Object Oriented Programming (Java Programming Language)

[James Gosling](#)

Jun 1991
Gosling, Sheridan and Naughton started development of new programming language with the name 'Oak'

Jul 1991
Later it was named as **Green** and then as **Java** to remind their favourite Java Coffee from Indonesia.

Jun 1991
Initially named as **Oak** when they discussed under Oak tree near Gosling's office

1995
JavaSoft was taken over by Sun Microsystems and announced the release of Java soon.

Jan 23, 1996
Sun Microsystems released first official Java 1.0

Feb, 1997
First stable release of Java as Dev Kit (1.1) released. Arthur van Hoff rewritten Java 1.0 compiler using Java itself.

1998
Java 1.2 released as J2SE along with Swing graphical API library with codename **PlayArea**

2000
Java 1.3 released with JVM, JNDI with code name KESTRAL

2002
Java 1.4 with Regexp release with codename **Merlin**

2004
Java 1.5 (J2SE 1.5) released with most of standard features in core libraries (lang, util) like Generics

2006
Multiple variants of Java released as J2SE, J2ME, J2EE with codename **Mustang**

Nov, 2006
JVM officially released as free/OpenSource under GNU GPL License.

Java Sun Microsystems

Java became multi-faceted releases with from Jrocket, IBM, OpenJDK

Java ORACLE

2011
J2SE 7 released with codename **Dolphin** with dynamic language, string optimization and GC optimization

2014
J2SE 8 released with Major changes like Lambda and datetime API

March, 2018
J2SE 10 released with memory features like Heap allocation and Thread local

March, 2019
J2SE 12 released with Shenandoah GC, Switch Expressions, JVM Constants API

March 2020
J2SE 14 released with Incubator, JFR Event Streaming, Java Records

2017
J2SE 9 released with codename **Jigsaw** with features like Streaming API, JShell, improved features in Collections API

Sept, 2018
J2SE 11 released with Epsilon GC, Heap profiling and improved features of Lambda.

Sept, 2019
J2SE 13 released with Text Blocks, Improved File IO API, Socket API

Sept 2020
J2SE 15 released with Z GC, preview of sealed class, hidden classes.

What you will be able to do after Learning Java Programming??

- Develop programs by applying Object-Oriented Programming (OOP) concepts of JAVA to solve real-world problems.
- Achieve Robustness and Concurrency while developing programs.
- Design Graphical User Interface(GUI) using swing.

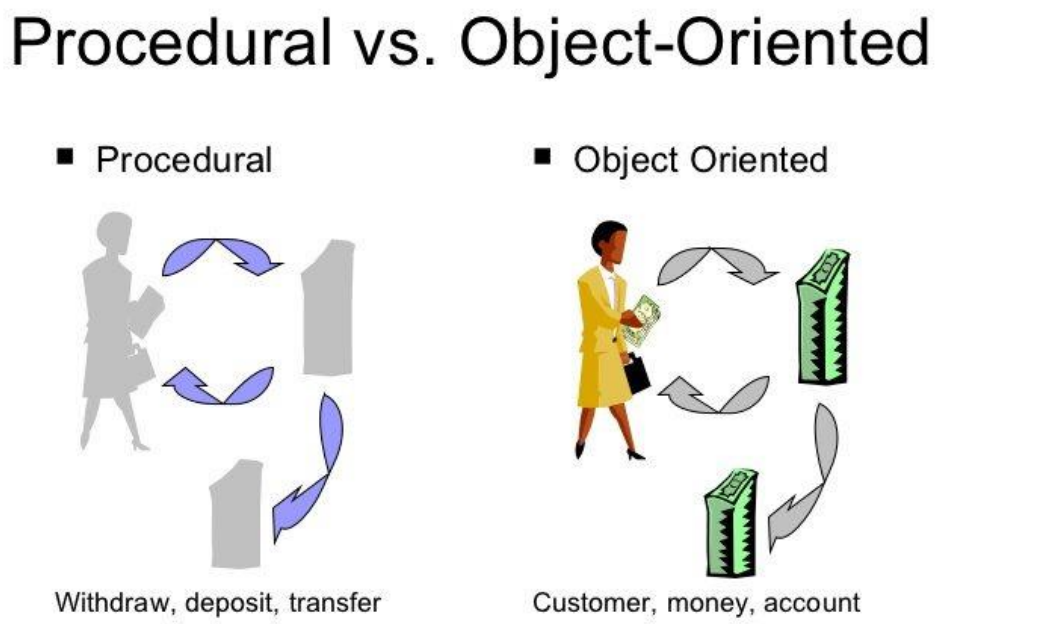
Module: 1

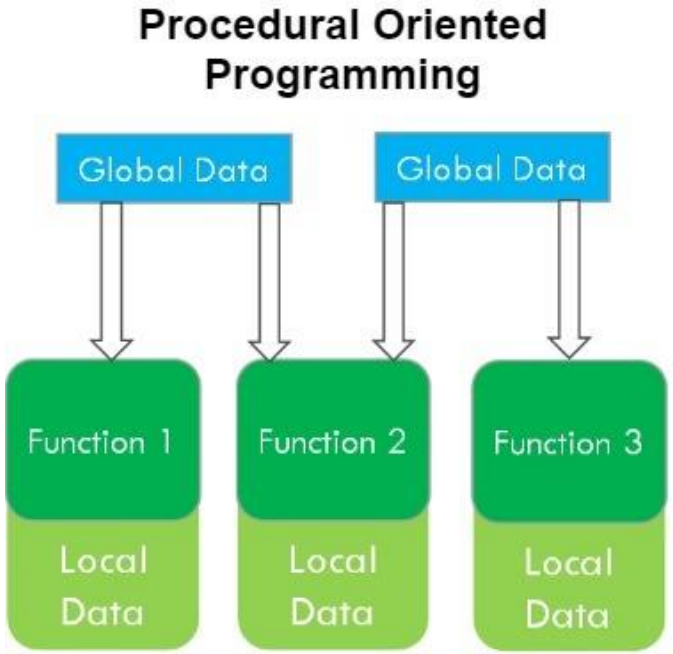
Introduction to Java as Object Oriented Programming Language

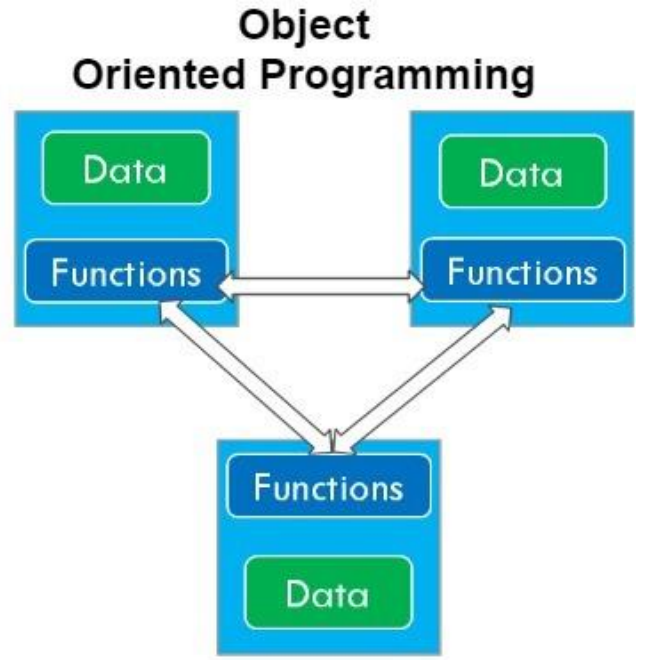
- Fundamentals of Java Programming:
- Overview of procedure and object-oriented programming,
- Features of Java,
- Java Virtual Machine
- Principles of OOP: Object, Class, Encapsulation, Abstraction, Inheritance, Polymorphism
- Basic Constructs: Constants, variables and data types, Wrapper classes, Operators and Expressions
- Input & Output in Java: command line arguments, BufferedReader class and Scanner class

Fundamentals of Java Programming:

- **Overview of procedure and object-oriented programming:**

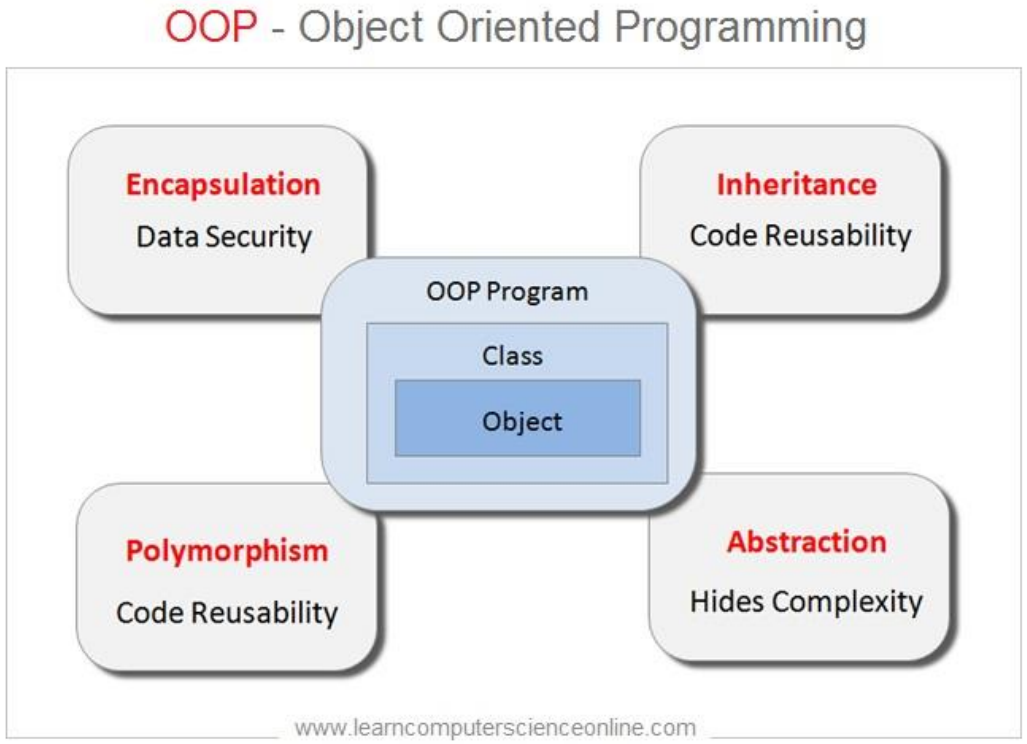






Fundamentals of Java Programming:

- **Overview of procedure and object-oriented programming:**



Structured/Procedural Programming	Object Oriented Programming
Code is divided into modules or functions	Code is made up of classes and objects
Town-down approach	Bottom-up approach
Difficult to modify / manage	Easy to modify / manage
Main function calls other functions	Objects communicate by passing messages
Data is not secured	Data is secure
Less reusability of code	More reusability of code
Less flexibility and abstraction	More flexibility and abstraction

