

# The history and Overview of Java

Object Oriented Programming

By

Abdul Ghafoor

---

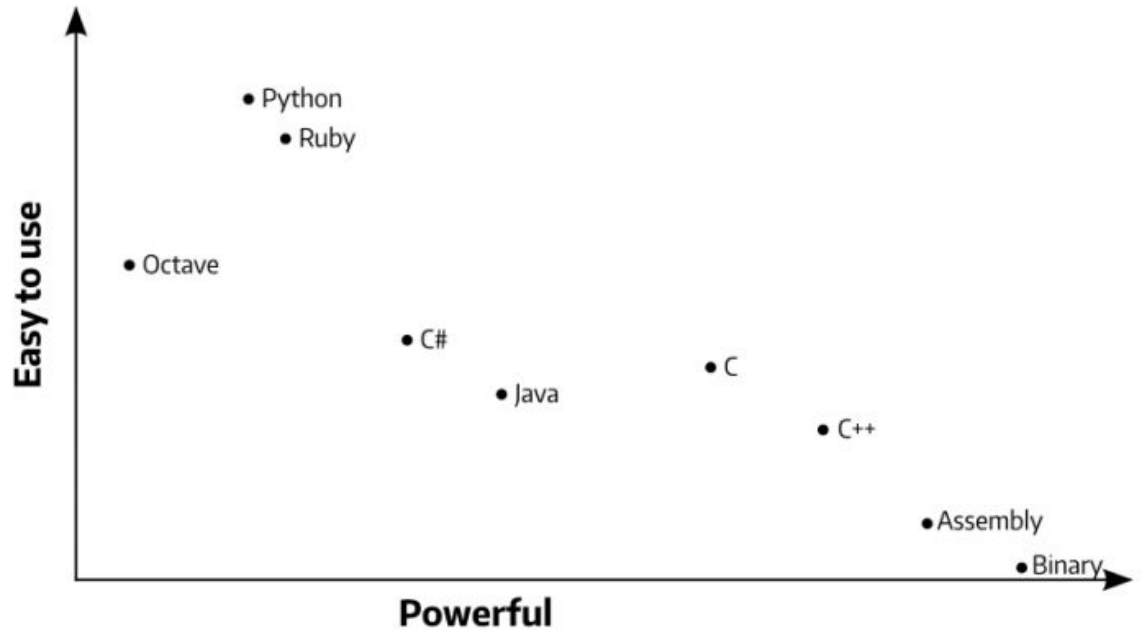
# Introduction to Course

- Assessment (Theory 100 Marks)
  - 30% Mid Term
  - 30% Sessional:
    - 50% Project
    - 50% Quiz and Assignments
- Course Outlines
- 75% Attendance is mandatory for appearing in Final Term Exam, and there will be no compromise on attendance.
- Assignments Should be submitted on due time.
- Quiz will not be retaken.
- During the class, if any one found using cellphone, will get penalty.

