PROGRAMMING IN JAVA

RAMAIAH
Institute of Technology

Dr. Manish Kumar
Assistant Professor
Department of Master of Computer Applications
M S Ramaiah Institute of Technology
Bangalore-54

Chapter 1 - The History and Evolution of Java

Java's Lineage

- Java is related to C++, which is a direct descendant of C. Much of the character of Java is inherited from these two languages.
- Many of Java's object oriented features were influenced by C++.

The Birth of Modern Programming: C

- The creation of C was a direct result of the need for a structured, efficient, high-level language that could replace assembly code when creating systems programs.
- Prior to C, programmers usually had to choose between languages that optimized one set of traits or the other.

- Another compounding problem was that early computer languages such as BASIC, COBOL, and FORTRAN were not designed around structured principles.
- Invented and first implemented by Dennis Ritchie on a DEC PDP-11 running the UNIX operating system, C was the result of a development process that started with an older language called BCPL, developed by Martin Richards.
- BCPL influenced a language called B, invented by Ken Thompson, which led to the development of C in the 1970s. For many years, the de facto standard for C was the one supplied with the UNIX operating system and described in The C Programming Language by Brian Kernighan and Dennis Ritchie (Prentice-Hall, 1978).
- C was formally standardized in December 1989, when the American National Standards Institute (ANSI) standard for C was adopted.

C++: The Next Step

- During the late 1970s and early 1980s, C became the dominant computer programming language, and it is still widely used today.
- Since C is a successful and useful language, you might ask why a need for something else existed. The answer is *complexity*.
- The use of structured languages enabled programmers to write, for the first time, moderately complex programs fairly easily. However, even with structured programming methods, once a project reaches a certain size, its complexity exceeds what a programmer can manage.
- By the early 1980s, many projects were pushing the structured approach past its limits. To solve this problem, a new way to program was invented, called *object-oriented programming (OOP)*.
- OOP is a programming methodology that helps organize complex programs through the use of inheritance, encapsulation, and polymorphism.

The Stage Is Set for Java

- By the end of the 1980s and the early 1990s, object-oriented programming using C++ took hold.
- Indeed, for a brief moment it seemed as if programmers had finally found the perfect language.
- Because C++ blended the high efficiency and stylistic elements of C with the object-oriented paradigm, it was a language that could be used to create a wide range of programs.
- However, just as in the past, forces were brewing that would, once again, drive computer language evolution forward.
- Within a few years, the World Wide Web and the Internet would reach critical mass. This event would precipitate another revolution in programming.

The Creation of JAVA

- Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991.
- It took 18 months to develop the first working version. This language was initially called "Oak," but was renamed "Java" in 1995.
- Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of thelanguage.
- Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin, and Tim Lindholm were key contributors to the maturing of the original prototype.

- The original impetus for Java was not the Internet! Instead, the primary motivation was the need for a platform-independent language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls.
- By 1993, it became obvious to members of the Java design team that the problems of portability frequently encountered when creating code for embedded controllers are also found when attempting to create code for the Internet.
- In fact, the same problem that Java was initially designed to solve on a small scale could also be applied to the Internet on a large scale.
- This realization caused the focus of Java to switch from consumer electronics to Internet programming. So, while the desire for an architecture-neutral programming language provided the initial spark, the Internet ultimately led to Java's large-scale success.

The C# Connection

- The reach and power of Java continues to be felt in the world of computer language development. Many of its innovative features, constructs, and concepts have become part of the baseline for any new language.
- The success of Java is simply too important to ignore. Perhaps the most important example of Java's influence is C#.
- Created by Microsoft to support the .NET Framework, C# is closely related to Java.
- For example, both share the same general syntax, support distributed programming, and utilize the same object model.
- There are, of course, differences between Java and C#, but the overall "look and feel" of these languages is very similar. This "cross-pollination" from Java to C# is the strongest testimonial to date that Java redefined the way we think about and use a computer language.

