INTRODUCTION TO JAVA

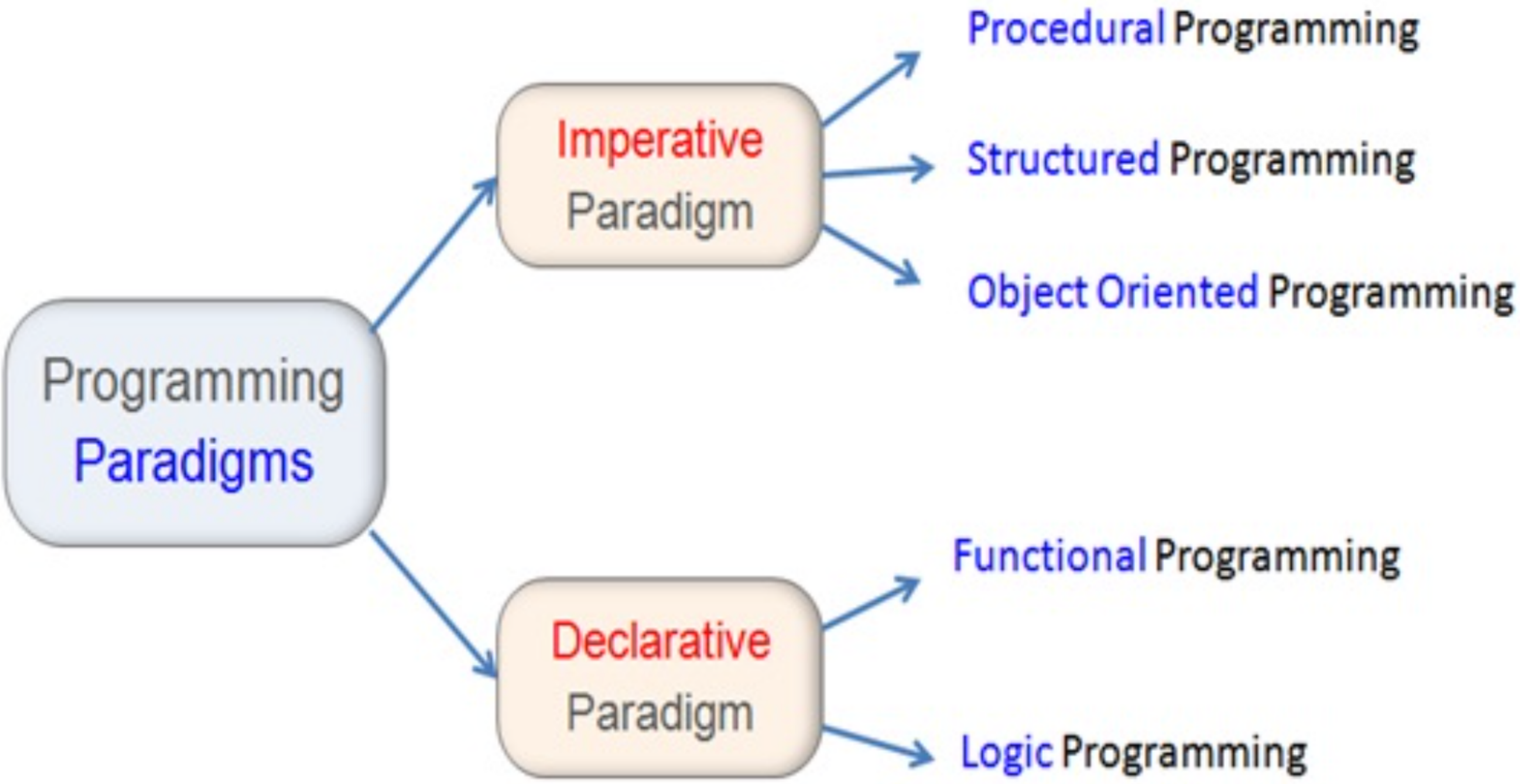
Java programming fundamentals

- Introduction to Java
- Overview of JDK/JRE/JVM
- Java Language Constructs
- Object Oriented Programming with Java
- Exception Handling

JAVA BACKGROUND AND HISTORY

Intro to Programming Language Paradigms

Programming paradigms are a way to classify programming languages based on their features
Imperative Paradigm - programmer instructs the machine how to change its state
Declarative Paradigm - programmer declares properties of the desired result, but not how to compute it



What is Java and it's Background?

