Object-Oriented Programming
2305CS102

Object Oriented Programming

Prof. Swati R Sharma
Computer Engineering Department
Darshan University, Rajkot

✉ swati.sharma@darshan.ac.in
☎

Teaching and Examination Scheme

Teaching Scheme (Contact Hours)				Examination Scheme				
Lecture	Tutorial	Lab	Credit	Theory Marks		Practical Marks		Total Marks
				SEE	CIA	SEE	CIA	
4	0	2	5	40	30	20	10	100

SEE *- Semester End Examination,* ***CIA*** *- Continuous Internal Assessment (It consists of Assignments/Seminars/Presentations/MCQ Tests, etc.)*

Syllabus

Syllabus

Sr.	Content	T Hrs	W %
1	Introduction to JAVA	10	21
2	Object Oriented Concepts	15	23
3	Inheritance and Abstraction	12	15
4	Package, Exception Handling and Multithreading	13	23
5	IO Programming and Collection Framework	10	18

Object-Oriented Programming Using Java
2305CS102

Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Unit-1
Introduction to java

Prof. Swati R Sharma
Computer Engineering Department
Darshan University, Rajkot

✉ swati.sharma@darshan.ac.in
☎

What we will learn

- ✓ Evolution of Java
- ✓ Features of JAVA,
- ✓ JDK, JRE, JVM, Byte Code
- ✓ Installing and configuring JAVA
- ✓ Basic Structure of Java program
- ✓ Compiling and Running Java files

Unit-1

Introduction

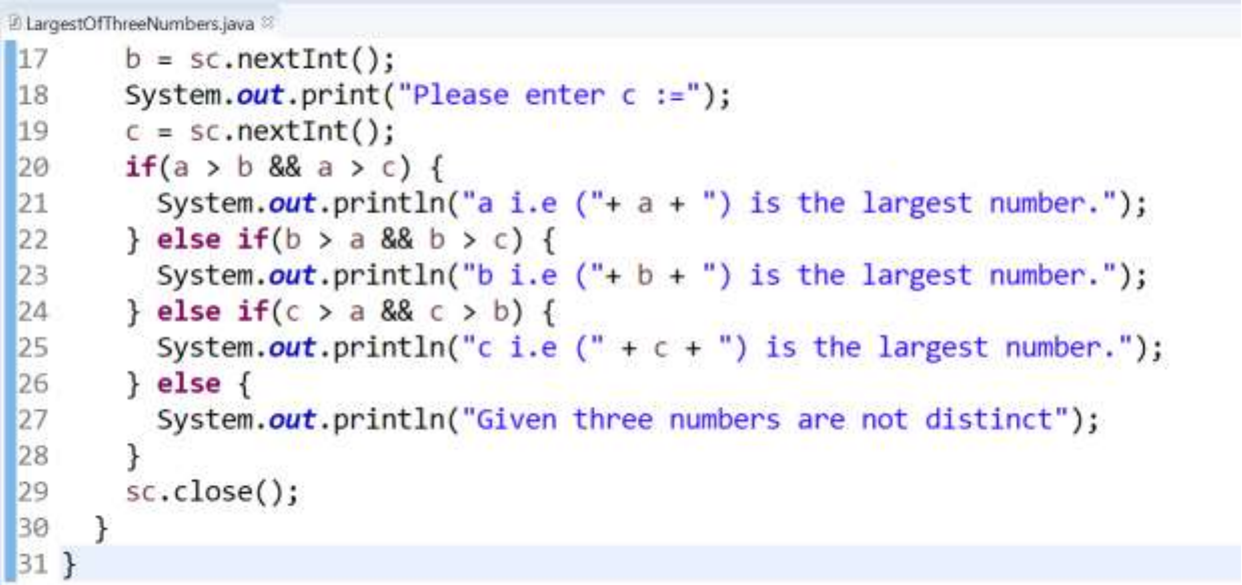
Welcome to the world of Programming

What is Language?

- A language is a system of communication used by a particular country or community.
- It is a structured system of communication used by humans, based on speech and gesture (spoken language), sign or often writing.
- Both person should understand each other’s language.

Programming Language

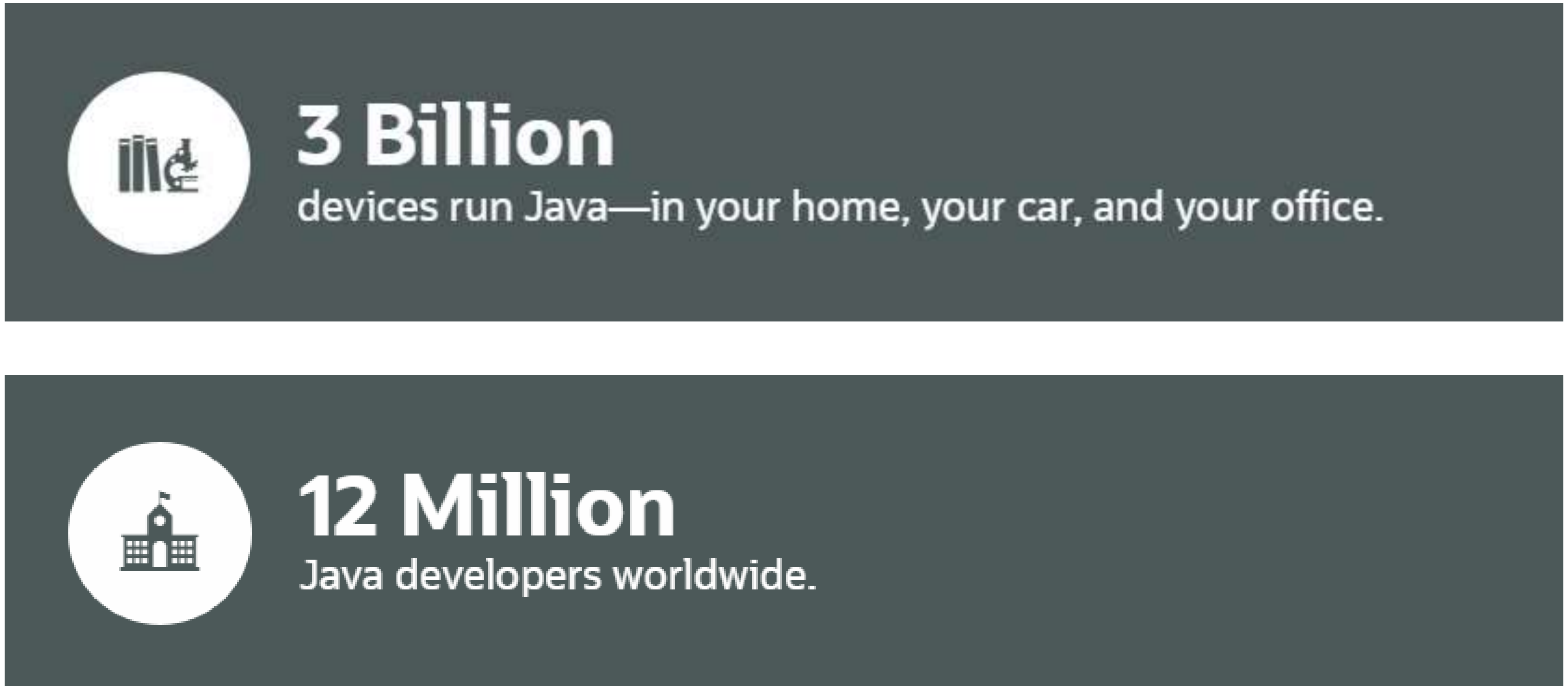
- A programming language is a computer language that is used by programmers (developers) to communicate with computers.
- It is a set of instructions written in a specific language to perform a specific task.
- It allow us to give instructions to a computer in a language the computer understands.
- 5000+ programming languages are there, notably used are approximate 250.
- E.g. C, C++, C#, Java, Python, PHP, Pascal, etc.



```
        b = sc.nextInt();
        System.out.print("Please enter c :=");
        c = sc.nextInt();
        if(a > b && a > c) {
          System.out.println("a i.e ("+ a + ") is the largest number.");
        } else if(b > a && b > c) {
          System.out.println("b i.e ("+ b + ") is the largest number.");
        } else if(c > a && c > b) {
          System.out.println("c i.e (" + c + ") is the largest number.");
        } else {
          System.out.println("Given three numbers are not distinct");
        }
        sc.close();
    }
}
```

Types of Programming Languages

1. **Low-level programming language**
   - It is a machine-dependent (hardware specific) programming language.
   - It consists of a set of instructions that are either in the binary form (0 or 1) or in a symbolic and human-understandable mnemonics (ADD, MOV, SUB).
   - E.g. Machine level language, Assembly language, etc.
2. **High-level programming language**
   - It is closer to human languages than machine-level languages.
   - It is easy to read, write, and maintain as it is written in English like words.
   - It allows to write the programs which are independent of a particular type of machine (hardware).
   - A compiler is required to translate a high-level language into a low-level language.
   - E.g. Python, Java, JavaScript, PHP, C#, LISP, FORTRAN, etc.
3. **Middle-level programming language**
   - Middle-level programming language lies between the low-level and high-level programming language.
   - E.g. C, C++, etc.



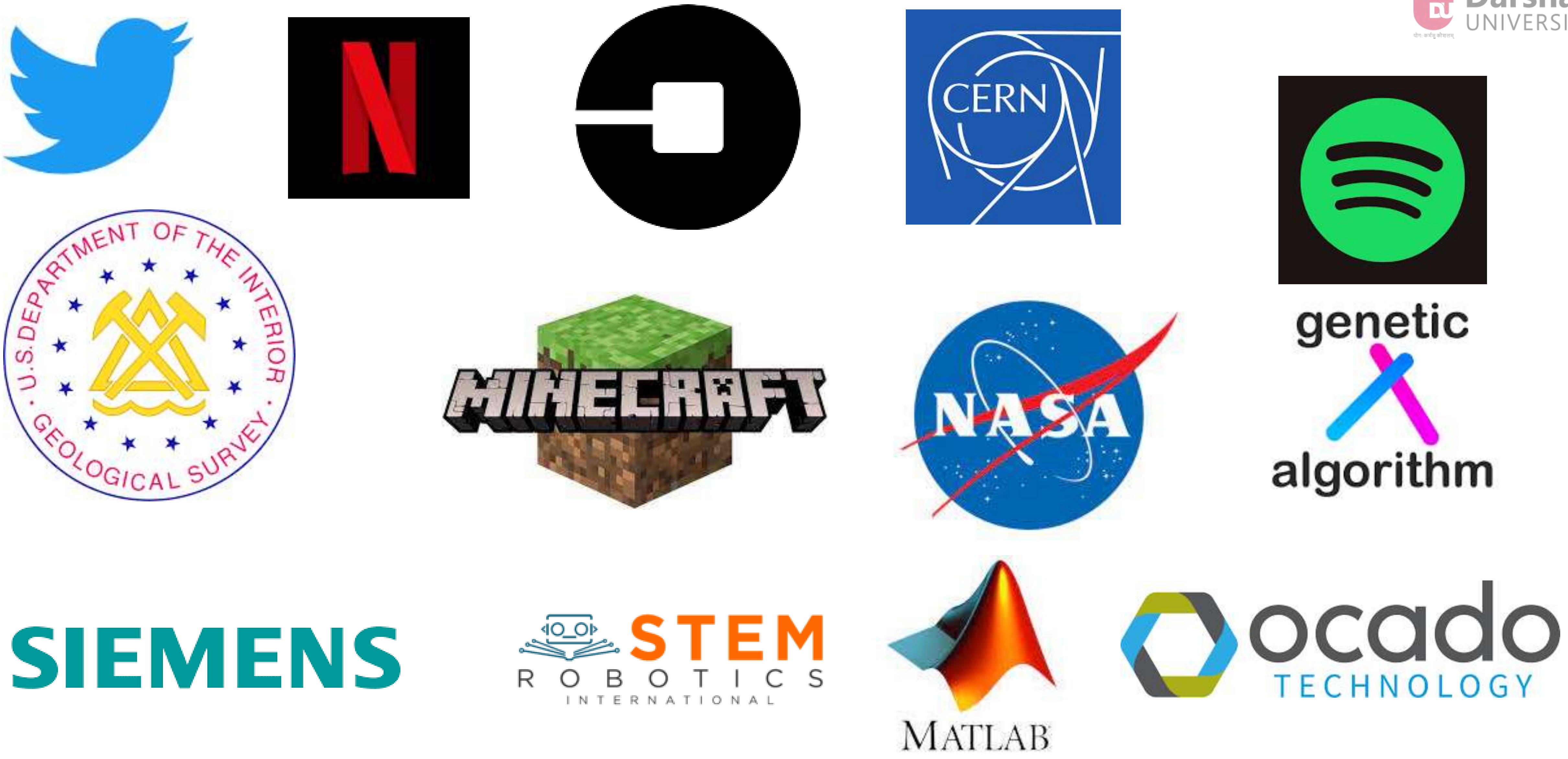
3 Billion

devices run Java—in your home, your car, and your office.

12 Million

Java developers worldwide.

Applications implemented in Java


N
CERN
U.S. DEPARTMENT OF THE INTERIOR · GEOLOGICAL SURVEY ·
MINECRAFT
NASA
genetic algorithm
SIEMENS
STEM ROBOTICS INTERNATIONAL
MATLAB
ocado TECHNOLOGY

Introduction to Java

JAVA Installation

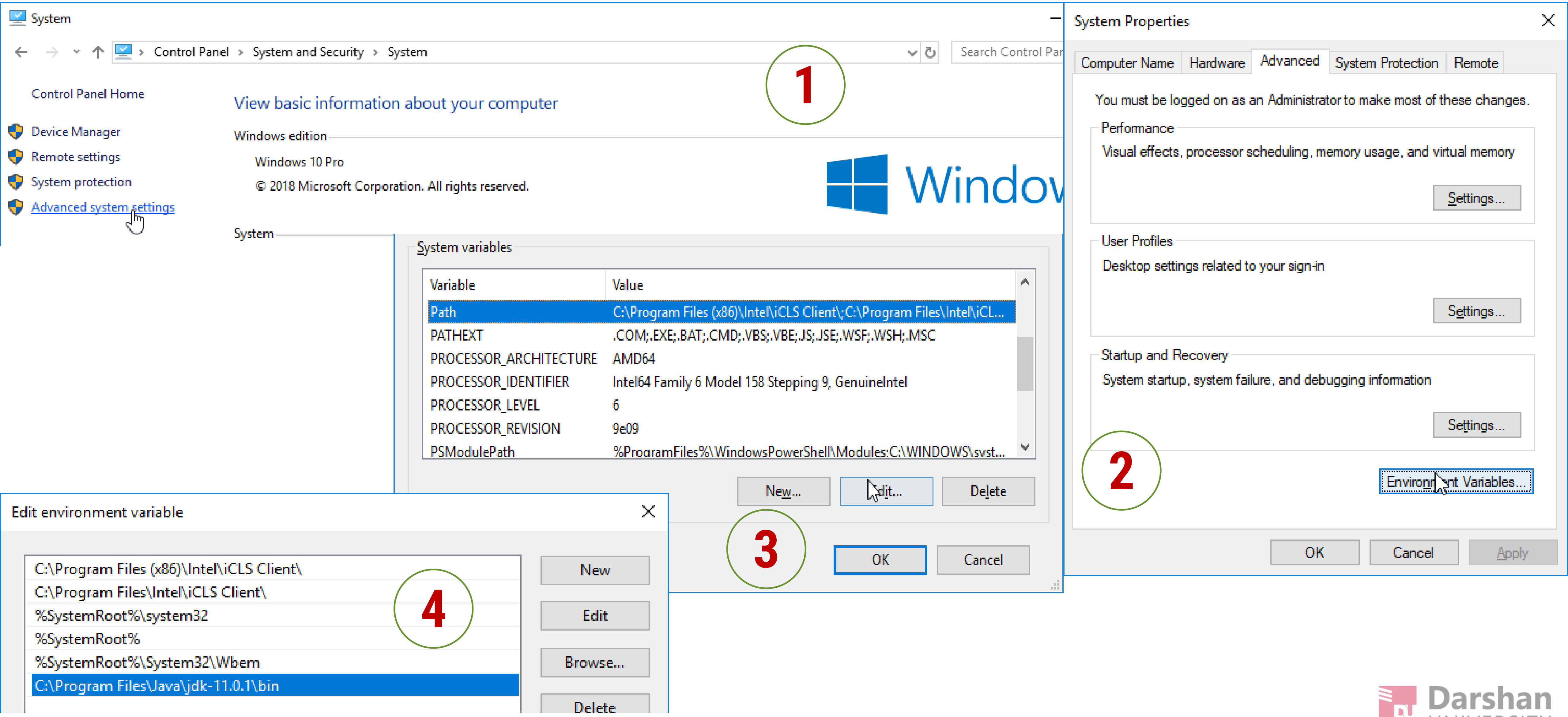
- Java is a general-purpose computer-programming language that is **open source**, **platform independent, object-oriented** and specifically designed to have as few implementation dependencies as possible.
- Java was originally developed by **James Gosling** at Sun Microsystems and released in 1995.
- Java was initially named as Oak language and renamed to JAVA in 1995.

Current Version	**Java SE 22** (as of September, 14th 2021)
Version we will use	Java SE 21 (LTS)
Setup size	149 MB (Linux), 152 MB (Windows x64)
Download Link	https://www.oracle.com/in/java/technologies/javase-jdk11-downloads.html
Official Website	https://java.com
Integrated **D**evelopment **E**nvironment (IDE)	1. Eclipse NetBeans 2. IntelliJ IDEA Community Edition 3. BlueJ

JAVA Installation: help

- Download JDK for Windows platform (.exe) from
  - https://www.oracle.com/technetwork/java/javase/downloads/index.html
- Install the executable of JDK
- Set the path variable of System variables by performing following steps
  - Go to "System Properties" (Right click This PC → Properties → Advanced System Settings)
  - Click on the "Environment variables" button under the "Advanced" tab
  - Then, select the "Path" variable in System variables and click on the "Edit" button
  - Click on the "New" button and add the path where Java is installed, followed by `\bin`. By default, Java is installed in `C:\Program Files\Java\jdk-11.0.1` (If nothing else was specified when you installed it). In that case, You will have to add a new path with: `C:\Program Files\Java\jdk-11.0.1\bin`
  - Then, click "OK", and save the settings
  - At last, open Command Prompt (cmd.exe) and type java -version to see if Java is running on your machine

JAVA Installation : Setting Path Variable



View basic information about your computer

Windows 10 Pro

© 2018 Microsoft Corporation. All rights reserved.