How Java Changed the Internet

- The Internet helped catapult Java to the forefront of programming, and Java, in turn, had a profound effect on the Internet.
- In addition to simplifying web programming in general, Java innovated a new type of networked program called the applet that changed the way the online world thought about content.
- Java also addressed some of the thorniest issues associated with the Internet: portability and security. Let’s look more closely at each of these.

Java Applets

- An *applet* is a special kind of Java program that is designed to be transmitted over the Internet and automatically executed by a Java-compatible web browser.
- Furthermore, an applet is downloaded on demand, without further interaction with the user.
- If the user clicks a link that contains an applet, the applet will be automatically downloaded and run in the browser.
- Applets are intended to be small programs. They are typically used to display data provided by the server, handle user input, or provide simple functions, such as a loan calculator, that execute locally, rather than on the server.
- In essence, the applet allows some functionality to be moved from the server to the client.

Security

- The code you are downloading might contain a virus, Trojan horse, or other harmful code.
- At the core of the problem is the fact that malicious code can cause its damage because it has gained unauthorized access to system resources
- In order for Java to enable applets to be downloaded and executed on the client computer safely, it was necessary to prevent an applet from launching such an attack.
- Java achieved this protection by confining an applet to the Java execution environment and not allowing it access to other parts of the computer.

Portability

- Portability is a major aspect of the Internet because there are many different types of computers and operating systems connected to it.
- If a Java program were to be run on virtually any computer connected to the Internet, there needed to be some way to enable that program to execute on different systems.
- For example, in the case of an applet, the same applet must be able to be downloaded and executed by the wide variety of CPUs, operating systems, and browsers connected to the Internet.
- It is not practical to have different versions of the applet for different computers. The *same* code must work on *all* computers.
- Therefore, some means of generating portable executable code was needed. As you will soon see, the same mechanism that helps ensure security also helps create portability.

Java's Magic: The Bytecode

- The key that allows Java to solve both the security and the portability problems just described is that the output of a Java compiler is not executable code. Rather, it is bytecode.
- *Bytecode* is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the *Java Virtual Machine (JVM)*.
- The fact that a Java program is executed by the JVM also helps to make it secure. Because the JVM is in control, it can contain the program and prevent it from generating side effects outside of the system.

RAMAIAH
Institute of Technology

Chapter 2 - An Overview of Java

Object-Oriented Programming

Two Paradigms

- All computer programs consist of two elements: code and data.
- Furthermore, a program can be conceptually organized around its code or around its data.
- That is, some programs are written around “what is happening” and others are written around “who is being affected.”
- These are the two paradigms that govern how a program is constructed.
- The first way is called the *process-oriented model*. This approach characterizes a program as a series of linear steps (that is, code).
- The process-oriented model can be thought of as *code acting on data*. Procedural languages such as C employ this model to considerable success.

- To manage increasing complexity, the second approach, called *object-oriented programming*, was conceived.
- Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data.
- An object-oriented program can be characterized as *data controlling access to code*.