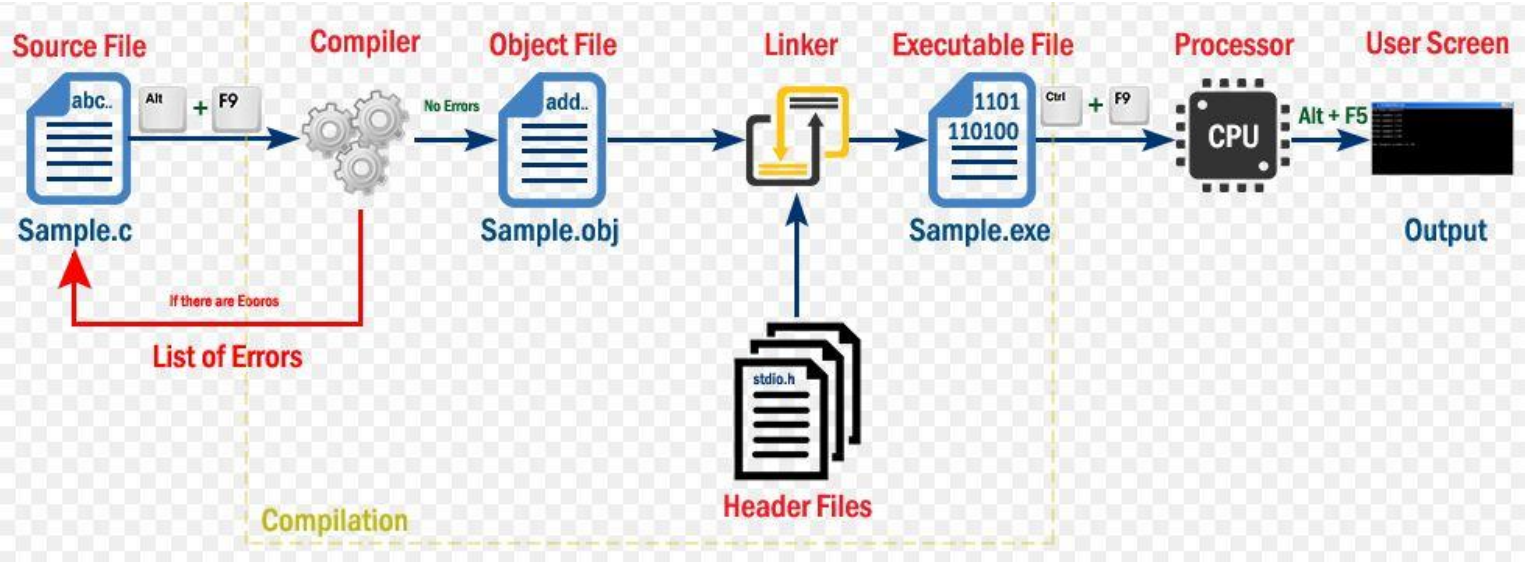
Fundamentals of Java Programming:

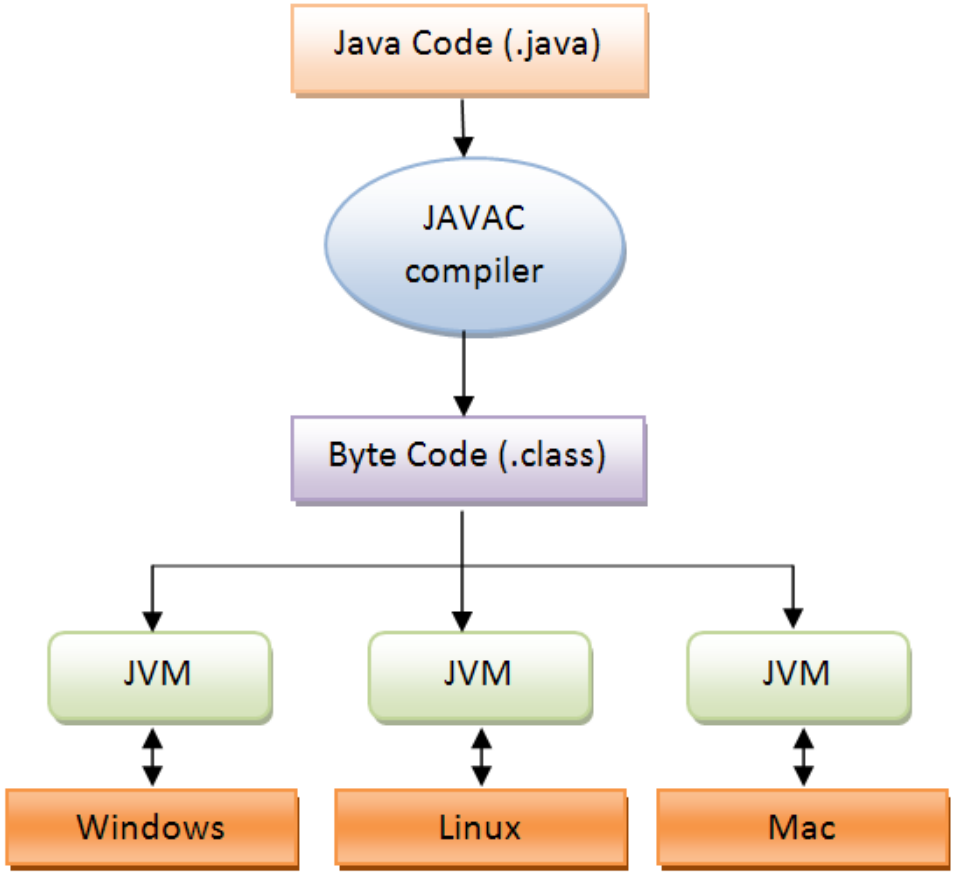
- **C Programming Vs Java Programming**

Sno.	C	Java
1.	C is platform-dependent.	Java is platform-independent.
2.	C supports goto statement.	Java doesn't support goto statement.
3.	C supports pointer.	Java doesn't support Pointer.(implicit pointer support)
4.	C supports structures and unions.	Java doesn't support structures and unions.
5.	C uses compiler only.	Java uses compiler and interpreter both.
6.	C is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
7.	C supports sizeof() operator.	Java doesn't support sizeof() operator.
8.	C supports unsigned datatype.	Java doesn't support unsigned datatype.
9.	C supports auto , extern, register variable.	Java doesn't support auto , extern, register variable.
10.	C is procedure oriented Programming.	Java is object oriented programming.
11.	C supports Preprocessors,Headerfiles.(#define,#include).	JAVA does not support Preprocessors,Headerfiles.(#define,#include).

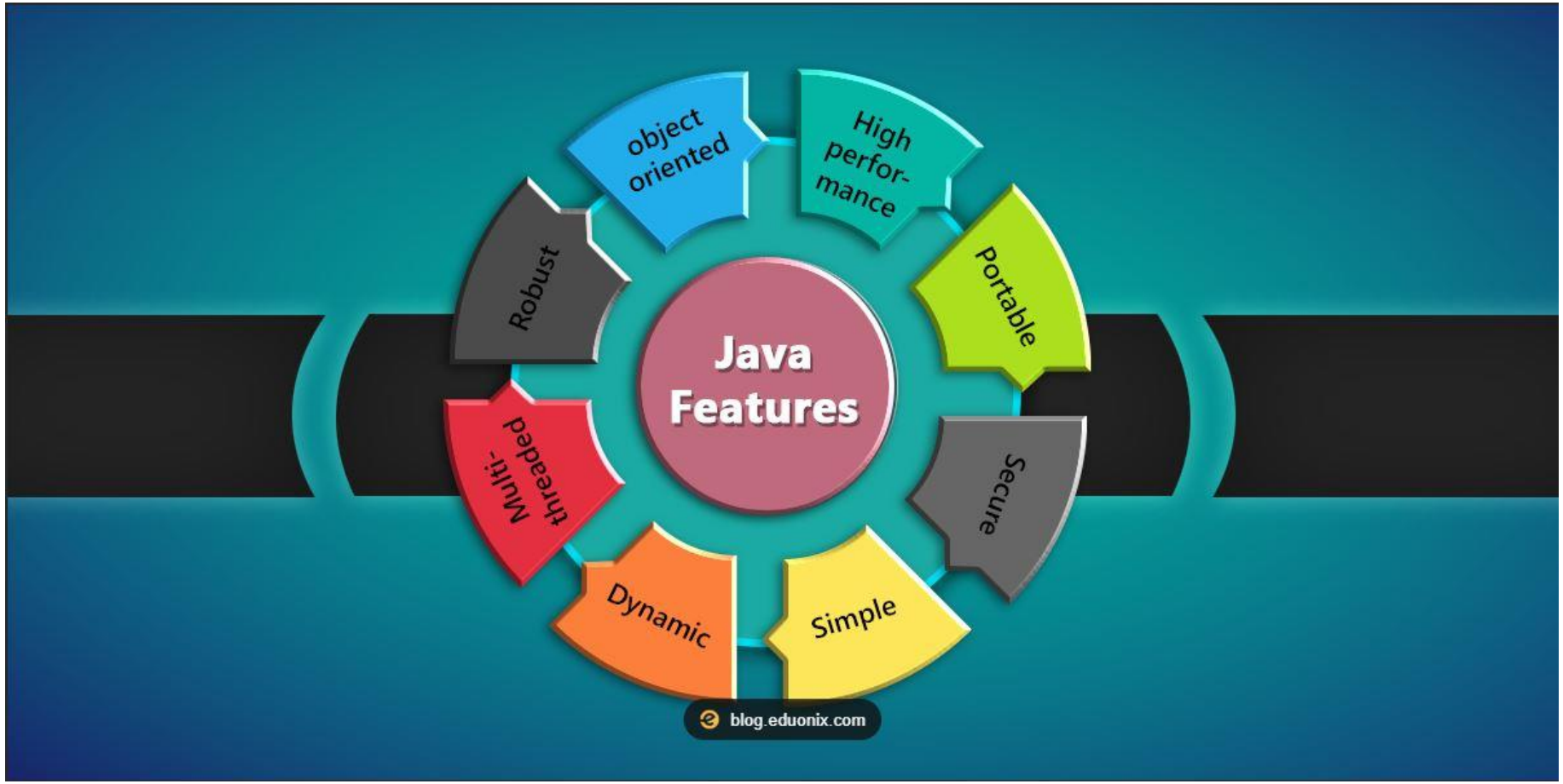
Fundamentals of Java Programming:

- **C Programming** **Vs** **Java Programming**

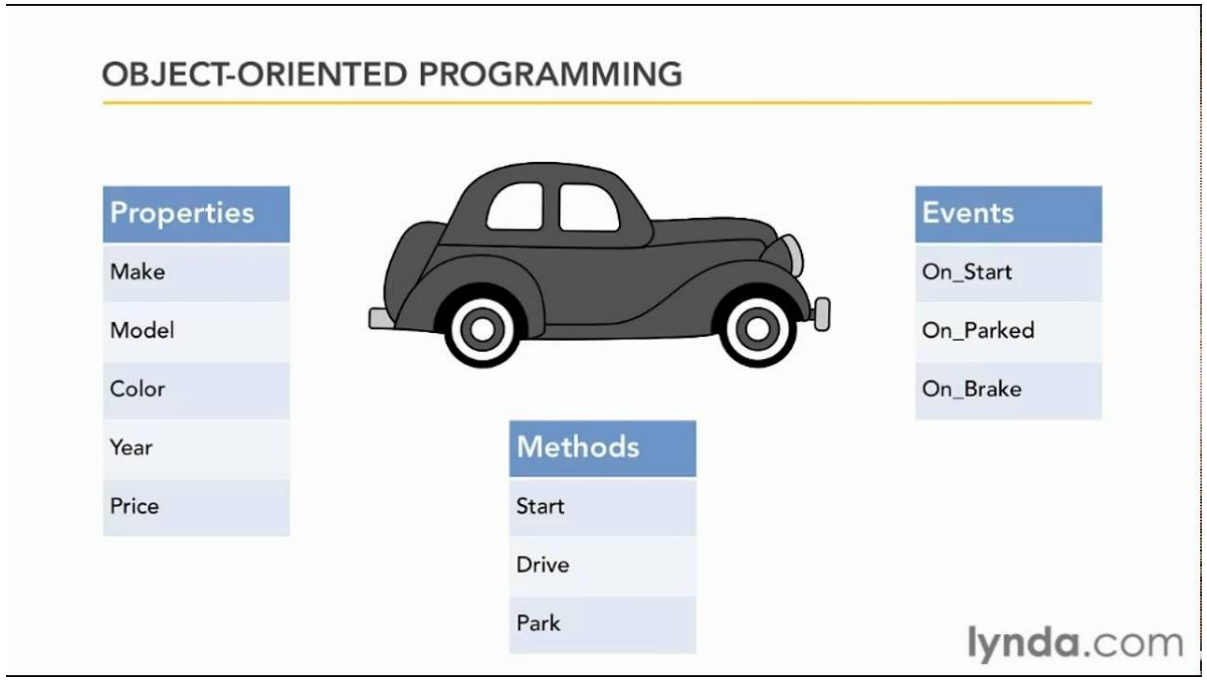


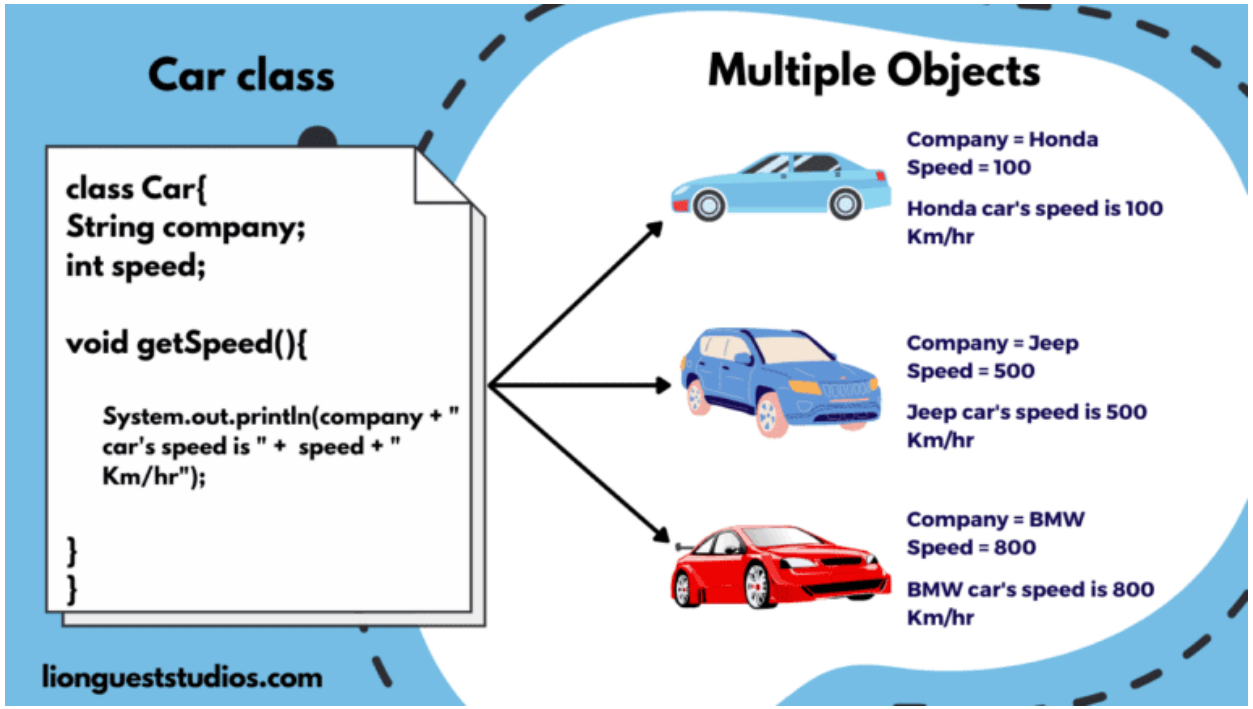


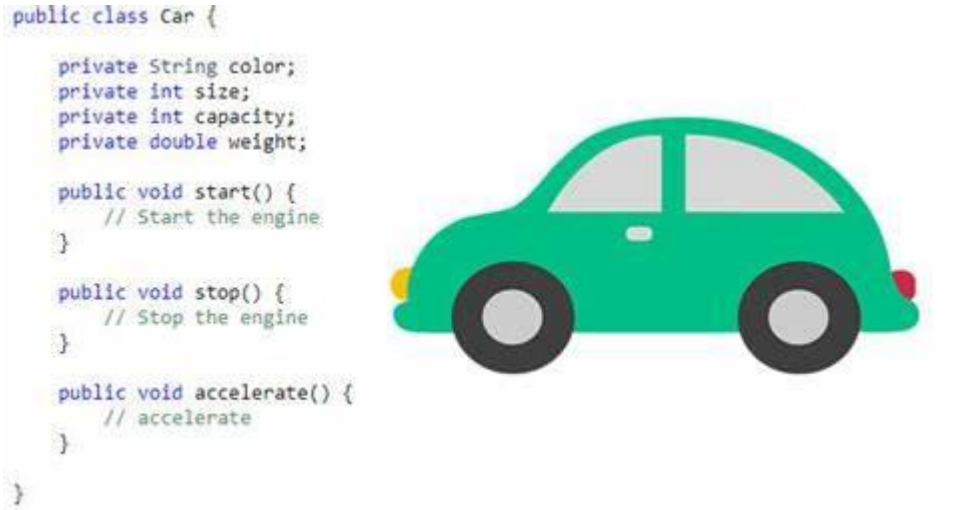
Features of Java



Java is Object Oriented







```
public class Car {

    private String color;
    private int size;
    private int capacity;
    private double weight;

    public void start() {
        // Start the engine
    }

    public void stop() {
        // Stop the engine
    }

    public void accelerate() {
        // accelerate
    }

}
```

Java Object:

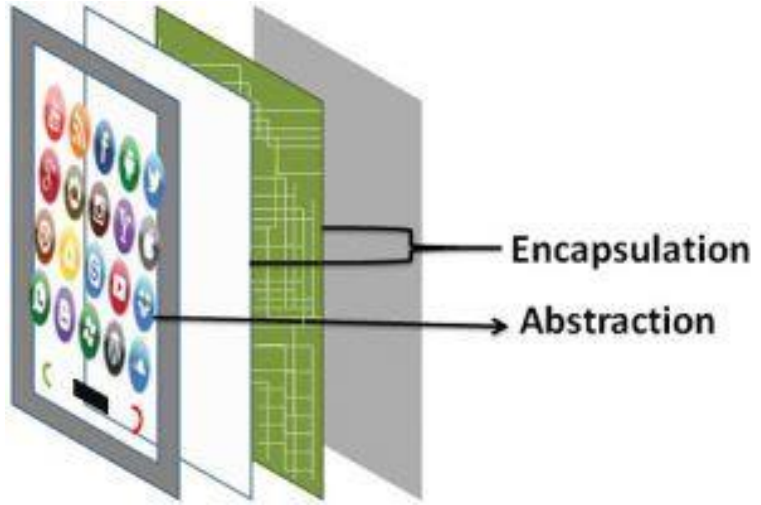
Java object is any real time entity in the world associated with some states/properties and different behaviours/ Methods where states are represented using data types(static/instance variables) and behaviours are represented using methods/functions. Modification in the states achieve the correct behaviours of any object.

4 main OOPS of java:

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

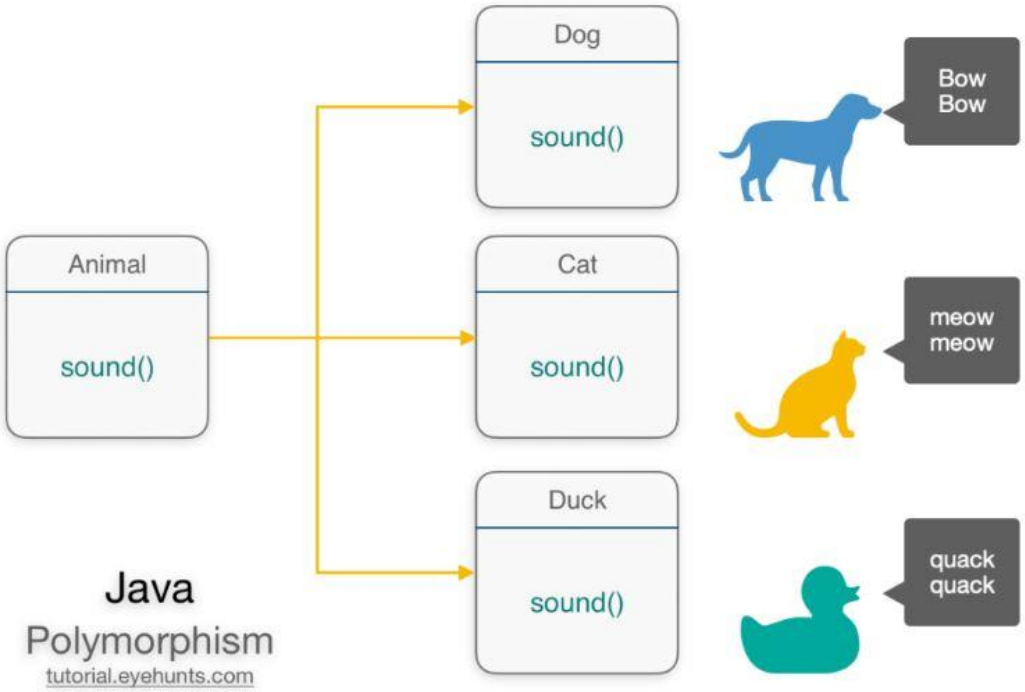
Java does support Abstraction & Encapsulation

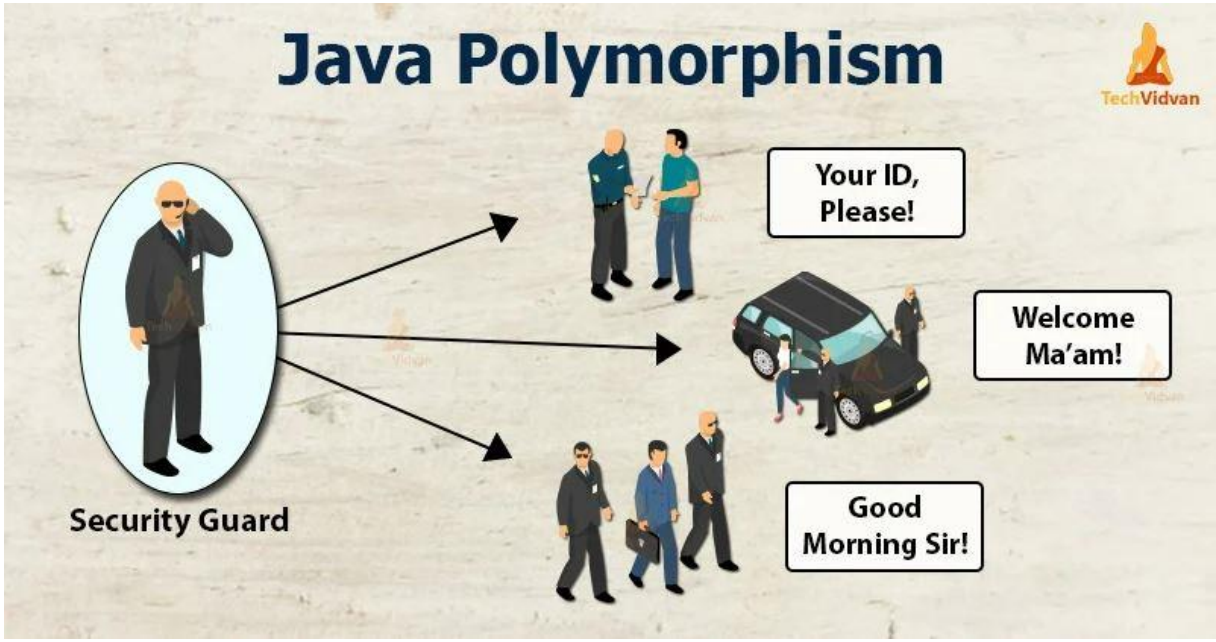
Abstraction	Encapsulation
1. Abstraction solves the problem in the design level.	1. Encapsulation solves the problem in the implementation level.
2. Abstraction is used for hiding the unwanted data and giving relevant data.	2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world.
3. Abstraction lets you focus on what the object does instead of how it does it	3. Encapsulation means hiding the internal details or mechanics of how an object does something.
4. **Abstraction**- Outer layout, used in terms of design. For Example:- Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number.	4. **Encapsulation**- Inner layout, used in terms of implementation. For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits.