Windows

Variable	Value
Path	C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCL...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 158 Stepping 9, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	9e09
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\WINDOWS\syst...

```
C:\Program Files (x86)\Intel\iCLS Client\
C:\Program Files\Intel\iCLS Client\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
C:\Program Files\Java\jdk-11.0.1\bin
```

You must be logged on as an Administrator to make most of these changes.

Performance: Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles: Desktop settings related to your sign-in

Startup and Recovery: System startup, system failure, and debugging information

Features of JAVA

Simple: Java inherits C/C++ syntax and many object-oriented features of C++.

Object Oriented: "Everything is an object" paradigm, which possess some state, behavior and all the operations are performed using these objects.

Robust: Java has a strong memory management system. It helps in eliminating error as it checks the code during compile and runtime.

Multithreaded: Java supports multiple threads of execution, including a set of synchronization primitives. This makes programming with threads much easier.

Features of JAVA (Cont.)

Portable

Architectural Neutral: Java is **platform independent** which means that any application written on one platform can be easily ported to another platform.

Interpreted

Interpreted: Java is compiled to **bytecodes**, which are interpreted by a Java run-time environment.

High Performance

High Performance: Java achieves high performance through the use of bytecode which can be easily translated into native machine code. With the use of JIT (Just-In-Time) compilers, Java enables high performance.

Features of JAVA (Cont.)

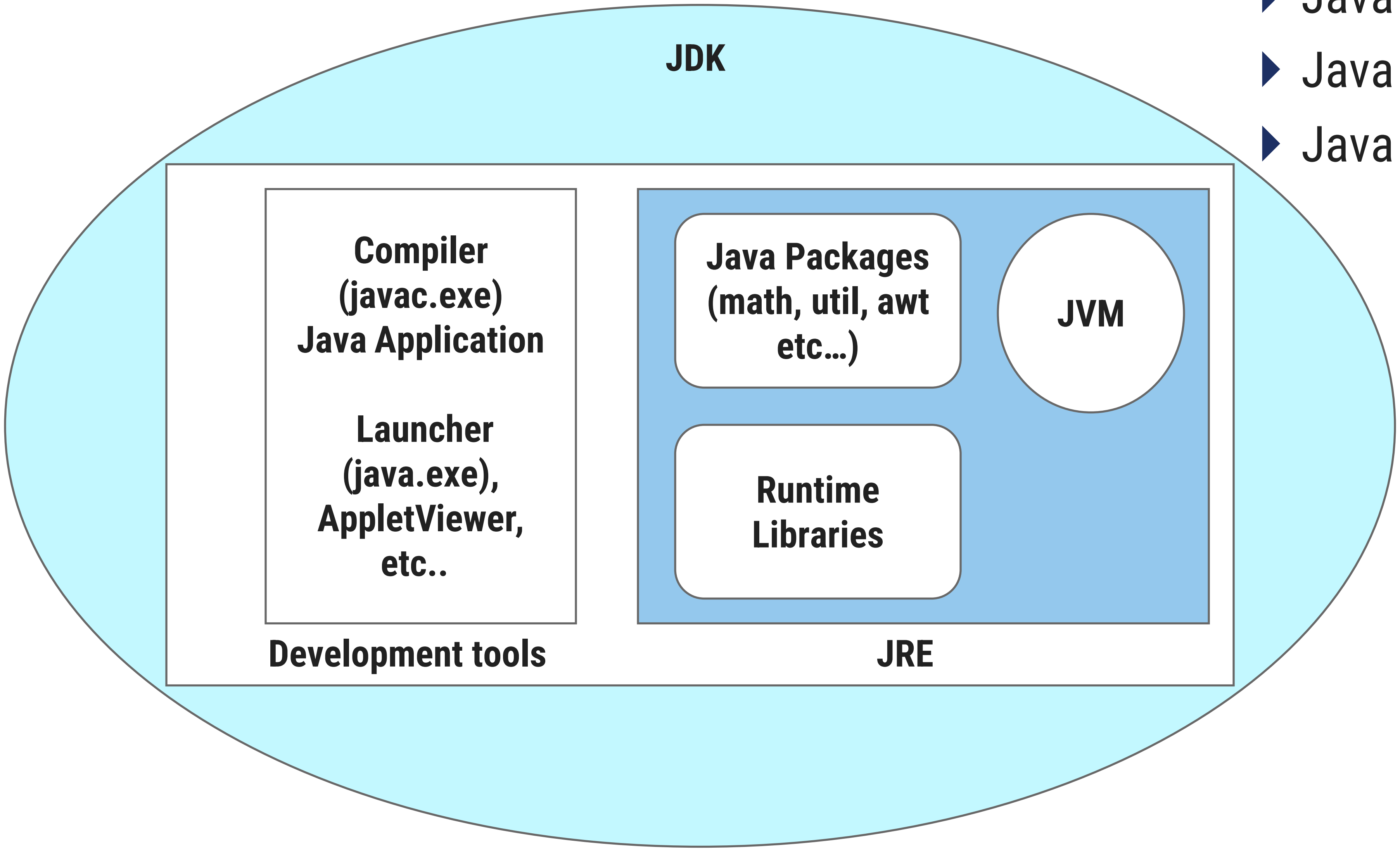
Distributed

Distributed: Java provides a feature which helps to create distributed applications. Using Remote Method Invocation (RMI), a program can invoke a method of another program across a network and get the output. You can access files by calling the methods from any machine on the internet.

Dynamic

Dynamic: Java has ability to adapt to an evolving environment which supports dynamic memory allocation due to which memory wastage is reduced and performance of the application is increased.

Components of Java

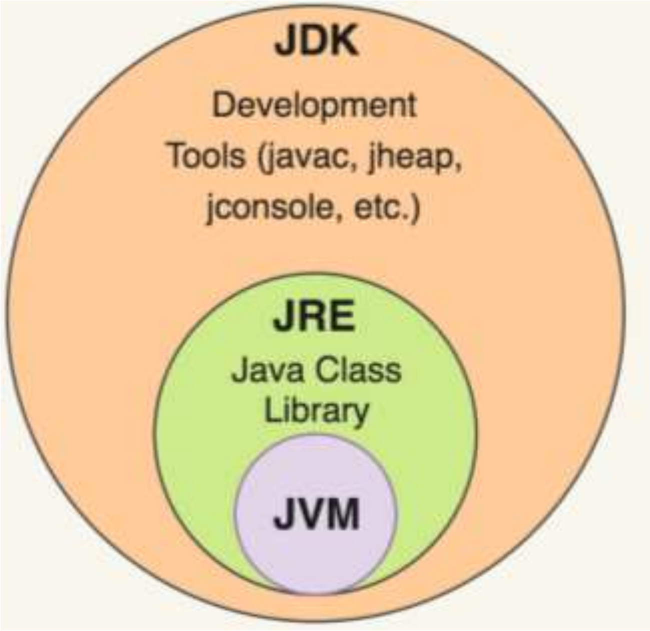


JDK
Compiler (javac.exe) Java Application
Launcher (java.exe), AppletViewer, etc..
Development tools
Java Packages (math, util, awt etc…)
JVM
Runtime Libraries
JRE

- Java Virtual Machine (JVM)
- Java Runtime Environment (JRE)
- Java Development Kit (JDK)

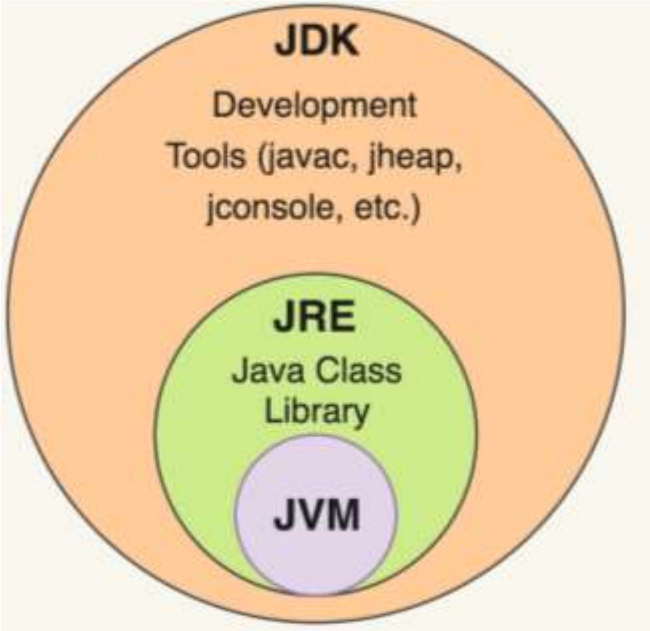
Java Development Kit (JDK)

- JDK contains tools needed ,
  - To develop the Java programs and
  - JRE to run the programs.
- The tools include
  - compiler (javac.exe),
  - Java application launcher (java.exe),
  - Appletviewer, etc…
- Java application launcher (java.exe) opens a JRE, loads the class, and invokes its main method.



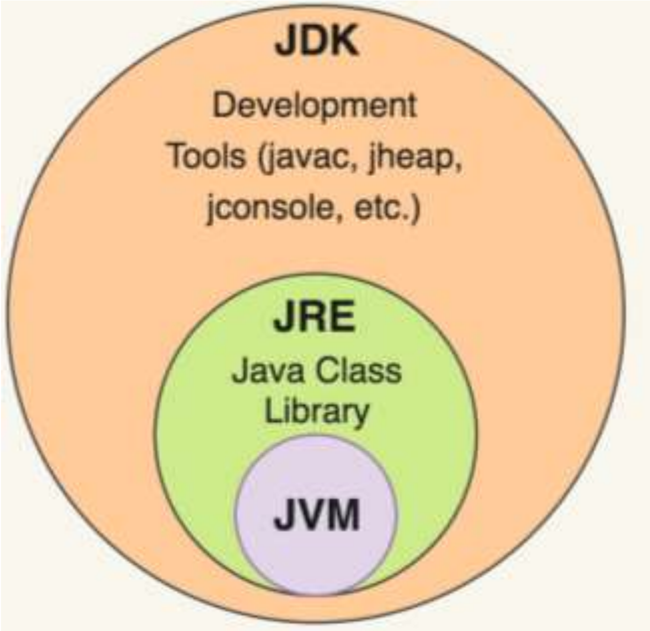
Java Runtime Environment (JRE)

- The JRE is required to run java applications.
- It combines the Java Virtual Machine (JVM), platform core classes and supporting libraries.
- JRE is part of the Java Development Kit (JDK), but can be downloaded separately.
- It does not contain any development tools such as compiler, debugger, etc.

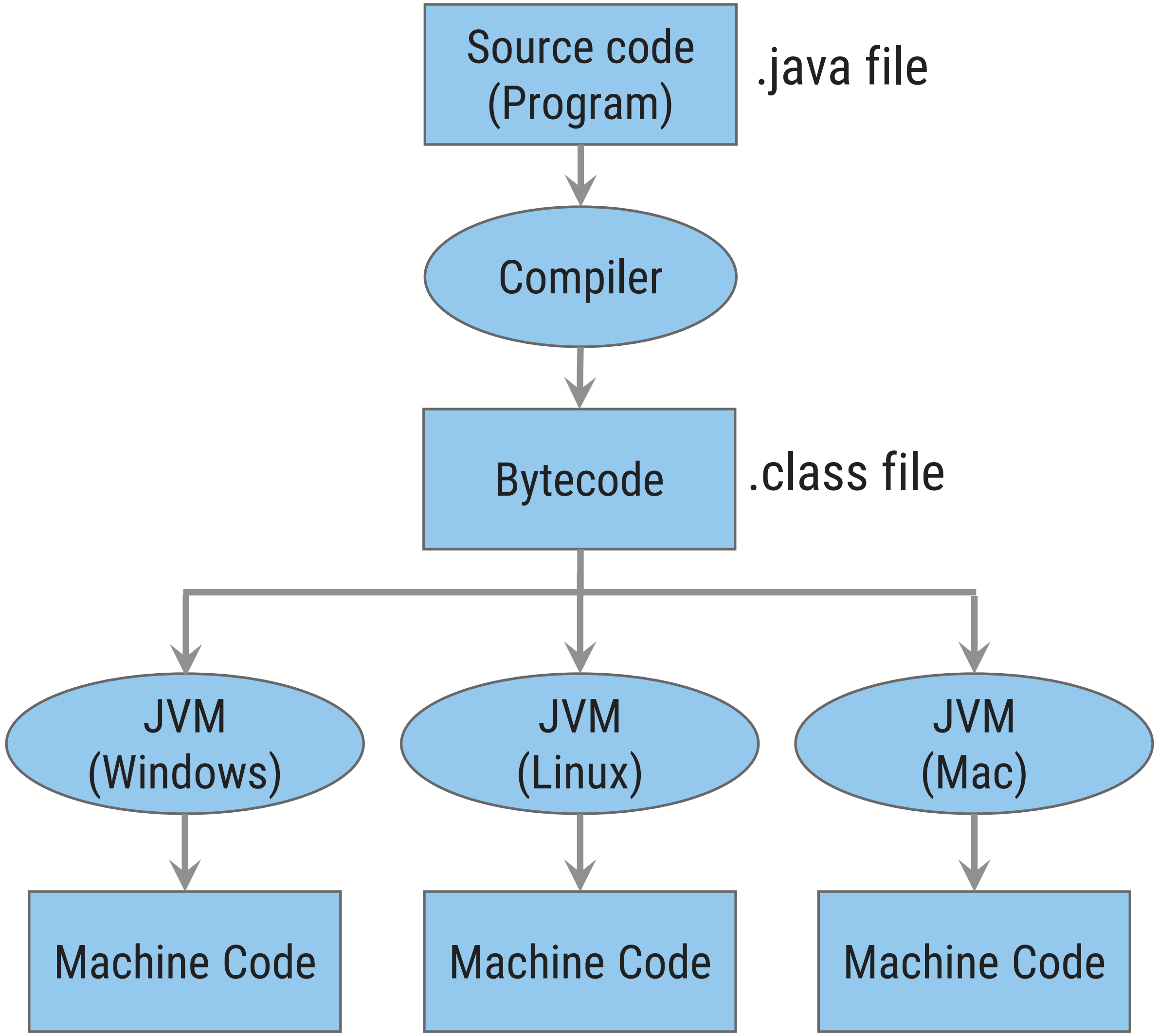


Java Virtual Machine (JVM)

- JVM is a virtual machine that enables a computer to run Java programs as well as programs written in other languages and compiled to Java Bytecode.
- Bytecode is a highly optimized set of instructions designed to be executed by the Java Virtual Machine(JVM).
- Byte code is intermediate representation of java source code.
- Java compiler provides byte code by compiling Java Source Code.
- Extension for java class file or byte code is '.class', which is platform independent.
- JVM is virtual because , It provides a machine interface that does not depend on the operating system and machine hardware architecture.
- JVM interprets the byte code into the machine code.
- **JVM** itself is **platform dependent**, but **Java** is **Not**.



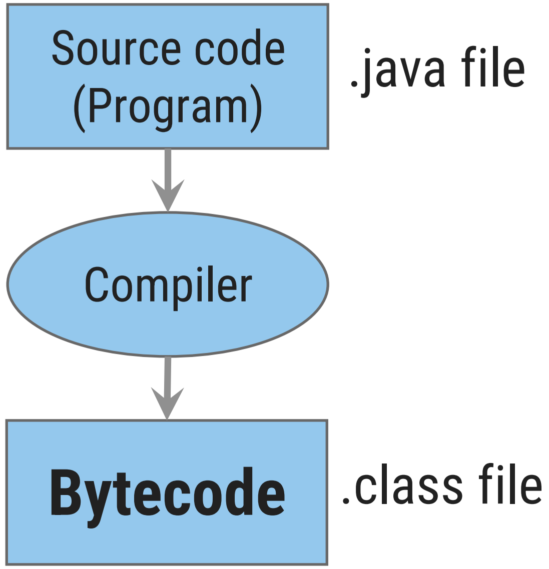
How Java become Platform Independent?



Bytecode: Java’s Magic

- ***Bytecode*** is a highly optimized set of instructions designed to be executed by the JVM
- Here, JVM is an *interpreter for bytecode.*
- Translating a Java program into bytecode helps makes it much easier to run a program in a wide variety of environments.

  i.e “***WRITE ONCE RUN ANYWHERE***”
- Thus, the interpretation of bytecode is the easiest way to create truly portable programs.
- Interpreted Java program are highly secured, because the execution of every Java program is under the control of the JVM.



