II B.Tech I Sem

23CS303 : Object Oriented Programming Through Java

Dr.K.Anuradha

Senior Professor,

Dean-SDC,

Department of CSE

Narsimha Reddy Engineering College

UNIT- I

Object Oriented Thinking

OBJECT ORIENTED THINKING

01 Introduction, Need of object-oriented programming

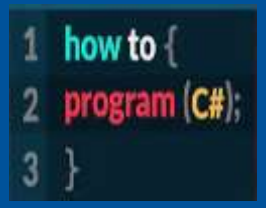
02 principles of object-oriented languages, Applications of OOP, history of JAVA

03 Java Virtual Machine, Java features, Program structures, Installation of JDK

04 Variables, Primitive data types, Identifiers-Naming Conventions, Keywords, Literals

05 Operators-Binary, Unary and Ternary, Expressions, Primitive Type conversion and casting, flow of control- branching, conditional, loops.

1



Q. What is a program?

Ans. A sequence of instructions that a computer can interpret and execute.

2

What is a Programming Language?

A programming language is a computer language that is used by programmers (developers) to communicate with computers.

3

As we know, to communicate with a person, we need a specific language, similarly to communicate with computers, programmers also need a language is called Programming language.

High-level programming language includes **Python, Java, JavaScript, PHP, C#, C++, Objective C, Cobol, Perl, Pascal, LISP, FORTRAN, and Swift programming language**.

Object Oriented Programming (OOP)

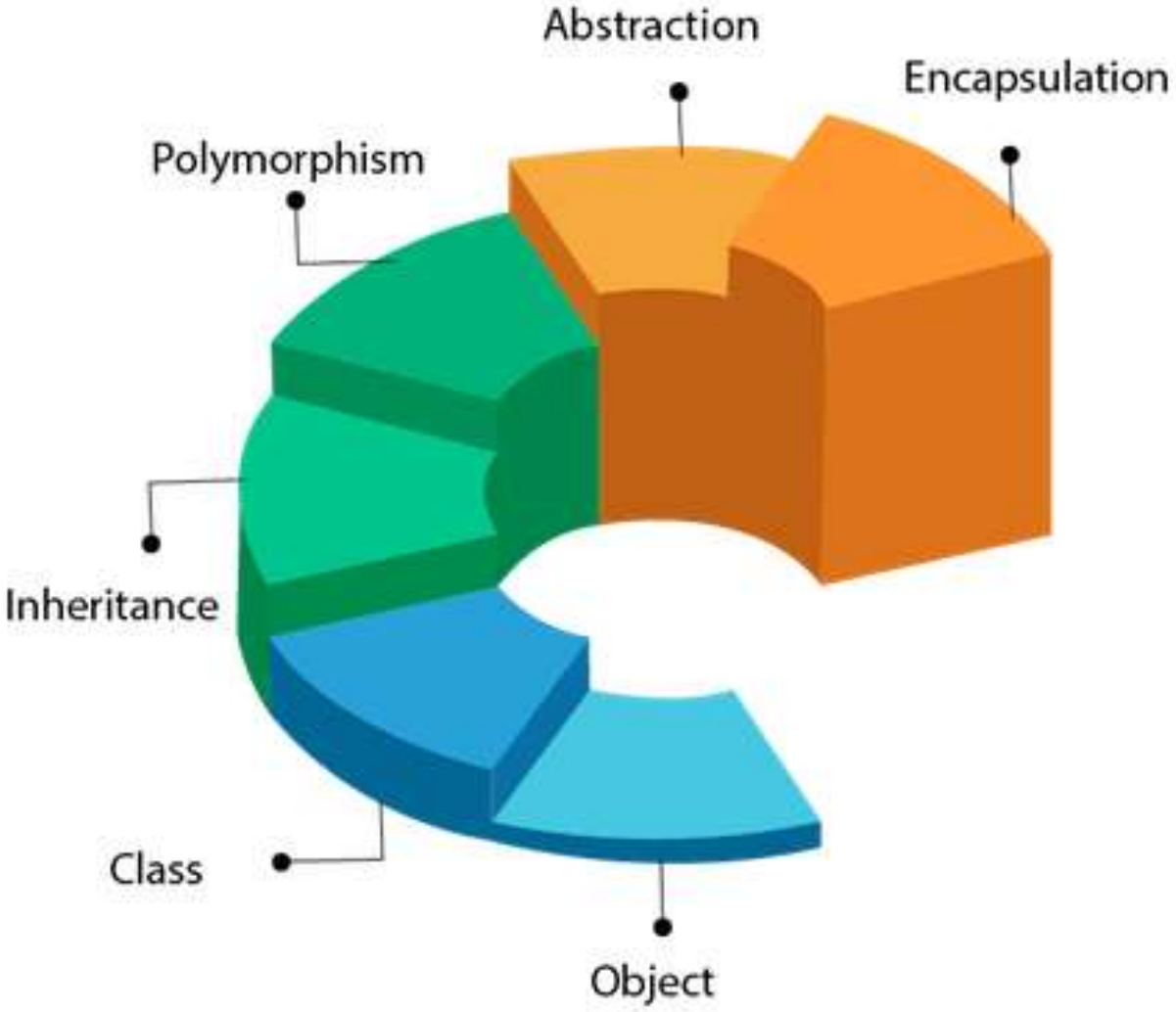
- It is an approach to program organization and development that attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several powerful new concepts.
- It is a new way of organizing and developing programs and has nothing to do with any particular language.
- However, not all languages are suitable to implement the OOP concepts easily.
- OOP allows us to decompose a problem into number of entities called objects and then build data and methods (functions) around these entities.

Need of Object Oriented Programming (OOP)

The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Using the OOPs methodology, we can enhance the **code reusability and save development time**.

Principles of object-oriented languages



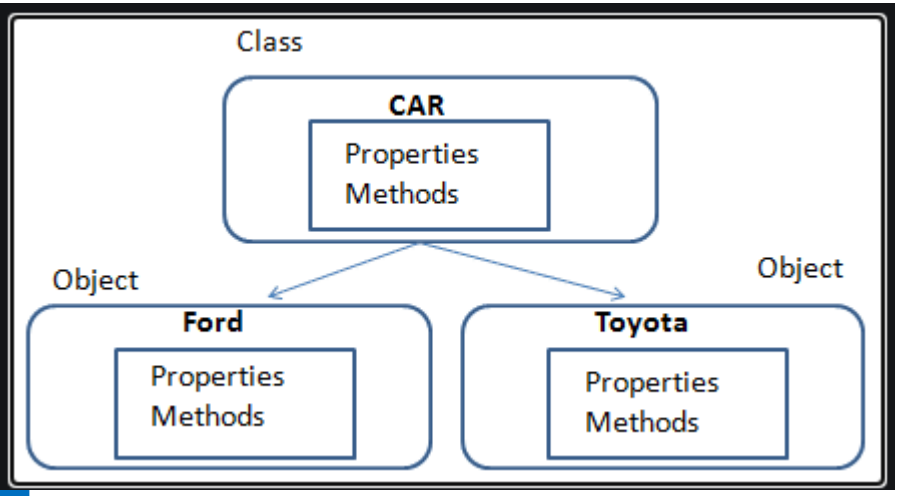
Object

Any entity that has state and behavior is known as an object.
For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called class. It is a logical entity.



Polymorphism

- If one task is performed in different ways, it is known as polymorphism.
- For example: to convince the customer differently, to draw something,

for example, shape, triangle, rectangle, etc.

- Another example can be to speak something;
- for example, a cat speaks meow, dog barks woof, etc.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability.

For example

We all have seen the updates which receive on gadgets both in terms of hardware and software. In software, regular updates are with some new features are also the inheritance.

Abstraction

Hiding internal details (implementation) and showing essential information is known as abstraction.

For example phone call, we don't know the internal processing.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. It is the fully encapsulated class because all the data members are private here.

Applications of OOPs

- Computer graphics applications
- Object-oriented database
- User-interface design such as windows
- Real-time systems
- Simulation and modeling
- Client-Server System
- Artificial Intelligence System
- CAD/CAM Software
- Office automation system

History of Java

1) James Gosling **, Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

JVM (Java Virtual Machine)

- **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
- **An implementation** Its implementation is known as JRE (Java Runtime Environment).
- **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

JVM (Java Virtual Machine)

The JVM performs following operation:

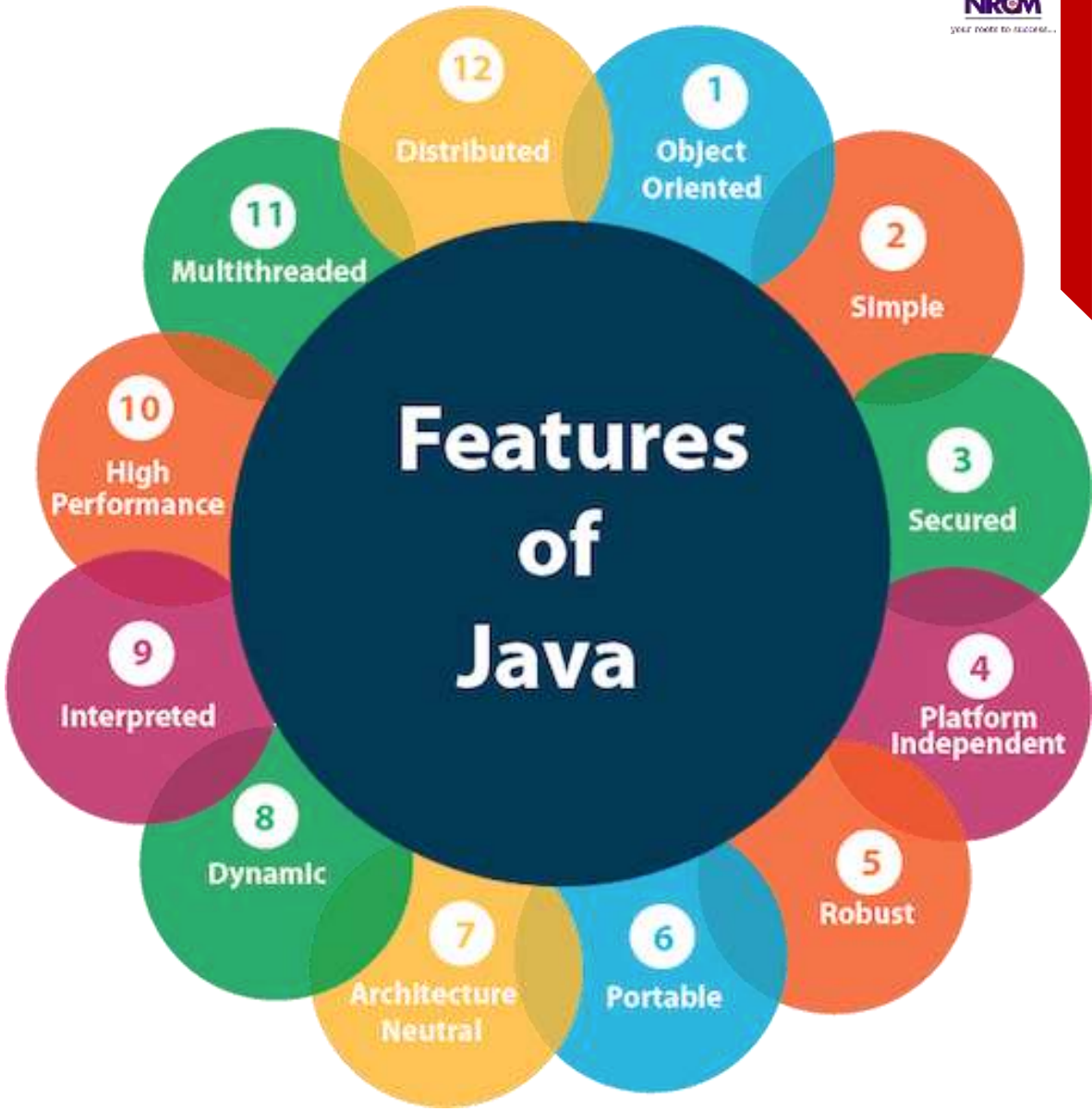
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Java Features

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic



Simple

- Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:
- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

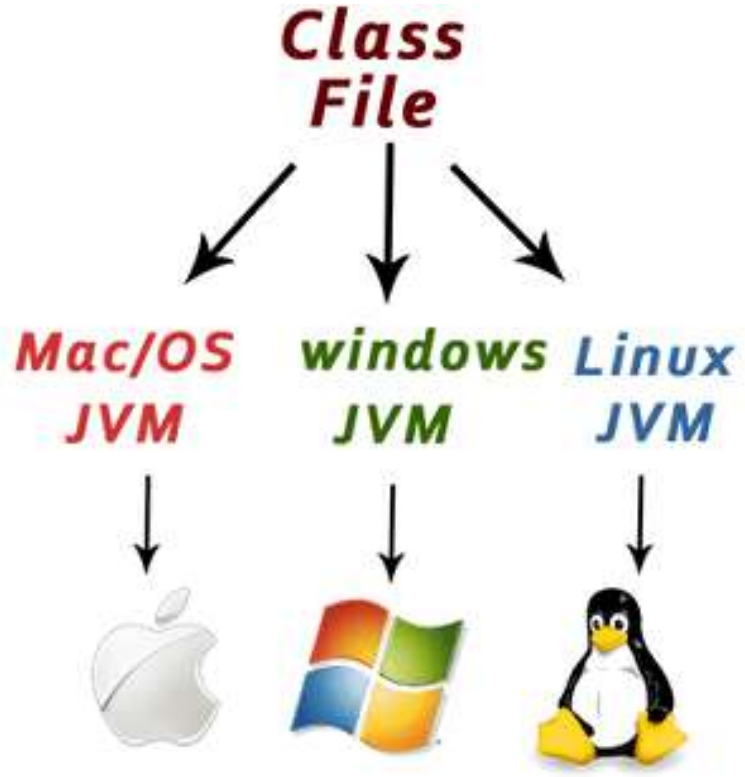
- Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Portable

- Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

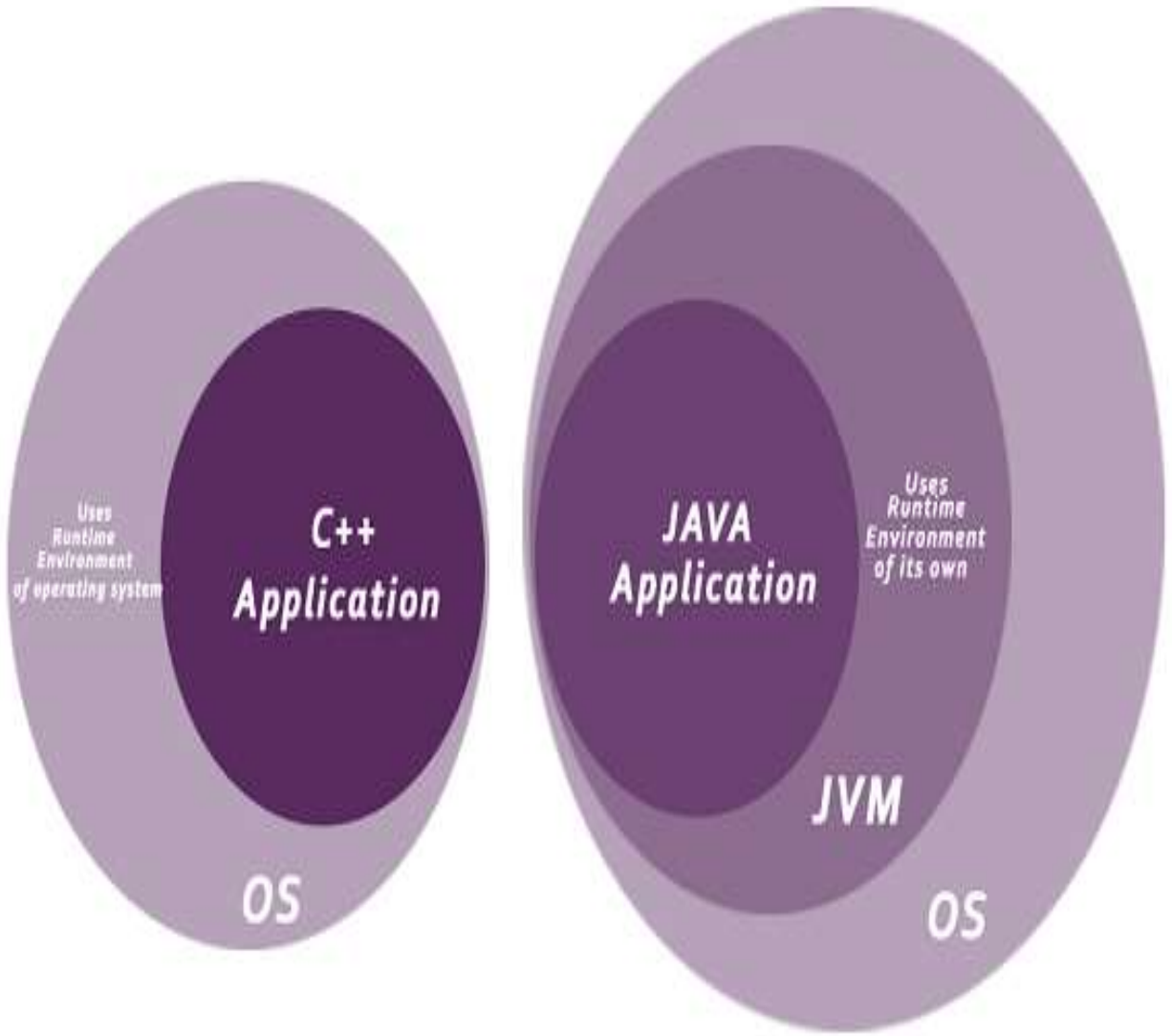
Platform Independent

- Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.
- Java code can be executed on multiple platforms
- for example, Windows, Linux, Sun Solaris, Mac/OS, etc.
- Java code is compiled by the compiler and converted into bytecode.
- This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).



Secured

- Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**



Robust

- Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

Architecture-neutral

- Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

High Performance

- With the use of Just-In-Time compilers, Java enables high performance.

