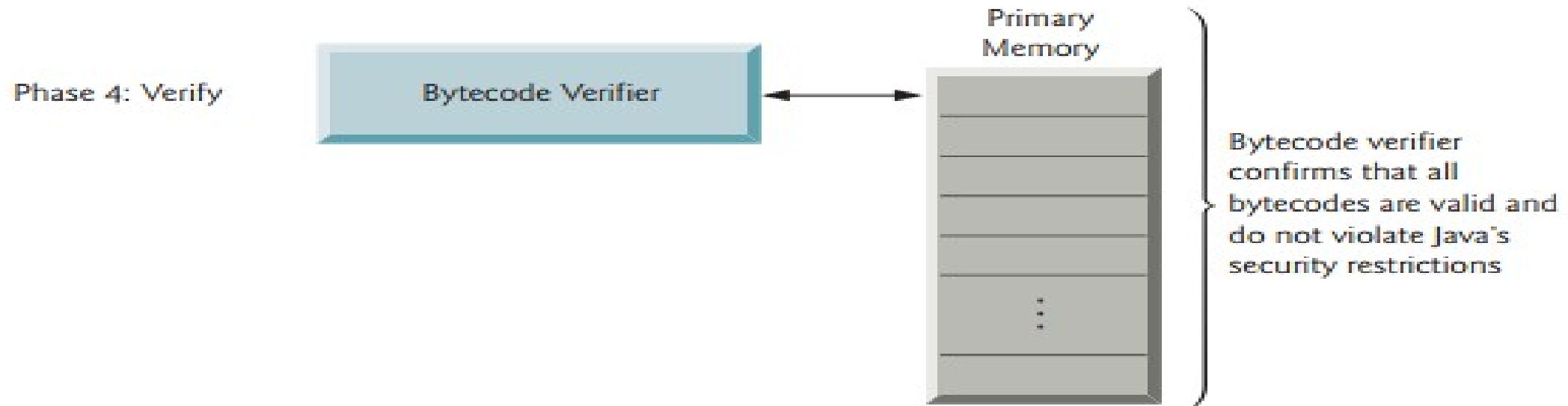
- In Phase 3, the JVM places the program in memory for execute it—this is known as **loading**.
- The JVM's **class loader** takes the .class files containing the program's bytecodes and **transfers them to primary memory**.
- The class loader also loads any of the .class files provided by Java that your program uses. The .class files can be loaded from a disk on your system or over a network

Phase 3: Load



PHASE 4: BYTECODE VERIFICATION

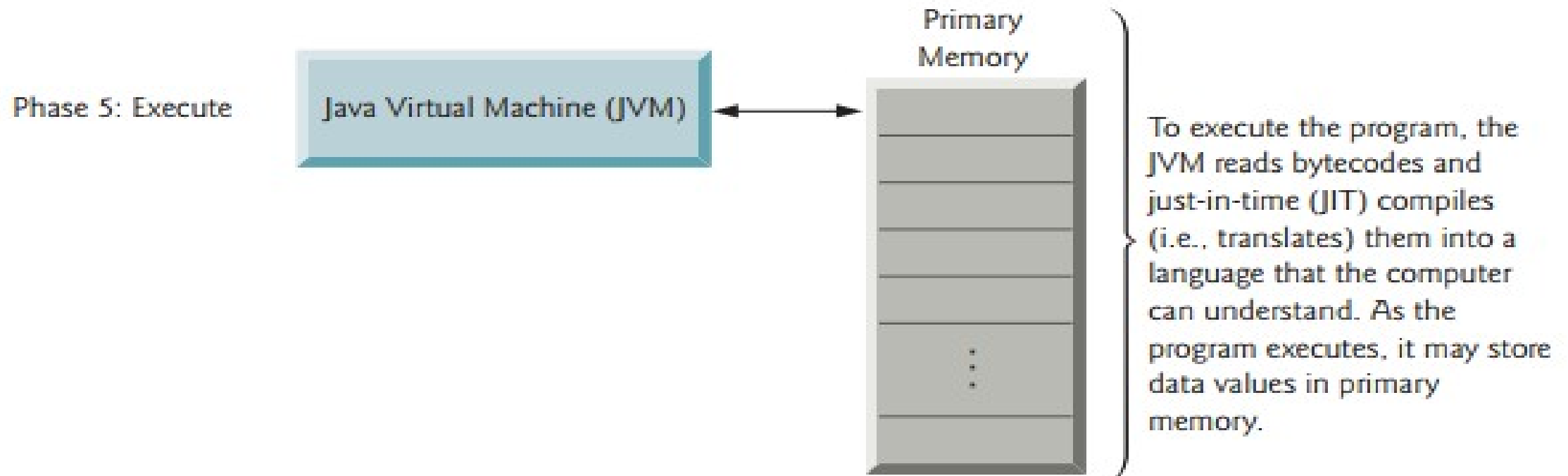
- In Phase 4, as the classes are loaded, the **bytecode verifier** examines their bytecodes **to ensure that they're valid** and do not violate Java's security restrictions .
- Java enforces strong security to make sure that Java programs arriving over the network do not damage your files or your system (as computer viruses and worms might)



PHASE 5: EXECUTION

- In Phase 5, the JVM executes the program's bytecodes, thus performing the actions specified by the program.
- In early Java versions, the **JVM was simply an interpreter** for Java bytecodes.
- This caused most Java programs to **execute slowly**, because the JVM would interpret and execute one bytecode at a time.
- Today's JVMs typically execute bytecodes using a **combination of interpretation and so-called just-in-time (JIT) compilation**.
- In this process, the JVM analyses the bytecodes as they're interpreted, searching for hot spots (***parts of the bytecodes that execute frequently***).