Java Interview Question

1. Difference between JRE and JVM?
2. Difference between interpreter and JIT compiler?
3. Justify Java is platform independent.
4. Justify Java gives high performance.
5. Justify Java is Robust language.
6. What are Java bytecodes?
7. JVM vs. JRE vs. JDK

First Java Program

Hello World Java Program

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

File must be saved as HelloWorld.java

Main method from where execution will start

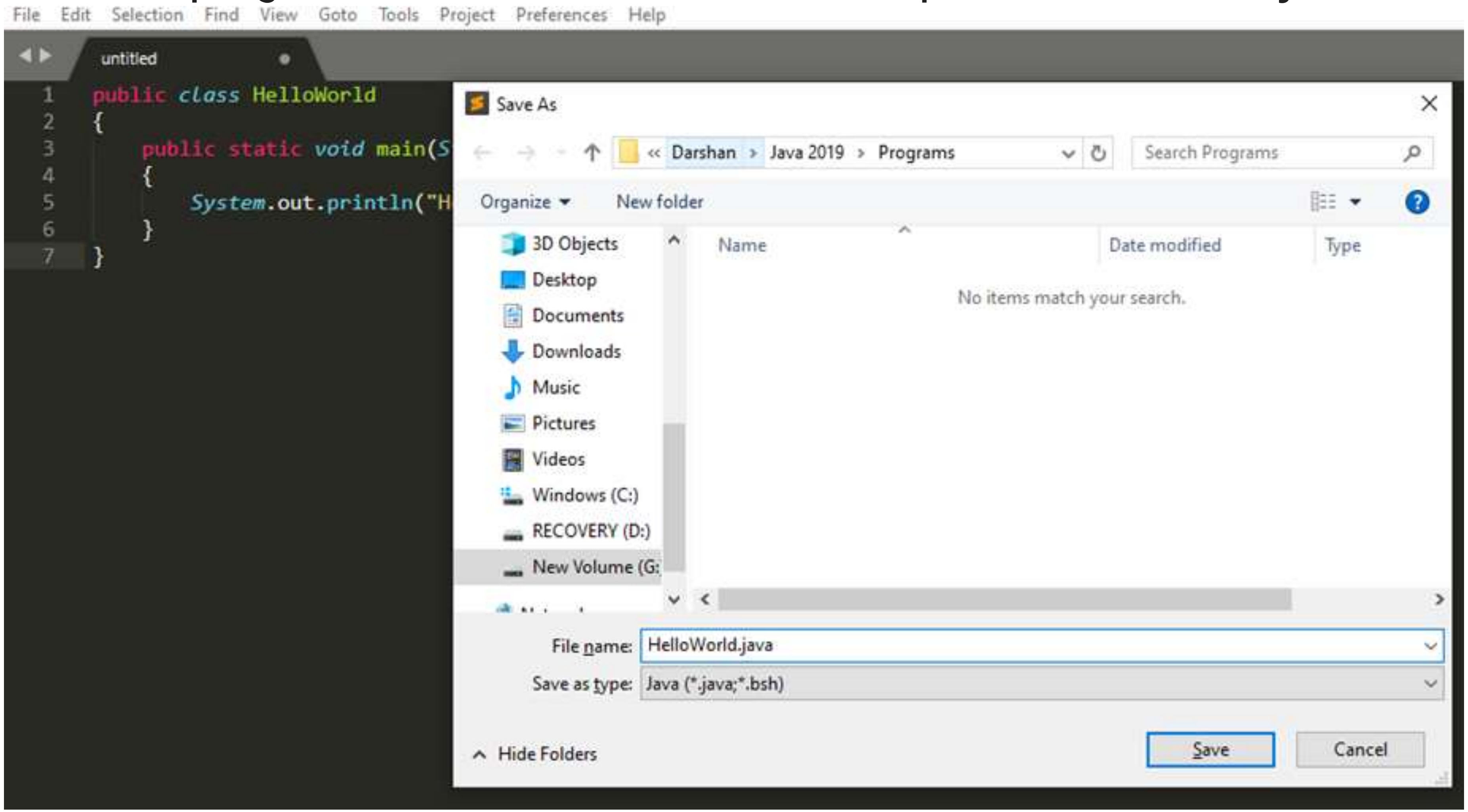
String must start with capital letter

System must start with capital letter

- We have to save this in **HelloWorld.java** file as it has public class named HelloWorld.
- String and System are inbuilt Java Classes.
- Classes in java are always written in Camel case.

How to execute Java Program?

1. Save the program with the same name as the public class with **.java** extension.



```
public class HelloWorld
{
    public static void main(S
    {
        System.out.println("H
    }
}
```

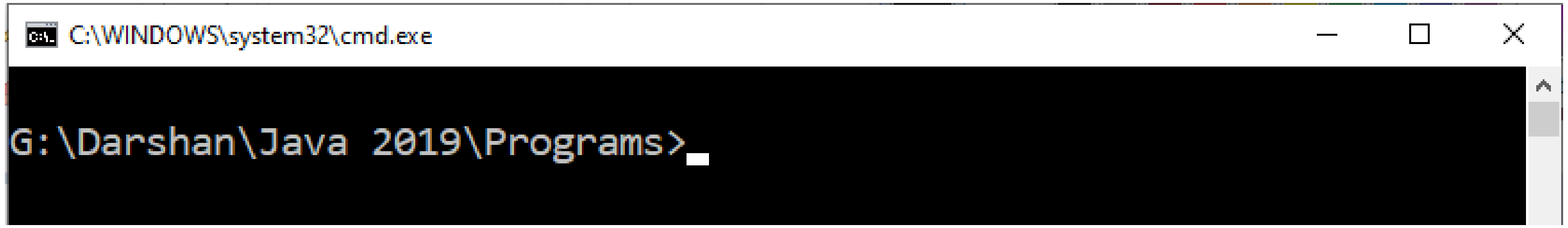
Save As

File name: HelloWorld.java

Save as type: Java (*.java;*.bsh)

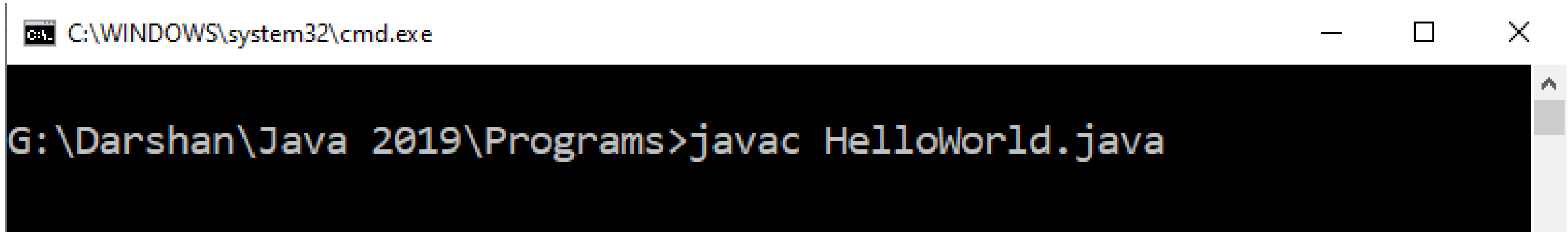
How to execute Java Program?

2. Open command prompt (cmd) / terminal & navigate to desired directory / folder.



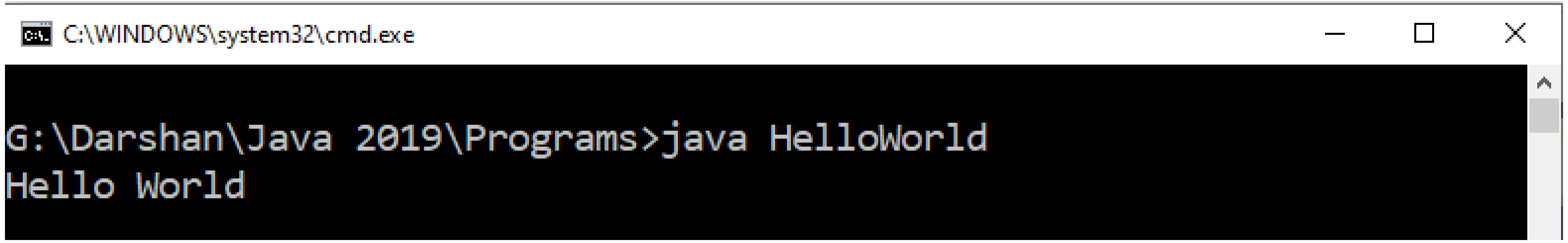
```
G:\Darshan\Java 2019\Programs>_
```

3. Compile the ".java" file with **javac** command.



```
G:\Darshan\Java 2019\Programs>javac HelloWorld.java
```

4. Execute the ".class" file with **java** command without extension.



```
G:\Darshan\Java 2019\Programs>java HelloWorld
Hello World
```

Assignment-1

1. Differentiate JRE,JDK and JVM.
2. Why Java is platform independent?

Object-Oriented Programming
2305CS102

Darshan
UNIVERSITY
योग: कर्मसु कौशलम्

Unit-1(part-II) Data types, Variables & Operators

Prof. Swati R Sharma
Computer Engineering Department
Darshan University, Rajkot

✉ swati.sharma@darshan.ac.in
☎

What we will learn

- ✓ Primitive data types
- ✓ String
- ✓ Variables
- ✓ Type casting
- ✓ Command line arguments
- ✓ Operators
- ✓ Operator precedence

Unit-1(part-II)

Tokens

Tokens

- The smallest individual unit of a language / program is known as a **token**.
- Tokens are basic building blocks of any language.

“Jane bakes tasty cookies.”

↪ Jane is noun

↪ bakes is verb

↪ tasty is adjective

↪ cookies is noun

↪ ‘.’ is special character to end the sentence.

- Each and every word and punctuation is a token.
- We divide sentence into tokens to understand the meaning of a sentence.
- Similarly, the compilers of programming language breaks a program into the tokens and proceeds to the various stages of the compilation.
- However, collection of tokens in appropriate sequence makes a meaningful sentence.

Classification of Tokens

Sr.	Token	Description	Examples
1	**Keywords**	Predefined reserved words	void, int, float, for, if
2	**Identifiers**	User-defined combination of alphanumeric characters. Name of a variable, function, class, etc.	a, i, sum, number, pi
3	**Constants**	Fixed values that do not change	17, -25.50, 82, 0
4	**Strings**	A sequence of characters	“Darshan”, “Hi!”
5	**Special Symbols**	Symbols that have special meaning	#, $, @, %, =, :, ;
6	**Operators**	A symbol that performs operation on a value or a variable	+, -, *, /

Identifiers

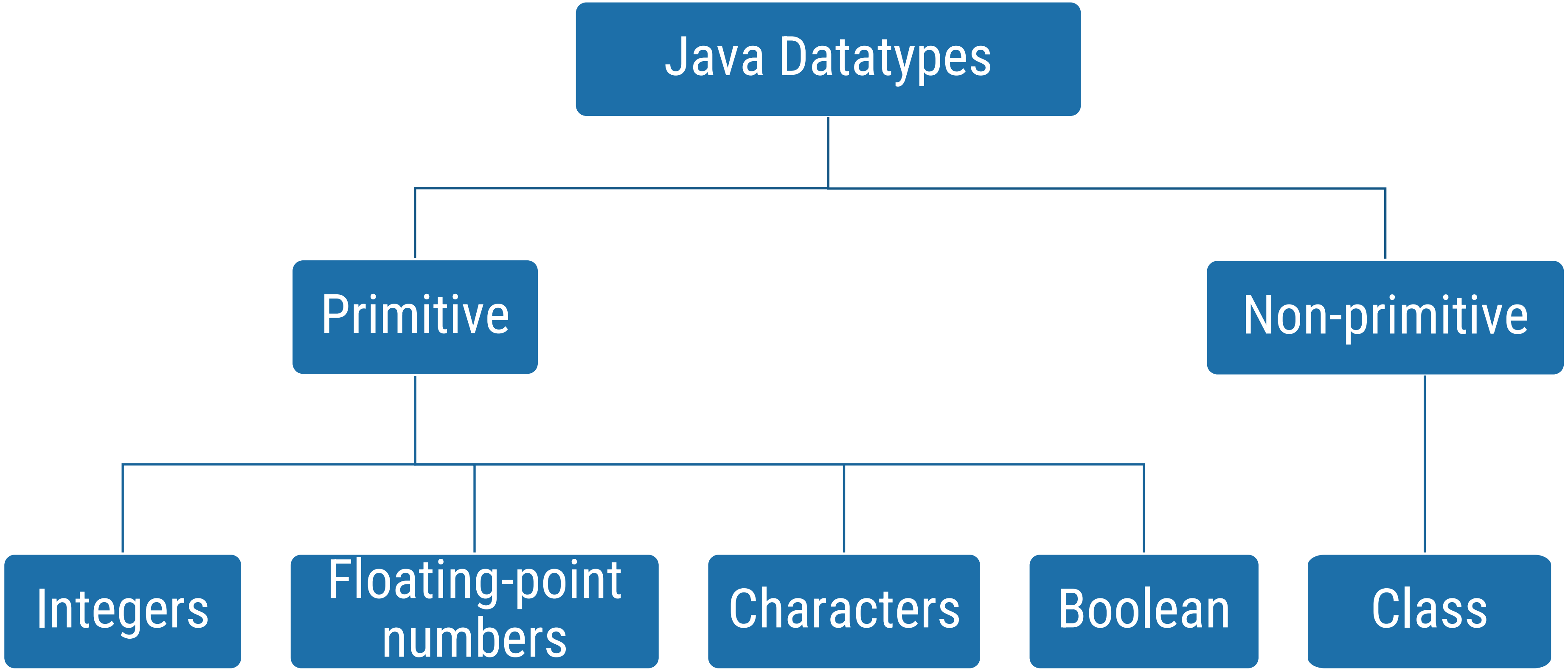
Identifiers

- They are used for **class** names, **method** names and **variable** names.
- An identifier may be any descriptive sequence of
  - uppercase(A…Z) and lowercase(a..z) letters
  - Numbers(0..9)
  - Underscore(_) and dollar-sign($) characters
- Examples for **valid** Identifiers,
  - AvgTemp
  - count
  - a4
  - $test
  - this_is_ok
- Examples for **invalid** Identifiers,
  - 2count (Identifiers can not start with digit)
  - High-temp (Identifiers can not contain dash)
  - Ok/NotOK (Identifiers can not contains slash)

Identifier Name Valid or Invalid?

🟥 for	🟩 Rajkot	🟩 _Name	🟩 If	🟥 Roll Number	🟥 Rs.
🟩 _________a	🟥 int	🟩 student	🟩 _7	🟩 Identifier	🟩 Valid
🟩 C	🟥 C++	🟩 Java	🟥 C#	🟩 Compiler	🟩 ___8__a__8
🟥 7Student	🟥 Who?	🟥 v.a.l.u.	🟩 A_B_C_D_E	🟩 i	🟥 if
🟩 Int	🟩 A2020	🟥 2021	🟩 sum	🟥 a,b,c	🟥 i;

Data Types


Java Datatypes
Primitive
Non-primitive
Integers
Floating-point numbers
Characters
Boolean
Class

Primitive Data Types

Data Type	Size	Range	Example
byte	1 Byte	-128 to 127	byte a = 10;
short	2 Bytes	-32,768 to 32,767	short a = 200;
int	4 Bytes	-2,147,483,648 to 2,147,483,647	int a = 50000;
long	8 Bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long a = 20;
float	4 Bytes	1.4e-045 to 3.4e+038	float a = 10.2f;
double	8 Bytes	4.9e-324 to 1.8e+308	double a = 10.2;
char	2 Bytes	0 to 65536 (Stores ASCII of character)	char a = ‘a’;
boolean	Not defined	true or false	boolean a = true;

Escape Sequences

- Escape sequences in general are used to signal an alternative interpretation of a series of characters.
- For example, if you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes.

```
System.out.println("Good Morning \"World\"");
```

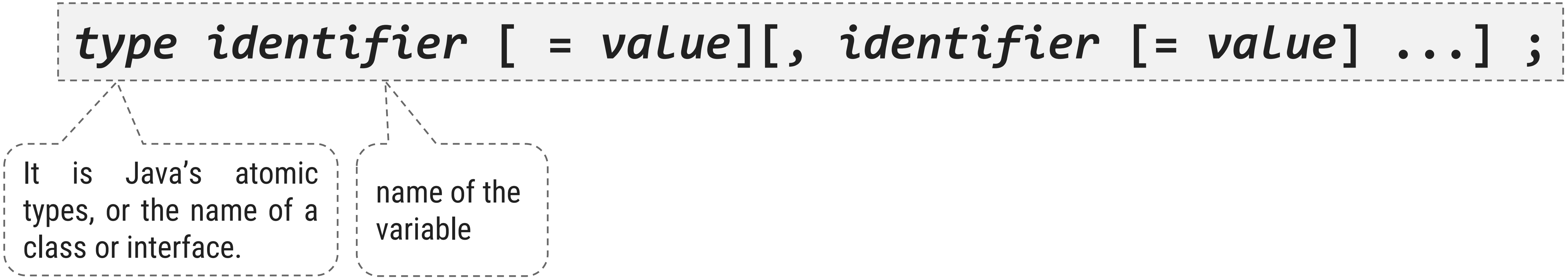
Escape Sequence	Description
\’	Single quote
\”	Double quote
\\	Backslash
\r	Carriage return
\n	New Line
\t	Tab

Variables in Java

Variables in Java

- The variable is the **basic unit of storage** in a Java program.
- A variable is defined by the combination of an identifier, a type, and an optional initializer.
- Variables have a scope, which defines their visibility, and a lifetime.

Declaring a Variable



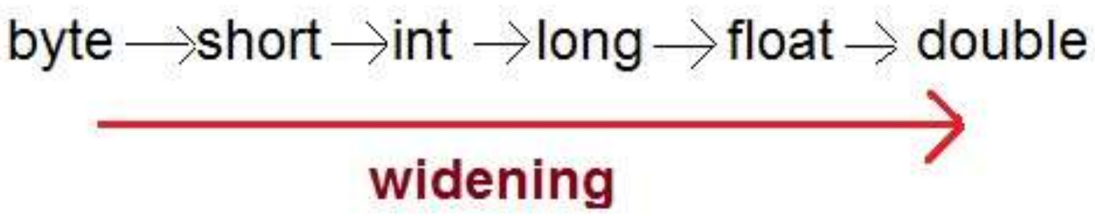
```
type identifier [ = value][, identifier [= value] ...] ;
```

It is Java's atomic types, or the name of a class or interface.