Multithreaded

- With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

Dynamic

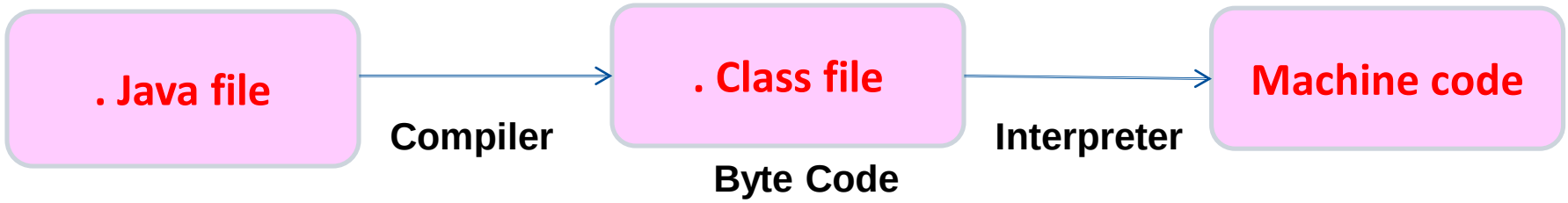
- Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand.

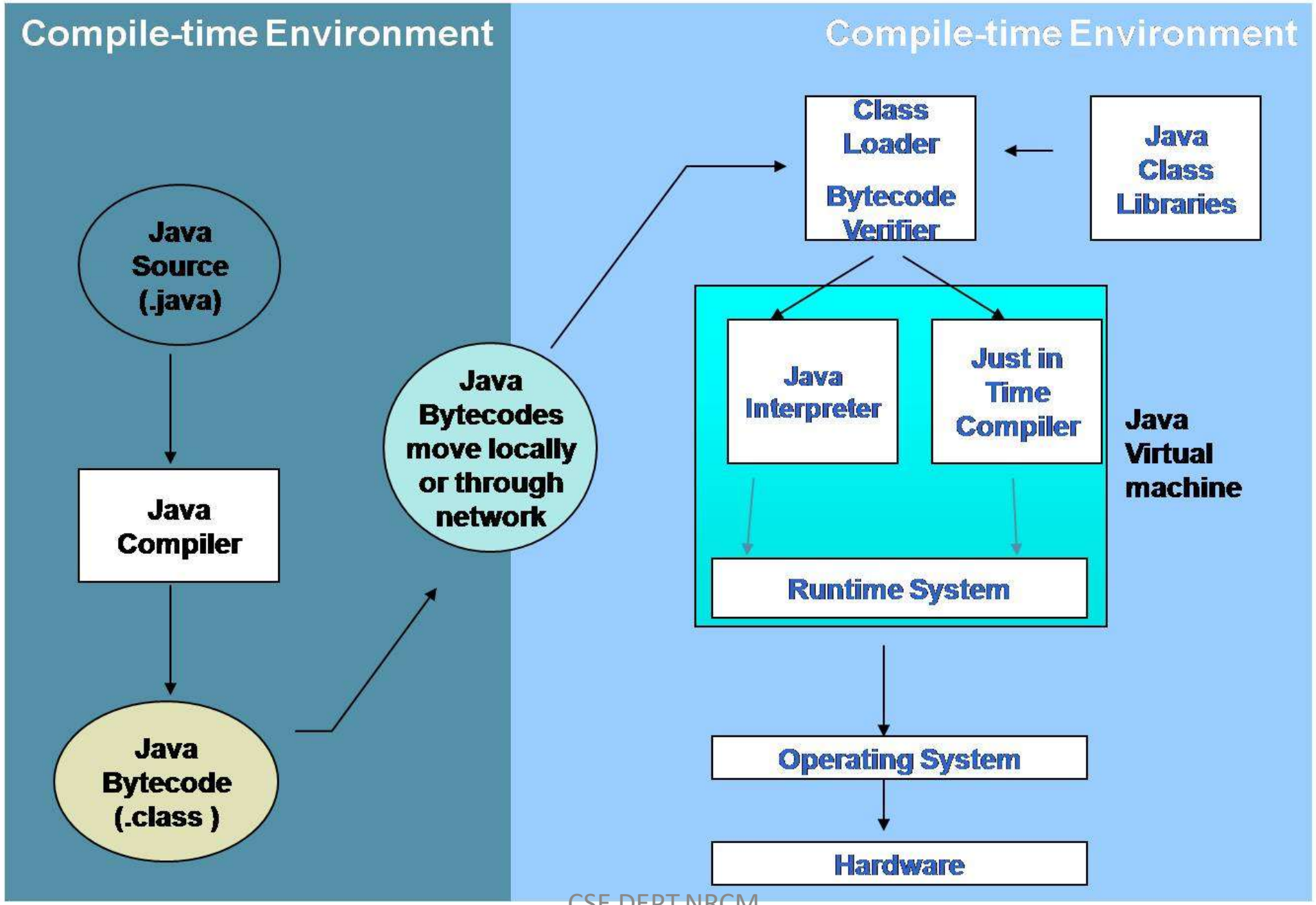
Distributed

- Java makes us able to access files by calling the methods from any machine on the internet.

Java Compiled & Interpreted

- Usually a computer language is either uses compiler or interpreter.
- But Java Combines both these approaches.
- First Java Compiler translates source code into byte code instructions. Byte code is not machine instructions.
- Second Java Interpreter directly executes the byte code.




Compile-time Environment
Compile-time Environment
Class Loader
Bytecode Verifier
Java Class Libraries
Java Source (.java)
Java Compiler
Java Bytecode (.class)
Java Bytecodes move locally or through network
Java Interpreter
Just in Time Compiler
Java Virtual machine
Runtime System
Operating System
Hardware

A simple Java application

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

- name of class is same as name of file (which has extension)
- body of class surrounded by { }
- this class has one method called main
  - all Java applications must have a main method in one of the classes
  - execution starts here
  - body of method within { }
- all other statements end with semicolon

