CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE, INDIA

MAI419-3 – JAVA PROGRAMMING

UNIT-1

By

Dr.Thirunavukkarasu, MCA.,M.Phil.,SET.,PhD &
Dr.Sridevi

Introduction to Object Oriented Programming (OOP)

Def of OOP: It is a programming approach in which data type of the data structure and its associated operations are defined. So that each data structure is considered as an object.

Why do we need object-oriented programming?

Need to understand the limitations and how OOP arose from traditional programming approach.

Conventional (or) Procedural Languages

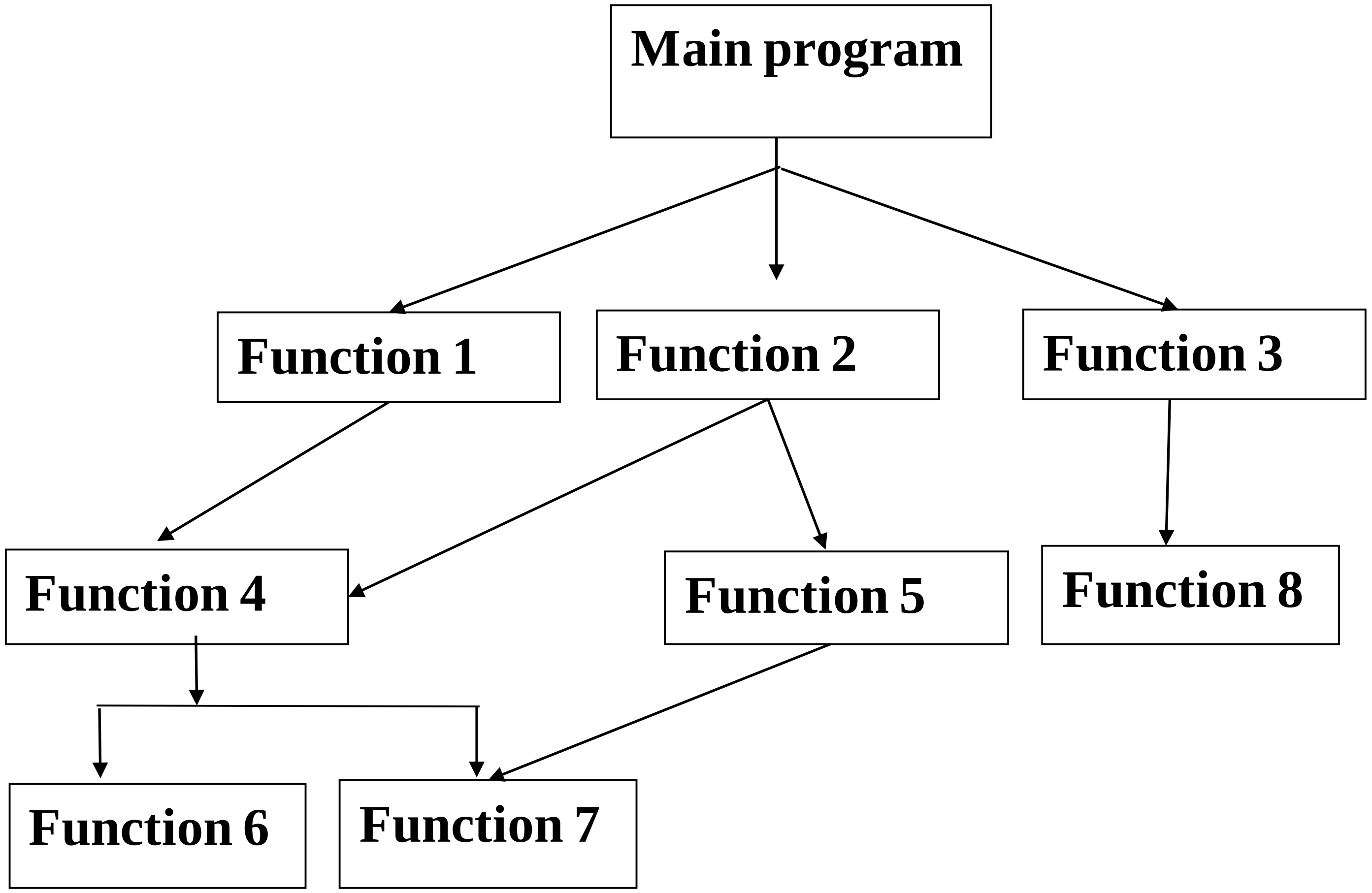
It makes use of high-level language such as C, COBOL, PASCAL, and FORTRAN.