Java is a high-level object-oriented programming language with platform independent deployment.

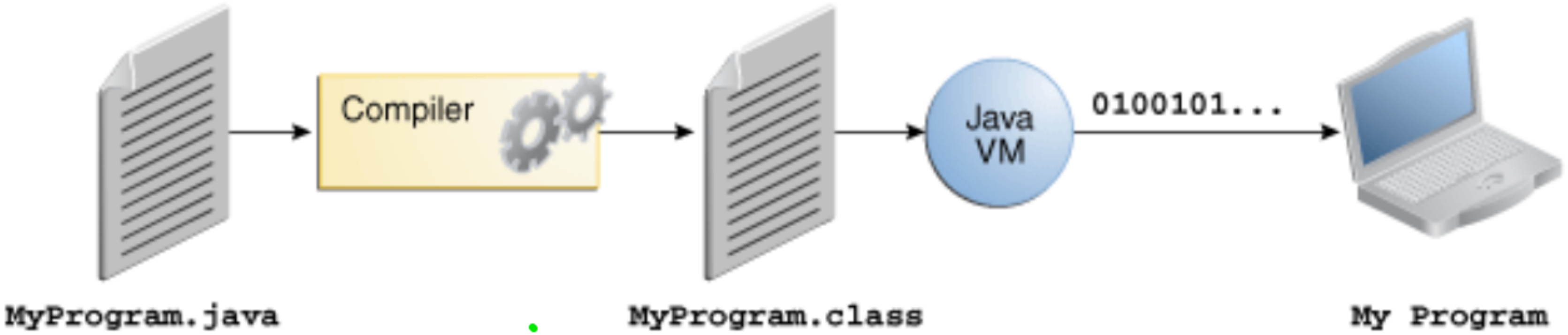
- Project started on 1991 by Sun Microsystems
- Developed by James Gosling with support from Mike Sheridan, Patrick Naughton
- Initially names as Oak
- v1.0 released on 1996
- JVM become open source on 2006/07 under FOSS (Free & Open Source Software)
- Oracle acquired Sun Microsystems and become owner of Java on 2009/10
- Latest version 23 and LTS versions are 8, 11, 17 and 21

Java Design Goals

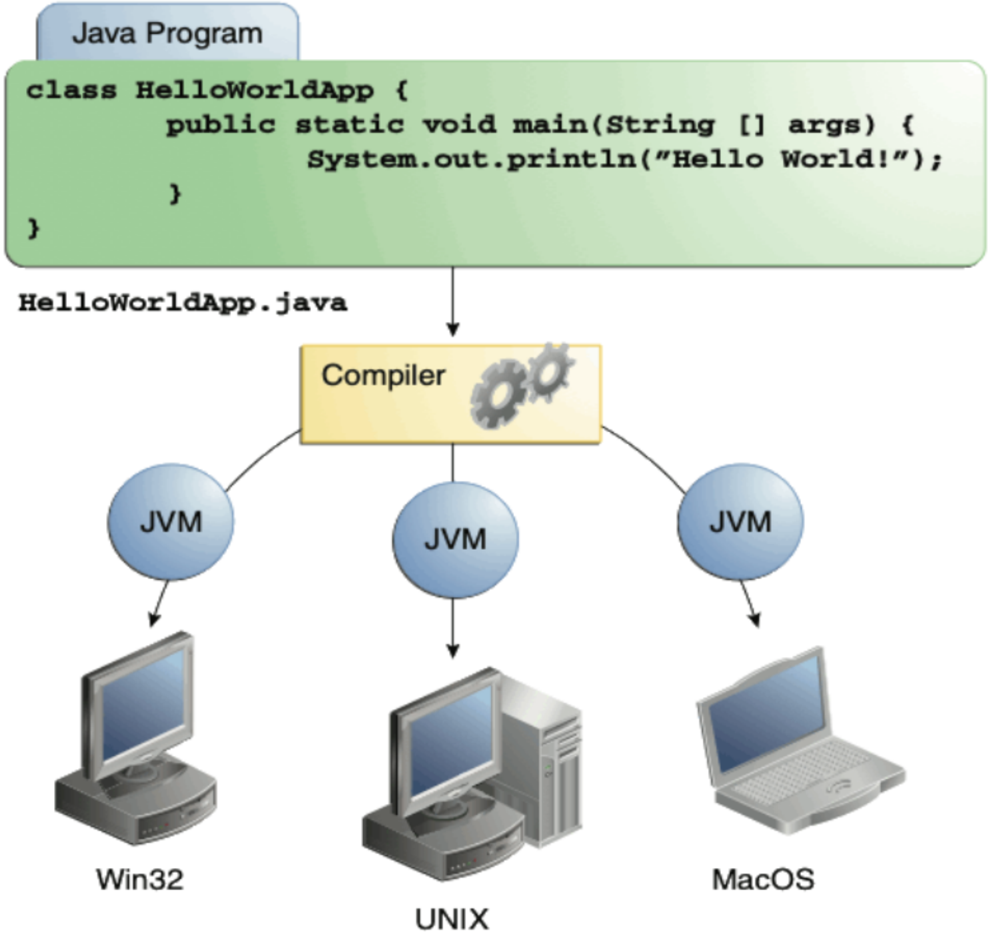
- simple, object oriented, familiar
- robust and secure
- architectural neutral and portable
- high performance (JIT)
- interpreted, threaded and dynamic