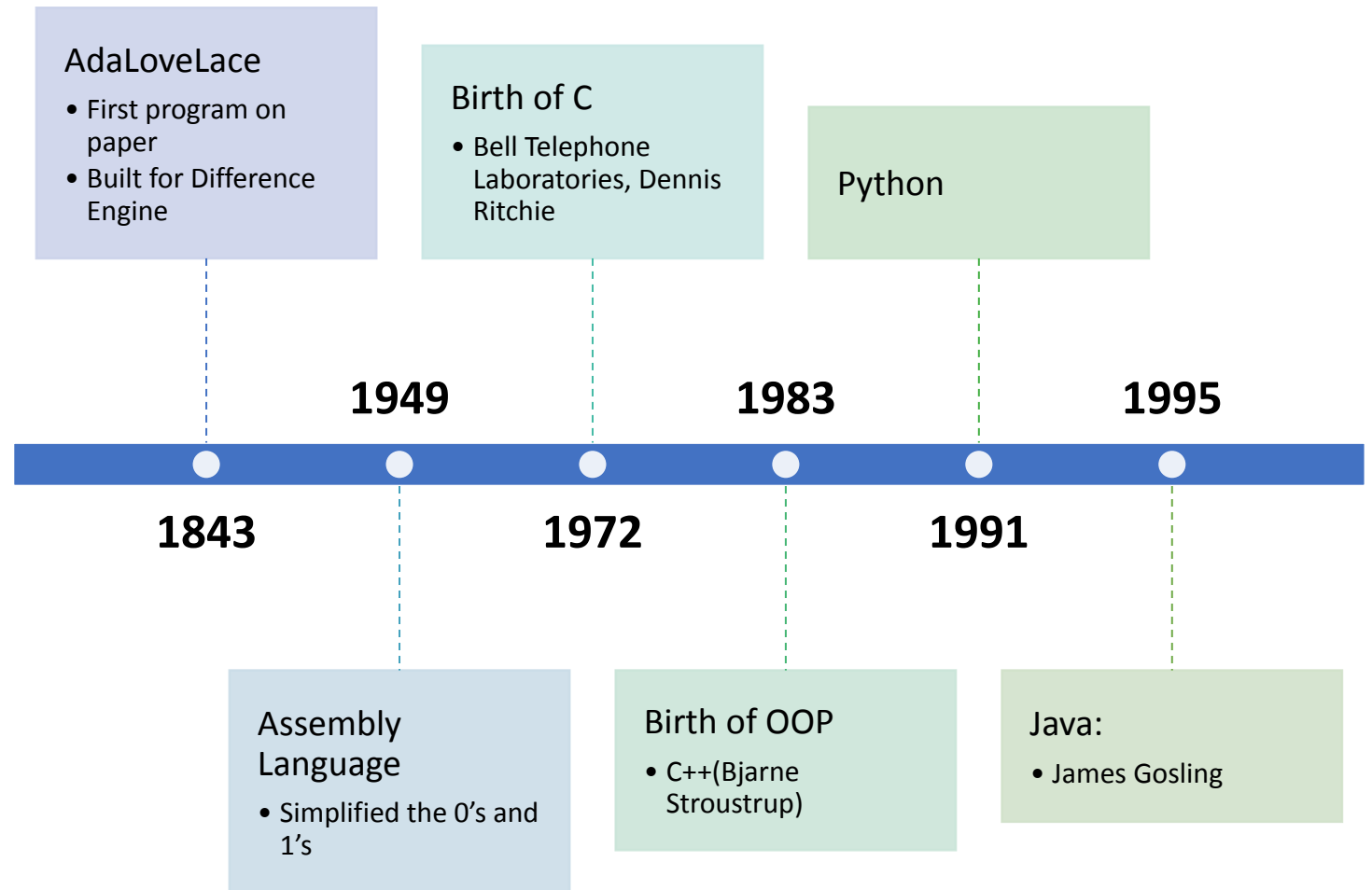
# Trade-offs in language design

---

- Ease of use vs power



# History of Programming



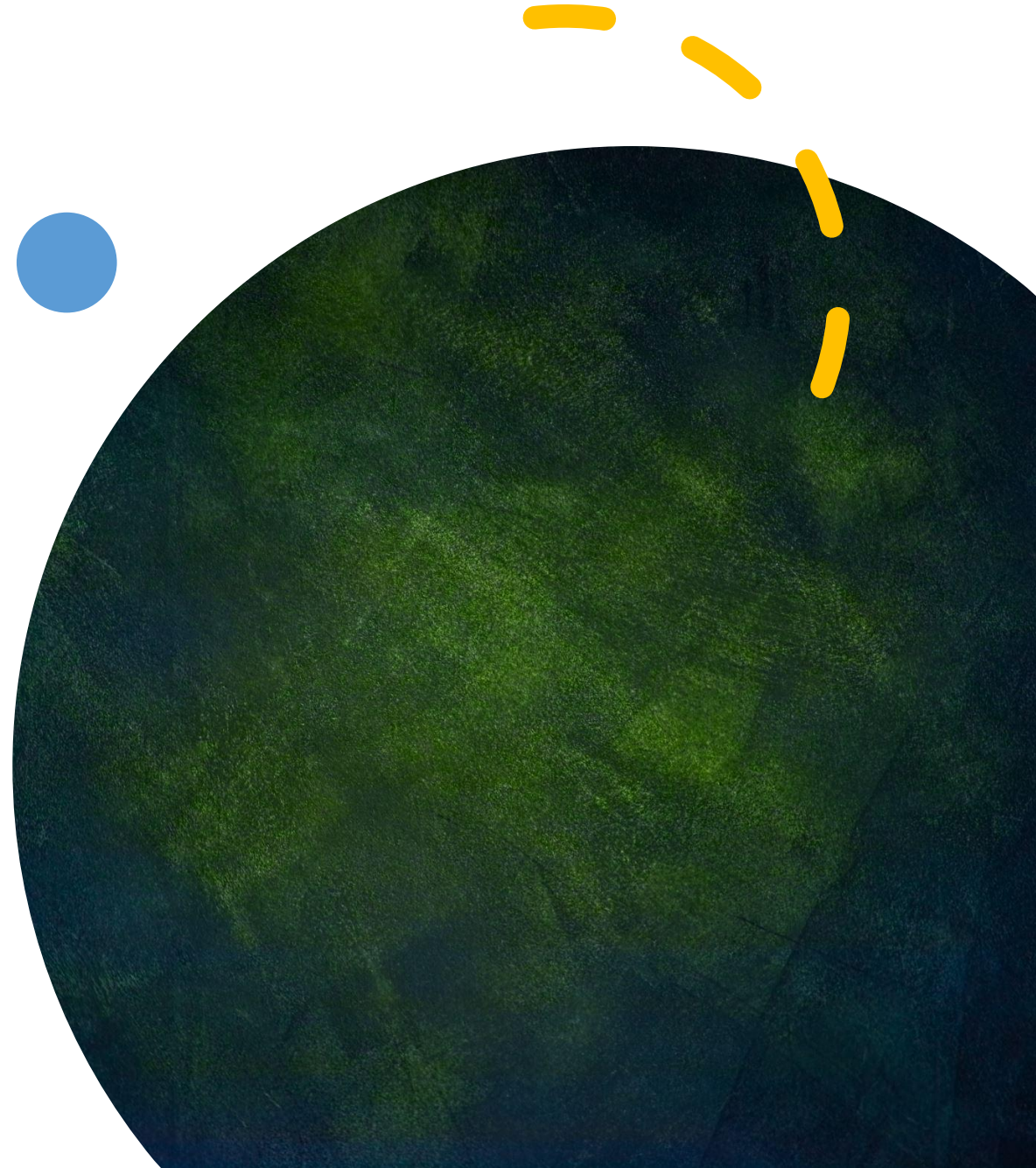
# History of Java

## The Java creation

- 1991 by James Gosling and Co. at Sun Microsystem
- Initially named as Oak, went through a name “Green” and finally in 1995 renamed to Java


## Primary motivation

- Platform independence
- Consumer Electronics





# What is Java?

- Java is a high-level, general-purpose, object-oriented, and secure programming language used for creating wide variety of software programs
- 

# Editions of Java



Java Standard Edition (JSE):

Programs for Desktops



Java Enterprise Edition (JEE):