The primary focus is on functions. Functions or otherwise called sub routine or sub program or procedure.

Characteristics of Conventional (or) Procedural Languages

- It focuses on process rather than data.
- Large programs are divided into function.
- Most of the function share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach.

Hierarchical structure of Conventional or procedure-oriented programming



Problems with structured Or Procedure oriented Programming

1. Unrestricted access of global data is vulnerable.

2. In large programs it is difficult to identify which data is shared by which function.

3. Unrelated functions and data leads poor model of real-world problem and sometimes difficult to formulate the real-world problems.

NOTE:

❖ In C global variables can be declared more than once. But in C++ global variables can be declared only once.

- **Lack of flexibility-**Small changes in requirements often need major changes in the code.

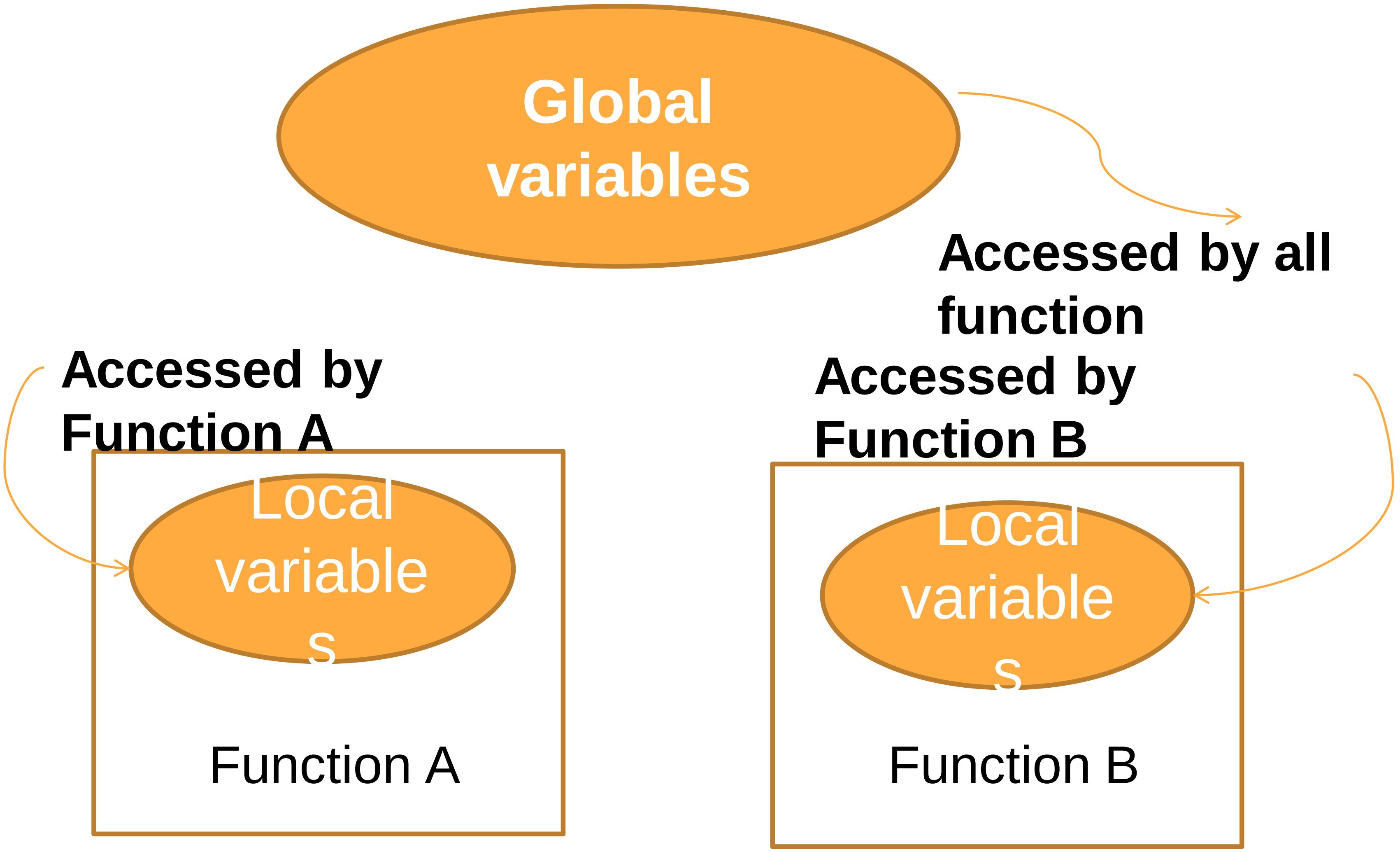
- **Difficulty handling complex problems-**Problems involving uncertainty, learning, or pattern recognition (e.g., speech or image recognition).