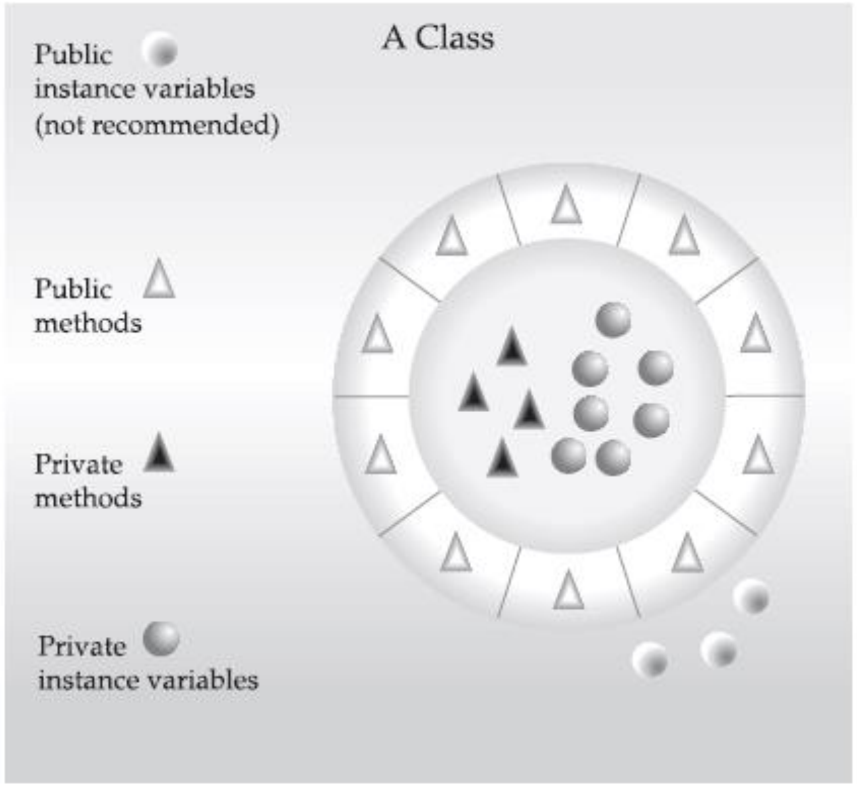
The Three OOP Principles

Encapsulation:- *Encapsulation* is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.

Inheritance:- *Inheritance* is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification.

Polymorphism:- *Polymorphism* (from Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation.

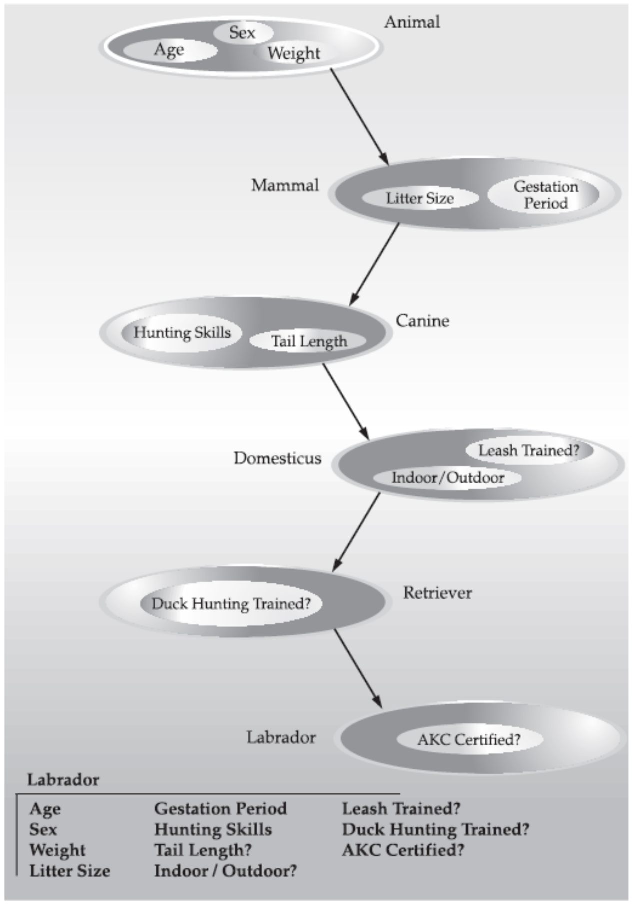
Inheritance


A Class
Public instance variables (not recommended)
Public methods
Private methods
Private instance variables

Encapsulation: public methods can be used to protect private data.

RAMAIAH
Institute of Technology

Inheritance Example



Labrador inherits the encapsulation of all its superclasses.

First Sample Program

```
/*
   This is a simple Java program.
   Call this file "Example.java".
*/
class Example {
  // Your program begins with a call to main().
  public static void main(String args[]) {
    System.out.println("This is a simple Java program.");
  }
}
```

- The first thing that you must learn about Java is that the name you give to a source file is very important. For this example, the name of the source file should be **Example.java**.
- Let's see why. In Java, a source file is officially called a *compilation unit*. It is a text file that contains (among other things) one or more class definitions. (For now, we will be using source files that contain only one class.) The Java compiler requires that a source file use the **.java** filename extension.