Encapsulation in Java

Java does support Polymorphism

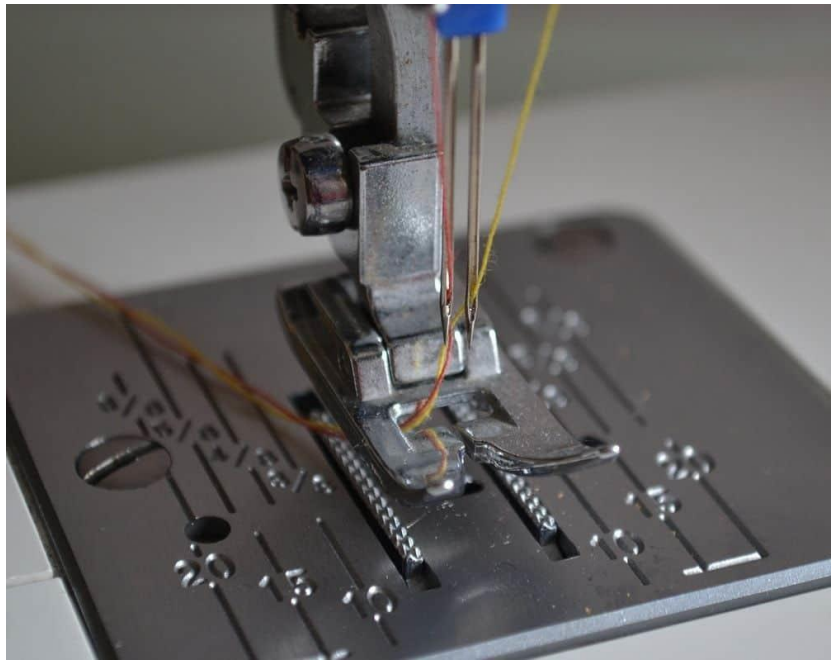


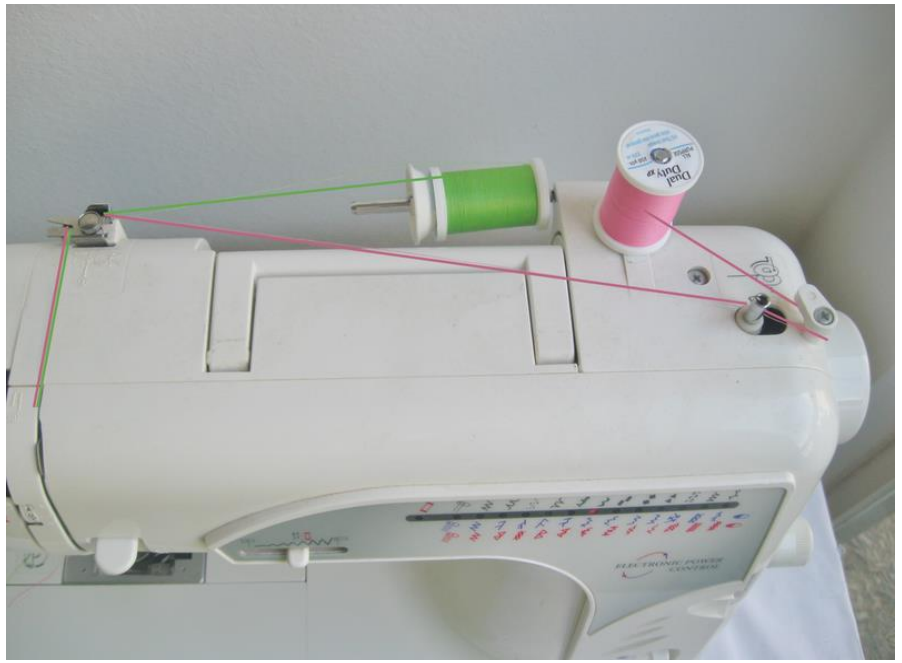


Polymorphism: One Form(Method) with many possibly different behaviours

Java is Multithreaded

Enables a program to perform several tasks simultaneously.





Java is Secure language

- **JVM**

JVM plays a vital role to provide security. It verifies the byte-code. The JVM provides guarantees that there is no unsafe operation going to execute. It also helps to diminish the possibilities of the programmers who suffer from memory safety flaws.

- **Security API's**

Java class libraries provide several API that leads to security. These APIs contain cryptographic algorithms and authentication protocols that lead to secure communication.

- **Byte Code**

Every time when a user compiles the Java program, the Java compiler creates a class file with Bytecode, which are tested by the JVM at the time of program execution for viruses and other malicious files.

- **Security Manager**

The security manager is responsible for checking the permissions and properties of the classes. It monitors the system resources accessed by the authorized classes. It also controls socket connections.

- **No Concept of Pointers**

Java does not provide support for pointers concept. It is the main security features of Java. The use of pointers may lead to unauthorized read or write operations. Therefore, the user cannot point to any memory locations.

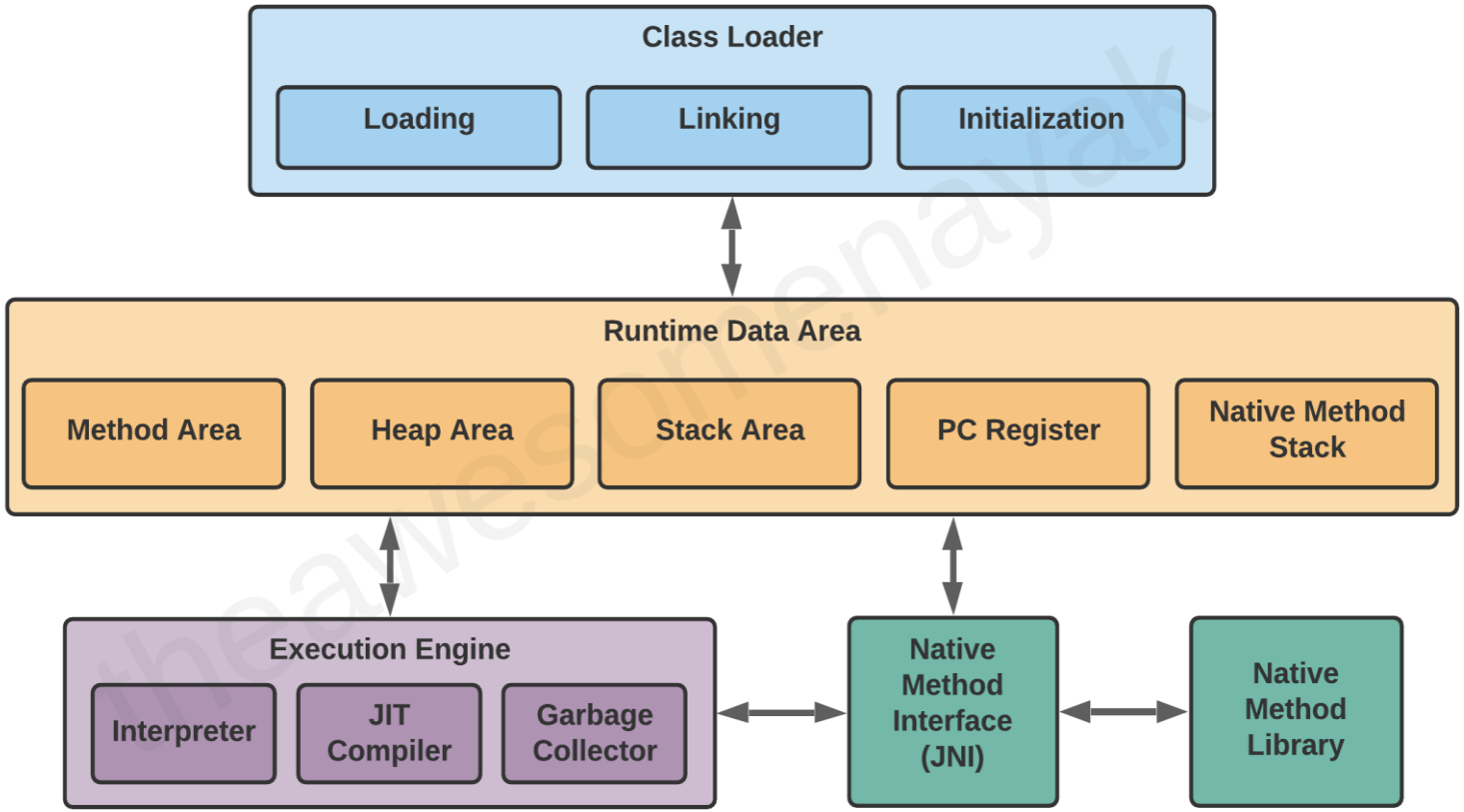
- **Memory management**

Java automatically manages memory which is known as garbage collection. The JVM manages memory itself. The programmers are free from memory management. Hence, there is no chance to fault in memory management.

- **Compile-time checking**

- Compile-time checking also makes the Java secure. Consider a scenario in which an unauthorized method is trying to access the private variable, in this case, the JVM gives the compile-time error. It prevents the system from the crash.

Java Virtual Machine



Interpreter

The interpreter reads and executes the bytecode instructions line by line. Due to the line by line execution, the interpreter is comparatively slower.

JIT Compiler

The JIT Compiler overcomes the disadvantage of the interpreter. The Execution Engine first uses the interpreter to execute the byte code, but when it finds some repeated code, it uses the JIT compiler.

The JIT compiler then compiles the entire bytecode and changes it to native machine code. This native machine code is used directly for repeated method calls, which improves the performance of the system.

Garbage Collector

The Garbage Collector (GC) collects and removes unreferenced objects from the heap area. It is the process of reclaiming the runtime unused memory automatically by destroying them.

Garbage collection makes Java memory efficient because it removes the unreferenced objects from heap memory and makes free space for new objects. It involves two phases:

Mark – in this step, the GC identifies the unused objects in memory

Sweep – in this step, the GC removes the objects identified during the previous phase

Object Oriented Programming

- Abstraction

Denotes the extraction of essential characteristics of an object that distinguish from all other kinds of objects.

- Encapsulation

Hiding the implementation details of a class.

Forces the user to use an interface to access the data.

- Inheritance

Process by which one class acquires the properties and functionalities of the other.