- **Poor handling of incomplete or vague data**

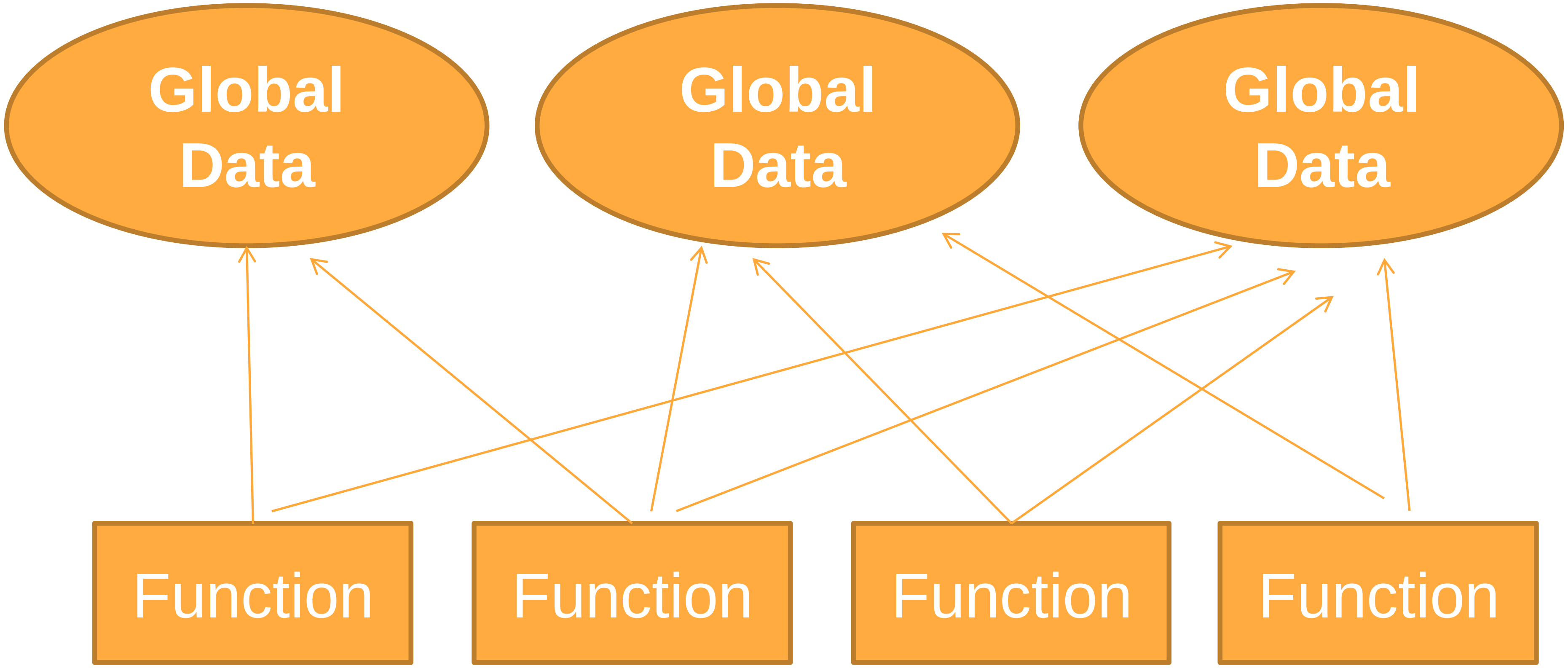
- **Time-consuming development-**Writing detailed step-by-step instructions can be lengthy and harder to maintain as programs grow.