- As you can see by looking at the program, the name of the class defined by the program is also **Example**. This is not a coincidence.
- In Java, all code must reside inside a class.
- By convention, the name of the main class should match the name of the file that holds the program.
- You should also make sure that the capitalization of the filename matches the class name.
- The reason for this is that Java is case-sensitive. At this point, the convention that filenames correspond to class names may seem arbitrary. However, this convention makes it easier to maintain and organize your programs.

RAMAIAH
Institute of Technology

Compiling the Program

- To compile the **Example** program, execute the compiler, **javac**, specifying the name of the source file on the command line, as shown here:

```
C:\>javac Example.java
```

The **javac** compiler creates a file called **Example.class** that contains the bytecode version of the program. As discussed earlier, the Java bytecode is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute. Thus, the output of **javac** is not code that can be directly executed.

- To actually run the program, you must use the Java application launcher called **java**. To do so, pass the class name **Example** as a command-line argument, as shown here:

```
C:\>java Example
```

- When the program is run, the following output is displayed:

```
This is a simple Java program.
```

A Closer Look at the First Sample Program

```
public static void main(String args[ ]) {
```

- This line begins the **main( )** method.
- As the comment preceding it suggests, this is the line at which the program will begin executing.
- All Java applications begin execution by calling **main( )**.
- Let's take a brief look at each part now.

- The **public** keyword is an *access modifier*, which allows the programmer to control the visibility of class members.
- When a class member is preceded by **public**, then that member may be accessed by code outside the class in which it is declared. (The opposite of **public** is **private**, which prevents a member from being used by code defined outside of its class.)
- In this case, **main( )** must be declared as **public**, since it must be called by code outside of its class when the program is started.
- The keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class. This is necessary since **main( )** is called by the Java Virtual Machine before any objects are made.
- The keyword **void** simply tells the compiler that **main( )** does not return a value. As you will see, methods may also return values.

A Second Short Program

```
/*
   Here is another short example.
   Call this file "Example2.java".
*/
class Example2 {
  public static void main(String args[]) {
    int num; // this declares a variable called num

    num = 100; // this assigns num the value 100

    System.out.println("This is num: " + num);

    num = num * 2;

    System.out.print("The value of num * 2 is ");
    System.out.println(num);
  }
}
```

When you run this program, you will see the following output:

```
This is num: 100
The value of num * 2 is 200
```

RAMAIAH
Institute of Technology

Two Control Statements

The if Statement

The Java **if** statement works much like the IF statement in any other language. Further, it is syntactically identical to the **if** statements in C, C++, and C#. Its simplest form is shown here:

if*(condition) statement;*

Here, *condition* is a Boolean expression. If *condition* is true, then the statement is executed. If *condition* is false, then the statement is bypassed. Here is an example:

```
if(num < 100) System.out.println("num is less than 100");
```

```
/*
  Demonstrate the if.

  Call this file "IfSample.java".
*/
class IfSample {
  public static void main(String args[]) {
    int x, y;

    x = 10;
    y = 20;

    if(x < y) System.out.println("x is less than y");

    x = x * 2;
    if(x == y) System.out.println("x now equal to y");

    x = x * 2;
    if(x > y) System.out.println("x now greater than y");

    // this won't display anything
    if(x == y) System.out.println("you won't see this");
  }
}
```

The output generated by this program is shown here:

```
x is less than y
x now equal to y
x now greater than y
```

Notice one other thing in this program. The line

```
int x, y;
```

declares two variables, **x** and **y**, by use of a comma-separated list.