- Polymorphism

Means the ability of methods to exist in several different forms

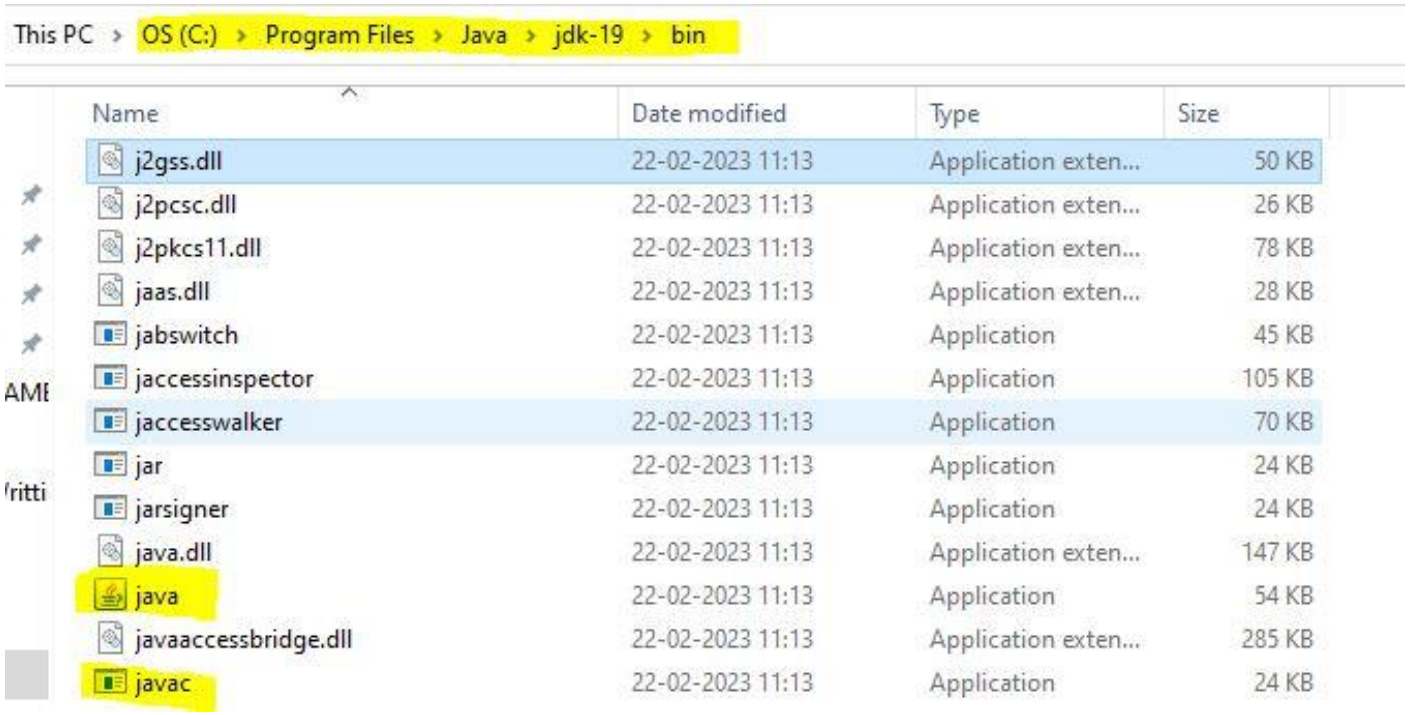
Java is a Robust Language

- Two main hurdles which cause program failures i.e memory management mistakes and mishandled runtime errors can be overcome.
    - Memory management mistakes can be overcome by garbage collection. Garbage collection is automatic de-allocation of objects which are no longer needed.
    - Mishandled runtime errors are resolved by Exception Handling procedures.

How to Write→Compile→ Execute Java Code/Program

- Download & install jdk from below link

https://www.oracle.com/java/technologies/downloads/#jdk20-windows

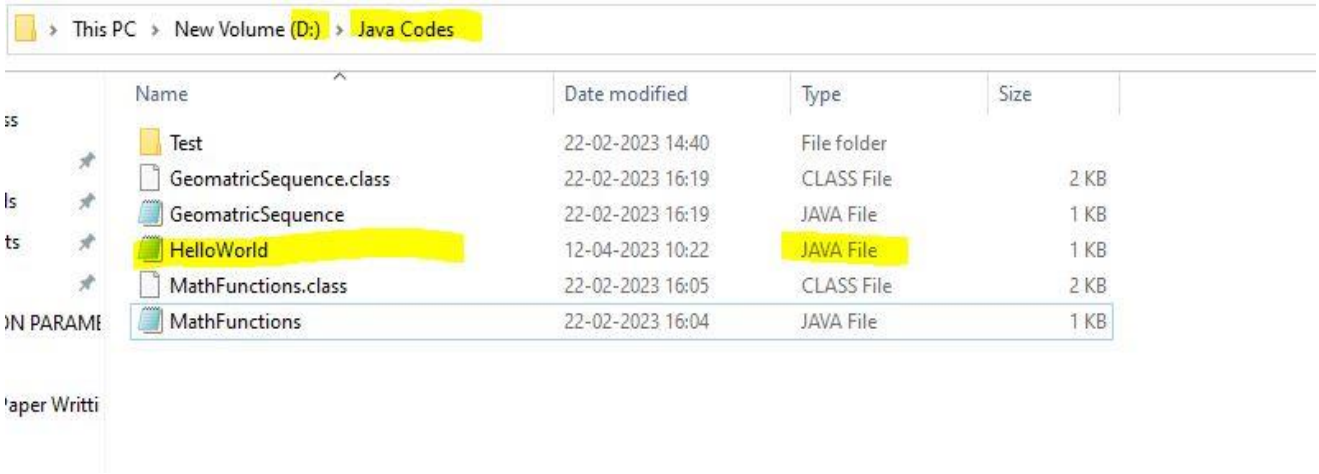


javac - The Java Compiler

java - The Java Interpreter

How to Write→Compile→ Execute Java Code/Program

1. Create folder on any drive on the PC
2. In the folder create notepad file and write java code in it and save the file with Classname.java



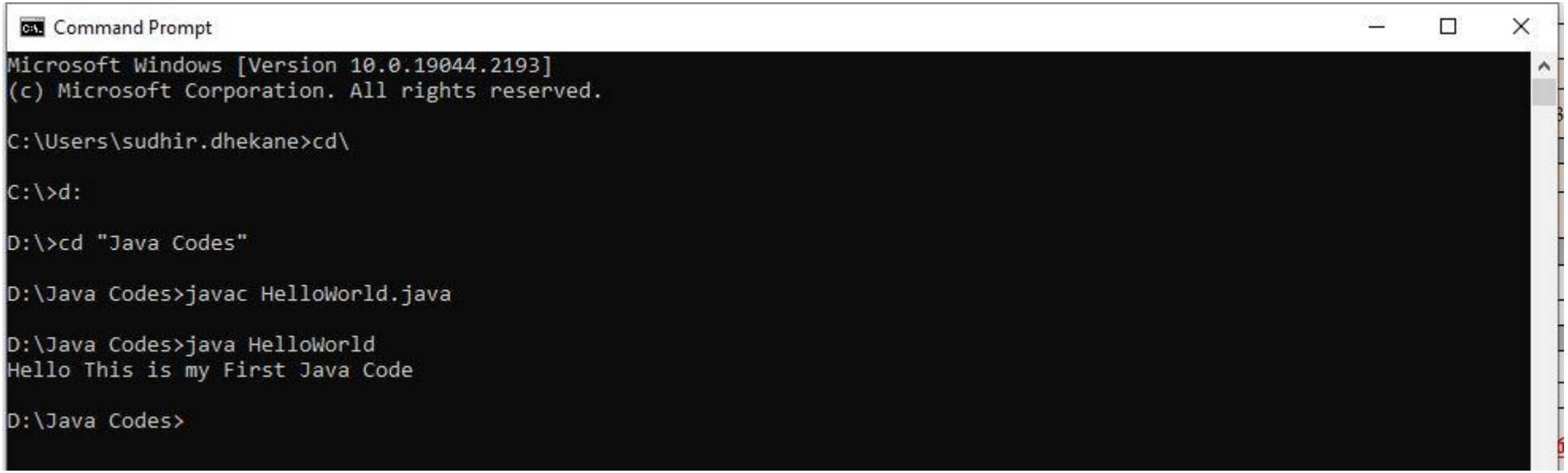

```
import java.lang.*;
class HelloWorld
{
    public static void main(String  args[])
    {
        System.out.println("Hello This is my First Java Code");
    }

}
```

How to Write→Compile→ Execute Java Code/Program

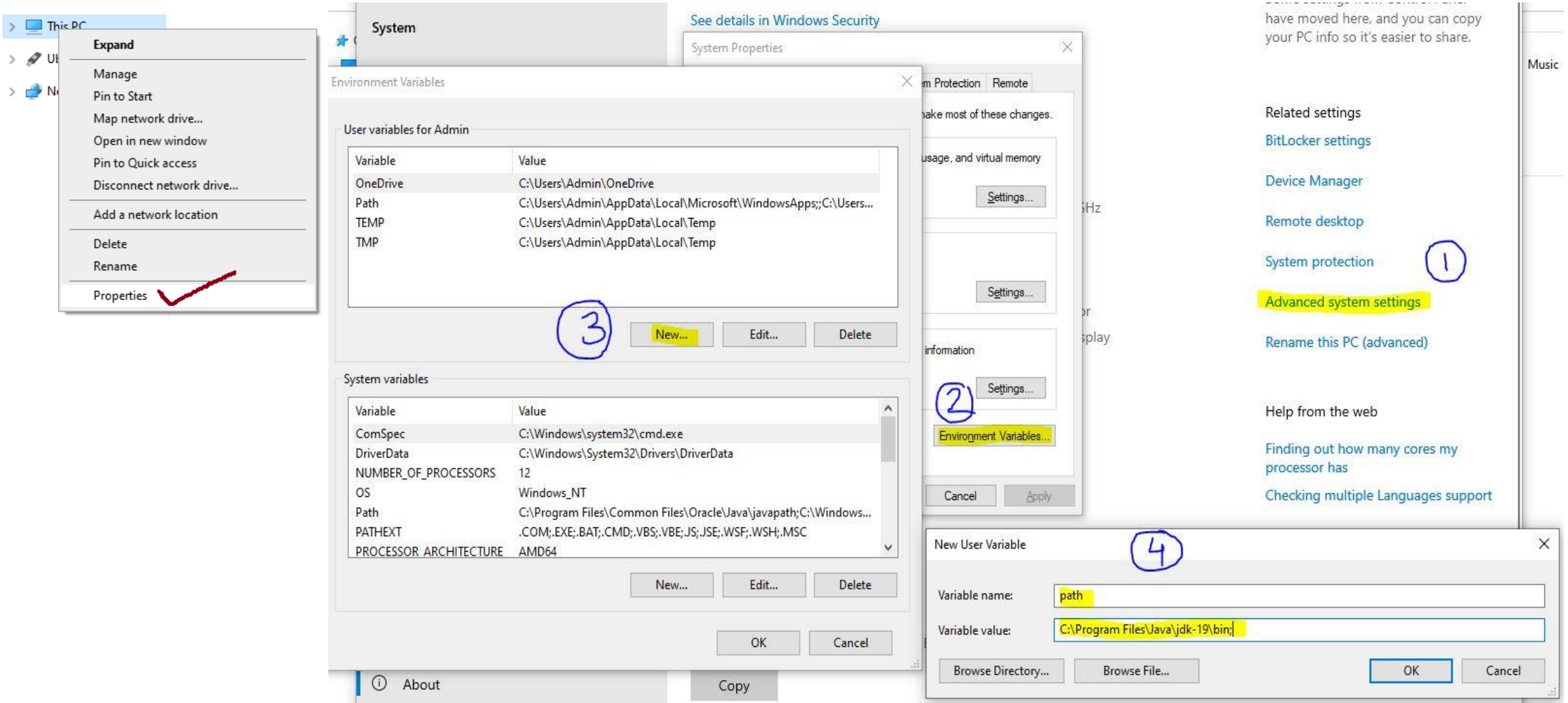
- **Step1:** Save the source file as **HelloWorld.java**
- **Step2:** Open command prompt and navigate to the directory where you have stored the file.
- **Step 3:** To compile, type **javac HelloWorld.java** and press Enter.
- **Step 4:** On successful compilation, you will see the command prompt and **HelloWorld.class** file in the same folder where **HelloWorld.java** is stored. This is the byte code of the program.
- **Step 5:** To execute, type **java HelloWorld** Do not type the extension while executing.
- **Step 6:** See the output **Hello This is my First Java Code** displayed on the console.