- **Compile Welcome.java**

```
javac Welcome.java
```

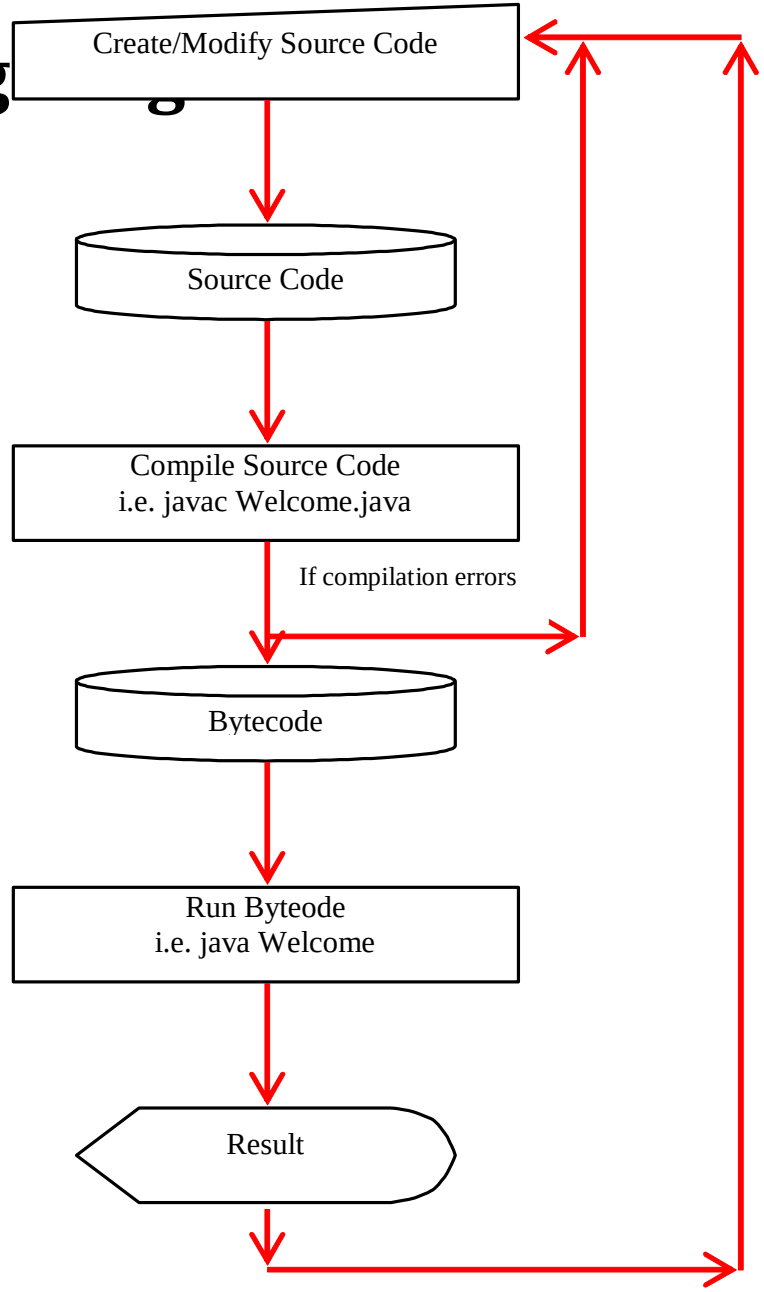
Output: Welcome.class

- **Run**

```
java Welcome
```

Output: Welcome to Java

Creating and Compiling



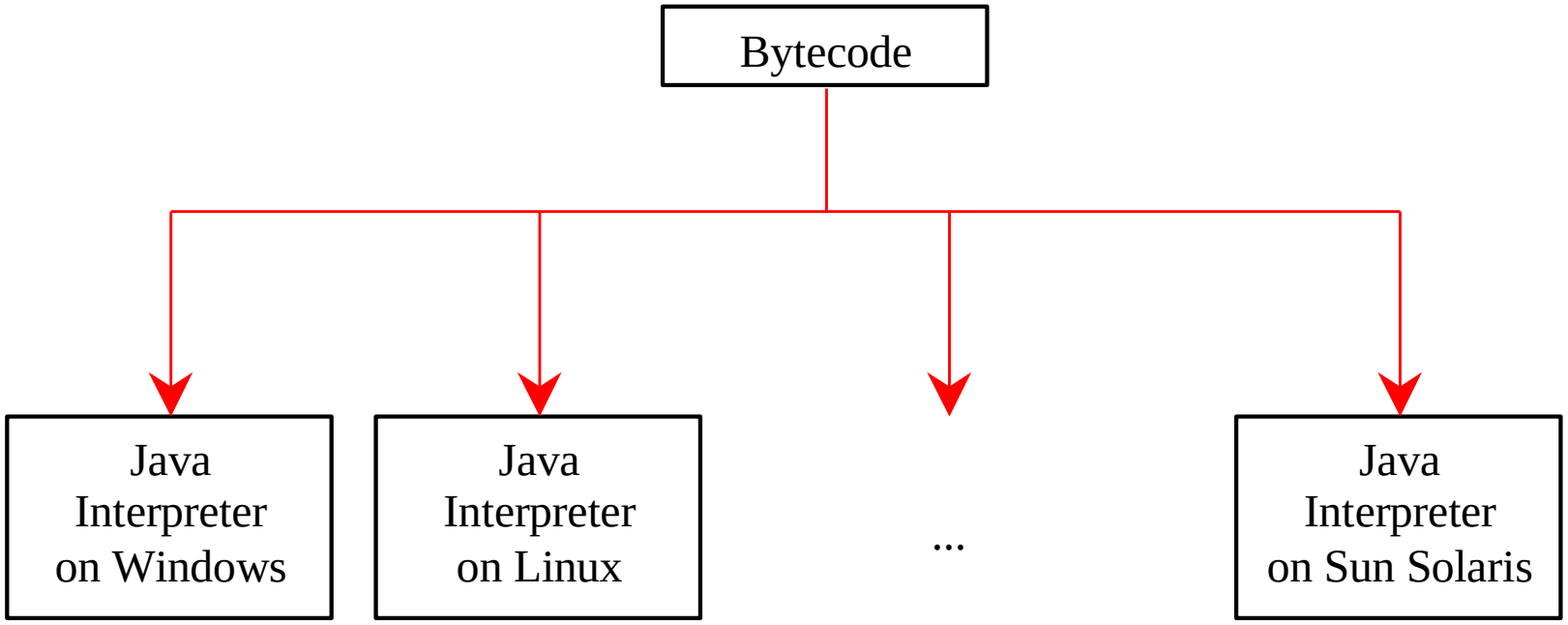
On command line

```
javac file.java
```

Executing Application

On command line

```
java classname
```



Variables

- Variable is a name of memory location, that can hold values of a certain *data type*
- Each variable must be *declared* before it is used
- The declaration allocates a location in memory to hold values of this type

Variables

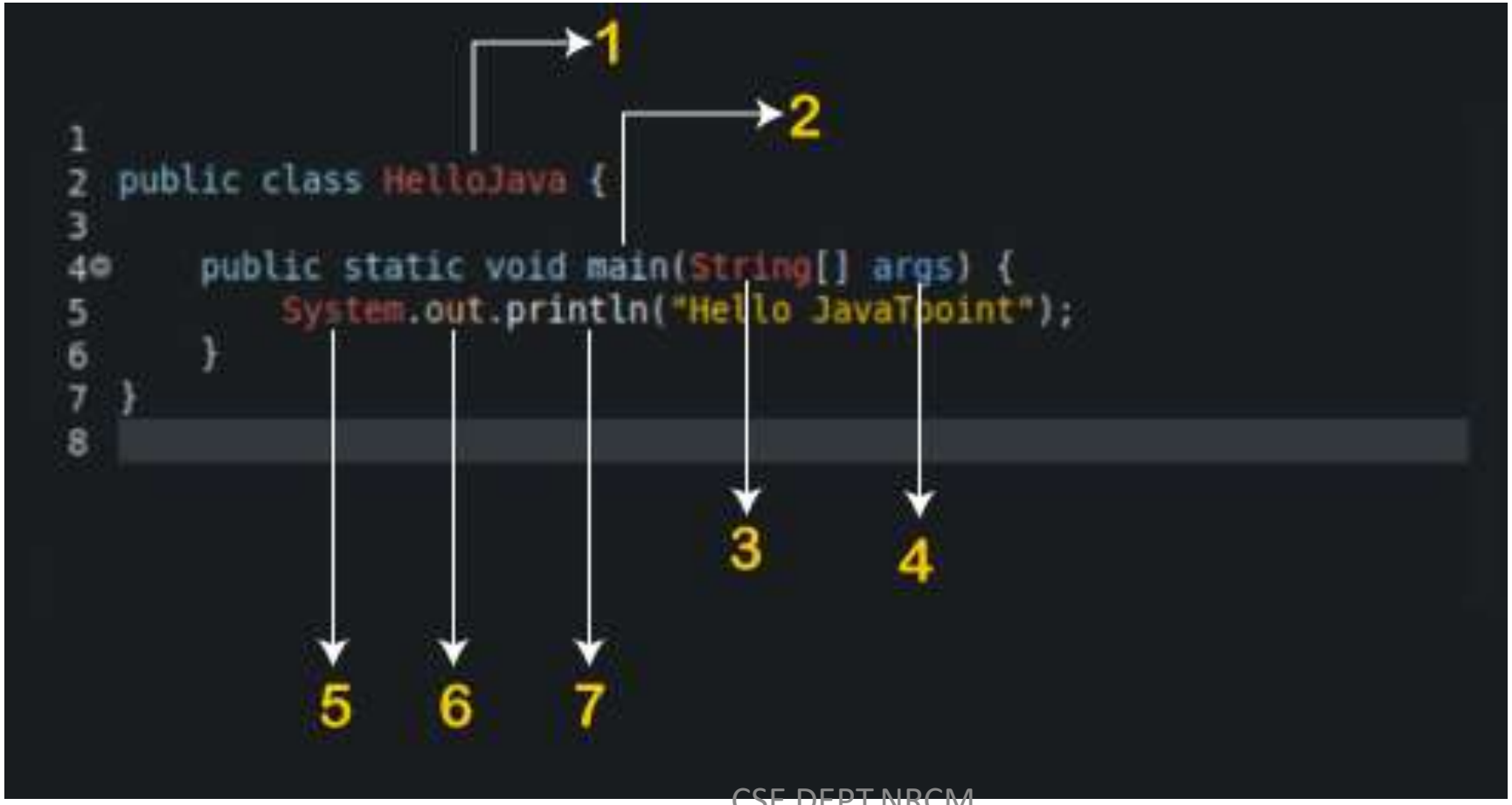
- Unlike C, variables can be declared anywhere within block
- Use meaningful names
- Start with lower case, capitalize first letter of subsequent words

VariableExample Program

```
public class VariableExample {
  public static void main(String[] args){
    int x;
    x = 3;
    System.out.println(x);
    x = 4;
    System.out.println(x);
  }
}
```

Identifiers

- In programming languages, identifiers are used for identification purposes. In Java, an identifier can be a class name, method name, variable name, or label.



Rules for Identifiers in Java

- **Identifier syntactic rules:**
  - A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(_) or a dollar sign ($). Cannot begin with a digit
  - There should not be any space in an identifier.
  - There is no limit on its length. But, it is good to follow the standard conventions.
  - We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.

- Java is *case sensitive*
  - `User` and `user` are completely different identifiers

- In Java, There are some reserved words that can not be used as an identifier.