- **Limited abstraction for real-world problems**

Unrestricted Access



The procedural paradigm



Difficulties in procedural paradigm

- Number of connections causes a problem in several ways.
- First it makes program structure difficult to conceptualize.
- It makes the program difficult to modify.
- Change in global data item may result in rewriting all the functions that access the items.
- For example, in inventory program the product code is changed from 5 digit to 12 digit leads changes in data type short to long data type.
- All function that access product code should be modified to deal with the changes.

Object oriented Programming

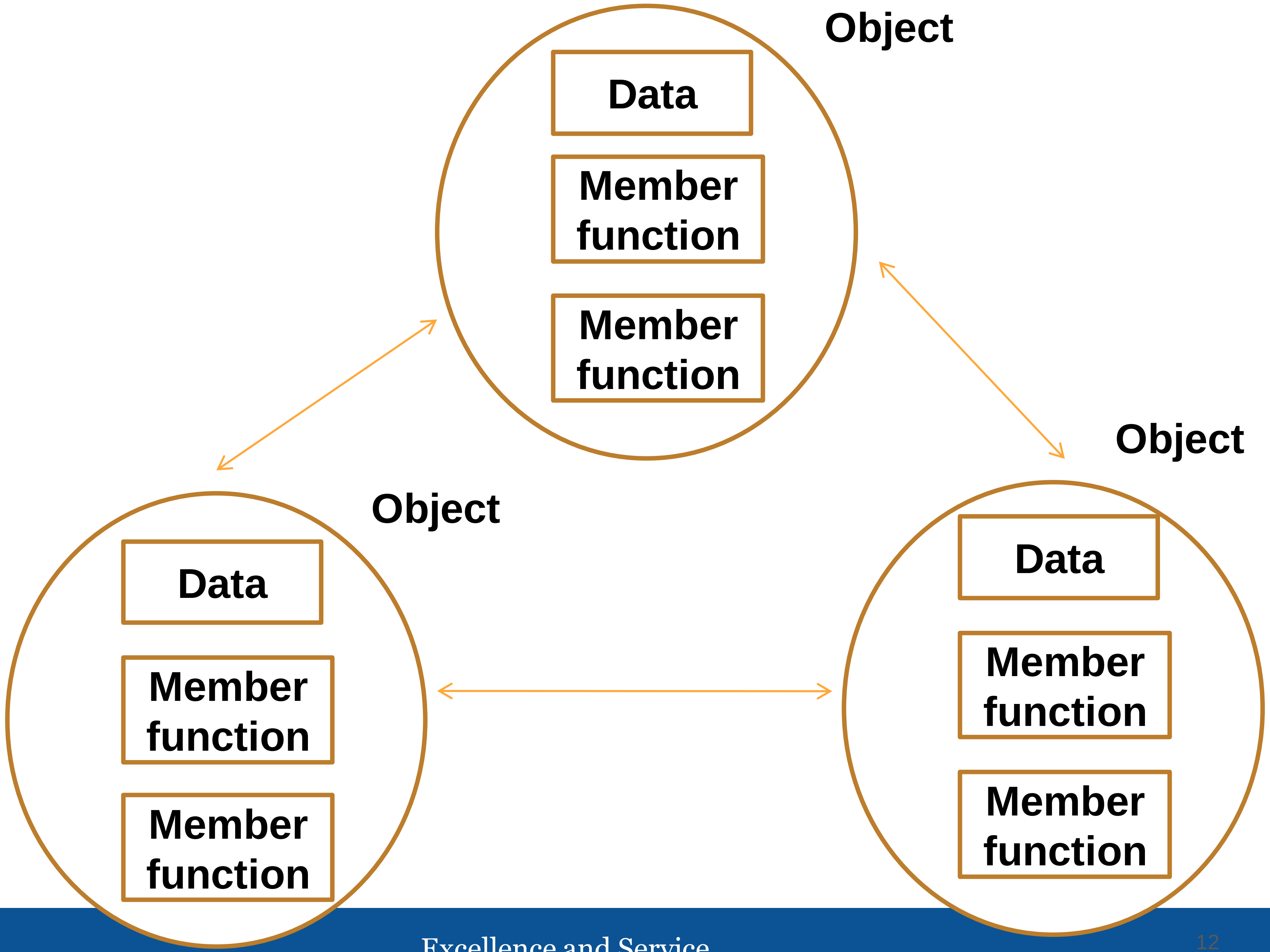
- The fundamental idea behind object-oriented language is to combine data and function into a single unit. Such unit is called object.
- An object's function is called member function and data is called member data or member variables.
- Member variables are accessed through member functions.