How to Write→Compile→ Execute Java Code/Program

1. Create folder on any drive on the PC
2. In the folder create notepad file and write java code in it and save the file with Classname.java
3. Compile java program using command: javac HelloWorld.java
4. Execute java program using command: java HelloWorld



Setting Path one time to run java program on any drive any folder:



Basic Constructs:

- Constants
- variables and data types
- Wrapper classes
- Operators and Expressions

Java Constants

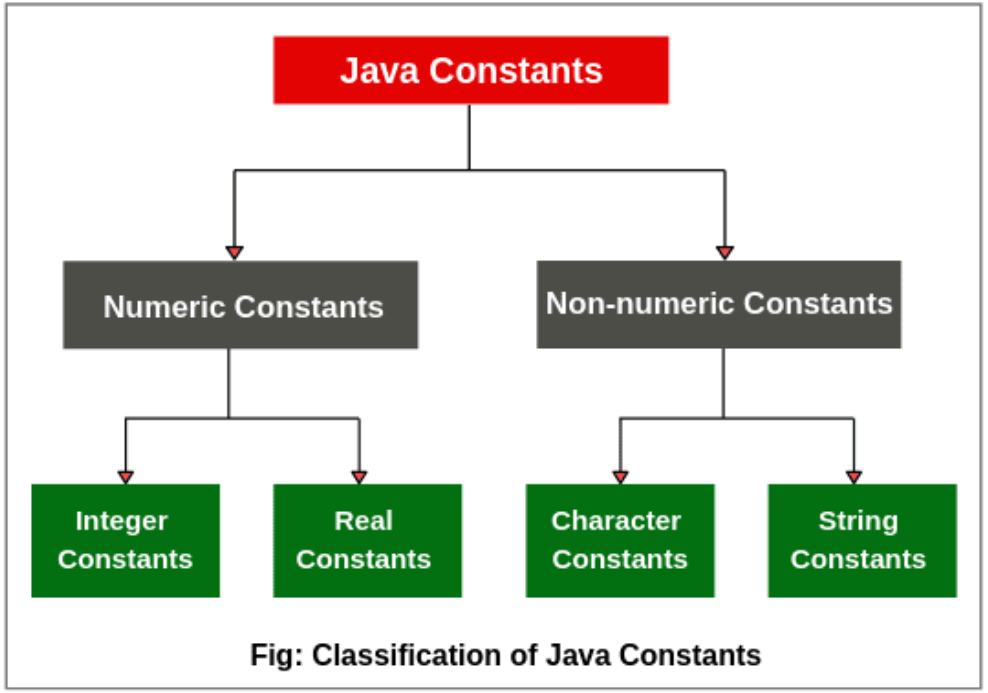


Fig: Classification of Java Constants

- A constant is a fixed value. Examples:
  - 8, -45, 2000000 : integer constants
  - -34.6, .009, 8. : floating point constants
  - "Hello, Bob", "hi" : string constants
- The default type for an integer constant is `int` (not `long`).
- The default type for a floating point constant is `double` (not `float`).

Naming Constants

Instead of using "anonymous" numbers in a program, always declare them as named constants, and use their name instead

```
public static final int INCHES_PER_FOOT = 12;
public static final double RATE = 0.14;
```

- This prevents a value from being changed inadvertently
- It has the added advantage that when a value must be modified, it need only be changed in one place
- Note the naming convention for constants: Use all uppercase letters, and designate word boundaries with an underscore character

Data Types

There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types:

Primitive Data Types:

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types.

- **boolean:** boolean data type represents one bit of information.
  - There are only two possible values : true and false.
  - This data type is used for simple flags that track true/false conditions.
  - Default value is false.
  - Example : boolean one = true
- **char:** char data type is a single 16-bit Unicode character.
  - Minimum value is '\u0000' (or 0).
  - Maximum value is '\uffff' (or 65,535 inclusive).
  - Char data type is used to store any character.
  - Example . char letterA ='A'
- **byte:** Byte data type is a 8-bit signed two's complement integer.
  - Minimum value is -128 (-2^7)
  - Maximum value is 127 (inclusive)(2^7 -1)
  - Default value is 0
  - Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
  - Example : byte a = 100 , byte b = -50

Primitive Data Types:

- **short:** Short data type is a 16-bit signed two's complement integer.
  - Minimum value is -32,768 (-2^15)
  - Maximum value is 32,767(inclusive) (2^15 -1)
  - Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
  - Default value is 0.
  - Example : short s= 10000 , short r = -20000
- **int:** Int data type is a 32-bit signed two's complement integer.
  - Minimum value is - 2,147,483,648. (-2^31)
  - Maximum value is 2,147,483,647. (2^31 -1)
  - Int is generally used as the default data type for integral values unless there is a concern about memory.
  - The default value is 0.
  - Example : int a = 100000, int b = -200000
- **long:** Long data type is a 64-bit signed two's complement integer.
  - Minimum value is -9,223,372,036,854,775,808.(-2^63)
  - Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)
  - This type is used when a wider range than int is needed.
  - Default value is 0L.
  - Example : int a = 100000L, int b = -200000L

Primitive Data Types:

float: Float data type is a single-precision 32-bit IEEE 754 floating point.

Float is mainly used to save memory in large arrays of floating point numbers.

Default value is 0.0f.

Float data type is never used for precise values such as currency.

Example : float f1 = 234.5f

double: double data type is a double-precision 64-bit IEEE 754 floating point.

This data type is generally used as the default data type for decimal values. generally the default choice.

Double data type should never be used for precise values such as currency.

Default value is 0.0d.

Example : double d1 = 123.4

Identifiers

- Important Note:
  - Java identifiers are case-sensitive – this means that upper and lower case letters are considered to be different – be careful to be consistent!
  - Ex: AccountNo and accountno are NOT the same
- Naming Convention:
  - Many Java programmers use the following conventions:
    - **Classes:** start with upper case, then start each word with an upper case letter
    - Ex: StringBuffer, BufferedInputStream, ArrayIndexOutOfBoundsException
    - **Methods and variables:** start with lower case, then start each word with an upper case letter
    - Ex: compareTo, lastIndexOf, mousePressed

Literals

- Values that are hard-coded into a program
  - They are **literal**ly in the code!
- Different types have different rules for specifying literal values
  - They are fairly intuitive and similar across most programming languages
  - Integer
    - An optional +/- followed by a sequence of digits
    - **Ex: 1024, -78, 1024786074**
  - Character
    - A single character in single quotes
    - **Ex: ‘a’, ‘y’, ‘q’**
  - String
    - A sequence of characters contained within double quotes
    - Ex: **“This is a string literal”**
  - See p. 75-77 for more literals

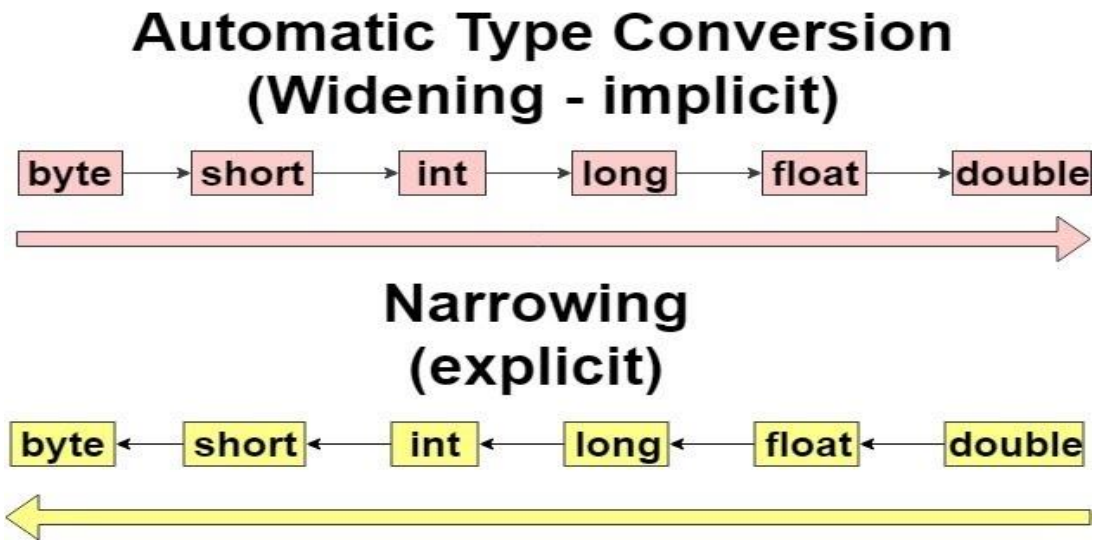
Variables

- Note: For numeric types, you even get an error if the value assigned will “lose precision” if placed into the variable
  - Generally speaking this means we can place “smaller” values into “larger” variables but we cannot place “larger” values into “smaller” variables
    - Ex: byte < short < int < long < float < double
    - Ex: `int i = 3.5;`

```
possible loss of precision found   : double
                          required: int
                int i = 3.5;
                        ^
```

  - Ex: `double x = 100;`
    - » **This is ok**

Type casting in Java



This is done forcefully by a programmer to specific data type.

Syntax:

(data_type) variable;

or

(data type) expression;

Example 1:

```
int a=7,b=2, int q;
double Q;
q = a/b;
Q = (double)a/b;
```

Example 2:

```
int a=8;
double b=2.2;
int q;
q =(int) a/b;
```

Variables

- Floating point literals in Java are by default double
  - If you assign one to a float variable, you will get a “loss of precision error” as shown in the previous slide
- If you want to assign a “more precise” value to a “less precise” variable, you must explicitly **cast the value** to that variable type

Error check each of the statements in the box to the right

```
int i = 5;
int j = 4.5;
float x = 3.5;
float y = (float) 3.5;
double z = 100;
i = z;
y = z;
z = i;
j = (long) y;
j = (byte) y;
```