The for Loop

The simplest form of the **for** loop is shown here:

for*(initialization; condition; iteration) statement;*

```
/*
  Demonstrate the for loop.

  Call this file "ForTest.java".
*/
class ForTest {
  public static void main(String args[]) {
    int x;

    for(x = 0; x<10; x = x+1)
      System.out.println("This is x: " + x);
  }
}
```

This program generates the following output:

```
This is x: 0
This is x: 1
This is x: 2
This is x: 3
This is x: 4
This is x: 5
This is x: 6
This is x: 7
This is x: 8
This is x: 9
```

Using Blocks of Code

Once a block of code has been created, it becomes a logical unit that can be used any place that a single statement can. For example, a block can be a target for Java's **if** and **for** statements.

Consider this **if** statement:

```
if(x < y) { // begin a block
x = y;
y = 0;
} // end of block
```

Let’s look at another example. The following program uses a block of code as the target of a **for** loop.

```
/*
  Demonstrate a block of code.

  Call this file "BlockTest.java"
*/
class BlockTest {
  public static void main(String args[]) {
    int x, y;

    y = 20;

    // the target of this loop is a block
    for(x = 0; x<10; x++) {
      System.out.println("This is x: " + x);
      System.out.println("This is y: " + y);
      y = y - 2;
    }
  }
}
```

The output generated by this program is shown here:

```
This is x: 0
This is y: 20
This is x: 1
This is y: 18
This is x: 2
This is y: 16
This is x: 3
This is y: 14
This is x: 4
This is y: 12
This is x: 5
This is y: 10
This is x: 6
This is y: 8
This is x: 7
This is y: 6
This is x: 8
This is y: 4
This is x: 9
This is y: 2
```

Lexical Issues

- Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.

Whitespace

- Java is a free-form language. This means that you do not need to follow any special indentation rules. For instance, the **Example** program could have been written all on one line or in any other strange way you felt like typing it, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator. In Java, whitespace is a space, tab, or newline.

Identifiers

Java is case-sensitive, so **VALUE** is a different identifier than **Value**. Some examples of valid identifiers are

AvgTemp	count	a4	$test	this_is_ok

Invalid identifier names include these:

2count	high-temp	Not/ok

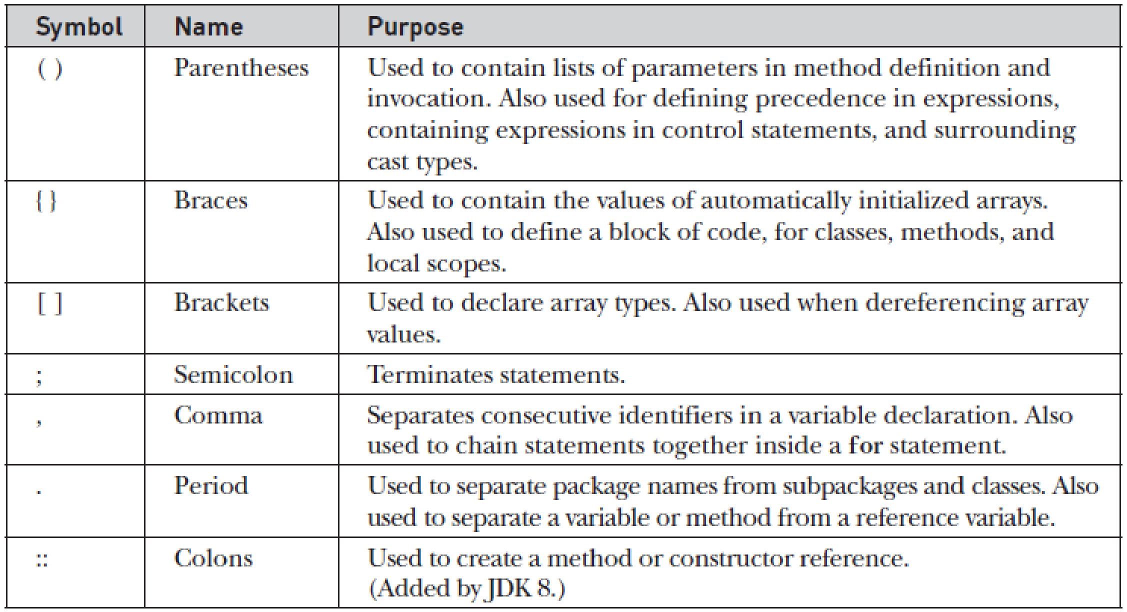
Literals

A constant value in Java is created by using a *literal* representation of it. For example, here are some literals:

100	98.6	'X'	"This is a test"

Separators

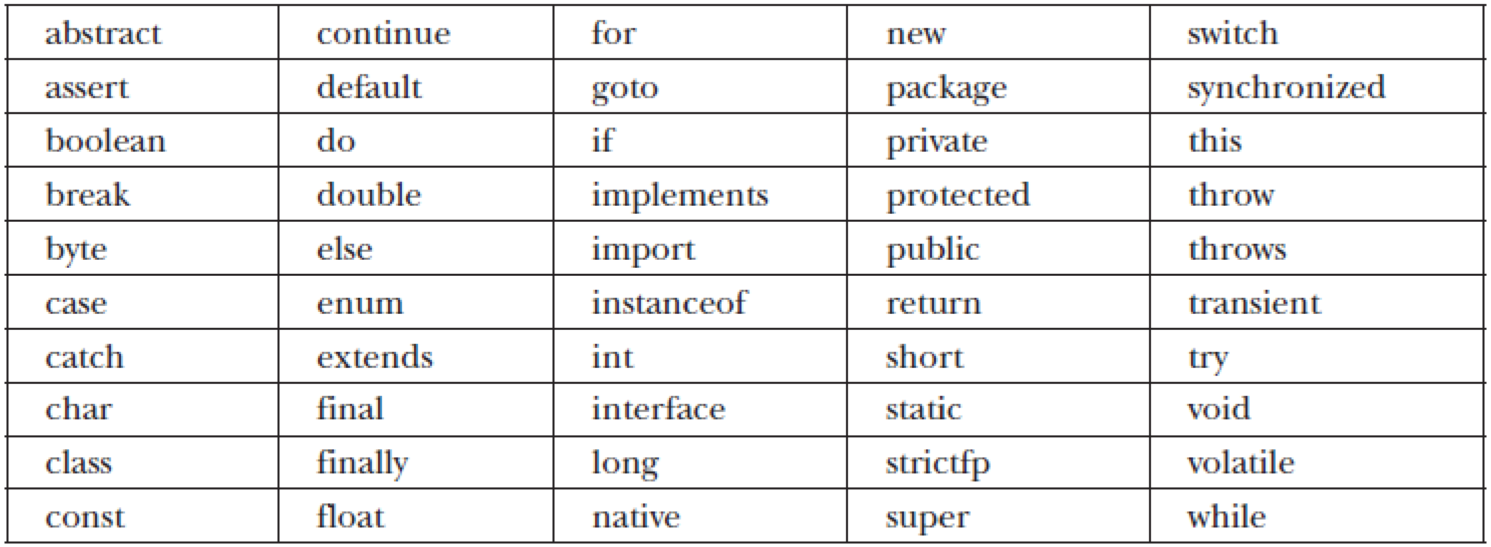
- In Java, there are a few characters that are used as separators. The most commonly used separator in Java is the semicolon. As you have seen, it is used to terminate statements. The separators are shown in the following table:



Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference. (Added by JDK 8.)

The Java Keywords

- There are 50 keywords currently defined in the Java language



abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

The Java Class Libraries

- The sample programs shown in this chapter make use of two of Java's built-in methods: **println( )** and **print( )**.
- These methods are available through **System.out**.
- **System** is a class predefined by Java that is automatically included in your programs. In the larger view, the Java environment relies on several built-in class libraries that contain many built-in methods that provide support for such things as I/O, string handling, networking, and graphics.
- The standard classes also provide support for a graphical user interface (GUI). Thus, Java as a totality is a combination of the Java language itself, plus its standard classes.

Questions !

RAMAIAH
Institute of Technology

Thank You!