Example:

Valid Identifiers

- TestVariable
- testvariable
- a
- i
- Test_Variable

Invalid Identifiers

- Test Variable (We can not include a space in an identifier)
- 123javatpoint (The identifier should not begin with numbers)
- java+tpoint (The plus (+) symbol cannot be used)
- a-javatpoint (Hyphen symbol is not allowed)
- java_&_Tpoint (ampersand symbol is not allowed)
- Java'tpoint (we can not use an apostrophe symbol in an identifier)

Naming Conventions for Identifiers

- Classes
  - Names given in lowercase except for first letter of each word in the name
- Variables
  - first letter is lowercase.
- Constants
  - All caps with _ between words
- Methods
  - Starts with lowercase letters and all subsequent words will start with capital case

Constants

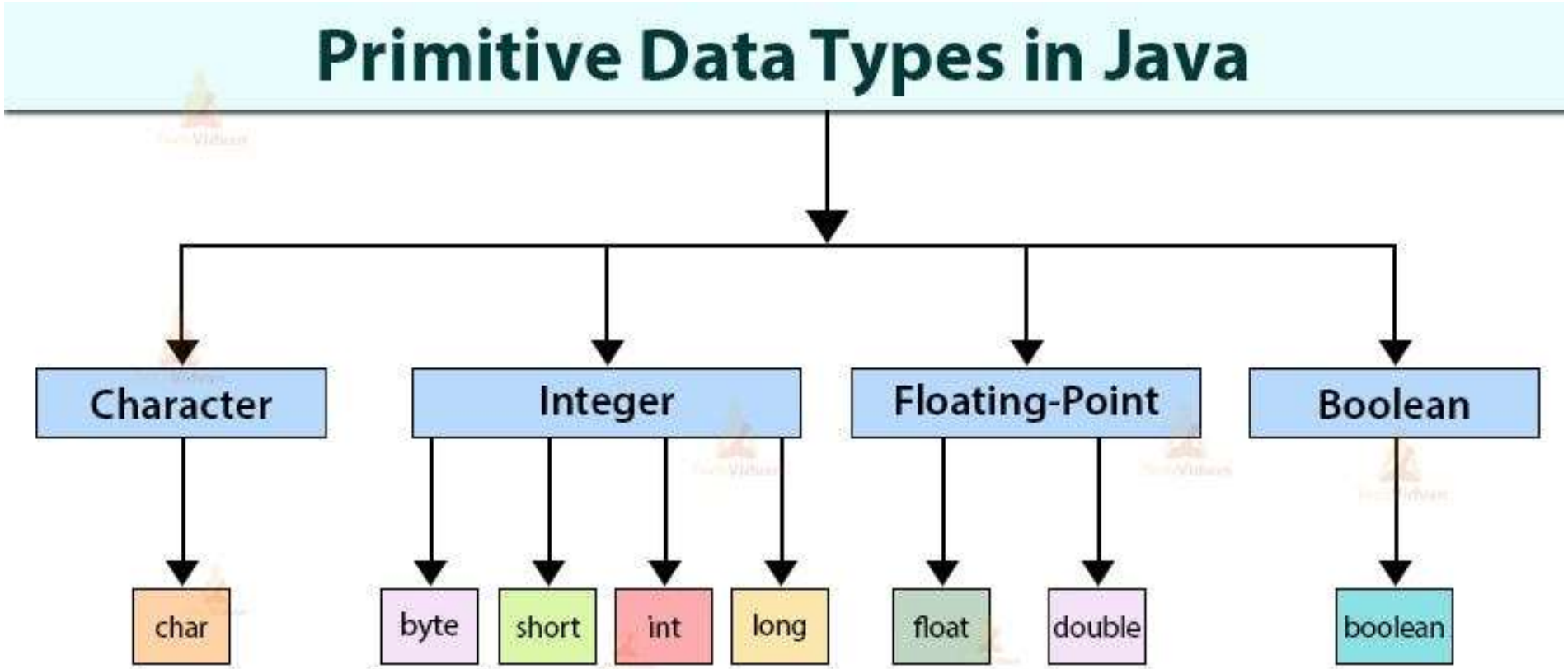
- **Constant** is a value that cannot be changed after assigning it.

```
final double PI = 3.14159;
final int CHINA_OLYMPICS_YEAR = 2008;
```

Data Types in Java

- Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

- **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.


Primitive Data Types in Java
Character
Integer
Floating-Point
Boolean
char
byte
short
int
long
float
double
boolean

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Primitive Data types

- **int** 4 byte integer (whole number)
  - range -2147483648 to +2147483648
- **float** 4 byte floating point number
  - decimal points, numbers outside range of **int**
- **double** 8 byte floating point number
  - 15 decimal digits (float has 7) so bigger precision and range
- **char** 2 byte letter
- **String** string of letters
- **boolean** true or false (not 1 or 0)

Type	*Domain*
`byte`	8-bit integers in the range –128 to 127
`short`	16-bit integers in the range –32768 to 32767
`int`	32-bit integers in the range –2146483648 to 2146483647
`long`	64-bit integers in the range –9223372036754775808 to 9223372036754775807
`float`	32-bit floating-point numbers in the range $\pm 1.4 \times 10^{-45}$ to $\pm 3.4028235 \times 10^{-38}$
`double`	64-bit floating-point numbers in the range $\pm 4.39 \times 10^{-322}$ to $\pm 1.7976931348623157 \times 10^{308}$
`char`	16-bit characters encoded using Unicode
`boolean`	the values `true` and `false`

Characters

- A `char` value stores a single character from the *Unicode character set*
- A *character set* is an ordered list of characters
  - ‘A’, ‘B’, ‘C’, … , ‘a’, ‘b’, … ,‘0’, ‘1’, … , ‘$’, …

Boolean

- A `boolean` value represents a true/false condition.
- It can also be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a boolean type

- Comments are ignored and are treated as white space
- They should be written to enhance readability
  - Explain what a piece of code does (its *interface*)
  - Explain any special tricks, limitations, …
- Java has three comment formats:
  - `// comment to end of line`
  - `/* comment until closing */`
  - `/** API specification comment */`

Operators

- Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

Operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Relational Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Unary Operator

- The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:
- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Example:

```
public class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
int a=10,b=-10;
boolean c=true,d=false;
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
}}
```

Arithmetic Operators

- Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.
- **Binary Operators:**

```
a + b       adds a and b
a - b       subtracts b from a
a * b       multiplies a and b
a / b       divides a by b
a % b       the reminder of divining a by b
```

Java Arithmetic Operator Example

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Operator Precedence

- Multiplication, division, and remainder (%) have a higher *precedence* than addition and subtraction.
- Operators with same precedence evaluate from left to right.
- Parenthesis can be used to force order of evaluation.

Operator Precedence Examples

Expression	Result
`10 - 7 - 1`	2
`10 - (7 - 1)`	4
`1 + 2 * 3`	7
`(1 + 2) * 3`	9
`1 - 2 * 3 + 4 * 5`	15

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
//21
```

Relational Operators

Relational operators are used to compare two values.
it helps us to find answers and make decisions.
The return value of a comparison is either true or false.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

```
  int y = 3;
System.out.println(x > y); // returns true,
because 5 is higher than 3
```

Java Logical Operators

Logical operators are used to determine the logic between variables or values
You can also test for `true` or `false` values with logical operators.

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Assignment Statements

- An assignment statement takes the following form

  variable-name = expression;

- The expression is first *evaluated*
- Then, the result is stored in the variable, overwriting the value currently stored in the variable

Java Ternary Operator

Java ternary operator is the only conditional operator that takes three operands. It's a replacement for the if-then-else statement and is used a lot in Java programming.

```
public class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);  //2
}}
```

Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java Control Statements

- Java provides three types of control flow statements.
- Decision Making statements
  - if statements
  - switch statement
- Loop statements
  - do while loop
  - while loop
  - for loop
- Jump statements
  - break statement
  - continue statement

Decision-Making statements:

Decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e.,

If statement

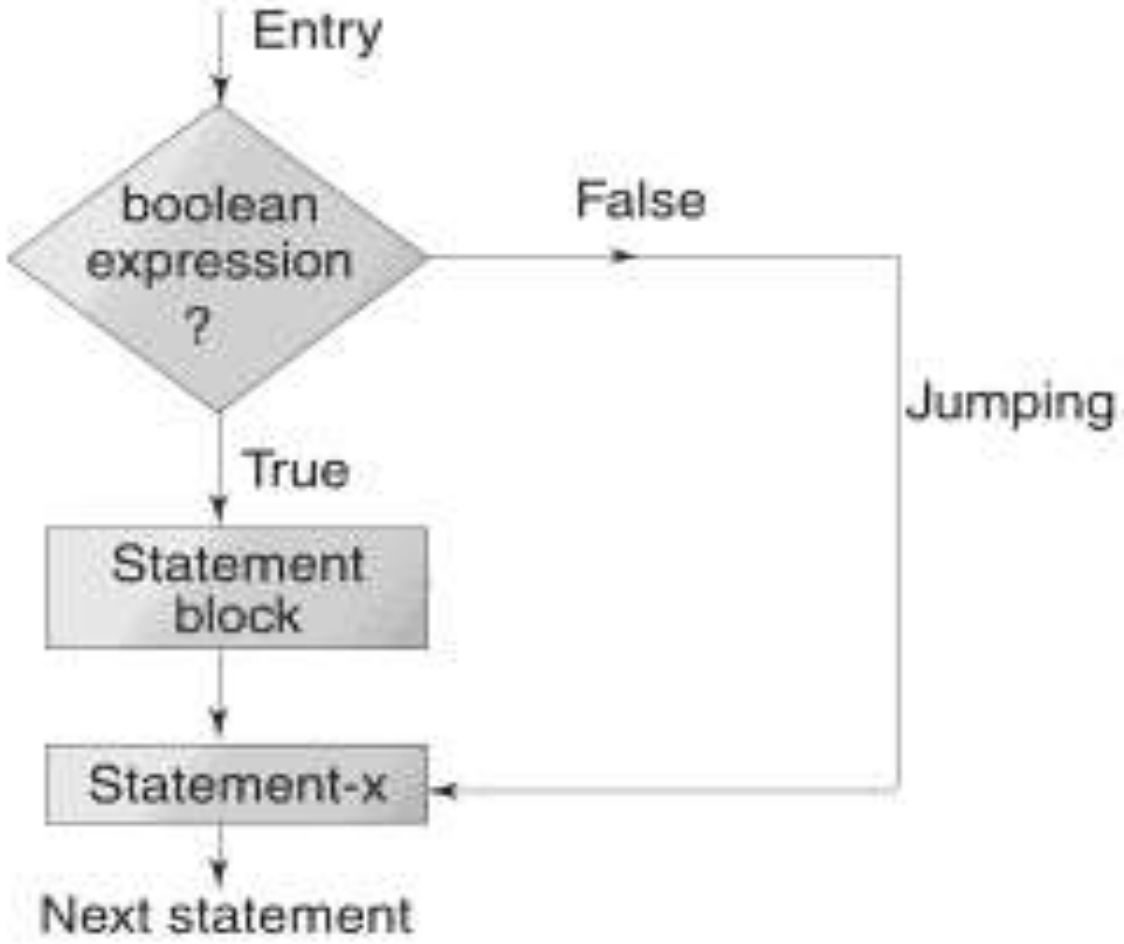
and

switch statement.

If Statement:

- In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false.
- In Java, there are four types of if-statements given below.
- Simple if statement
- if-else statement
- if-else-if ladder
- Nested if-statement
- Let's understand the if-statements one by one.

Simple If


Entry
boolean expression ?
False
Jumping
True
Statement block
Statement-x
Next statement

If – The Conditional Statement

- The if statement evaluates an expression and if that evaluation is true then the specified action is taken