Java Characteristics / Features

- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High performance
- Robust
- Secure



Java is Platform Independent



```
class HelloWorldApp {
        public static void main(String [] args) {
                System.out.println(”Hello World!”);
        }
}
```

Java Release History

- v1.0 -> 1996
- v1.1 -> 1997
- v1.2 -> 1998 => J2SE, J2EE, J2ME
- v1.3 -> 2000
- v1.4 -> 2002
- v5.0 -> 2004 => JSE, JEE, JME
- v6.0 -> 2006
- v7.0 -> 2011
- v8.0 -> 2014 (LTS) => OOP + FP (Lambda Expr + Stream API)
- v9.0 -> 2017
- v10 -> 2018(Mar)
- v11 -> 2018(Sep) (LTS)
- v12 -> 2019(Mar)
- v13 -> 2019(Sep)
- v14 -> 2020(Mar)
- v15 -> 2020(Sep)
- v16 -> 2021(Mar)
- v17 -> 2021(Sep) (LTS)
- v18 -> 2022(Mar)
- v19 -> 2022(Sep)
- v20 -> 2023(Mar)
- v21 -> 2023(Sep)
- v22 -> 2024(Mar)
- v23 -> 2024(Sep)

Java Flavors

- Java SE (Standard Edition)
- Java EE (Enterprise Edition) / Jakarta EE - Servlet, JSP, EJB, JAX-RS, etc..
- Java ME (Micro Edition)

Language Features

- Object-oriented
- Platform independent
- Multi-threaded
- Auto memory management
- Robust
- Secure
- Dynamic binding
- Interfacing & enhancing legacy code
- Distributed Computing

Features: Object-oriented

- Programming Methodologies:
  - Programming around code (Structured Approach)
  - Programming around data (OO Approach)
- OO Approach is more realistic & natural.
- Objects comprise state (data) & behavior (methods).
- Objects encapsulate data.
- An Object may use features of another object.
- Objects show polymorphic behavior.

Features: Platform Independent

- Java is based on the concept of WORA.
- Java code requires both compiler & interpreter.
- Java Compiler produces byte code file (.class file).
- Byte code is meant for JVM, not for real machine.
- JVM is specific to a platform & produces platform-specific machine code.

Features: Multi-threaded

- Multi-threading helps in achieving Multi-tasking.
- Java has language-level support of Multi-threading.
- A thread is an independent path of execution.
- Multi-threading saves wastage of CPU cycles. It makes the application more productive & responsive.

Features: Auto Memory Management

- In Java, we don’t need to care of de-allocation of garbage (Un-referenced objects).
- JVM delegates the job of garbage collection to a thread, called garbage collector.
- Working of garbage collector is monitored & controlled by JVM itself.

Features: Robust

- Java is a strongly typed language (that is, all variables must be assigned an explicit data type).
- Java has language-level support for exception handling.
- Java automatically checks the array boundary. It’s not the case of its predecessors.
- From the beginning, Java was designed to make certain kinds of attacks impossible, among them:
    - Overrunning the runtime stack—a common attack of worms and viruses
    - Corrupting memory outside its own process space
    - Reading or writing files without permission

Features: Secure

- Elimination of direct memory pointers & automatic array limit checking prevents rogue programs from reaching into sections of memory where they shouldn’t.
- Untrusted programs are restricted to run inside the virtual machine. Access to the platform can be strictly controlled by a security manager.
- Code is checked for pathologies by a class loader and a bytecode verifier.