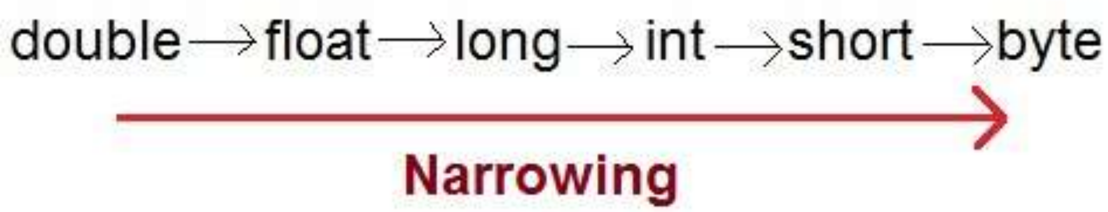
name of the variable

Type Casting

- Assigning a value of one type to a variable of another type is known as **Type Casting**.
- In Java, type casting is classified into two types,
  - Widening/Automatic Type Casting (**Implicit**)



  - Narrowing Type Casting(**Explicitl**y done)



Automatic Type Casting

- When one type of data is assigned to other type of variable , an *automatic type conversion* will take place if the following two conditions are satisfied:
  - The two types are compatible
  - The destination type is larger than the source type
- Such type of casting is called “*widening conversion*”.
- Example:

  int can always hold values of **byte** and **short**

```
public static void main(String[] args) {
    byte b = 5;
    // √ this is correct
    int a = b;
}
```

Casting Incompatible Types

- To create a conversion between two incompatible types, you must use a *cast*
- A ***cast*** is an explicit type conversion.
- Such type is called “*narrowing conversion”.*
- Syntax:

  (target-type) value
- Example:

```
public static void main(String[] args) {
    int a = 5;
    // × this is not correct
    byte b = a;
    // √ this is correct
    byte b = (byte)a ;
}
```

Operator

Perform definite operation

Operators

1. Arithmetic Operators
2. Relational Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Conditional / Ternary Operator
7. Instance of Operator

Operators

- An operator is a symbol to perform specific mathematical or logical functions.
- We use operators in maths to perform certain operations, e.g. +, -, *, /, etc.
- Unary operators (++, --) take one operand, Binary operators (+, -, *, /) take two operands.
- Programming languages are rich in operators which can be divided in following categories,

Sr.	Operator	Examples
1	Arithmetic Operators	+, -, *, /, %
2	Relational Operators	<, <=, >, >=, ==, !=
3	Logical Operators	&&, ||, !
4	Assignment Operators	=, +=, -=, *=, /=
5	Increment and Decrement Operators	++, --
6	Conditional Operator	?:
7	Bitwise Operators	&, |, ^, <<, >>

Arithmetic Operators

- An arithmetic operator performs basic mathematical calculations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning	Example	Description
+	Addition	a + b	Addition of a and b
-	Subtraction	a – b	Subtraction of b from a
*	Multiplication	a * b	Multiplication of a and b
/	Division	a / b	Division of a by b
%	Modulo division- remainder	a % b	Modulo of a by b

Relational Operators

- A relational operators are used to compare two values.
- They check the relationship between two operands, if the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational expressions are used in decision statements such as if, for, while, etc…

Operator	Meaning	Example	Description
<	is less than	a < b	a is less than b
<=	is less than or equal to	a <= b	a is less than or equal to b
>	is greater than	a > b	a is greater than b
>=	is greater than or equal to	a >= b	a is greater than or equal to b
==	is equal to	a == b	a is equal to b
!=	is not equal to	a != b	a is not equal to b

Relational Operators

Note : A = 10 & B = 20

Operator	Description	Example
==	Equals	(A == B) is not true.
!=	Not Equals	(A != B) is true.
>	Greater than	(A > B) is not true.
<	Less than	(A < B) is true.
>=	Greater than equals	(A >= B) is not true.
<=	Less than equals	(A <= B) is true.

Bitwise Operators

Note : A = 60 & B = 13

Operator	Description	Example
&	Binary AND Operator	A & B = 12 which is 0000 1100
|	Binary OR Operator	A | B = 61 which is 0011 1101
^	Binary XOR Operator	A ^ B = 49 which is 0011 0001
~	Binary Ones Complement Operator	~A = -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator	A << 2 = 240 which is 1111 0000
>>	Binary Right Shift Operator.	A >> 2 = 15 which is 1111
>>>	Shift right zero fill operator.	A >>>2 = 15 which is 0000 1111

Logical Operators

- Logical operators are decision making operators.
- They are used to combine two expressions and make decisions.
- An expression containing logical operator returns either 0 or 1 depending upon whether expression results false or true.

Operator	Meaning	Example (Let's assume c=5 and d=2)
&&	Logical AND. True only if all operands are true	expression ((c==5) && (d>5)) equals to 0.
| |	Logical OR. True only if either one operand is true	expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is false	expression !(c==5) equals to 0.

a	b	a && b	a || b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Logical Operators

Note : A = true & B = false

Operator	Description	Example
&&	Logical AND operator	(A && B) is false.
||	Called Logical OR Operator	(A || B) is true.
!	Called Logical NOT Operator	!(A && B) is true.

Assignment Operators

- Assignment operators are used to assign a new value to the variable.
- The left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value or a result of an expression.
- Meaning of = in Maths and Programming is different.
  - Value of LHS & RHS is always same in Math.
  - In programming, value of RHS is assigned to the LHS

Operator	Meaning	
=	Assigns value of right side to left side. Suppose a=5 and b=10. a=b means now value of a is 10.	
+=	a += 1 is same as a = a + 1	Shorthand Assignment Operators
-=	a -= 5 is same as a = a – 5	
*=	a *= b is same as a = a * b	
/=	a /= c is same as a = a / c	
%=	a %= 10 is same as a = a % 10	

Assignment Operators

Operator	Description	Example
=	Simple assignment operator	C = A + B will assign value of A + B into C
+=	Add AND assignment operator	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Increment / Decrement Operators

- Increment and decrement operators are unary operators that add or subtract one, to or from their operand.
- the increment operator ++ increases the value of a variable by 1, e.g. a++ means a=a+1
- the decrement operator -- decreases the value of a variable by 1. e.g. a−− means a=a−1
- If ++ operator is used as a prefix (++a) then the value of a is incremented by 1 first then it returns the value.
- If ++ operator is used as a postfix (a++) then the value of a is returned first then it increments value of a by 1.

Expression	Evaluation (Let's say a=10, c=15)
b = a++	Value of b would be **10** and value of a would be **11**.
b = ++a	Value of b & a would be **11**.
b = a--	Value of b would be **10** and value of a would be **9**.
b = --a	Value of b & a would be **9**.

Expression	Evaluation (Let's say a=10, c=15)
b = --a + c++	b = 24
b = a++ + ++c	b = 26
b = ++a - ++c	b = −5

Conditional Operator (Ternary)

- Conditional Operator (? :)
  - Syntax:

    variable x = (expression) ? value if true : value if false
  - Example:

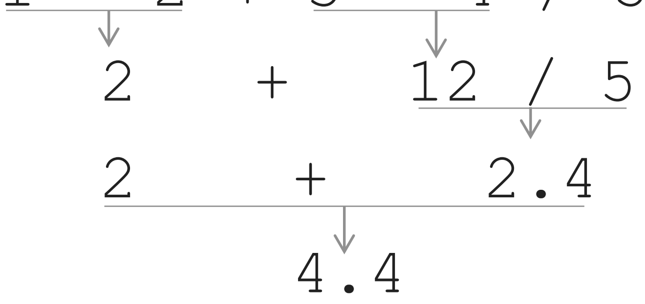
```
b = (a == 1) ? 20 : 30;
```

Operators Precedence & Associativity

Priority matters!

Operator Precedence & Associativity

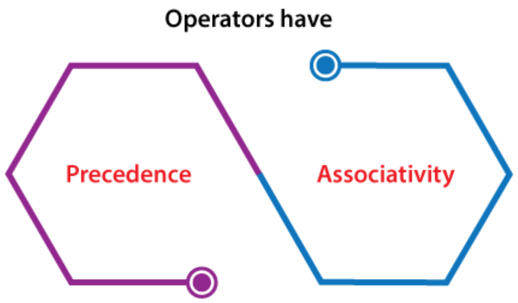
- How does java evaluate `1 + 10 * 9 ?`
  - `(1 + 10 ) * 9 = 99` **OR** `1 + (10 * 9) = 91`
- To get the correct answer for the given problem Java came up with **Operator precedence**.
- Multiplication have higher precedence than addition so correct answer will be **91** in this case.
- For Operator, associativity means that when the same operator appears in a row, then to which direction the expression will be evaluated.
- How does java evaluate `1 * 2 + 3 * 4 / 5 ???`



```
2 + 12 / 5
2 + 2.4
4.4
```

Operators Precedence & Associativity

- Operators Precedence and Associativity are two characteristics of operators that determine the evaluation order of sub-expressions in absence of brackets.
- Operator precedence determines which operation is performed first in an expression with more than one operators with different precedence.
  - a=10 + 20 * 30 is calculated as 10 + (20 * 30) and not as (10 + 20) * 30 so answer is 610.
- Operators Associativity is used when two operators of same precedence appear in an expression. Associativity can be either Left to Right (L to R) or Right to Left (R to L).
  - a=100 / 10 * 10
    - If Left to Right means (100 / 10) * 10 then answer is 100
    - If Right to Left means 100 / (10 * 10) then answer is 1
    - Division (/) & Multiplication (*) are Left to Right associative so the answer is **100.**



Precedence of Java Operators

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Operators Precedence and Associativity

1) Associativity is only used when there are two or more operators of same precedence.

2) All operators with the same precedence have same associativity

Priority	Operator	Description	Associativity
1	( ) [ ] . a++ a—	Parentheses (function call) Brackets (array subscript) Member selection via object name Postfix increment/decrement	left-to-right
2	++a —a + – ! ~ (*type*) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of *type*) De-reference Address (of operand) Determine size in bytes on this implementation	**right-to-left**
3	* / %	Multiplication/division/modulus	left-to-right
4	+ –	Addition/subtraction	left-to-right
5	<< >>	Bitwise shift left, Bitwise shift right	left-to-right
6	< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
7	== !=	is equal to/is not equal to	left-to-right
8	&	Bitwise AND	left-to-right
9	^	Bitwise exclusive OR	left-to-right
10	|	Bitwise OR	left-to-right
11	&&	Logical AND	left-to-right
12	| |	Logical OR	left-to-right
13	? :	Ternary conditional	**right-to-left**
14	= += -= *= /= %= &= ^= |= <<= >>=	Assignment Shorthand Assignments Bitwise exclusive/inclusive assignment Bitwise shift left/right assignment	**right-to-left**
15	,	Comma (separate expressions)	left-to-right

Exercise

Sr.	Exercise	Answer

1.	int i=1;
	i=2+2*i++;
		4

2.	int a=2,b=7,c=10;
	c=a==b;
		0

3.	int a=2,b=7,c=10;
	c=a!=b;
		1

4.	int a=100 + 200 / 10 - 3 * 10	90

5.	int a = 2, b = 6, c = 12, d;
	d = a * b + c / b;
	System.out.println("The value of d ="+d);
		14

Interview Questions

1. What is the **default value** of local variables in Java?
2. Can you compare a **boolean** with an **int** variable in Java?
3. Difference between **double** and **float** variables in Java.
4. Determine Output

```
class Main {
   public static void main(String args[]) {
         int t;
         System.out.println(t);
    }
}
```

Assignment

1.	int i = 11;
	i = i++ + ++i;	

2.	int a=11, b=22, c;
	c = a + b + a++ + b++ + ++a + ++b;	

3.	int i=0;
	i = i++ - --i + ++i - i--;	

Object-Oriented Programming
2305CS102

Unit-1(Part-III) Array and Control Statements

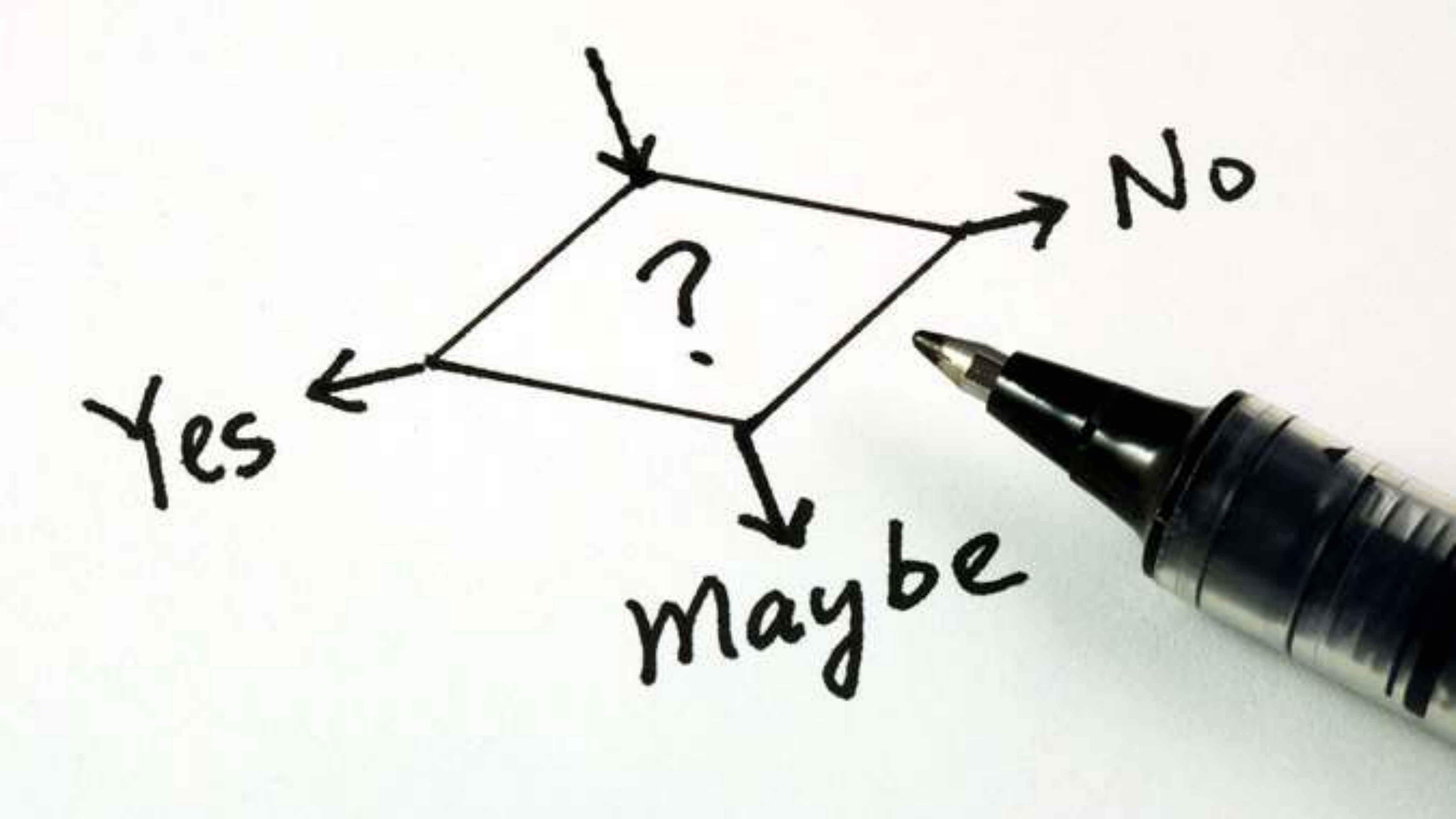
Prof. Swati R Sharma
Computer Engineering Department
Darshan University, Rajkot

✉ swati.sharma@darshan.ac.in
☎

What we will learn

- ✓ if, else, nested if, if-else ladder
- ✓ switch
- ✓ Looping Statement: while, do-while, for, foreach,
- ✓ break and continue statement
- ✓ Single and Multidimensional Array

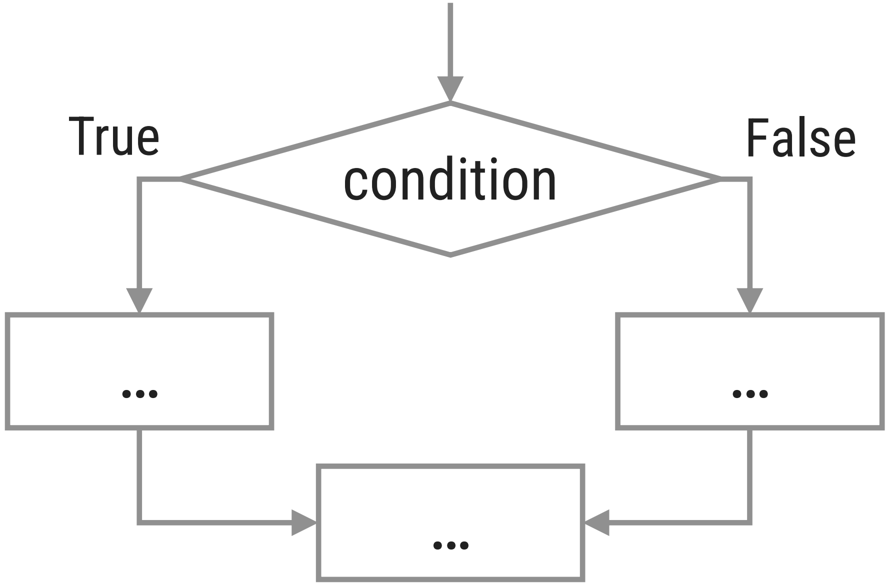
Unit-1



?
No
Yes
Maybe

Decision Making

- Compiler executes program statements sequentially.
- Decision making statements are used to control the flow of program execution.
- It allows us to control whether a set of program statement should be executed or not.
- It evaluates condition or logical expression first and based on its result (true or false), the control is transferred to the particular statement.
- If result is true then it takes one path else it takes another path.



Decision Making Statements

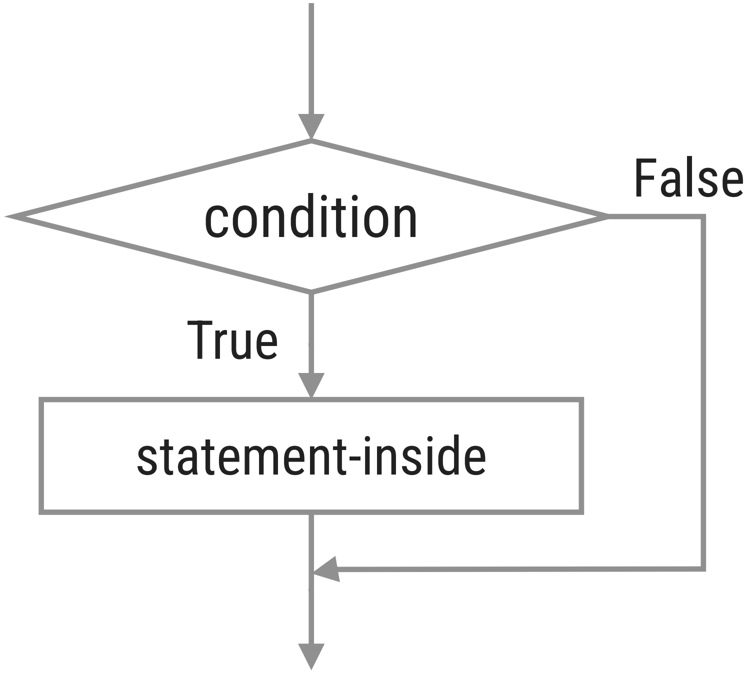
- Commonly used decision making statements are
  - One way Decision: `if` (Also known as simple if)
  - Two way Decision: `if…else`
  - Multi way Decision: `if…else if…else if…else`
  - Decision within Decision: `nested if`
  - Two way Decision: `?: (Conditional Operator)`
  - n-way Decision: `switch…case`

if

One way Decision

if

- **if** statement is the most simple decision-making statement also known as **simple if**.
- An **if** statement consists of a boolean expression followed by one or more statements.
- **If** the expression is true, then 'statement-inside' will be executed, otherwise 'statement-inside' is skipped and only 'statement-outside' will be executed.
- It is used to decide whether a block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.



```
if(condition)
{
        // Statements to execute if condition is true
}
```

WAP to print if a number is positive

```java
import java.util.*;
class MyProgram{
public static void main (String[] args){
int x;
Scanner sc = new Scanner(System.in);
    x = sc.nextInt();
    if(x > 0){
      System.out.println("number is a positive");
    }
}
```

Exercise

- Write a program which reads two numbers and based on different between it prints either of following message DIFFERENCE IS POSITIVE or DIFFERENCE IS NEGATIVE.