Operators and Expressions

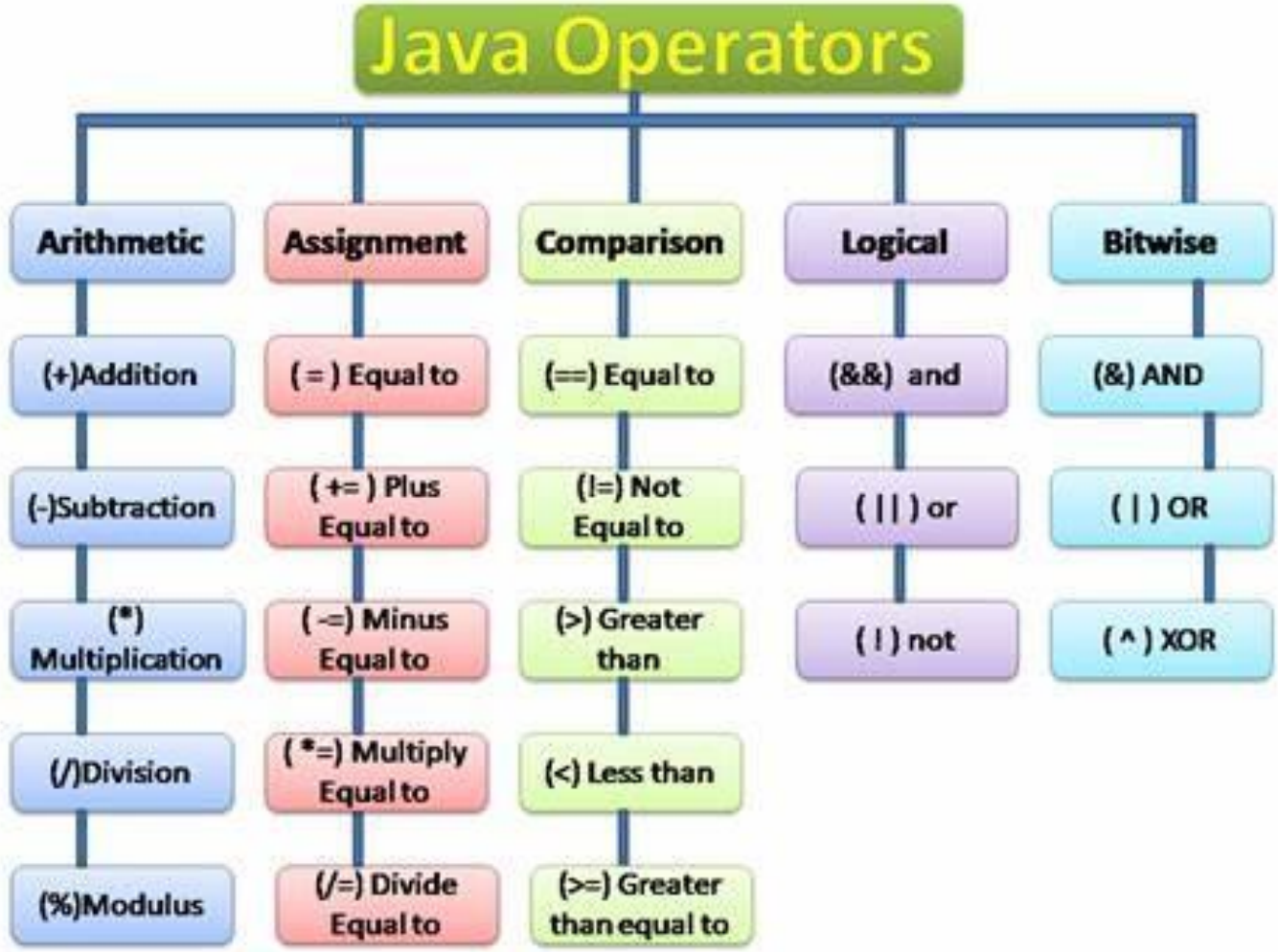
- Numeric operators in Java include

  +, –, *, /, %

  - These are typical across most languages
  - A couple points, however:
    - If both operands are integer, / will give integer division, always producing an integer result – discuss implications
    - The % operator was designed for integer operands and gives the remainder of integer division
    - However, % can be used with floating point as well

```
int i, j, k, m;
i = 16;  j = 7;
k = i / j;                    // answer?
m = i % j;          // answer?
```

Operators and Expressions



Symbol/s	Number of operands	Description	Example
=	Two	Assigns the value from the right hand operand to the left hand operand	`int i;` `i = 10;`
+=	Two	Adds the right hand operand to the left hand operand and assign the result to the left hand operand.	`int i = 10;` `i += 10;// The value of i is now 20`
-=	Two	Subtracts the right hand operand from the left hand operand and assign the result to the left hand operand.	`int i = 10;` `i -= 10;// The value of i is now 0`
/=	Two	Divides the left hand operand by the right hand operand and assign the result to the left hand operand.	`int i = 10;` `i /= 5;// The value of i is now 2`
*=	Two	Multiply the left hand operand with the right hand operand and assign the result to the left hand operand.	`int i = 10;` `i *= 5;// The value of i is now 50`
%=	Two	Divide the left hand operand by the right hand operand and assign the reminder to the left hand operand.	`int i = 10;` `i %= 4;// The value of i is now 2`

Operators and Expressions

Symbol/s	Number of Operands	Description	Example

+	Two	Adds two operands and returns the result	`double i{ 10 };`
			`char j{ 20 };`
			`auto a = i + j; //type of a will be double and value will be 30`

-	Two	Subtracts the 2nd operand from the first operand	`double i{ 10 };`
			`char j{ 20 };`
			`auto a = j-i; //type of a will be double and the value will be 10`

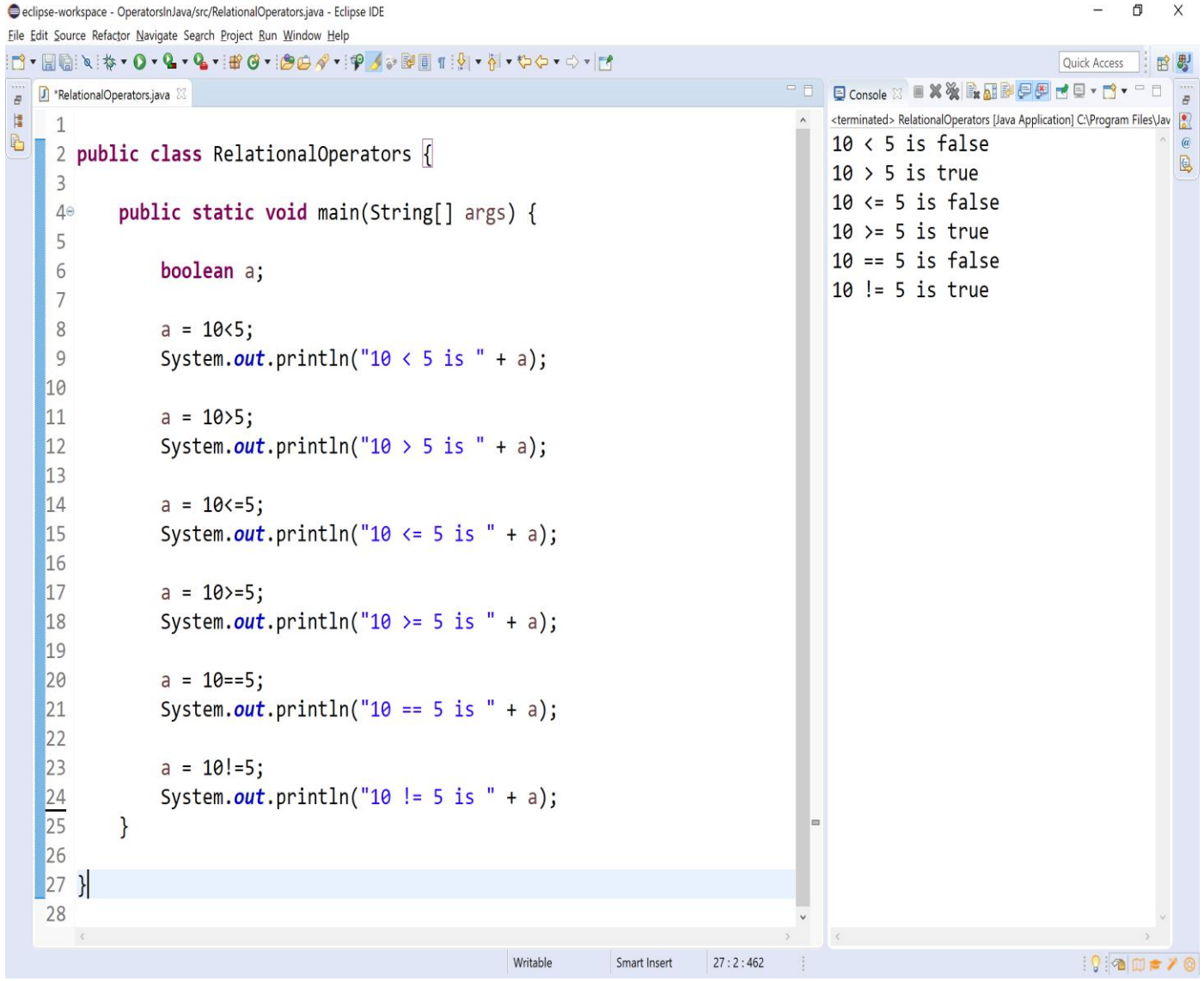
*	Two	Multiplies two operands and returns the result	`double i{ 10 };`
			`char j{ 20 };`
			`auto a = j*i; //type of a will be double and the value will be 200`

/	Two	Divides the 1st operand by the 2nd operand and returns the result	`double i{ 15 };`
			`char j{ 50 };`
			`auto a = j/i; //type of a will be double and the value will be 3.33333`

++	One	Increases the operand value by one. It can be post-fix and pre-fix	`double i{ 15.3 };`
			`int j{ 50 };`
			`int k = j++; // Post-fix: value of k will be 50 and j will be 51`
			`double l = ++i; //pre-fix: value of l will be 16.3 and i will be 16.3`

--	One	Decreases the operand value by 1. It can be post-fix and pre-fix	`double i{ 15.3 };`
			`int j{ 50 };`
			`int k = j--; // Post-fix: value of k will be 50 and j will be 49`
			`double l = --i; //Pre-fix: value of l will be 14.3 and i will be 14.3`

%	Two	Divides the 1st operand by the 2nd operand and returns the remainder	`long i{ 15 };`
			`int j{ 50 };`
			`int k = j%i; // value of k will be 5`



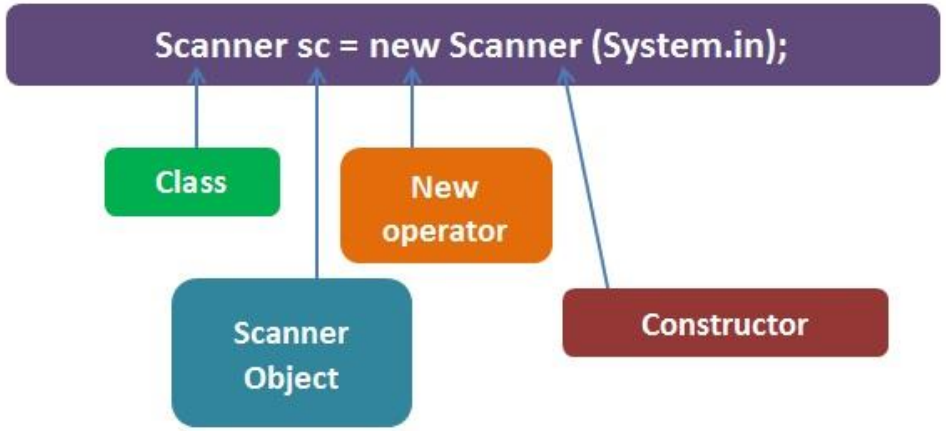
Operators and Expressions

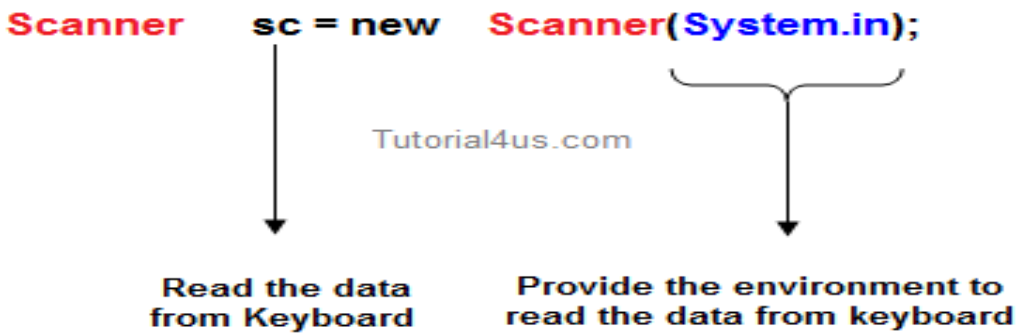
Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === “5”	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == “5”	True

```
public class LogicalOperatorExample{
    public static void main(String args[]){
        boolean a=true;
        boolean b=true;
        boolean c=false;
        System.out.println(a&&b);//True as both are true
        System.out.println(a&&c);//Fasle as c is false
        System.out.println(a||c);//True as a is true
        System.out.println(!c);//True as c is false
    }
}
```

How to Take input in java (Using Scanner Class)






```
import java.util.Scanner
```
1. Import Scanner Class

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


2. Construct Scanner class Object

```
Scanner s=new Scanner(System.in);
System.out.println("Enter first no= ");

int num1, num2;
```
3. Define Variable to Receive Input

```
num1=s.nextInt();

System.out.println("Enter 2nd no ");
```
4. Read Input from Keyboard

```
num2=s.nextInt();
System.out.println("Sum of no is= "+(num1+num2));
}
}
```

How to Take input in java (Using Scanner Class)