Features: Interfacing & Enhancing Legacy Code

- Java’s strong graphics and networking capabilities can be applied to existing C programs.
- A Java graphical user interface (GUI) can bring enhanced ease of use to a C program, which then acts as a computational engine behind the GUI.

A very basic Java Application

```
class HelloWorld{
 public static void main(String [] argv) {
               System.out.print(“Welcome to Java");
 }
}
```

Steps:-

1. Save the source file as HelloWorld.java
2. On the command line, compile the source file

```
javac HelloWorld.java
```

3. Execute the class

```
java HelloWord
```

Understanding the main() method

main(String [] argv) method is the entry point for all Java applications.

- An application must have a class definition that includes a main(String [] argv) method.
- We execute the application by typing java at the command line, followed by the name of the class which contains the main method.
- argv refers to a 1-D array of String type. It’s generally used to retrieve command-line arguments.

Understanding Path & Classpath

- Path refers to the file-system location of an executable file.

  syntax:-

  ```
  set path = %path%;c:\program files\Java\jdk21\bin
  ```

- Classpath refers to the file-system location of a .class file or .jar file.

  syntax:-

  ```
  set classpath = %classpath%;c:\JavaPrograms
  ```

Java Benefits

- Get started quickly
- Write less code
- Write better code
- Develop programs more quickly
- Avoid platform dependencies
- Write once, run anywhere (WORA)
- Distribute software more easily

JAVA KEYWORDS

Keywords

- *Keywords* are special reserved words in Java that you cannot use as identifiers (names) for classes, methods, or variables.
- These have meaning to the compiler; it uses them to figure out what your source code is trying to do.

Java Keywords

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

Cont….

- **Reserved Literals in Java**

- null
- true
- false

- **Reserved Keywords not Currently in Use**

- const
- goto

JAVA BASICS

Language Basic Constructs

- Data Types
- Variables
- Constants
- Operators
- Expressions, Statements, Blocks
- Control Flow Statements
- Loop Statements
- Branching Statements
- Naming Conventions
- Comments
- Arrays
- Strings

Primitive Data Types

Data Types	Size in Bytes
byte	1
short	2
int	4
long	8
float	4
double	8
char	2
boolean	1/8

Range of Integer Values

Data Type	Width (bits)	Minimum value MIN_VALUE	Maximum value MAX_VALUE
byte	8	-2^{7} (-128)	$2^{7}-1$ (+127)
short	16	-2^{15} (-32768)	$2^{15}-1$ (+32767)
int	32	-2^{31} (-2147483648)	$2^{31}-1$ (+2147483647)
long	64	-2^{63} (-9223372036854775808L)	$2^{63}-1$ (+9223372036854775807L)

Character Type

Range of Character Values

Data Type	Width (bits)	Minimum Unicode value	Maximum Unicode value
char	16	0x0 (\u0000)	0xffff (\uffff)

Floating-point Types

Range of Floating-point Values

Data Type	Width (bits)	Minimum Positive Value MIN_VALUE	Maximum Positive Value MAX_VALUE
float	32	1.401298464324817E-45f	3.402823476638528860e+38f
double	64	4.94065645841246544e-324	1.79769313486231570e+308

Boolean Type

Boolean Values

Data Type	Width	True Value Literal	False Value Literal
boolean	not applicable	true	false

Identifiers

- Identifiers are programmer defined tokens.
- They are used for naming classes, methods, variables, packages, and interfaces in a program.

- **Rules for a Legal Identifier:-**
    - They can be a set of alphabet, digit, underscore and dollar characters.
    - They must not begin with digit.
    - Uppercase and Lowercase letter are distinct.
    - They can be of any length.