- For these parts, a **just-in-time** (JIT) compiler—known as the **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.



- Thus Java programs actually go through **two compilation phases**
 - One in which **source code is translated into bytecodes** (for portability across JVMs on different computer platforms)
 - Second in which, during execution, the **bytecodes are translated into machine language** for the actual computer on which the program executes.

MY FIRST JAVA APPLICATION

1st Line: **public class HelloWorld**

- Java is a true object-oriented language and therefor, Everything must be placed inside a class
- '**class**' is a keyword and declares a new class definition follows by the name - HelloWorld
- The class uses an access specifier '**public**', which indicates that our class is accessible from other packages(packages are collection of classes)

```
J HelloWorld.java > ...  
1  public class HelloWorld  
2  {  
3      public static void main(String[] args)  
4      {  
5          System.out.println("Hello World!");  
6      }  
7  }
```

2nd Line: {

- Every class definition in Java begins with Opening curly brace '{' and ends with a matching closing brace '}'
- We can place '{' next to the first line
 - E.g. public class HelloWorld
{

3rd Line: **public static void main (String[] args)**

- Indicates a method called 'main' in our class
- The '**main**' method is the starting point of a Java program. It is where JVM begins execution of the program.
- A java application can have any number of classes but only one of them must include a main method to initiate the execution.
- "**public**" : is an access specifier that says the "main" method is accessible to any object (or to all classes)
- "**static**" : declares this method does not belong to an instance of a class, rather it belongs to the class as a whole. "main" must always be declared "static" since the interpreter uses this method before any objects are created.
- "**void**" : indicates main method does not return any value
- "**String[] args**" : declares parameter named args which contains an array of strings. Array of strings is passed by command line.

NB: main method must always follow the above signature

- 5th Lne: `System.out.println("Hello World!");`
 - Since Java is a true Object Oriented language, every method must be part of an object. The “println” method is a member of the out object, which is static data member of the “System” class.
 - Prints a given text to the screen.
 - “println” appends a newline to the end of string, “print” does not.
- Every Java program consists of at least one class declaration that is defined by the programmer.
- By convention all class names in Java begin with capitalized first letters of each word in a name.
- Class name can contain: digits, letters, underscore, dollar sign but do not begin with a digit.
- Java is **CASE SENSITIVE!**
- When we save a public class declaration in a file, the file name must be “class name” followed by “.java” extension.
- Every Java statement must end with a semicolon.

COMMENTING YOUR PROGRAMS

- Comments used to increase the readability of document programs.
- Java compiler ignores comments , so they do not cause the computer to perform any action when the program is run.
- There are two types of comments in Java
 - I. Single line comment begins with `//`, indicating that it is an end-of-line comment- it terminates at the end of the line on which the `//` appears.
 - II. Traditional comments (multiline comments):- which can spread over several lines
 - These begin and end with delimiters `/*` and `*/`
- *Forgetting one of the delimiters of a traditional or Javadoc comment is a syntax error.*

- A **syntax error** occurs when the compiler encounters code that **violates Java's language rules** (i.e., its syntax)
- These rules are similar to a natural language's grammar rules specifying sentence structure.
- Syntax errors are also called **compiler errors**, **compile-time errors** or **compilation errors**, because the compiler detects them during the compilation phase.
- The compiler responds by issuing an error message and preventing your program from compiling.

DECLARING A CLASS

- the Phrase “public class FirstJavaCode “begins a **class declaration**
- Every Java program consists of at least one class that you (the programmer) define.
- The **class keyword** introduces a class declaration and is immediately followed by the **class name** (HelloWorld).
- **Keywords** (sometimes called **reserved words**) are reserved for use by Java and are always spelled with all **lowercase letters**.

DECLARING A METHOD

- `public static void main( String[] args )` is the starting point of every Java application.
- The **parentheses** after the identifier `main` indicate that it's a program building block called a **method**. Java class declarations normally contain one or more methods.
- For a Java application, one of the methods *must* be called `main`.
- Methods perform tasks and can return information when they complete their tasks.
- Keyword **void** indicates that this method will *not* return any information.
- the `String[] args` in parentheses is a required part of the method `main`'s declaration

PERFORMING OUTPUT WITH SYSTEM.OUT.PRINTLN

- `System.out.println( "Hello World!" );` instructs the computer to perform an action—namely, to print the **string** of characters contained between the double quotation marks (but not the quotation marks themselves).
- A string is sometimes called a **character string** or a **string literal**.
- White-space characters in strings *are not ignored* by the compiler.
- The **System.out** object is known as the **standard output object**. It allows a Java applications to display information in the **command window** from which it executes.
- Method **System.out.println** displays (or prints) a line of text in the command window.
- The string in the parentheses is the **argument** to the method.

COMPILING AND EXECUTING YOUR FIRST JAVA APPLICATION

- Using the Java Development Kit's command-line tools, not an IDE.
- lets assume we have a folder in drive C : called "java"

```
PS C:\java> javac HelloWorld.java
PS C:\java> java HelloWorld
Hello World!
```

```
public class EndOfChapter1 {  
  
    public static void main(String [] args){  
  
        System.out.println("End of Chapter One. Thank you") ;  
        System.out.println("Reading Assignment- study about variables and  
data types in java") ;  
  
    }  
}
```

END OF CHAPTER 1