Programs for servers



Java Micro Edition (JME):

Programs for small devices:

- Set Top Boxes

# Types of Java Applications

Standalone Applications

Enterprise Applications

Web Applications

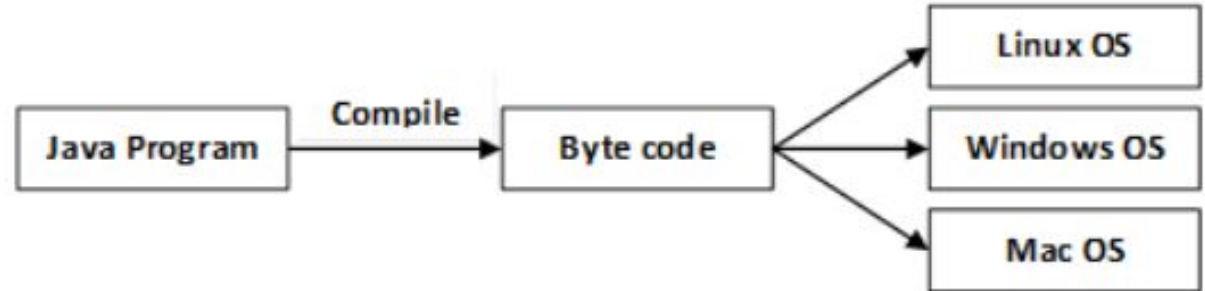
Mobile Applications



# Bytecode — the magic of Java (WORA)

A.java  
A.class

---



# Compilation and Execution of Bytecode

1. Write the java program and saved with .java extention: example Hello.java
2. The Java compiler (javac) converts the source code (A.java) into **bytecode**.
3. When you run the Java program using the command java A, the **JVM**:
  - Converts the bytecode into **machine-specific instructions** using a **Just-In-Time (JIT) compiler**.