WAP to print if a number is odd or even

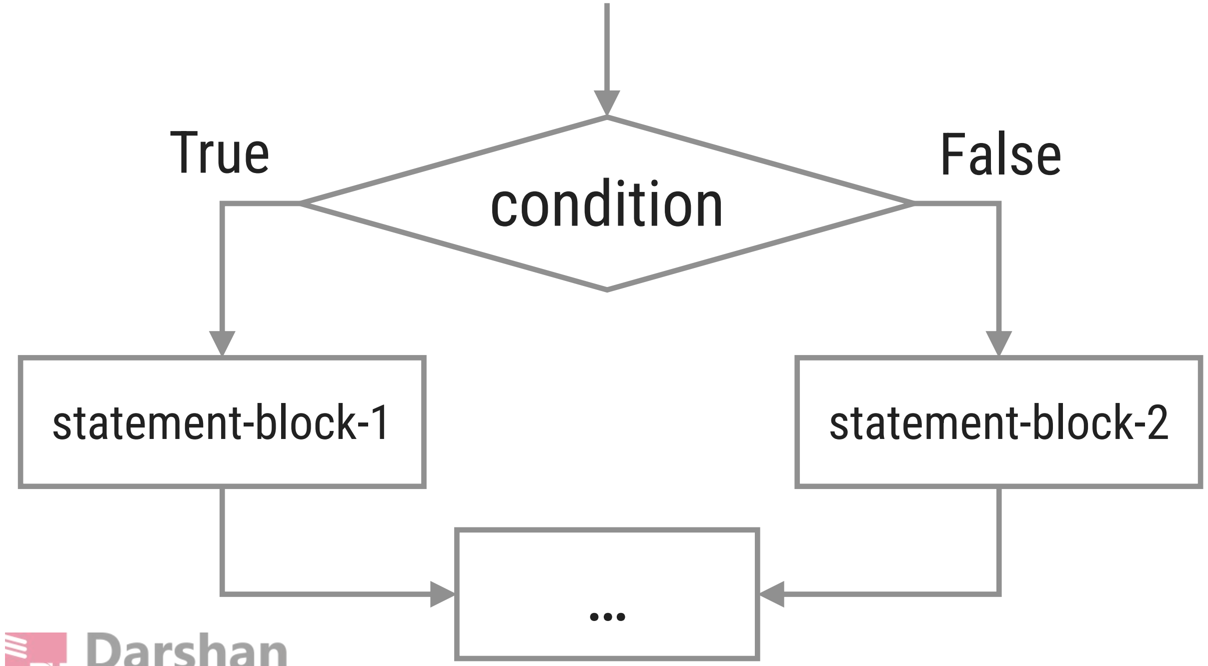
```java
import java.util.*;
class MyProgram{
public static void main (String[] args){
int x;
Scanner sc = new Scanner(System.in);
    x = sc.nextInt();
    if( x % 2 == 1 ){
         System.out.println("number is a odd");
    }
     if( x % 2 == 0 ){
        System.out.println("number is a even");
     }
}
```

if…else

Two way Decision

If…else

- For a **simple if**, if a condition is true, the compiler executes a block of statements, if condition is false then it doesn’t do anything.
- What if we want to do something when the condition is false? if…else is used for the same.
- If the 'expression' is true then the 'statement-block-1' will get executed else 'statement-block-2' will be executed.



```
if(condition)
{
    // statement-block-1
    // to execute if condition is
    true
}
else
{
    // statement-block-2
    // to execute if condition is
    false
}
```

WAP to print if a number is positive or negative

```
import java.util.*;
class MyProgram{
public static void main (String[] args){
int x;
     Scanner sc = new Scanner(System.in);
     x = sc.nextInt();
     if (x > 0){
         System.out.println("Number is positive");
     }//if
     else{
         System.out.println("Number is negative");
       }//else
      }//main
}//class
```

WAP to print if a number is odd or even

```
import java.util.*;
class MyProgram{
public static void main (String[] args){
     int x;
     Scanner sc = new Scanner(System.in);
      x = sc.nextInt();
      if( x % 2 == 1 ){
              System.out.println("number is a odd");
      }
       else{
             System.out.println("number is a even");
       }
}
```

Exercise

1. Any year is entered through the keyboard, write a program to determine whether the year is leap or not.
2. Write a program to check whether a triangle is valid or not, when the three angles of the triangle are entered through the keyboard. A triangle is valid if the sum of all the three angles is equal to 180 degrees.

if…else if…else

Multi way Decision

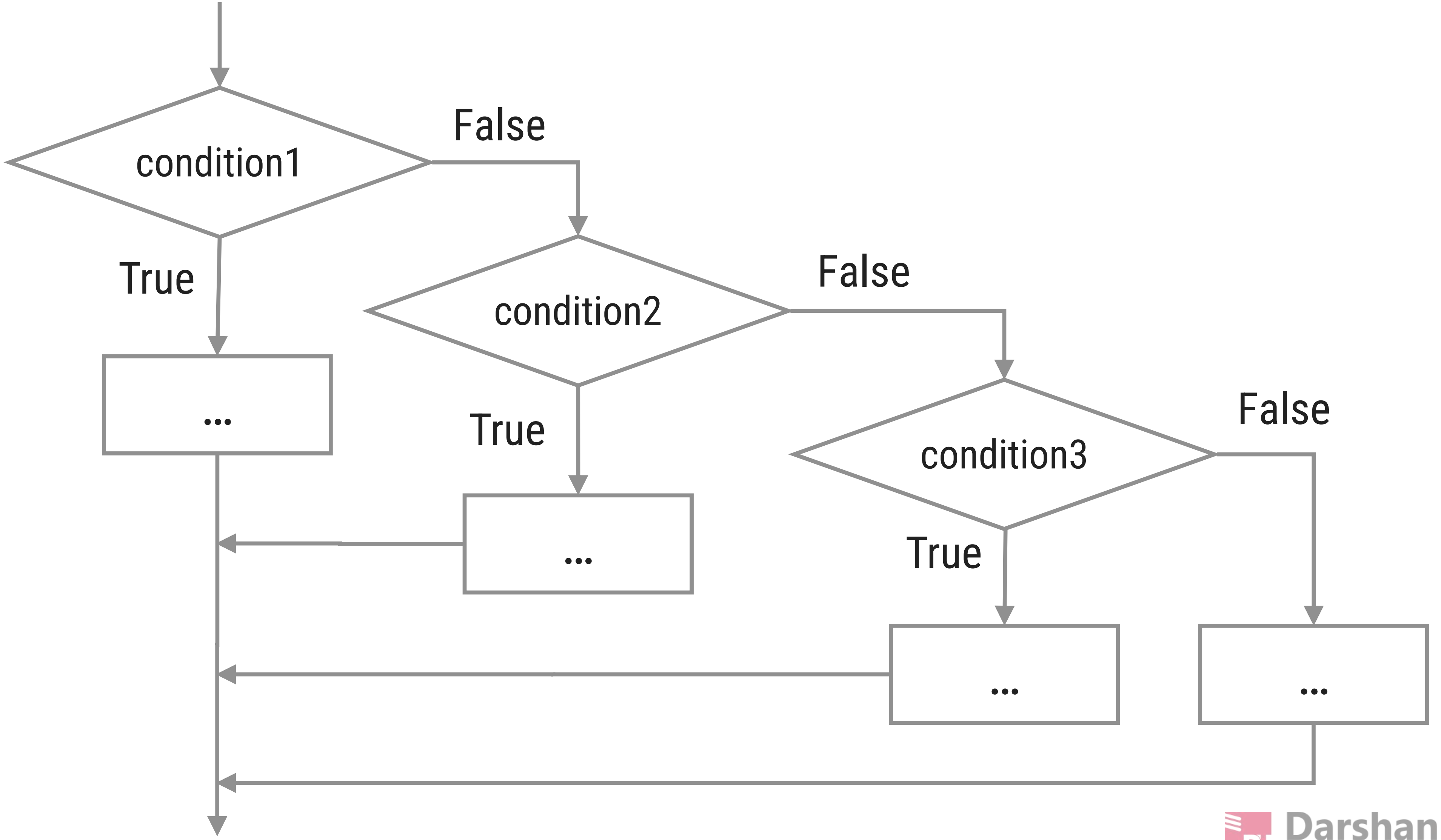
if…else if…else

- **if…else if…else** statement is also known as **if-else-if ladder** which is used for multi way decision making.
- It is used when there are more than two different conditions.
- It tests conditions in a **sequence**, from top to bottom.
- If first condition is true then the associated block with if statement is executed and rest of the conditions are skipped.
- If condition is false then then the next if condition will be tested, if it is true then the associated block is executed and rest of the conditions are skipped. Thus it checks till last condition.
- Condition is tested only and only when all previous conditions are false.
- The last else is the default block which will be executed if none of the conditions are true.
- The last else is not mandatory. If there are no default statements then it can be skipped.

```
if(condition 1)
{
        statement-block1;
}
else if(condition 2)
{
         statement-block2;
}
else if(condition 3)
{
         statement-block3;
}
else if(condition 4)
{
        statement-block4;
}
else
        default-statement;
```

if-else-if ladder

```
if(condition 1)
{
         statement-block1;
}
else if(condition 2)
{
         statement-block2;
}
else if(condition 3)
{
         statement-block3;
}
else if(condition 4)
{
         statement-block4;
}
else
         default-
statement;
```



WAP to print if a number is zero or positive or negative

```
import java.util.*;
class MyProgram{
public static void main (String[] args){
     int x;
     Scanner sc = new Scanner(System.in);
      x = sc.nextInt();
     if(x > 0){
          System.out.println(" number is a positive");
     }
     else if(x < 0) {
          System.out.println(" number is a negative");
     }
     else{
          System.out.println(" number is a zero");
  }
}
```

WAP to print day name from day number

```
public class Demo  {
  public static void main(String[] args)   {
    int d;
     Scanner sc = new Scanner(System.in);
    d = sc.nextInt();
    if (d == 1 )         System.out.println(“Monday”);
    else if (d == 2)     System.out.println(“Tuesday“);
    else if (d == 3)     System.out.println(“Wednesday“);
    else if (d == 4)     System.out.println(“Thursday“);
    else if (d == 5)     System.out.println(“Friday“);
    else if (d == 6)     System.out.println(“Saturday“);
    else                 System.out.println(“Sunday“);
  }
}
```

if-else statement: WAP to display Exam Result

```java
int marks = 65;
if (marks < 60) {
    System.out.println("fail");
} else if (marks >= 60 && marks < 80) {
    System.out.println("B grade");
} else if (marks >= 80 && marks < 90) {
    System.out.println("A grade");
} else if (marks >= 90 && marks < 100) {
    System.out.println("A+ grade");
} else {
    System.out.println("Invalid!");
}
```

Nested If

- A nested if is an if statement that is the target of another if statement.
- Nested if statements mean an if statement inside another if statement.
- The statement connected to the nested if statement is only executed when -:
    - condition of outer if statement is true, and
    - condition of the nested if statement is also true.
- Note: There could be an optional else statement associated with the outer if statement, which is only executed when the condition of the outer if statement is evaluated to be false and in this case, the condition of nested if condition won't be checked at all.

```
if(condition 1)
{
        if(condition 2)
        {
                nested-block;
        }
        else
        {
                nested-block;
        }
}//if
else if(condition 3)
{
         statement-block3;
}
else(condition 4)
{
        statement-block4;
}
```

Nested If statement:WAP Login

- We can also use if/else if statement inside another if/else if statement, this is known as nested if statement.

```java
int username = Integer.parseInt(args[0]);
int password = Integer.parseInt(args[1]);
double balance = 123456.25;

if(username==1234){
    if(password==987654){
        System.out.println("Your Balance is ="+balance);
    }
    else{
        System.out.println("Password is invalid");
    }
}
else{
    System.out.println("Username is invalid");
}
```

Exercise

- In a company an employee is paid as under:
  - If his basic salary is less than Rs. 1500, then HRA = 10% of basic salary and DA = 90% of basic salary.
  - If his salary is either equal to or above Rs. 1500, then HRA = Rs. 500 and DA = 98% of basic salary.
  - Employee's salary is input through the keyboard, write a program to find his gross salary.

switch…case

n-way Decision

Switch…case

- switch…case is a multi-way decision making statement.
- It is similar to if-else-if ladder statement.
- It executes one statement from multiple conditions.

```
switch (expression)
{
   case constant 1:
        // Statement-1
        break;

   case constant 2:
        // Statement-2
        break;

   case constant 3:
        // Statement-3
        break;

default:
      // Statement-default
//  if none of the above case matches
then this block would be executed.
}
```

Switch…case: WAP to print day based on number entered

```
public class Demo  {
 public static void main(String[] args){
 int d;
 Scanner sc= new Scanner(System.in);
 d = sc.nextInt();
```

```
switch (d) {
    case 1:
        System.out.println(“Monday“); break;
    case 2:
        System.out.println(“Tuesday“); break;
    case 3:
        System.out.println(“Wednesday“); break;
    case 4:
        System.out.println(“Thursday“); break;
    case 5:
        System.out.println(“Friday“); break;
    case 6:
        System.out.println(“Saturday“); break;
    case 7:
        System.out.println(“Sunday“); break;
    default:
        System.out.println(“Invalid Day“);
    } //switch
  }
}
```

Switch…case

- switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement.

```
public class SwitchExampleDemo {
    public static void main(String[] args)
    {
        int number = 20;
        switch (number) {
            case 10:
                System.out.println("10");
                break;
            case 20:
                System.out.println("20");
                break;
            default:
                System.out.println("Not 10 or 20");
        }
    }
}
```

Exercise

- Write a menu driven program that allows user to enters five numbers and then choose between finding the smallest, largest, sum or average. Use switch case to determine what action to take. Provide error message if an invalid choice is entered.

Points to remember for switch…case

- The condition in the switch should result in a constant value otherwise it would be invalid.
- In some languages, switch statements can be used for **integer values** only.
- Duplicate case values are not allowed.
- The value for a case must be of the same data type as the variable in the switch.
- The value for a case must be a constant.
- variables are not allowed as an argument in switch statement.
- The break statement is used inside the switch to terminate a statement sequence.
- The break statement is optional, if eliminated, execution will continue on into the next case.
- The default statement is optional and can appear anywhere inside the switch block.

Exercise

- Write a Java program to get a number from the user and print whether it is positive or negative.
- Write a program to find maximum no from given 3 no.
- The marks obtained by a student in 5 different subjects are input through the keyboard.
  - The student gets a division as per the following rules:
    - Percentage above or equals to 60-first division
    - Percentage between 50 to 59-second division
    - Percentage between 40 and 49-Third division
    - Percentage less than 40-fail

  Write a program to calculate the division obtained by the student.
- Write a Java program that takes a number from the user and displays the name of the weekday accordingly (For example if user enter 1 program should return Monday) .

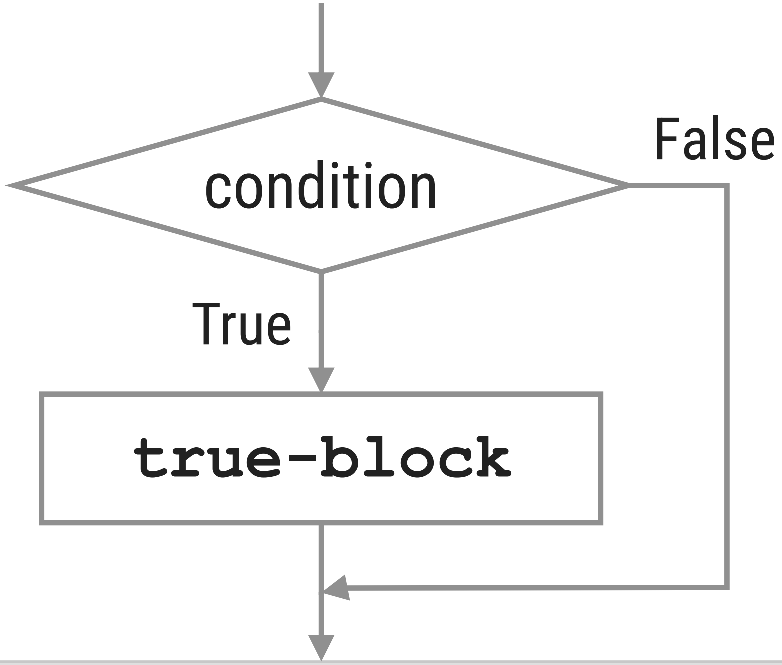
Introduction to loop

Repeatedly execute a block of statements

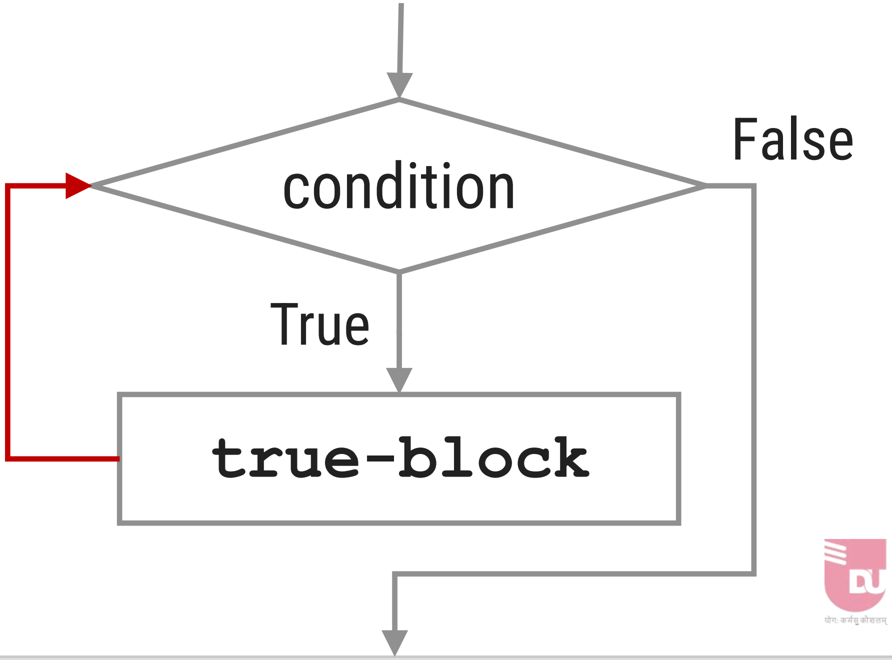
Loop

- Sometimes we need to repeat certain actions several times or till the some criteria is satisfied.
- Loop constructs are used to iterate a block of statements several times.
- Loop constructs repeatedly execute a block of statements for a fixed number of times or till some condition is satisfied