Literals

- A Java literal is a sequence of characters (digits, letters, and other characters) that represent constant value to be stored in variables.
- Java specifies five major types of literals:-
  - Integer Literals
  - Floating_point Literals
  - Character Literals
  - String Literals
  - Boolean Literals

Integer Literals

- There are three ways to represent integer numbers in the Java language:
  - Decimal (base 10)
  - Octal (base 8)
  - Hexadecimal (base 16)

Ex:-

198 (in decimal)

010 (in octal, equal to 8 in decimal)

0xA (in hex, equal to 11 in decimal)

Floating-Point Literals

- Floating-point numbers are defined as a number, a decimal symbol, and one or more numbers representing the fraction.

  For example:-

  double d = 11301874.9881024;

- Floating-point literals are defined as double (64 bits) by default, so if you want to assign a floating-point literal to a variable of type float (32 bits), you *must* attach the suffix *F* or *f* to the number.

Boolean Literals

- Boolean literals are the source code representation for boolean values. A boolean value can only be defined as true or false.

- Although in C (and some other languages) it is common to use numbers to represent true or false, *this will not work in Java*.

```
boolean t = true; // Legal

boolean f = 0; // Compiler error!
```

Character Literals

- A character literal is represented by a single character in single quotes.

  char a = 'a';

  char b = '@';

- You can also type in the Unicode value of the character, using the Unicode notation of prefixing the value with \u .

  For Example :

  char ch = '\u0041'; // The letter ‘A'

Character Literals contd.

- Remember, characters are just 16-bit unsigned integers under the hood. That means you can assign a number literal, assuming it will fit into the unsigned 16-bit range (65535 or less). For example, the following are all legal:

```
char a = 0x892; // hexadecimal literal
char b = 982; // int literal
char c = (char) 70000; // The cast is required; 70000 is out of
                             // char range
char d = (char) -98; // Ridiculous, but legal
```

And the following are not legal and produce compiler errors:

```
char e = -29; // Possible loss of precision; needs a cast
char f = 70000 // Possible loss of precision; needs a cast
```

String Literals

- In Java, string literals are enclosed within double quotes. For example:-

  "hpes "

- String literals are not only a sequence of characters as in C, these are objects of type java.lang.String.
- Java stores all string literals in String Constant Pool.

Escape Sequences

Escape Sequence	Unicode Value	Character
\b	\u0008	Backspace (BS)
\t	\u0009	Horizontal tab (HT or TAB)
\n	\u000a	Linefeed (LF) a.k.a., Newline (NL)
\f	\u000c	Form feed (FF)
\r	\u000d	Carriage return (CR)
\'	\u0027	Apostrophe-quote
\"	\u0022	Quotation mark
\\	\u005c	Backslash

Comments

- Comments are used to provide description for some section of code.
- Comments are not compiled by the compiler.
- 3 types of comments in Java.

1. Documentation comment:-

```
/**
* …..
*……
*/
```

2. Single line comment

```
//…………….
```

3. Multi line comment

```
/*…………………
……………….
………………..*/
```

Declaring & Initializing variables

- Variables in java can be of
  - primitive type
  - class type
  - interface type.
- **Syntax:-**

  type identifier=literal;
- Variables can be
  - Static
  - Non-static
  - Local

Declaring constants

- final keyword in java is used to declare constants.
- Variables once declared final don’t allow their value to be change after initialization.
- Syntax:-

```
final float PI = 3.14F;
```

- It a coding convention to use uppercase for constants.

Object Oriented Programming and Related Concepts

- Class
- Object
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

- Interface
- Package
- Wrapper Classes
- Object Class
- Methods
- Access Modifiers

Implementing Object-Oriented Concept: Defining a class

- Features of object-oriented programming are:-
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction
- A Class is used to implement these concepts.
- An object-oriented program can’t do without objects. A class is a blueprint for one or more objects.

Defining a class: An example

```
public class Employee{
    private String name;
    private float salary;
    public void setInfo(String name, float salary){
        //your code goes here
    }
    public String getName(){
        //your code goes here
    }
    public Float getSalary(){
        // your code goes here
    }
}
```

Instantiating a class: Creating Objects

- A class is just a blueprint. It’s of no much use until we create object(s).
- Syntax:-

```
Employee emp = new Employee();
Emp.setInfo("John",5000.0f);
```

- In the Employee class; name, salary, setInfo(), getName(), getSalary() are members of the class.
- Access to class members is specified using access specifers.
- Members defined in a class can either be static or non-static.

Access Specifiers

Visibility	public	protected	default	priv
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, through inheritance	No	No
From any non-subclass class outside the package	Yes	No	No	No

Thank You!