# Bytecode:

- Bytecode is an **intermediate representation** of Java code, generated after compiling Java source files. It plays a crucial role in making Java **platform-independent**.
- **File Format:** Bytecode is stored in .class files.
- **Platform Independence:** Bytecode is designed to run on any machine that has a **Java Virtual Machine (JVM)**.

# Just-In-Time Compiler

- The **Just-In-Time (JIT) compiler** is a part of the JVM compile the bytecode into **machine code** at runtime.
- JVM reads .class files (bytecode) generated by the Java compiler and executes them.
  - It translates platform-independent bytecode into platform-specific machine instructions.

# JDK,JRE AND JVM

## JDK:

- Java Development Kit
- Contains all set of necessary tools to work with java

## JRE:

- Java Run-Time Environment
- Includes JVM
  - Environment to execute a java program

## JVM:

- Software in form of interpreter
- Written in C Language
- Executes bytecode

# JVM, JRE AND JDK

## JDK:

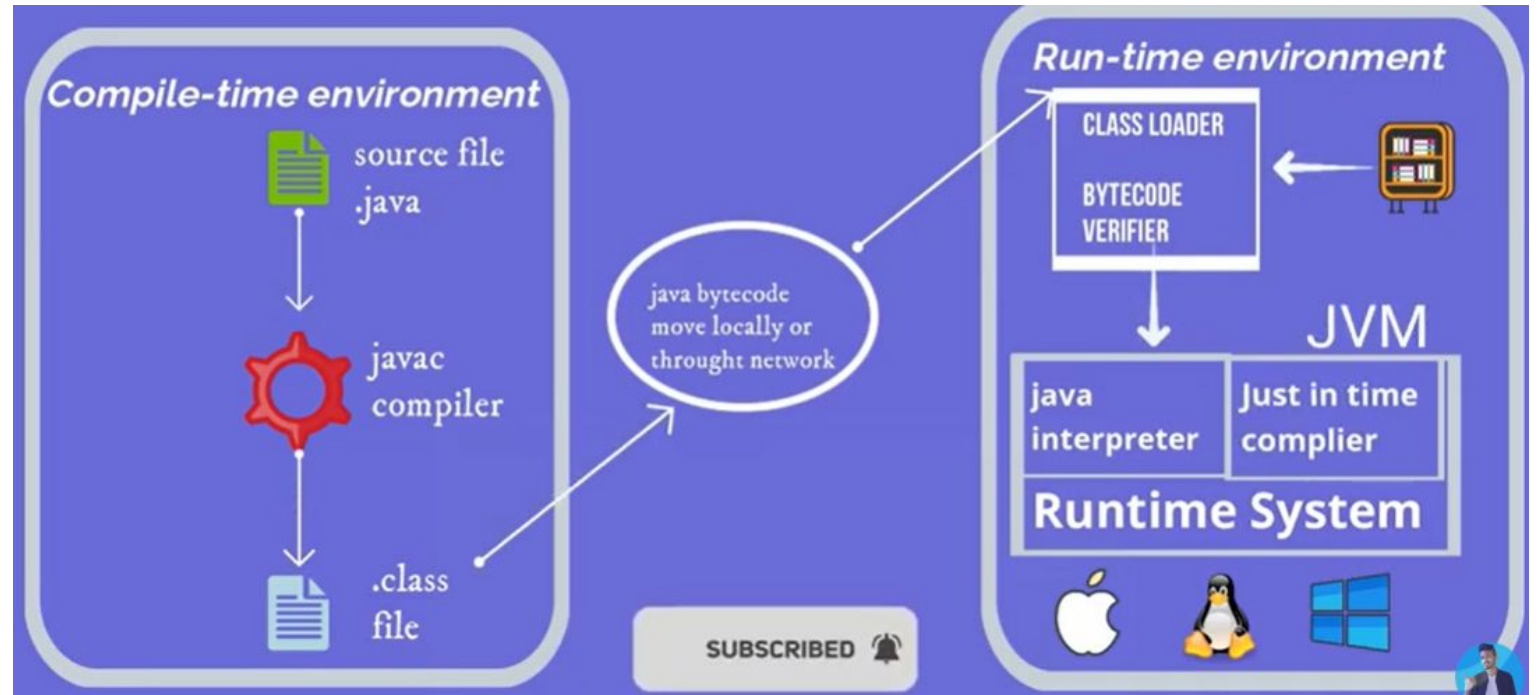
- JRE+JVM+Development Tools
- Programmers and students (Java Developers)