Flowchart of **`if`**
(**`true-block`** executed only once)



Flowchart of **`while`**
(**`true-block`** executed till condition is true)



Looping Statements

- Following are looping statements in any programming language,
  - Entry Controlled `while, for`
  - Exit Controlled `do…while`
  - Unconditional Jump `goto` (It is advised to never use **goto** in a program)

while

Entry Controlled Loop

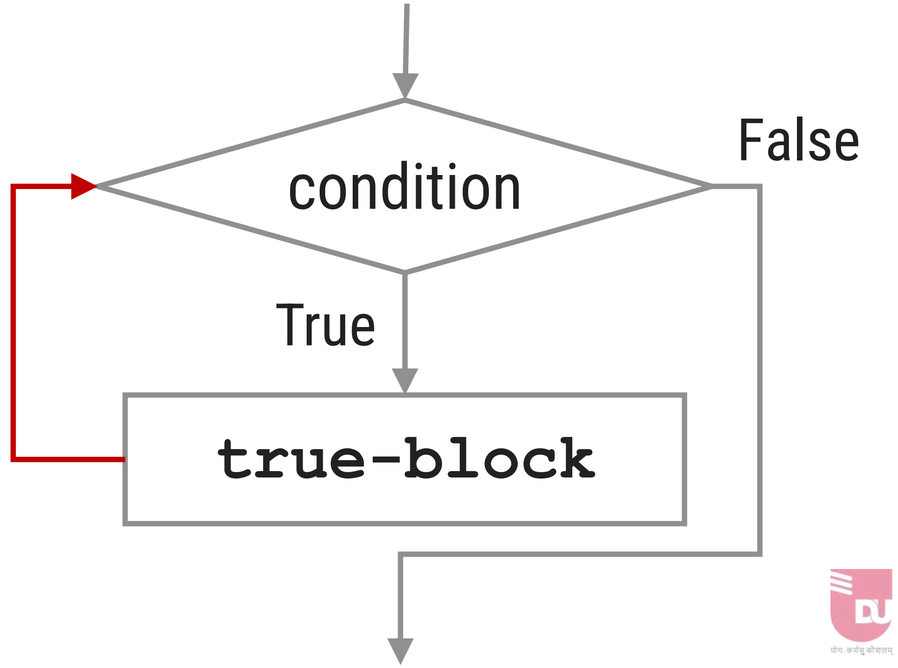
while

- **while** is a entry controlled loop.
- It executes a block of statements till the condition is true.

```
while(condition)
{
      // true-block
}
```

```
int i = 1;
while (i <= 5)
{
  System.out.println(i);
  i++;
}
```

Flowchart of **while**
(**true-block** executed till condition is true)



While Loop

- while loop is used to iterate a part of the program several times. while is entry control loop.
- If the number of iteration is not fixed, it is recommended to use while loop.

```
//code will print 1 to 9
public class WhileLoopDemo {
    public static void main(String[] args) {
        int number = 1;
        while(number < 10) {
            System.out.println(number);
            number++;
        }
    }
}
```

Do-while Loop

- do-while loop is executed at least once because condition is checked after loop body.

```
//code will print 1 to 9
public class DoWhileLoopDemo {
    public static void main(String[] args) {
        int number = 1;
        do {
                System.out.println(number);
                number++;
        }while(number < 10) ;
    }
}
```

WAP to print odd numbers between 1 to n

```java
import java.util.*;
class WhileDemo{
public static void main (String[] args){
 int n,i=1;
 Scanner sc = new Scanner(System.in);
 System.out.print("Enter a number:");
 n = sc.nextInt();
 while(i <= n){
   if(i%2!=0)
     System.out.println(i);
   i++;
 }
}}
```

Output

```
Enter a number:10
1
3
5
7
9
```

WAP to print factors of a given number

```java
import java.util.*;
class WhileDemo{
public static void main (String[] args){
int i=1,n;
Scanner sc = new Scanner(System.in);
System.out.print("Enter a Number:");
n = sc.nextInt();
System.out.print(" Factors:");
while(i <= n){
   if(n%i == 0)
     System.out.print(i +",");
   i++;
 }
}}
```

Output

```
Enter a Number:25
Factors:1,5,25
```

Exercise: while

1. WAP to print multiplication table using while loop
2. Write a program that calculates and prints the sum of the even integers from 1 to 10.

for(;;)

Entry Controlled Loop

for

- **for** is an entry controlled loop
- Statements inside the body of **for** are repeatedly executed till the condition is true

```
for (initialization; condition; increment
                                /decrement)
{
    // statements
}
```

```
int i = 1;
while (i <= 5) {

System.out.print("Hell
o World!");
    i++;
}
```

```
for(i=1; i <= 5; i++)
{

System.out.print("Hel
lo World!");

}
```

- The **initialization** statement is executed only once, at the beginning of the loop.
- Then, the **condition** is evaluated.
  - If the condition is true, statements inside the body of for loop are executed
  - If the condition is false, the for loop is terminated.
- Then, **increment / decrement** statement is executed
- Again the **condition** is evaluated and so on so forth till the condition is true.

For Loop

- for loop is used to iterate a part of the program several times.
- If the number of iteration is fixed, it is recommended to use for loop.

```
//code will print 1 to 9
public class ForLoopDemo {
    public static void main(String[] args)
    {
        for(int number=1;number<10;number++)
        {
                System.out.println(number);
        }
    }
}
```

WAP to print odd numbers between 1 to n

```
import java.util.*;
class MyProgram{
public static void main (String[] args){
    int i=1;
    Scanner sc = new Scanner(System.in);
    n = sc.nextInt();
    for(i=1; i<=n; i++)  {
            if(i%2==1)
                    System.out.println(i);
    }//for
  }//
}
```

WAP to print factors of a given number

```
import java.util.*;
class MyProgram{
public static void main (String[] args){
   int i=1;
   Scanner sc = new  Scanner(System.in);
   n = sc.nextInt();
   for(i=1; i<=n; i++){
      if(n%i == 0)
        System.out.println(i);
  }
 }
}
```

Exercise: for

- Write a program to print average of n numbers.
- Write a program that calculates and prints the sum of the even integers from 1 to 10.

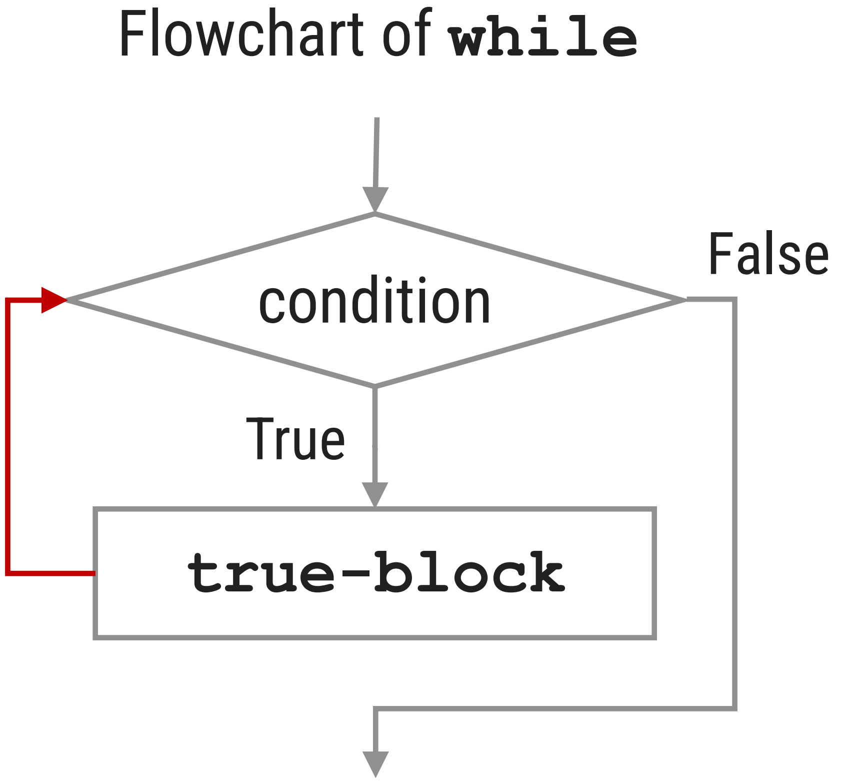
do…while

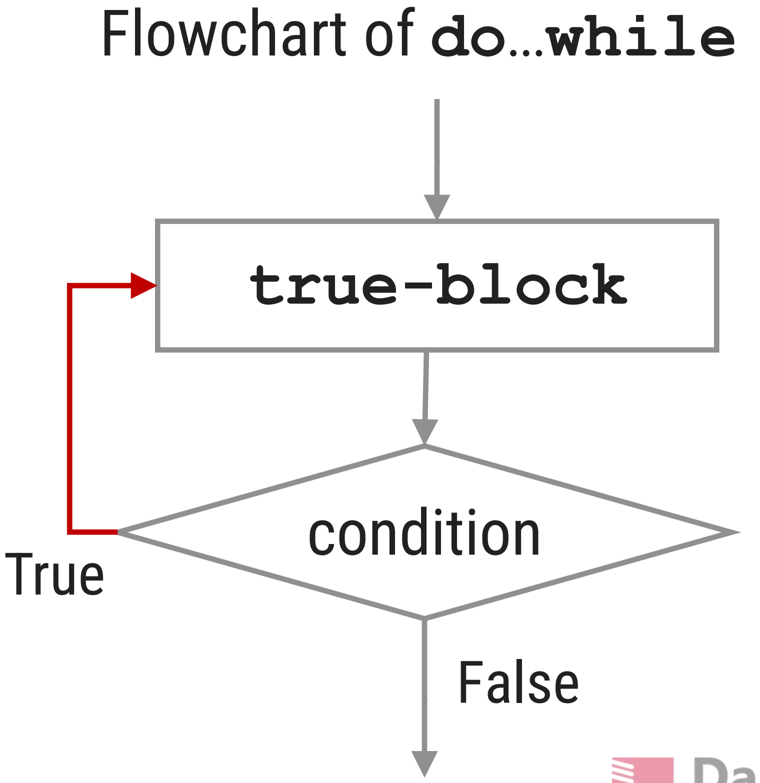
Exit Controlled Loop

do…while

- **do…while** is an exit controlled loop.
- Statements inside the body of **do…while** are repeatedly executed till the condition is true.
- **while** loop executes zero or more times, **do…while** loop executes one or more times.

```
do
{
        // true-block
}
while(condition) ;
```





WAP to print 1 to 10 using do-while loop

```
import java.util.*;
class MyProgram{
public static void main (String[] args){
int i=1;
  do{
      System.out.println(i);
      i++;
    }while(i <= 10);
 }
}
```

continue

Skip the statement in the iteration

continue

- Sometimes, it is required to skip the remaining statements in the loop and continue with the next iteration.
- `continue` statement is used to skip remaining statements in the loop.
- `continue` is keyword.

WAP to calculate the sum of positive numbers.

```java
import java.util.*;
class ContinueDemo{
public static void main(String[] args) {
int a,n,sum=0;
Scanner sc = new Scanner(System.in);
n = sc.nextInt();
for(int i=0;i<n;i++){
  a = sc.nextInt();
  if(a<0){
         continue;
         System.out.println("a="+a);//error:unreachable statement
  }//if
  sum=sum+a;
}//for
 System.out.println("sum="+sum);
 }
}
```

break

Early exit from the loop

break

- Sometimes, it is required to early exit the loop as soon as some situation occurs.
- E.g. searching a particular number in a set of 100 numbers. As soon as the number is found it is desirable to terminate the loop.
- `break` statement is used to jump out of a loop.
- `break` statement provides an early exit from **for**, **while**, **do…while** and **switch** constructs.
- `break` causes exit from the innermost loop or switch.
- `break` is keyword.

WAP to calculate the sum of given numbers. User will enter -1 to terminate.

```
import java.util.*;
class BreakDemo{
public static void main (String[] args){
    int a,sum=0;
    System.out.println("enter numbers_ enter -1 to break");
    Scanner sc = new Scanner(System.in);
    while(true){
        a = sc.nextInt();
        if(a==-1)
             break;
         sum=sum+a;
     }//while
     System.out.println("sum="+sum);
  }
 }
```

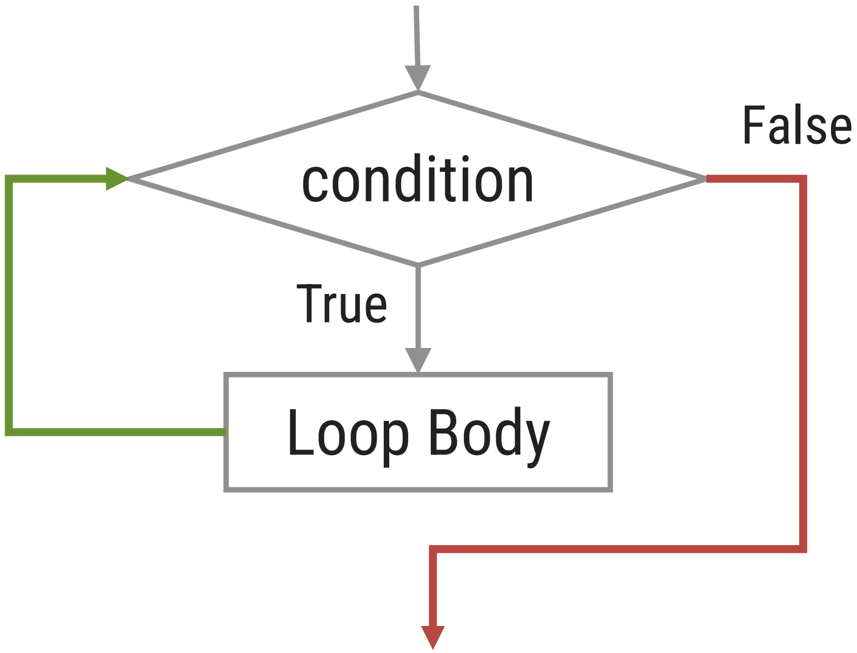
Types of loops

Entry Control Loop

```
int i=1;
while(i<=10)
{
    i++;
}
```

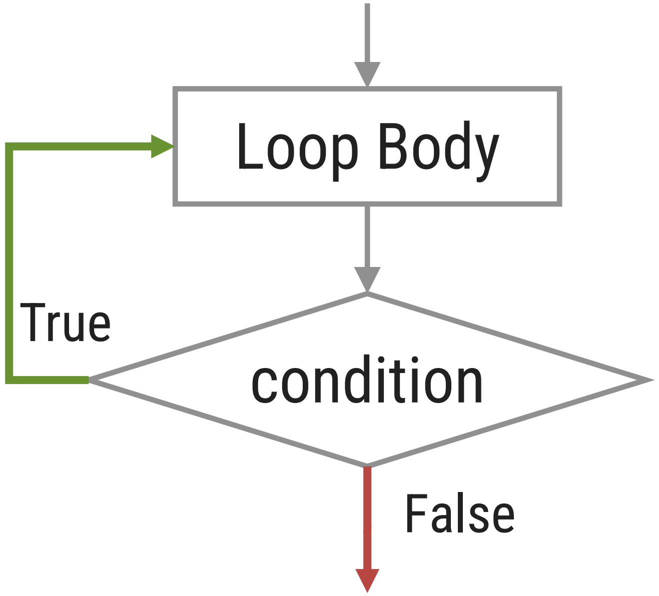
Entry Control Loop

```
int i;
for(i=1;i<=10;i++)
{
    i++;
}
```



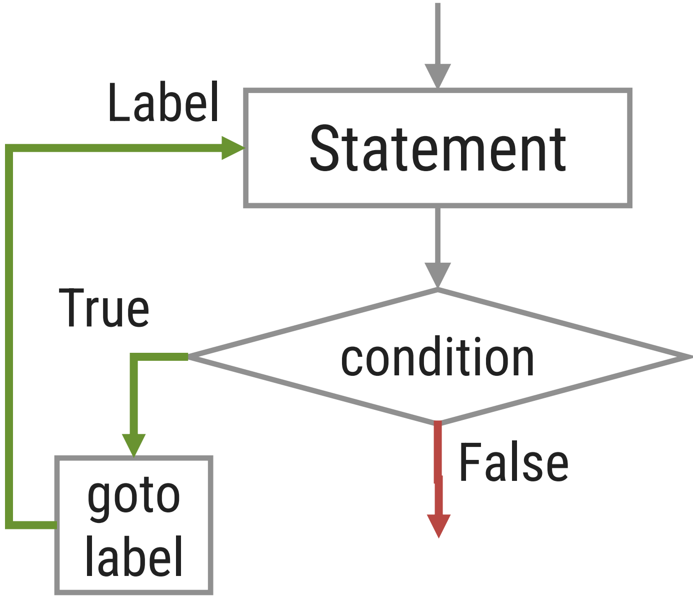
Exit Control Loop

```
int i=1;
do
{
        i++;
}
while(i<=10);
```



Virtual Loop

```
   int i=1;
p: i++;
   if(i<=10)
      goto p;
```



nested loop

loop within a loop

WAP to print given pattern (nested loop)

```
*
**
***
****
*****
```

```java
class PatternDemo{
public static void main(String[] args) {
  int n=5;
  for(int i=1;i<=n;i++){
     for(int j=1;j<=i;j++){
         System.out.print("*");
     }//for j
     System.out.println();
  }//outer for i
 }
}
```