```
if ( x < 10 ) x = 10;
```

- If the value of x is less than 10, make x equal to 10
- It could have been written:

```
if ( x < 10 )
x = 10;
```

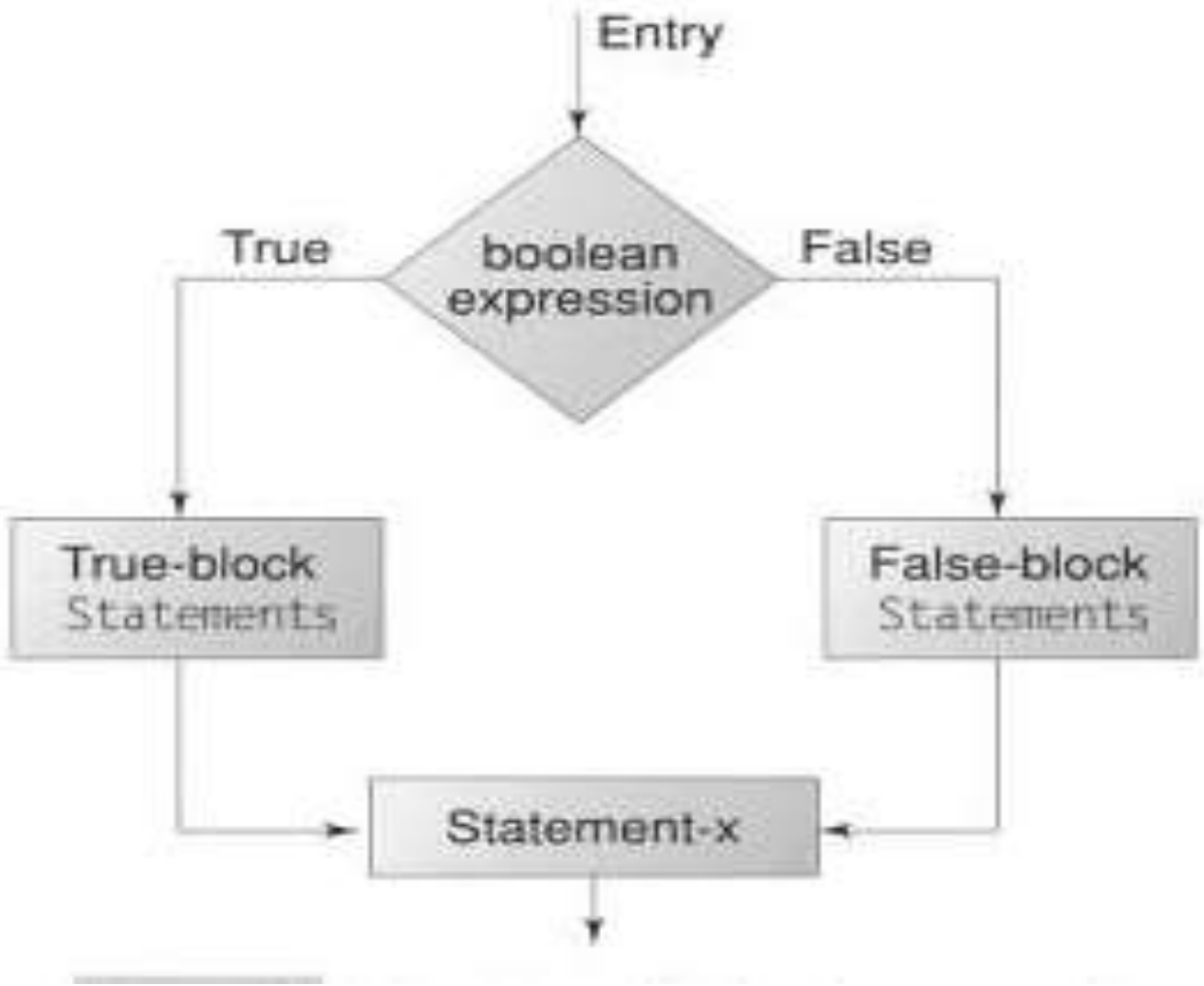
- Or, alternatively:

```
if ( x < 10 ) { x = 10; }
```

If... else

The if … else statement evaluates an expression and performs one action if that evaluation is true or a different action if it is false.

If.. else

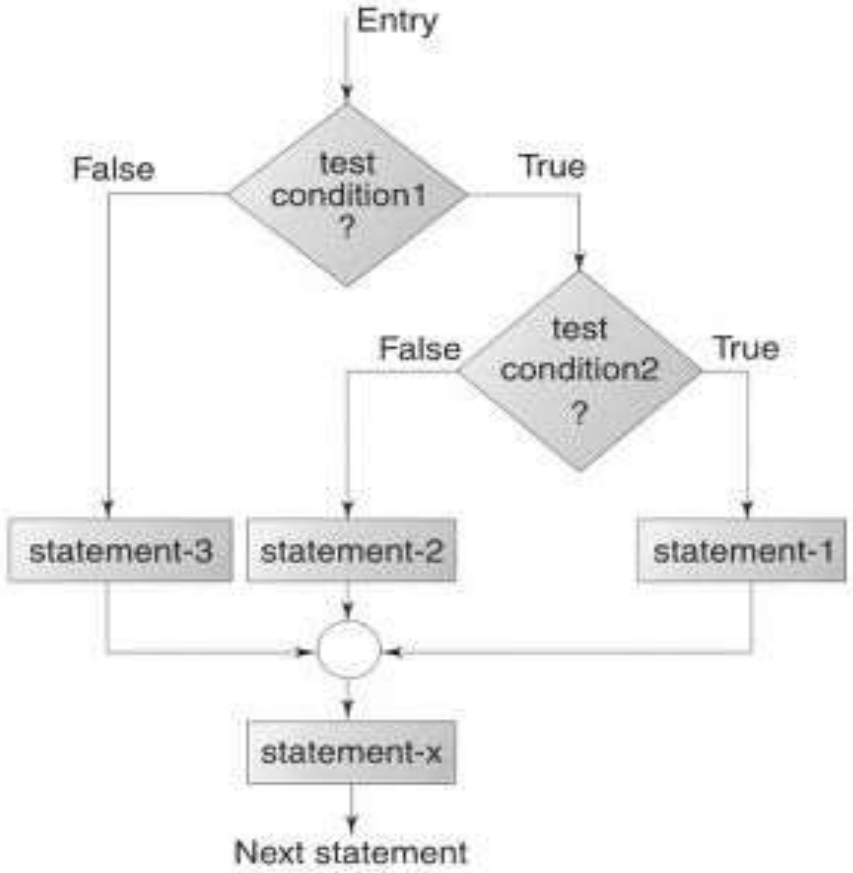



```
if (x != oldx) {
      System.out.print("x was
   changed");
   }
   else {
      System.out.print("x is
   unchanged");
   }
```

Nested if ... else

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

```
if (test condition1)
{
    if (test condition2)
    {
        statement-1;
    }
    else
    {
        statement-2;
    }
}
else
{
        statement-3;
}
statement-x;
```



Nested if ... else