## JRE:

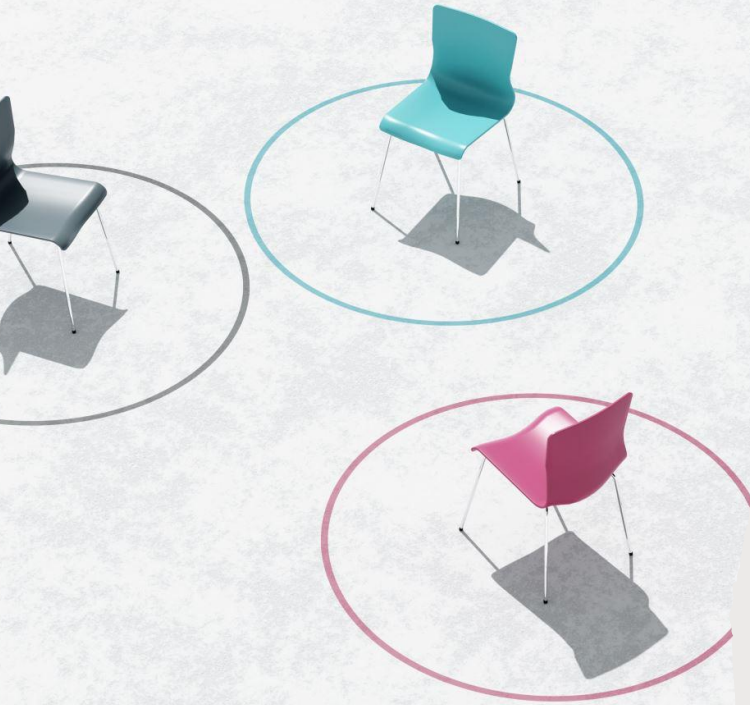
- JVM+Libraries
- Bytecode can be directly executed

# How Java Works?

---



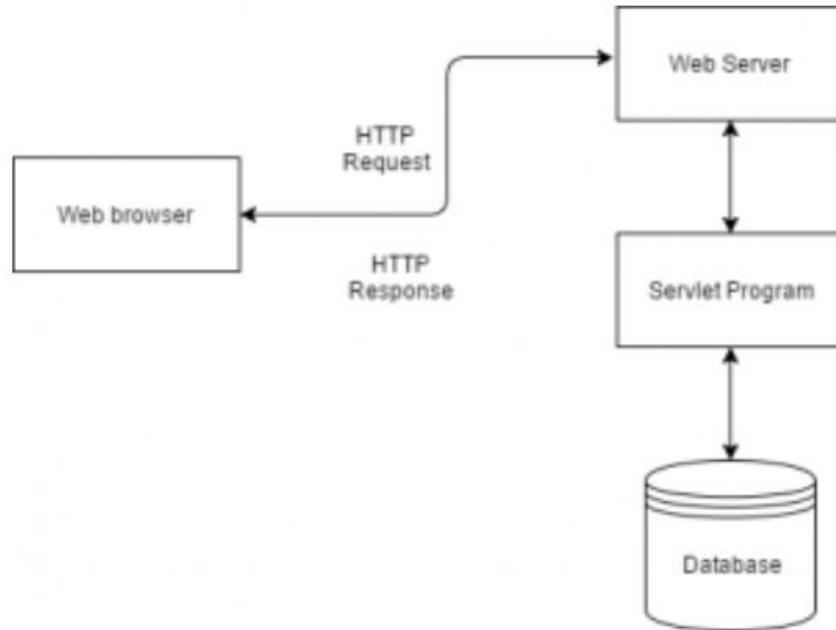
# Java Buzzwords



1. Simple
  - a) Easy to understand and use
  - b) Complex concepts like pointers are eliminated
2. Secure:
  - a) No pointers and memory conflicts
3. Portable
  - a) Bytecode can be carried to any platform
4. Object-oriented
  - a) Every thing is object (Dog, Chair, etc)
5. Robust (memory management, exception handling, reliability)
6. Architecture-neutral (write once; run anywhere, anytime, forever)
7. Interpreted and High Performance (JIT, no compromise on performance)



# Servlets



- Dynamic Web Pages:
  - Generate change content according to request
- Java Servlet allows Dynamic Web Pages
  - Server side programs
  - Handle the request, process and send back the response

# OOP (Programming Paradigm)

- Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around the concept of classes and objects.