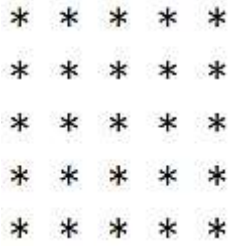
WAP to print given pattern (nested loop)

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

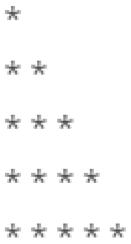
```java
class PatternDemo{
public static void main(String[] args) {
   int n=5;
   for(int i=1;i<=n;i++){
        for(int j=1;j<=i;j++){
              System.out.print(j+"\t");
        }//for j
         System.out.println();
   }//outer for i
   }
}
```

Programs to perform (Looping Statements)

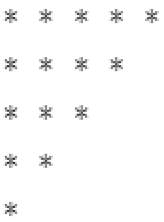
- Write a program to print first n odd numbers.
- Write a program to check that the given number is prime or not.
- Write a program to draw given patterns,



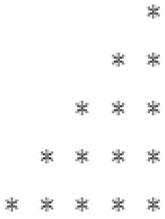
```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```



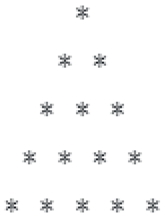
```
*
**
***
****
*****
```



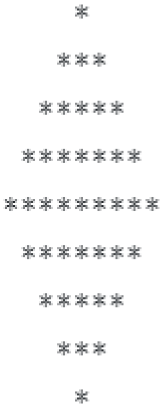
```
* * * * *
* * * *
* * *
* *
*
```



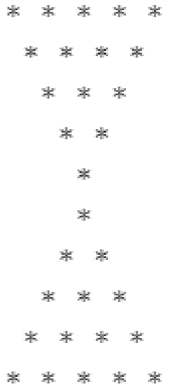
```
        *
      * *
    * * *
  * * * *
* * * * *
```



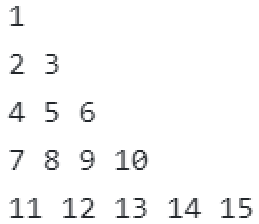
```
    *
   * *
  * * *
 * * * *
* * * * *
```



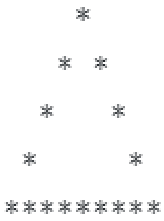
```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```



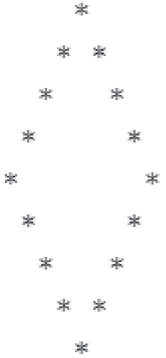
```
* * * * *
 * * * *
  * * *
   * *
    *
    *
   * *
  * * *
 * * * *
* * * * *
```



```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```



```
    *
   * *
  *   *
 *     *
*********
```



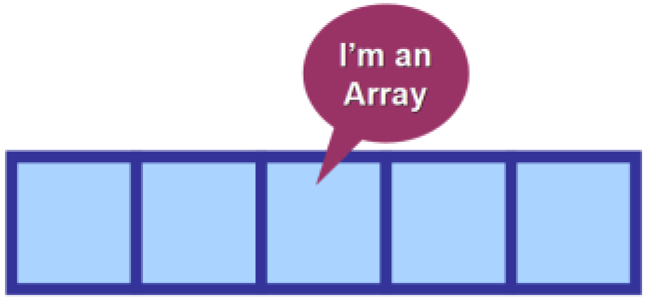
```
    *
   * *
  *   *
 *     *
*       *
 *     *
  *   *
   * *
    *
```

Introduction to Array

Large Data Handling

Why Array?

- Very often we need to deal with relatively large set of data.
- E.g.
  - Percentage of all the students of the college. (May be in thousands)
  - Age of all the citizens of the city. (May be lakhs)
- We need to declare thousands or lakhs of the variable to store the data which is practically not possible.
- We need a solution to store more data in a single variable.
- **Array** is the most appropriate way to handle such data.
- As per English Dictionary, "***Array means collection or group or arrangement in a specific order.***"



Array

- An array is a fixed size sequential collection of elements of same data type grouped under single variable name.

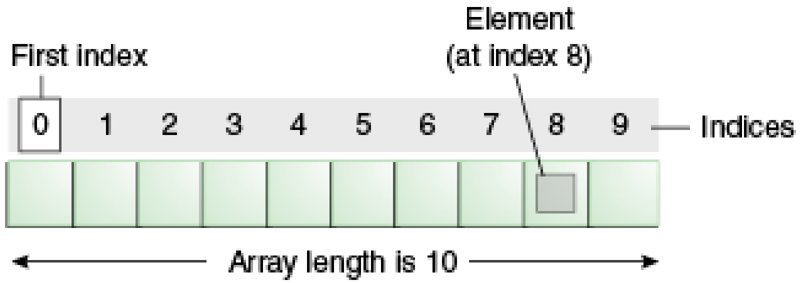
```
int percentage[10];
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fixed Size	Sequential	Same Data type	Single Variable Name
The size of an array is fixed at the time of declaration which cannot be changed later on. Here **array size** is **10**.	All the elements of an array are stored in a consecutive blocks in a memory. **10** (0 to 9)	Data type of all the elements of an array is same which is defined at the time of declaration. Here **data type** is **int**	All the elements of an array will be referred through common name. Here **array name** is **percentage**

Array declaration

- Normal Variable Declaration: int **a**;
- Array Variable Declaration: int **b**[10];
- Individual value or data stored in an array is known as an **element of an array**.
- Positioning / indexing of an elements in an array always **starts with 0 not 1**.
  - If 10 elements in an array then index is 0 to 9
  - If 100 elements in an array then index is 0 to 99
  - If 35 elements in an array then index is 0 to 34
- Variable **a** stores 1 integer number where as variable **b** stores 10 integer numbers which can be accessed as b[0], b[1], b[2], b[3], b[4], b[5], b[6], b[7], b[8] and b[9]



Array

- Important point about Java array.
  - An array is **derived** datatype.
  - An array is **dynamically** allocated.
  - The individual elements of an array is refereed by their **index/subscript** value.
  - The **subscript** for an array always begins with **0**.

35	13	28	106	35	42	5	83	97	14
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

One-Dimensional Array

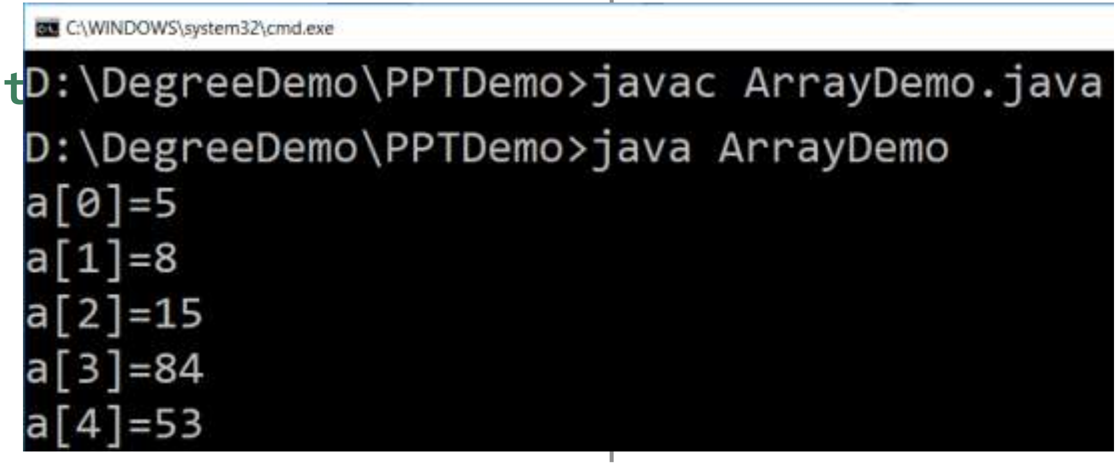
- An array using **one subscript** to represent the **list of elements** is called **one dimensional array**.
- A One-dimensional array is essentially a **list** of **like-typed variables.**
- Array declaration: type var-name[];

  Example: int student_marks[];
- Above example will represent array with no value (null).
- To link **student_marks** with actual array of integers, we must allocate one using ***new*** keyword.

  Example: int student_marks[] = ***new*** int[20];

Example (Array)

```
public class ArrayDemo{
    public static void main(String[] args) {
        int a[]; // or int[] a
        // till now it is null as it does not assigned any memory

        a = new int[5]; // here we actually creat
        a[0] = 5;
        a[1] = 8;
        a[2] = 15;
        a[3] = 84;
        a[4] = 53;

        /* in java we use length property to determine the length
         * of an array, unlike c where we used sizeof function */
        for (int i = 0; i < a.length; i++) {
                System.out.println("a["+i+"]="+a[i]);
        }
    }
}
```



```
D:\DegreeDemo\PPTDemo>javac ArrayDemo.java
D:\DegreeDemo\PPTDemo>java ArrayDemo
a[0]=5
a[1]=8
a[2]=15
a[3]=84
a[4]=53
```

WAP to store 5 numbers in an array and print them

```java
import java.util.*;
class ArrayDemo1{
public static void main (String[] args){
        int i, n;
        int[] a=new int[5];
        Scanner sc = new Scanner(System.in);
        System.out.print("enter Array Length:");
        n = sc.nextInt();
        for(i=0; i<n; i++) {
            System.out.print("enter a["+i+"]:");
            a[i] = sc.nextInt();
        }
        for(i=0; i<n; i++)
            System.out.println(a[i]);
    }
}
```

Output:

```
enter Array
Length:5
enter a[0]:1
enter a[1]:2
enter a[2]:4
enter a[3]:5
enter a[4]:6
1
2
4
5
6
```

For-each loop

```
class Array1{
    public static void main(String[] args) {
        int a[ ]={10,20,30,40,50};
        for(int i : a){// for each
            System.out.println(i);
        }
    }
}
```

Limitations of for-each loop

1. For-each loops are not appropriate when you want to modify the array.
2. For-each loops **do not keep track of index**. So we can not obtain array index using For-Each loop.
3. For-each **only iterates forward over the array in single steps.**
4. For-each **cannot process two decision making statements** at once.
5. For-each also has some **performance overhead** over simple iteration.

WAP to print elements of an array in reverse order

```java
import java.util.*;
public class RevArray{
public static void  main(String[] args) {
  int i, n;
  int[] a;
  Scanner sc=new Scanner(System.in);
  System.out.print("Enter Size of an Array:");
  n=sc.nextInt();
  a=new int[n];
  for(i=0; i<n; i++){
      System.out.print("enter a["+i+"]:");
      a[i]=sc.nextInt();
  }
  System.out.println("Reverse Array");
  for(i=n-1; i>=0; i--)
      System.out.println(a[i]);
 }
}
```

Output:

```
Enter Size of an
Array:5
enter a[0]:1
enter a[1]:2
enter a[2]:3
enter a[3]:4
enter a[4]:5
Reverse Array
5
4
3
2
1
```

WAP to count positive number, negative number and zero from an array of n size

```java
import java.util.*;
class ArrayDemo1{
public static void main (String[] args){
     int n,pos=0,neg=0,z=0;
     int[] a=new int[5];
     Scanner sc = new Scanner(System.in);
     System.out.print("enter Array Length:");
     n = sc.nextInt();
     for(int i=0; i<n; i++) {
         System.out.print("enter a["+i+"]:");
         a[i] = sc.nextInt();
         if(a[i]>0)
           pos++;
         else if(a[i]<0)
           neg++;
         else
           z++;
     }
     System.out.println("Positive no="+pos);
     System.out.println("Negative no="+neg);
     System.out.println("Zero no="+z);
}}
```

Output:

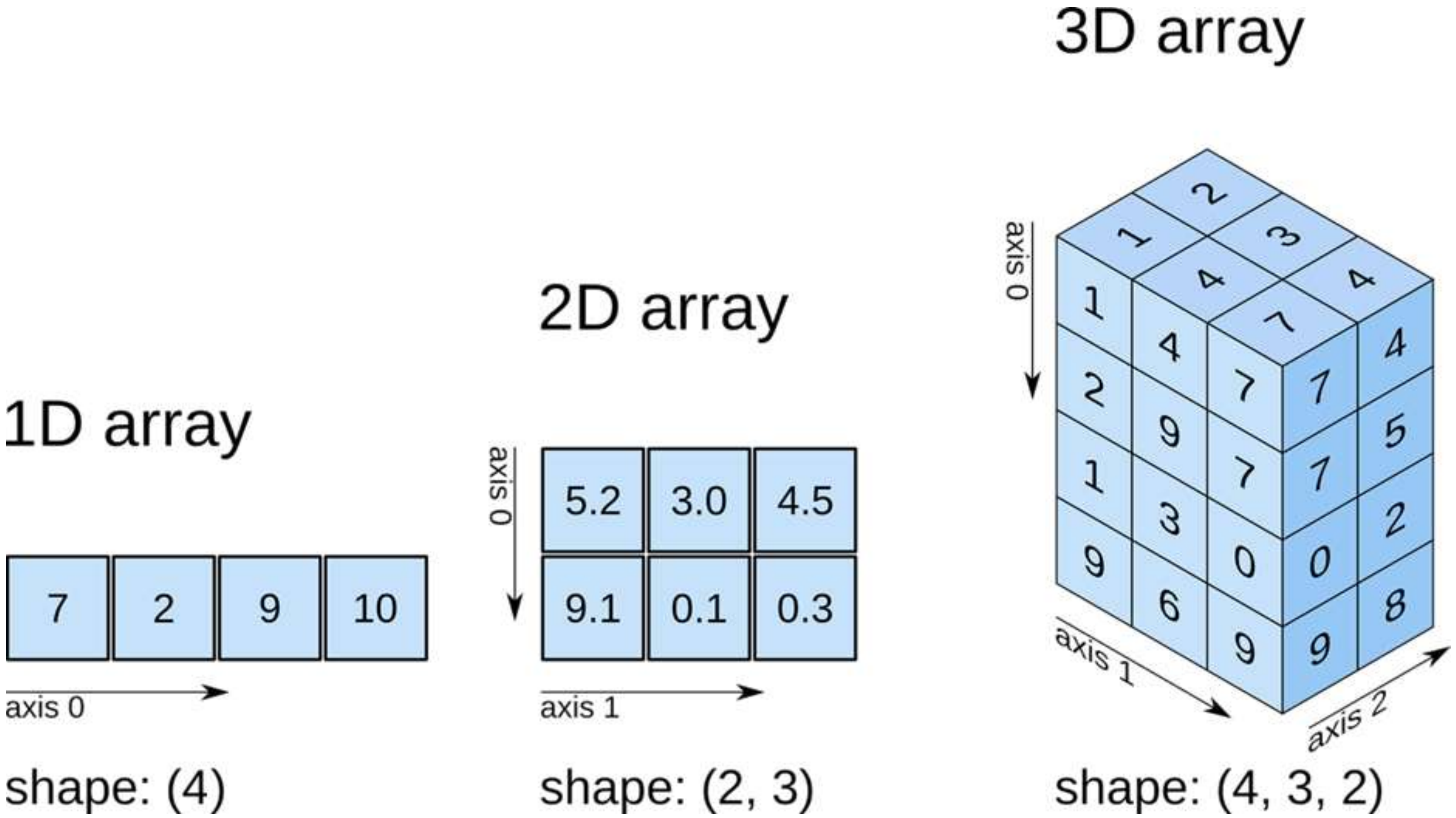
```
enter Array
Length:5
enter a[0]:-3
enter a[1]:5
enter a[2]:0
enter a[3]:-2
enter a[4]:00
Positive no=1
Negative no=2
Zero no=2
```

Exercise: Array

1. WAP to count odd and even elements of an array.
2. WAP to calculate sum and average of n numbers from an array.
3. WAP to find largest and smallest from an array.

Multidimensional Array

Multidimensional Array



WAP to read 3 x 3 elements in 2d array

```java
import java.util.*;
class Array2Demo{
public static void main(String[] args) {
  int size;
  Scanner sc=new Scanner(System.in);
  System.out.print("Enter size of an array");
  size=sc.nextInt();
  int a[][]=new int[size][size];
  for(int i=0;i<a.length;i++){
    for(int j=0;j<a.length;j++){
            a[i][j]=sc.nextInt();
    }
  }

  for(int i=0;i<a.length;i++){
    for(int j=0;j<a.length;j++){
          System.out.print("a["+i+"]["+j+"]:"+a[i][j]+
    }
    System.out.println();
  }
 }
}
```

	Column-0	Column-1	Column-2
Row-0	**11**	**18**	**-7**
Row-1	**25**	**100**	**0**
Row-2	**-4**	**50**	**88**

Output:

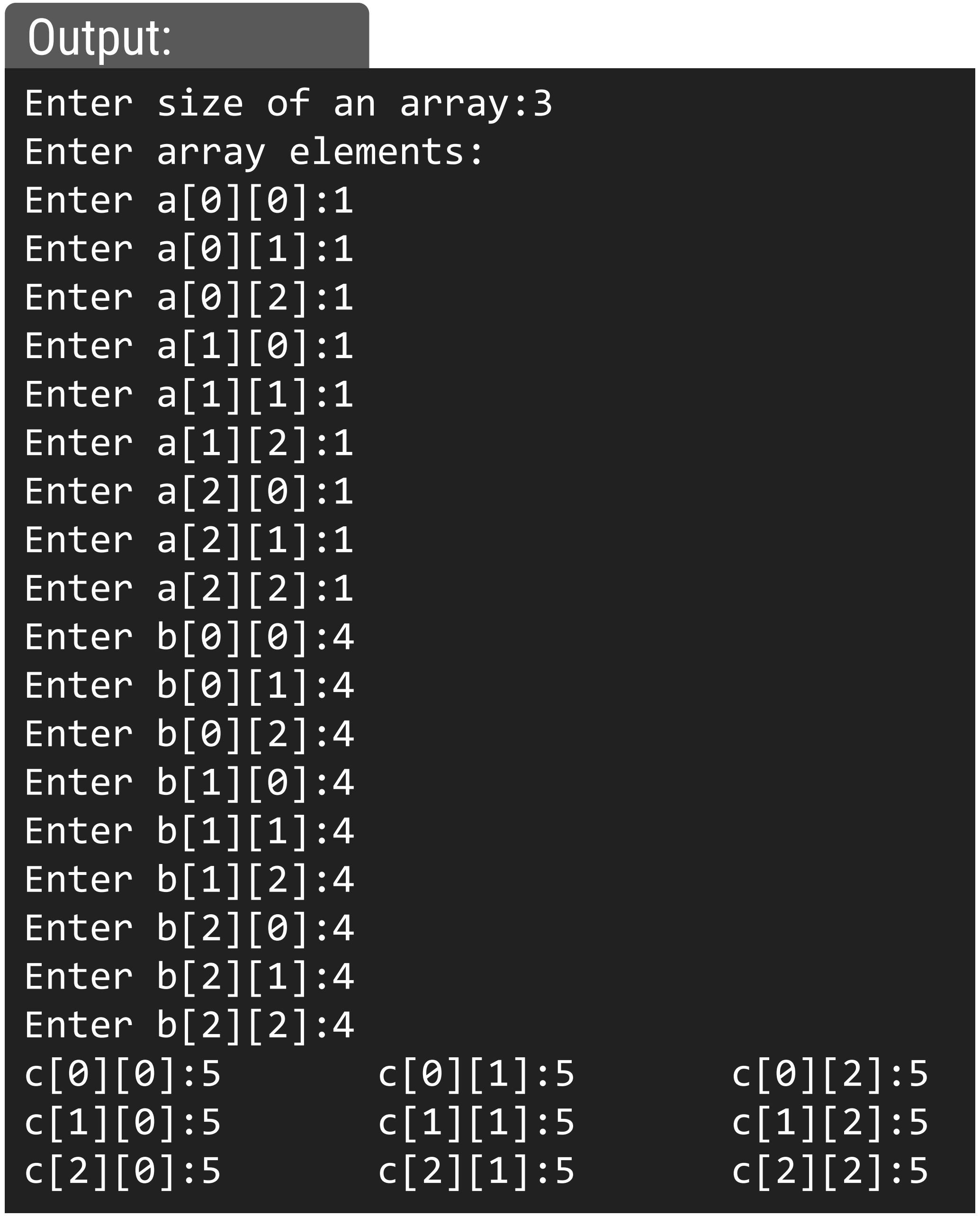
```
11
12
13
14
15
16
17
18
19
a[0][0]:11      a[0][1]:12      a[0][2]:13
a[1][0]:14      a[1][1]:15      a[1][2]:16
a[2][0]:17      a[2][1]:18      a[2][2]:19
```

WAP to perform addition of two 3 x 3 matrices

```java
import java.util.*;
class Array2Demo{
public static void main(String[] args) {
int size;
int a[][],b[][],c[][];
Scanner sc=new Scanner(System.in);
 System.out.print("Enter size of an array:");
 size=sc.nextInt();
 a=new int[size][size];
 System.out.println("Enter array elements:");
for(int i=0;i<a.length;i++){
    for(int j=0;j<a.length;j++){
        System.out.print("Enter a["+i+"]["+j+"]:");
        a[i][j]=sc.nextInt();
    }
}
```

```java
b=new int[size][size];
for(int i=0;i<b.length;i++){
 for(int j=0;j<b.length;j++){
   System.out.print("Enter b["+i+"]["+j+"]:");
   b[i][j]=sc.nextInt();
 }
}
c=new int[size][size];
for(int i=0;i<c.length;i++){
 for(int j=0;j<c.length;j++){
 System.out.print("c["+i+"]["+j+"]:“ +(a[i][j]+b[i][j])+"\t");
  }
  System.out.println();
  }
 }//main()
}//class
```

WAP to perform addition of two 3 x 3 matrices

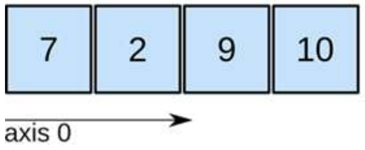


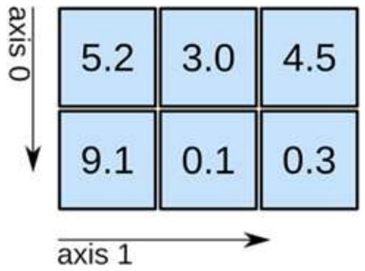
Output:

```
Enter size of an array:3
Enter array elements:
Enter a[0][0]:1
Enter a[0][1]:1
Enter a[0][2]:1
Enter a[1][0]:1
Enter a[1][1]:1
Enter a[1][2]:1
Enter a[2][0]:1
Enter a[2][1]:1
Enter a[2][2]:1
Enter b[0][0]:4
Enter b[0][1]:4
Enter b[0][2]:4
Enter b[1][0]:4
Enter b[1][1]:4
Enter b[1][2]:4
Enter b[2][0]:4
Enter b[2][1]:4
Enter b[2][2]:4
c[0][0]:5       c[0][1]:5       c[0][2]:5
c[1][0]:5       c[1][1]:5       c[1][2]:5
c[2][0]:5       c[2][1]:5       c[2][2]:5
```

Initialization of an array elements

1. **One dimensional Array**
    1. int a[ ] = { 7, 3, -5, 0, 11 }; // a[0]=7, a[1] = 3, a[2] = -5, a[3] = 0, a[4] = 11
    2. int a[ ] = { 7, 3 }; // a[0] = 7, a[1] = 3
    3. int a[ ] = { 0 }; // all elements of an array are initialized to 0





2. **Two dimensional Array**
    1. int a[ ][ ] = { { 7, 3, -5, 10 }, { 11, 13, -15, 2} }; // 1st row is 7, 3, -5, 10 & 2nd row is 11, 13, -15, 2

Multi-Dimensional Array

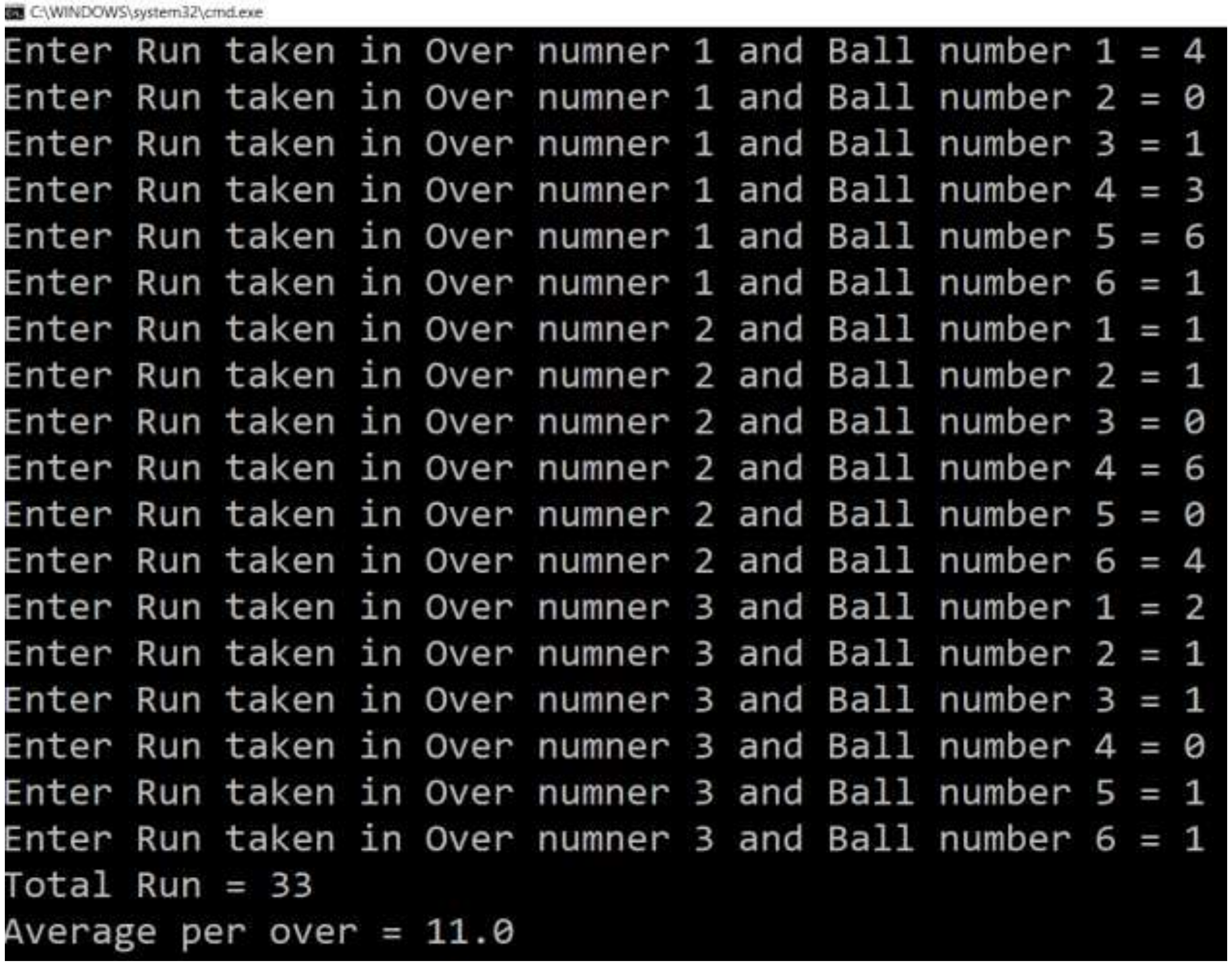
- In java, multidimensional array is actually **array of arrays.**
- **Example**: int runPerOver[][] = new int[3][6];

First Over (a[0])	**4** a[0][0]	**0** a[0][1]	**1** a[0][2]	**3** a[0][3]	**6** a[0][4]	**1** a[0][5]
Second Over (a[1])	**1** a[1][0]	**1** a[1][1]	**0** a[1][2]	**6** a[1][3]	**0** a[1][4]	**4** a[1][5]
Third Over (a[2])	**2** a[2][0]	**1** a[2][1]	**1** a[2][2]	**0** a[2][3]	**1** a[2][4]	**1** a[2][5]

- **`length` field:**
  - If we use length field with multidimensional array, it will return length of first dimension.
  - Here, if **runPerOver.length** is accessed it will return **3**
  - Also if **runPerOver[0].length** is accessed it will be **6**

Multi-Dimensional Array (Example)

```
Scanner s = new Scanner(System.in);
int runPerOver[][] = new int[3][6];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 6; j++) {
        System.out.print("Enter Run taken" +
        " in Over numner " + (i + 1) +
        " and Ball number " + (j + 1) + " = ");
        runPerOver[i][j] = s.nextInt();
    }
}
int totalRun = 0;
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 6; j++) {
        totalRun += runPerOver[i][j];
    }
}
double average = totalRun / (double) runPerOver.length;
System.out.println("Total Run = " + totalRun);
System.out.println("Average per over = " + average);
```



```
Enter Run taken in Over numner 1 and Ball number 1 = 4
Enter Run taken in Over numner 1 and Ball number 2 = 0
Enter Run taken in Over numner 1 and Ball number 3 = 1
Enter Run taken in Over numner 1 and Ball number 4 = 3
Enter Run taken in Over numner 1 and Ball number 5 = 6
Enter Run taken in Over numner 1 and Ball number 6 = 1
Enter Run taken in Over numner 2 and Ball number 1 = 1
Enter Run taken in Over numner 2 and Ball number 2 = 1
Enter Run taken in Over numner 2 and Ball number 3 = 0
Enter Run taken in Over numner 2 and Ball number 4 = 6
Enter Run taken in Over numner 2 and Ball number 5 = 0
Enter Run taken in Over numner 2 and Ball number 6 = 4
Enter Run taken in Over numner 3 and Ball number 1 = 2
Enter Run taken in Over numner 3 and Ball number 2 = 1
Enter Run taken in Over numner 3 and Ball number 3 = 1
Enter Run taken in Over numner 3 and Ball number 4 = 0
Enter Run taken in Over numner 3 and Ball number 5 = 1
Enter Run taken in Over numner 3 and Ball number 6 = 1
Total Run = 33
Average per over = 11.0
```

Multi-Dimensional Array (Cont.)

- **manually** allocate **different** size:

```
        int runPerOver[][] = new int[3][];
        runPerOver[0] =  new int[6];
        runPerOver[1]  =  new int[7];
        runPerOver[2]  =  new int[6];
```

- **initialization**:

```
    int runPerOver[][] = {
                        {0,4,2,1,0,6},
                        {1,-1,4,1,2,4,0},
                        {6,4,1,0,2,2},
              }
```

Note : here to specify extra runs (Wide, No Ball etc..) negative values are used

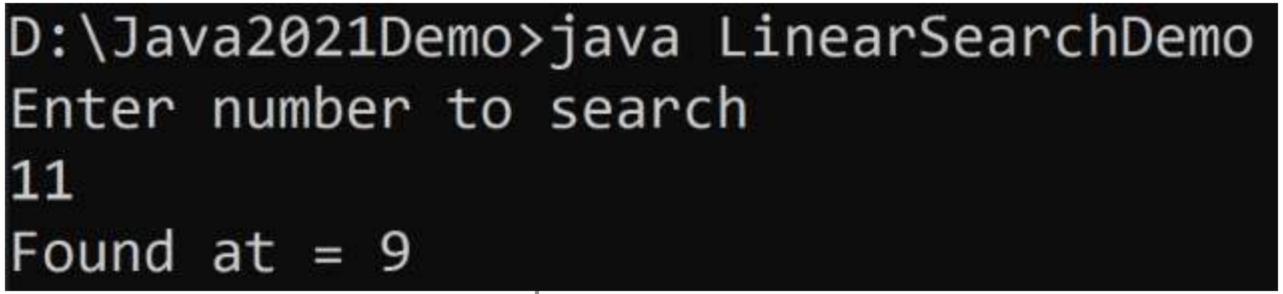
Searching in Array

- Searching is the process of looking for a specific element in an array. for example, discovering whether a certain element is included in the array.
- Searching is a common task in computer programming. Many algorithms and data structures are devoted to searching.
- We will discuss two commonly used approaches,
  - **Linear Search:** The linear search approach compares the key element key sequentially with each element in the array. It continues to do so until the key matches an element in the array or the array is exhausted without a match being found.
  - **Binary Search:** The binary search first compares the key with the element in the middle of the array. Consider the following three cases:
    - If the key is less than the middle element, you need to continue to search for the key only in the first half of the array.
    - If the key is equal to the middle element, the search ends with a match.
    - If the key is greater than the middle element, you need to continue to search for the key only in the second half of the array.

    Note: Array should be sorted in ascending order if we want to use Binary Search.

Linear Search

```java
import java.util.Scanner;
public class LinearSearchDemo{
    public static void main(String[] args){
        int[] myArray = {5,3,6,8,4,6,2,8,9,11};
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number to search");
        int searchNumber = sc.nextInt();
        sc.close();
        boolean flag = false;
        for(int i=0;i<myArray.length;i++){
            if(myArray[i]==searchNumber){
                System.out.println("Found at = "+i);
                flag = true;
                break;
            }
        }
        if(!flag){
            System.out.println("Number does not exist in array");
        }
    }
}
```



```
D:\Java2021Demo>java LinearSearchDemo
Enter number to search
11
Found at = 9
```

Binary Search (Animation)

0	2	5	10	32	51	52	56	57	61	64	76	79	84	87	91

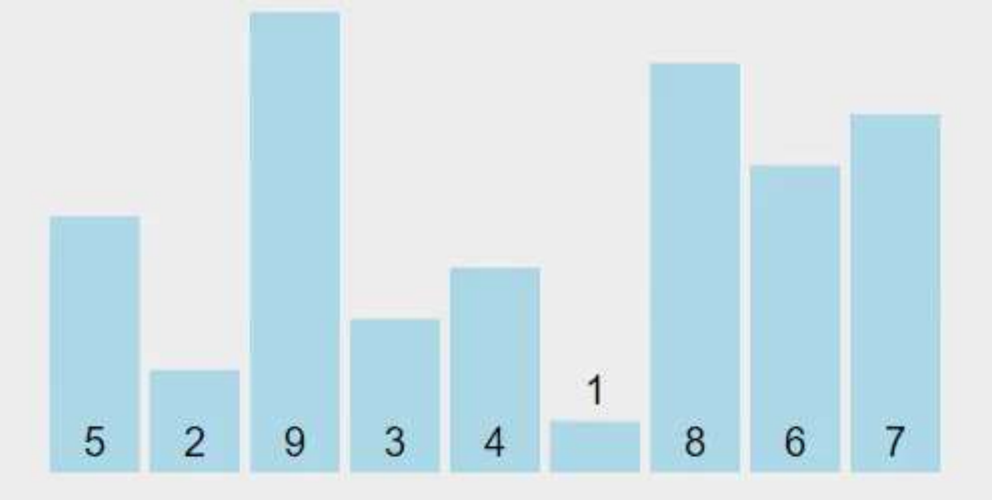
Binary Search

```
import java.util.Scanner;
public class BinarySearchDemo {

public static void main(String[] args) {

    int[] myArray
        = {2,3,5,6,8,9,11,18,29,65};

    Scanner sc = new Scanner(System.in);

    System.out.println
        ("Enter number to search");
    int searchNumber = sc.nextInt();
    sc.close();

    int low = 0;
    int high = myArray.length - 1;
    boolean isFound = false;
```

```
    while(high>=low) {
        int mid = (high + low) / 2 ;
        if(searchNumber < myArray[mid]) {
                high = mid - 1;
        }
        else if(searchNumber == myArray[mid]) {
            System.out.println("Found at = "+mid);
            isFound = true;
            break;
        }
        else {
                low = mid + 1;
        }
    }
    if(!isFound) {
        System.out.println
                ("Number does not exist in array");
    }
}
}
```

Sorting Array

- Sorting, like searching, is a common task in computer programming. Many different algorithms have been developed for sorting.
- There are many sorting techniques available, we are going to explore selection sort.
- Selection sort
  - finds the smallest number in the list and swaps it with the first element.
  - It then finds the smallest number remaining and swaps it with the second element, and so on, until only a single number remains.



Selection Sort (Example)

```java
int a[] = { 5, 2, 9, 3, 4, 1, 8, 6, 7 };
for (int i = 0; i < a.length - 1; i++) {
    // Find the minimum in the list[i..a.length-1]
    int currentMin = a[i];
    int currentMinIndex = i;
    for (int j = i + 1; j < a.length; j++) {
        if (currentMin > a[j]) {
            currentMin = a[j];
            currentMinIndex = j;
        }
    }
    // Swap a[i] with a[currentMinIndex] if necessary
    if (currentMinIndex != i) {
        a[currentMinIndex] = a[i];
        a[i] = currentMin;
    }
}
for(int temp: a) { // this is foreach loop
        System.out.print(temp + ", ");
}
```

Thank You