```
class GFG
{
    public static void main(String args[])
    {
        int a=10;
          int b=20;

        if(a==10){
            if(b==20){
              System.out.println("Welcome");
            }
        }
    }
}
```

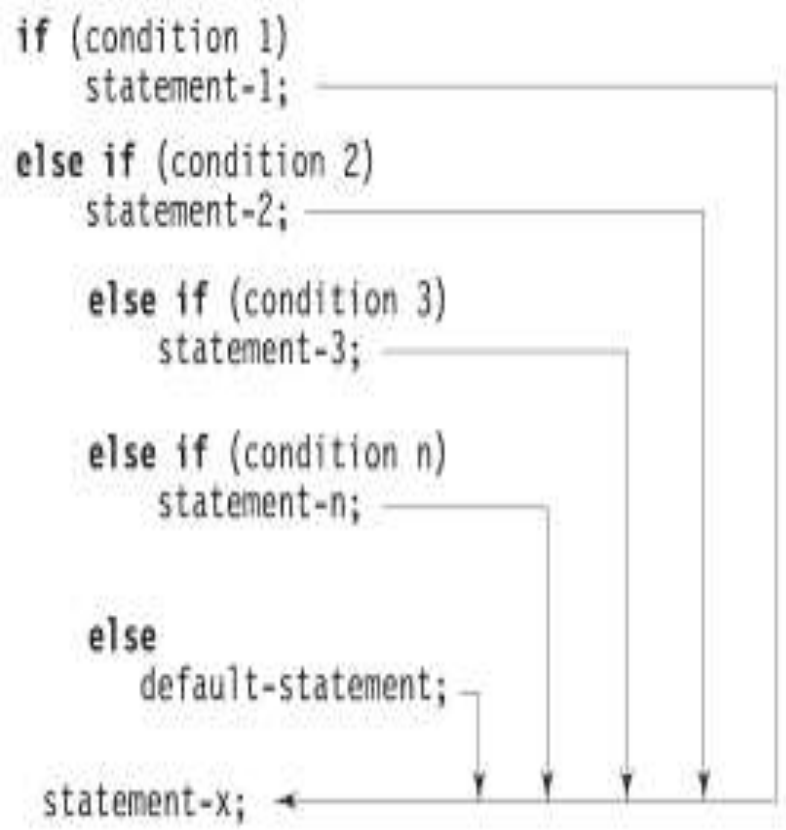
```
class GFG
{
   public static void main(String args[])
   {
      int a=10;
       int b=20;

      if(a==10){

         if(b!=20){
            System.out.println(“b is not 20");
                 }
          else{
            System.out.println(“b is 20");
                 }
      }
   }
}
```

else if

- Useful for choosing between alternatives:

```
if ( n == 1 )
{
   // execute code block #1
}
else if ( j == 2 ) {
   // execute code block #2
}
else {
   // if all previous tests have failed,
   execute code block #3
}
```

else if Ladder Java if-else-if ladder is used to decide among multiple options.



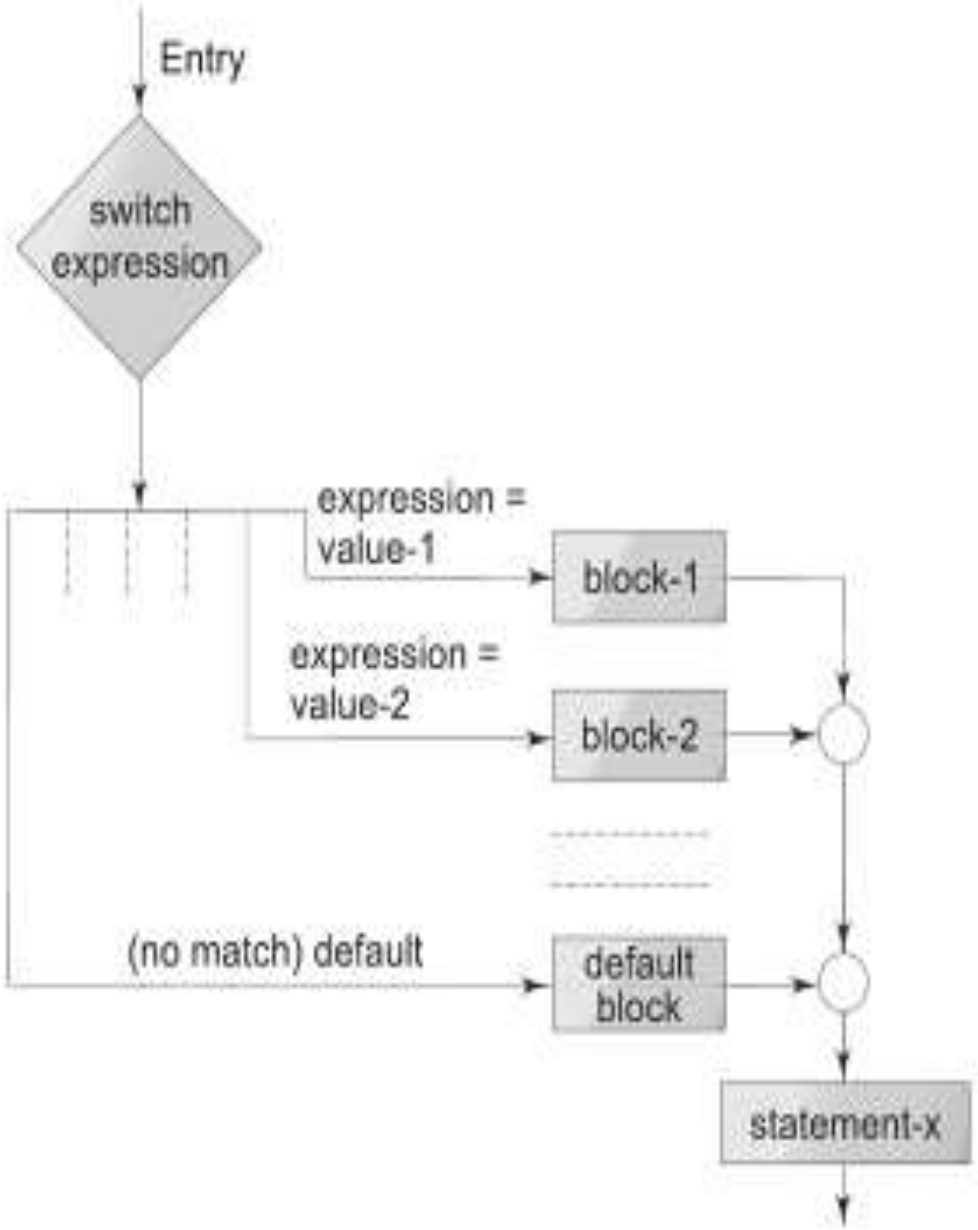

```
class IfElse
{
public static void main(String args[])
{
int month = 12;
String season;
if(month == 12 || month == 1 || month == 2)
season = "Winter";
else if(month == 3 || month == 4 || month == 5)
season = "Spring";
else if(month == 6 || month == 7 || month == 8)
season = "Summer";
else if(month == 9 || month == 10 || month == 11)
season = "Autumn";
else
season = "Dummy Month";
System.out.println("December is in the " + season + ".");
}
}
```

Switch Statement:

The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

The switch Statement

```
switch ( n ) {
  case 1:
          // execute code block #1
          break;
  case 2:
          // execute code block #2
          break;
  default:
   // if all previous tests fail then
   //execute code block #4
   break;
}
```


Entry
switch expression
expression = value-1
block-1
expression = value-2
block-2
(no match) default
default block
statement-x

the value of expression. It is optional.

Break statement terminates the switch block when the condition is satisfied.

It is optional, if not used, next case is executed.

While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