# Classes and Objects

## **Object:**

- Object is real world entity:
  - Dog, Chair, Laptop
- Easy to develop programs using real world entities

## **Class:**

- Class is a template or blueprint through which objects of different types are created
  - Animal, Furniture, Computer

# Characteristics of an Object

---

- An object has state, behavior and identity
  - State:
    - Data values of an object, like legs of a dog
  - Behavior:
    - Functionality of an object like barking of dog
  - Identity:
    - Unique id for each object
    - Used by JVM for unique identification



# OOP Principles




Encapsulation

Inheritance

Polymorphism

Abstraction





# Code: “Hello World”

```
class Simple{  
    public static void main(String args[])  
    {  
        System.out.println("Hello Java");  
    }  
}
```

# Code: “VariablesDemo”



print



println



comments



multiline  
comments



block of code

# Concatenate string value with a numeric value

It works in java

The Numeric value is converted to string before the concatenation occurs



# Example

```
/*
    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);
    }
}
```



# Introduction of Repetitions

---

```
/*  
    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
```

# Block of code

- Two or more statements form a code block

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

# Lexical Issues

- A program is collection of Lexicons:
  - Whitespaces
  - Identifiers
  - Literals
  - Comments
  - Operators
  - Separators
  - Keywords



# Collection of Lexicons

**Whitespaces:** Spaces, tabs, and newlines used to separate tokens in a program. They don't affect the program's logic but improve readability

**Identifiers:** Names given to variables, classes, methods, or objects to uniquely identify them.

- Class name: Student
- Variable name: age
- Method name: calculateMarks

**Literals:** Fixed values assigned to variables, such as numbers, characters, strings, or booleans.

- Numeric literal: 42
- String literal: "Hello"
- Boolean literal: true

# Collection of Lexicons

**Comments:** Descriptions or explanations in the program to improve readability and understanding, ignored by the compiler.

- Single-line comment: `// This is a comment`
- Multi-line comment: `/* This is a multi-line comment */`

**Operators:** Symbols used to perform operations on variables or values.

- Arithmetic operator: `+` (addition)
- Relational operator: `>` (greater than)
- Logical operator: `&&` (AND)

**Separators:** Special characters used to separate blocks of code or elements within a program.

- Parentheses: `()` for method calls
- Curly braces: `{}` for blocks of code
- Comma: `,` for separating arguments


# Collection of Lexicons

**Keywords:** Reserved words in a programming language that have predefined meanings and cannot be used as identifiers.

- class, public, if, while



# Whitespace

- Java is a free-form language
    - Don't worry about line breaks
    - Start on one line, end several lines below
  - Whitespace include,
    - Space
    - Tab
    - Newline
- 



# Identifiers

- Names of variables, methods, classes etc..
- Allowed
  - Uppercase lowercase letter
  - Numbers (must not be first character)
  - Dollar sign (\$)
  - Underscore (\_)

# Literals

100	98.6	'X'	"This is a test"
-----	------	-----	------------------

# Comments

- Single line `//`
- Multiline `/* */`
- Documentation comments `/** */`



# Separators

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 <b>for</b> 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.
...	Ellipsis	Indicates a variable-arity parameter.
@	Ampersand	Begins an annotation.

# Keywords

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

**Table 2-1** Java Keywords

# Java Class Libraries

- java.lang.\*
- System class



# Scanner

- The **Scanner class** in Java is part of the `java.util` package. It is widely used to read input from various sources, such as the keyboard (`System.in`), files, or strings.
  - It can read input as various data types, such as `int`, `float`, `double`, `String`, etc.

# Scanner Code

```
import java.util.Scanner;

public class sc {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Integer input
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Character input
        System.out.print("Enter a character: ");
        char character = scanner.next().charAt(0);

        // String input
        System.out.print("Enter a sentence: ");
        String sentence = scanner.nextLine();

        scanner.close(); // Close the scanner to free
resources
    }
}
```