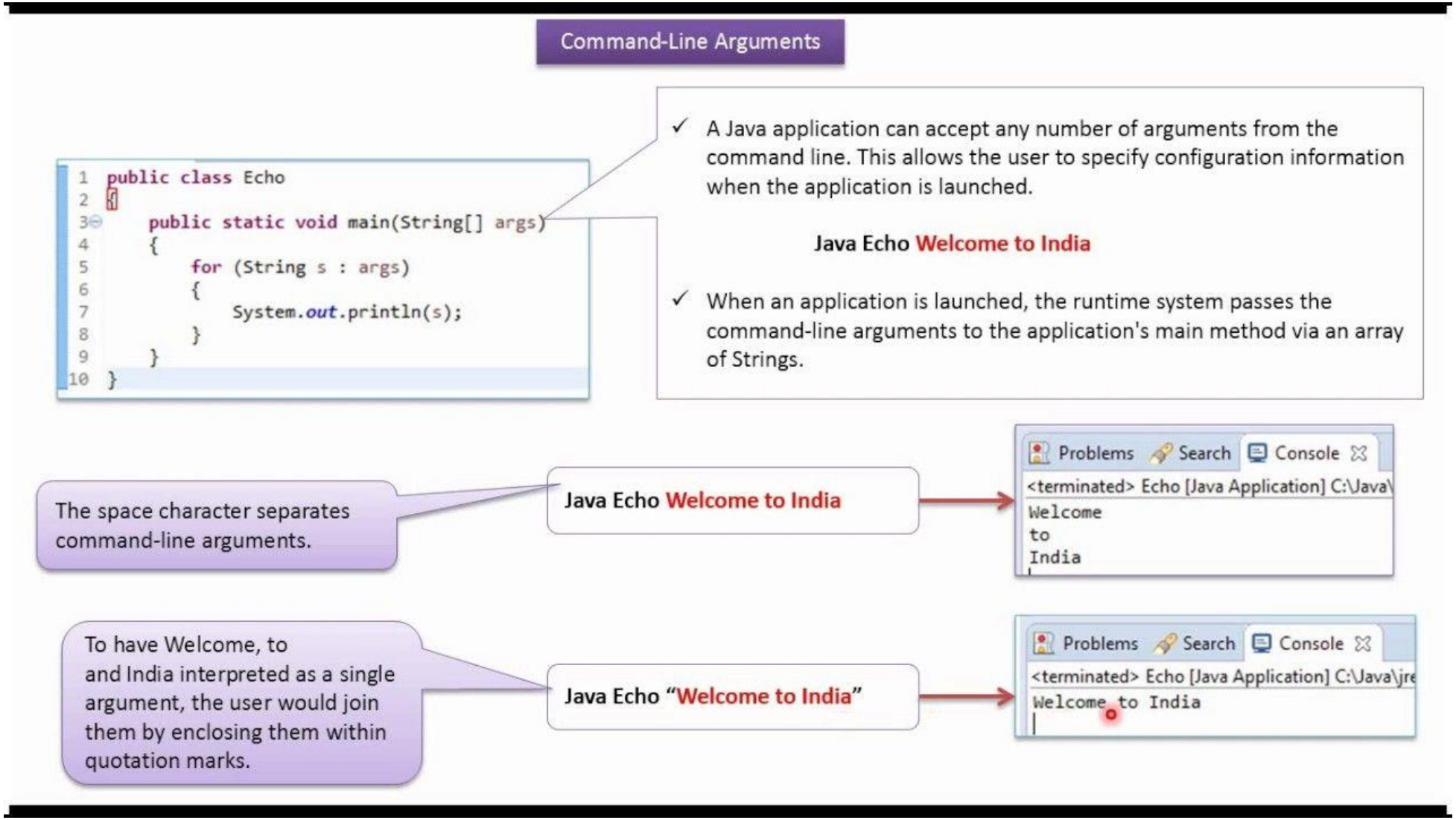
Method	Description
public String next()	it returns the next token from the scanner class.
public String nextLine()	it moves the Scanner position to the next line and returns the value as a string
public byte nextByte()	it reads the next token as a byte.
public int nextInt()	it reads the next token as an int.
public float nextFloat()	it reads the next token as a float.
public short nextShort()	it reads the next token as a short.
public long nextLong()	it reads the next token as a long.
public double nextDouble()	it reads the next token as a double.
public boolean nextBoolean()	it reads the boolean value.

```
public class ScannerClass {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the double data ");
        double num = sc.nextDouble();
        System.out.println("Double data: "+num);

        System.out.print("Enter the float data ");
        float num2 = sc.nextFloat();
        System.out.println("Float data: "+num2);

        System.out.print("Enter the byte data ");
        byte num3 = sc.nextByte();
        System.out.println("Byte data: "+num3);

        System.out.print("Enter the short data ");
        short num4 = sc.nextShort();
        System.out.println("Short data: "+num4);
    }
}
```

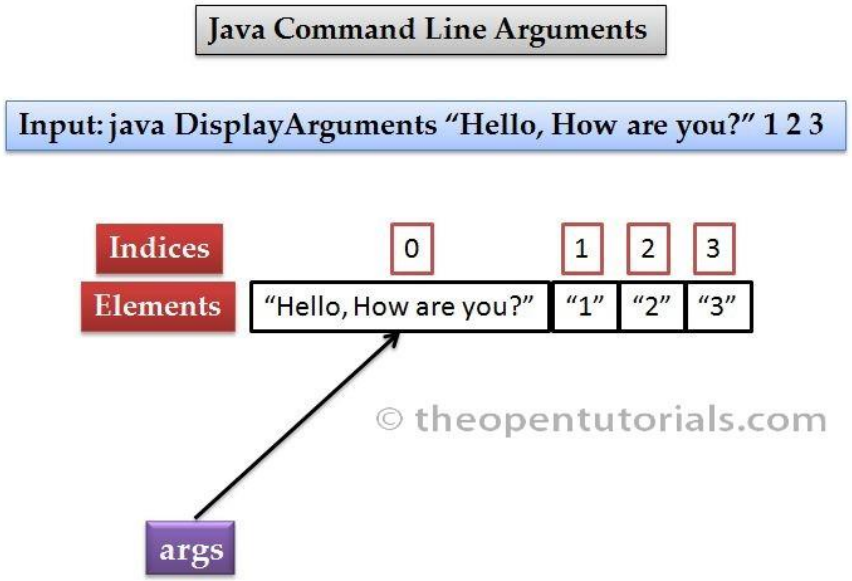
Output:

```
Enter the double data 78695.9865
Double data: 78695.9865
Enter the float data 7.6
Float data: 7.6
Enter the byte data 76
Byte data: 76
Enter the short data 6754
Short data: 6754
```

How to Take input in java (Using Command Line Argument)



How to Take input in java (Using Command Line Argument)



javac Mainclass.java

java Mainclass value1 value2 value3.....

Command Line Arguments

Command Line Arguments in Java

If any input value is passed through the command prompt at the time of running of the program is known as **command line argument** by default every command line argument will be treated as string value and those are stored in a string array of main() method.

Program Command Line Argument in Java

```
class CommandLineExample
{
public static void main(String args[])
{
System.out.println("Argument is: "+args[0]);
}
}
```

Compile and Run above programs

```
Compile By > Javac CommandLineExample.java
Run By > Java CommandLineExample Porter
```

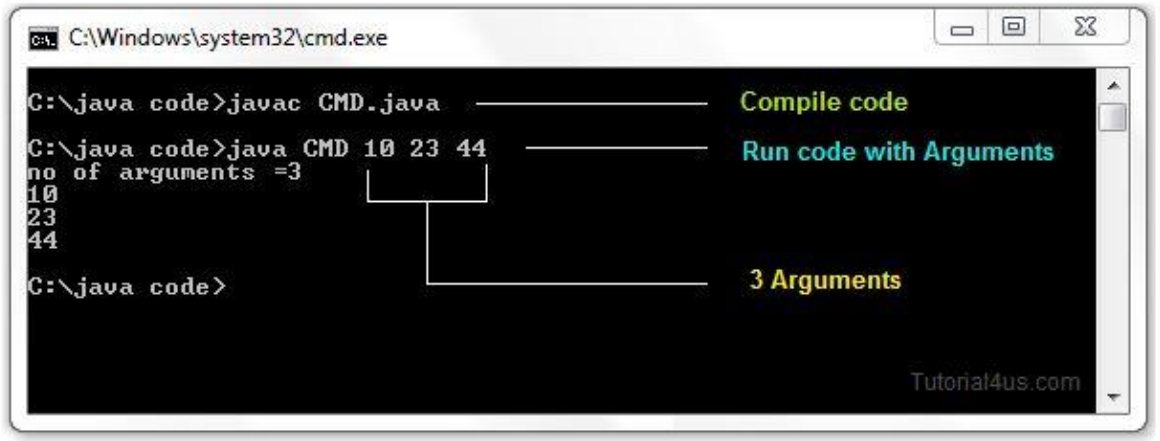
Output

```
Argument is: Porter
```

Wrapper class and their methods to convert default string type args[] command line input into respective data type

Accept command line arguments and display their values

```
class   CMD
{
public  static  void  main(String  k[])
{
System.out.println("no.  of  arguments  ="+k.length);
for(int  i=0;i<  k.length;i++)
{
System.out.println(k[i]);
}
}
}
```




```
public class CommandLineArguments {

   public static void main(String[] args) {

      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum = a + b;
      System.out.println("Sum is " + sum);
   }
}
```

java CommandLineArguments 8 5

Wrapper class and their methods to convert default string type args[] command line input into respective data type

Conversion From String To Number Using Wrapper Class

Sr No	Convert from string to	Method
1	Byte	Byte.ParseByte()
2	Short	Short.valueOf()
3	Integer	Integer.shortInt()
4	Long	Long.parseLong()
5	Float	Float.parseFloat()
6	Double	Double.parseDouble()

Input through BufferedReader class

BufferedReader is another way to take the input from the user, but it's a bit more complex than the Scanner class. java.io.BufferedReader reads text from the character-input stream.

BufferedReader is a bit faster than Scanner as Scanner does the parsing of input data, and BufferedReader simply reads the sequence of characters.

Wrapper classes and its methods are used to convert input string by BufferedReader class method readLine() in to its respective primitive

```
import java.io.*;
public class BufferedReaderclass {
	public static void main(String[] args) throws Exception {
		BufferedReader bf = new BufferedReader(new InputStreamReader(System.i
n));
		System.out.println("Enter your integer data here:");
		int x = Integer.parseInt(bf.readLine());
		System.out.println("Enter your float data here:");
		float y = Float.parseFloat(bf.readLine());
		System.out.println("Enter your string here:");
		String object = bf.readLine();
		System.out.println("Cast from string to Integer: "+ x);
		System.out.println("Cast from String to Float: "+ y);
		System.out.println(object);
	}
}
```

Output:

```
Enter your integer data here:
90
Enter your float data here:
9.654
Enter your string here:
Coding Ninjasss
Cast from string to Integer: 90
Cast from String to Float: 9.654
Coding Ninjasss
```

BufferReader vs Scanner class in Java

Scanner class	BufferedReader class
The scanner class is not synchronous and does not support threads.	The BufferedReader class is synchronous and Widely used with multiple threads.
Scanner breaks its input into tokens using a Delimiter pattern.	BufferedReader simply reads the sequence of characters in a portion that depends on the buffer size.
The scanner has a little buffer(1KB byte buffer).	It has a significantly larger buffer memory than Scanner.(8KB byte buffer)
The scanner is slow as it does the parsing of input data. Moreover, It hides IOException.	Unlike Scanner, BufferedReader simply reads the sequence of characters. Hence it is faster than the Scanner Class. It throws an IOException.

Oral questions exercise

- What is bytecode? What is its significance ?
- How java is platform independent?
- List features of java?
- What is abstraction and encapsulation means?
- How to define constants in java
- List different ways to read input in java
- Differentiate Scanner vs. BufferedReader
- List methods of Scanner Class & Methods of Wrapper classes
- Should I use a scanner or BufferedReader in Java?
- What is data typecasting how it is carried out in java
- What is JVM?