```
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}
}
}
```

Loop Statements

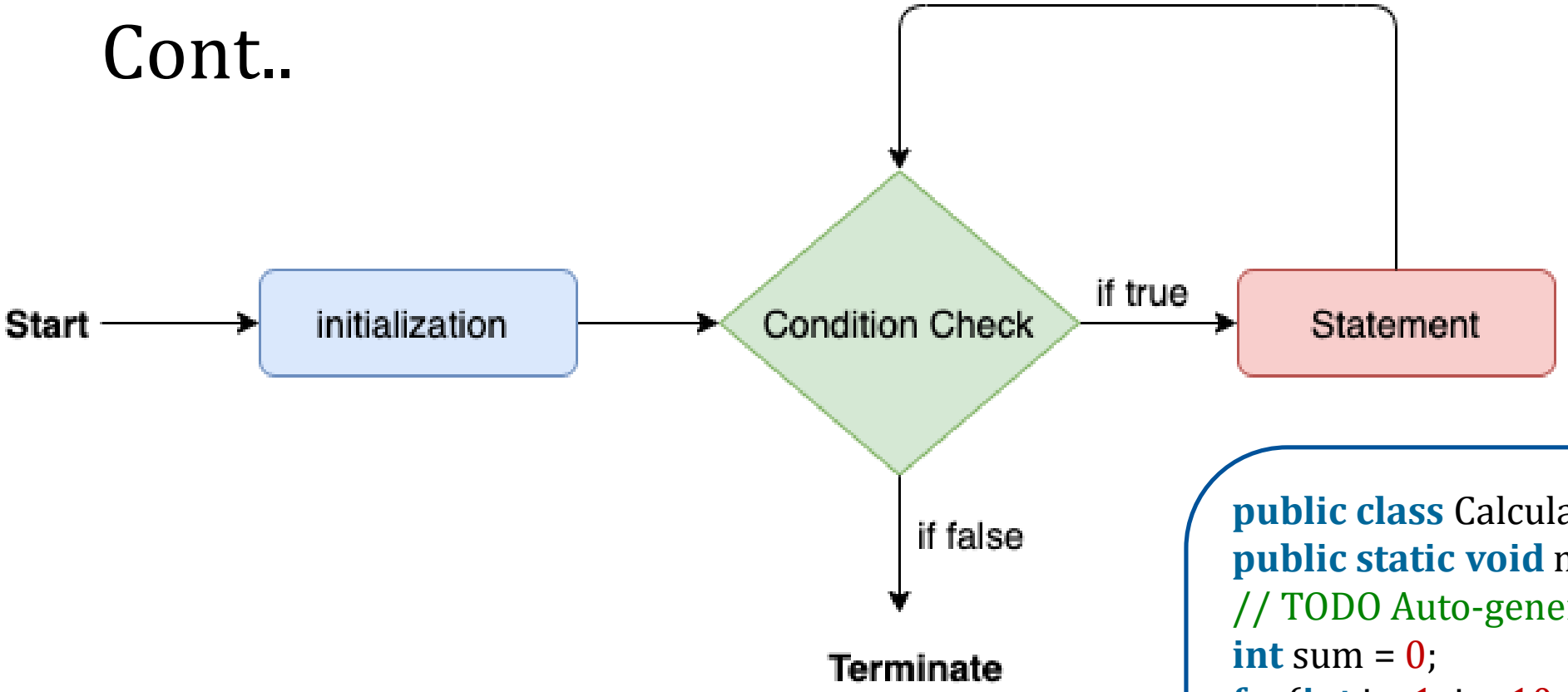
- In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.
- In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.
- for loop
- while loop
- do-while loop

Java for loop

- It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

- **for**(initialization, condition, increment/decrement) {
- //block of statements
- }

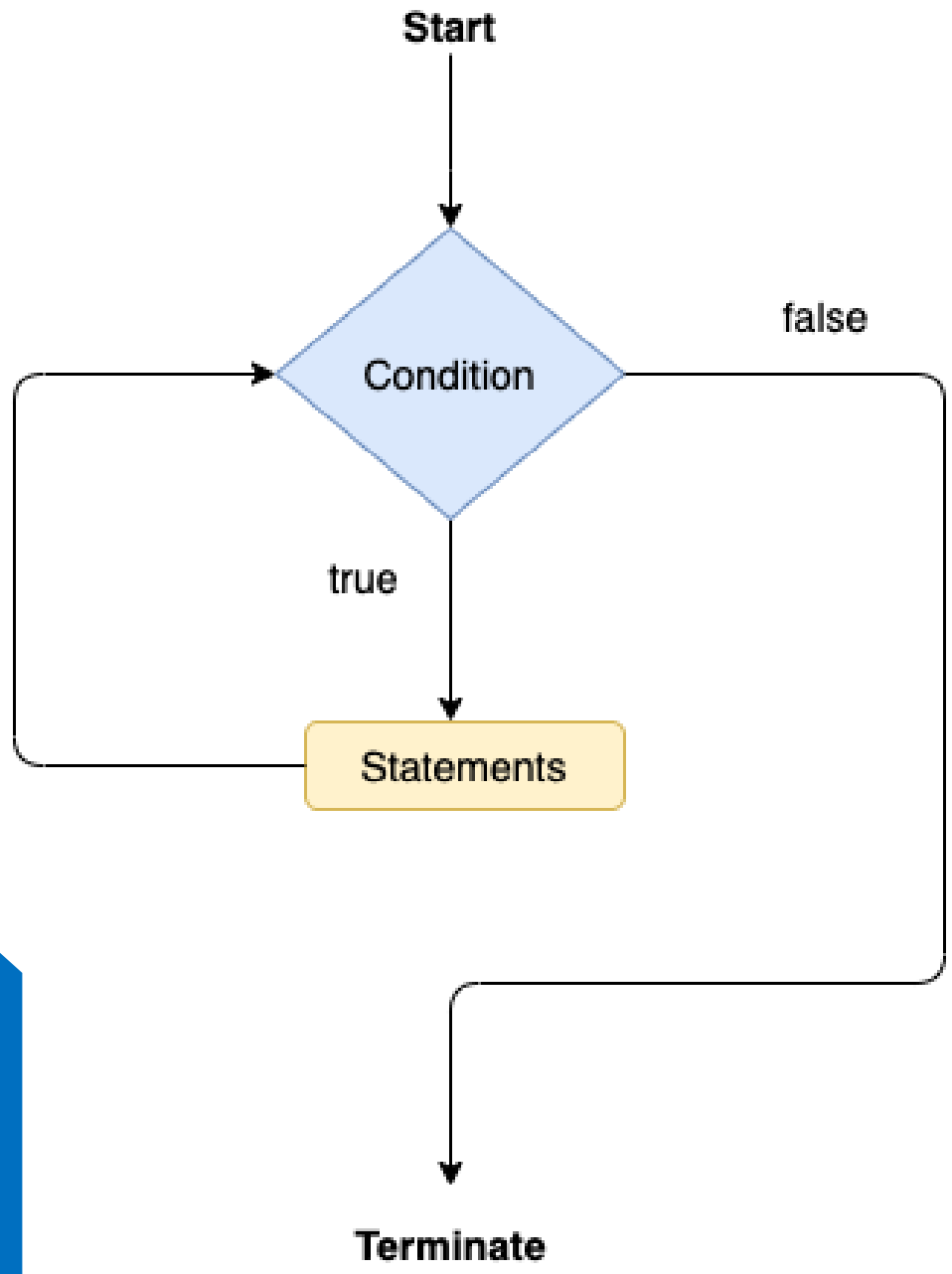
Cont..




```
public class Calculattion {
public static void main(String[] args) {
// TODO Auto-generated method stub
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println("The sum of first 10
 natural numbers is " + sum);
}
}
```

Java while loop

- The while loop is also used to iterate over the number of statements multiple times.
- Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.



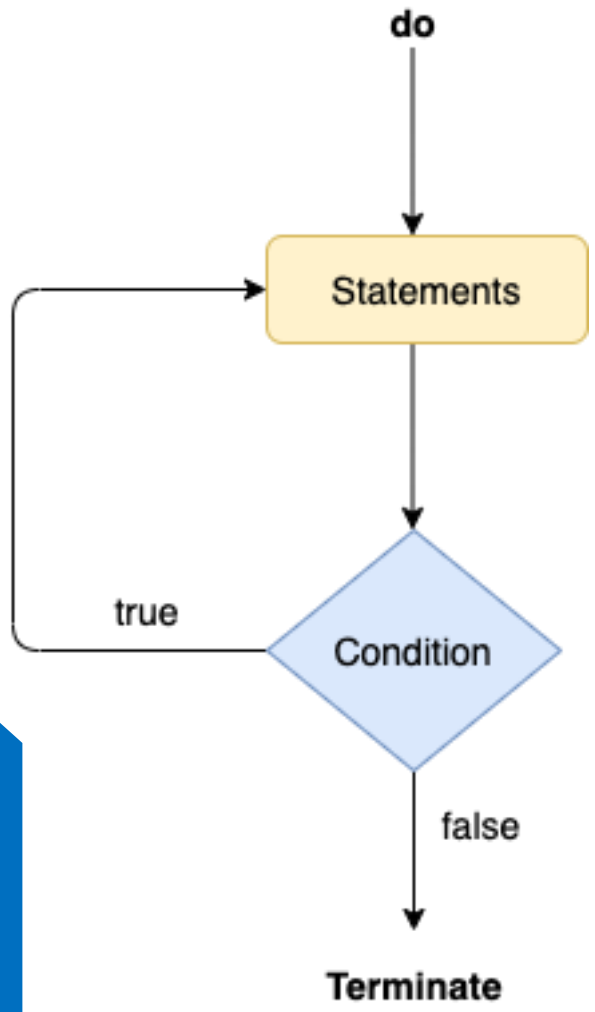

```
public class Calculation {
public static void main(String[] ar
gs) {

int i = 0;
System.out.println("Printing the list
 of first 10 even numbers \n");
while(i<=10) {
System.out.println(i);
i = i + 2;
}
}
}
```

Java do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
do
{
//statements
} while (condition);
```




```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even numb
ers \n");
do {
System.out.println(i);
i = i + 2;
}while(i<=10);
}
}
```

Jump Statements

- Jump statements are used to transfer the control of the program to the specific statements.
- In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e.,
- break
- continue.

Java break statement

- As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

- The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

Example

```
// Java program to illustrate using
// break to exit a loop
class BreakLoopDemo {
        public static void main(String args[])
        {
                // Initially loop is set to run from 0-9
                for (int i = 0; i < 10; i++) {
                        // terminate loop when i is 5.
                        if (i == 5)
                                break;

                        System.out.println("i: " + i);
                }
                System.out.println("Loop complete.");
        }
}
```

```
Output:
i: 0
i: 1
i: 2
i: 3
i: 4
Loop complete.
```

Java continue statement

- Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

- Consider the following example to understand the functioning of the continue statement in Java.

Example

```
public class ContinueExample {

public static void main(String[] args) {
// TODO Auto-generated method stub

for(int i = 0; i<= 2; i++) {

for (int j = i; j<=5; j++) {

if(j == 4) {
continue;
}
System.out.println(j);
}
}
}

}
```

```
0
1
2
3
5
1
2
3
5
2
3
5
```

Type Conversion

- Widening numeric conversions
  - `int` to `long`, `float`, or `double`
  - `long` to `double`
  - `float` to `double`
- Narrowing numeric conversions
  - `int` to `byte`, `short`, or `char`
  - `long` to `int`
  - `float` to `int`
  - `double` to `int`, `long`, or `float`

Type Casting

Implicit casting

double d = 3; (type widening)

Explicit casting

int i = (int)3.0; (type narrowing)

What is wrong? int x = 5/2.0;