Features of Object-oriented Programming

- It decompose a problem in to number of entities called objects.
- It will avoid the drawback of conventional programming.

- Emphasis is on data rather than procedure.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be added whenever necessary.
- It follows bottom-up approach

OO paradigm


Object
Data
Member function
Member function
Object
Data
Member function
Member function
Object
Data
Member function
Member function

Characteristics of Object-oriented languages

- OOP: An approach to organization:

- OOP is not primarily concerned with details of program operation instead it deals with overall organization of the program.

Object oriented concepts (Principles):

- Object
- Class
- Data abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message communication

Objects

Anything in the real world is called object.

- **Physical object**

- **Elements of the computer user environment**

  Windows, Menus, Graphic objects (Line, rectangle, circle), The mouse key board, disk drives, printers.
- **Data storage constructs**

  Customized arrays, Stacks, Linked list, Binary trees
- **Human Entities**

  Employees, Students, Customers, Sales people
- **Collection of data**

  An inventory, A personal tile, A dictionary
- **User defined data types**

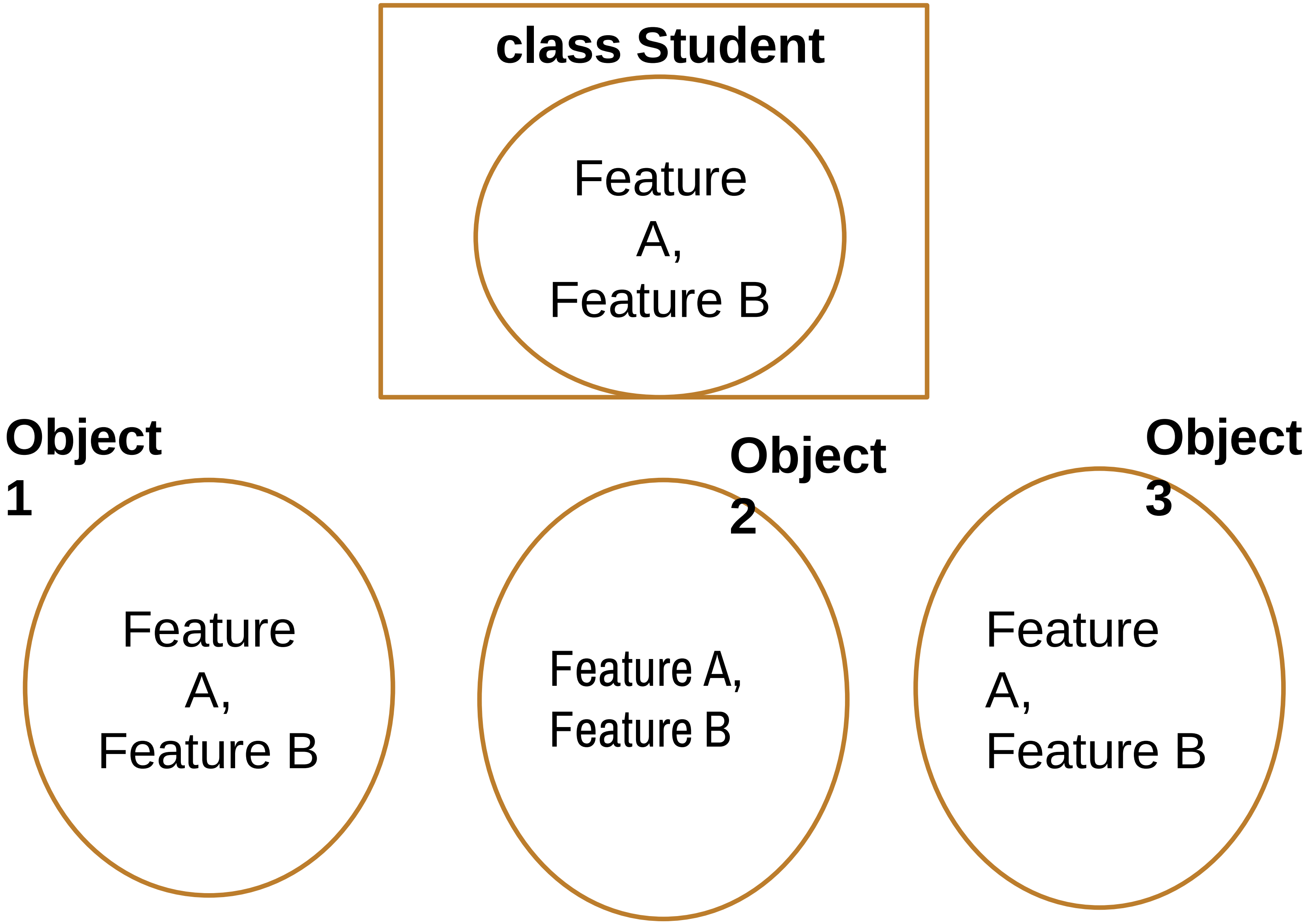
Time, Angles, Complex numbers, Points on the plane

Class

- A class encompasses data and function into a single unit.
- Class serves as plan or template and specifies what data and what function will be included in objects of that class.
- In OOP objects are the members of the class.
- We can define many object to the same class.
- Through class objects we can access member function and member variables(data).

Example for class

```
Class Student
{
// DATA
Name;
DOB; // date of birth
Marks;
//Function
Total();
Avg();
Display();
};
Void main()
{
Student s1,s2,s3;
}
```


class Student
Feature A, Feature B
Object 1
Feature A, Feature B
Object 2
Feature A, Feature B
Object 3
Feature A, Feature B

Data Abstraction

- Act of representing essential features without including background details or explanation such as size, weight, memory, etc.

- Example

  Class student

- Since the classes uses the concept of data abstraction, they are known as *Abstract Data Types* (ADT).

Data Encapsulation

Def: Wrapping of data and function in to a single unit is called encapsulation

Ex:

Class student

{

//DATA:

Private:

Name

DOB

Marks

//FUNCTIONS

Public:

Total()

Avg()

Display() };

Data Hiding

Definition:

Insulation of data from direct access is called data hiding

Example:

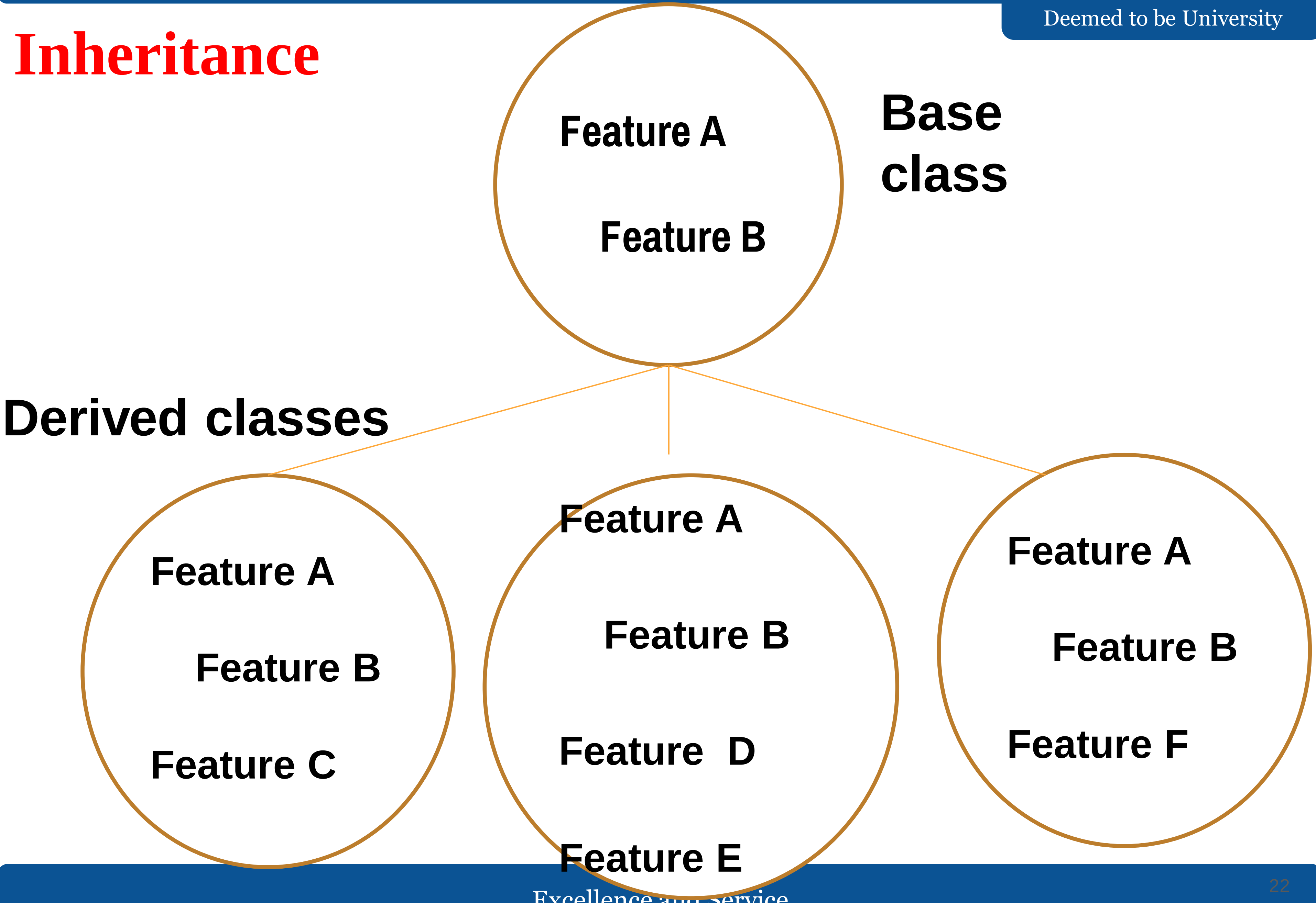
```
Class student
{
Private:
Name;
DOB;
Marks;
};
```

- In the above class the member variables cannot be accessed out side the class only it is accessible only with in the class.

Inheritance

- The idea of class leads to the idea of inheritance.
- The methods and properties of one class is invoked by another class.
- Inheritance is mainly invoked for code reusability.
- Ex : The class of vehicles is divided into cars, tucks, buses and motorcycles.
- All subclasses are share some common properties such as wheels and motors.
- In addition to common properties each class has its own characteristics.
- Ex: Buses have seats for many peoples while trucks has space for heavy loads.

Inheritance



Reusability

- Once the class has been written, created and debugged it can be distributed to other programmers for use in their own programs.

- It is similar to library of function in procedural language.

- Programmer can take existing class without any modification and add some additional features and capabilities.

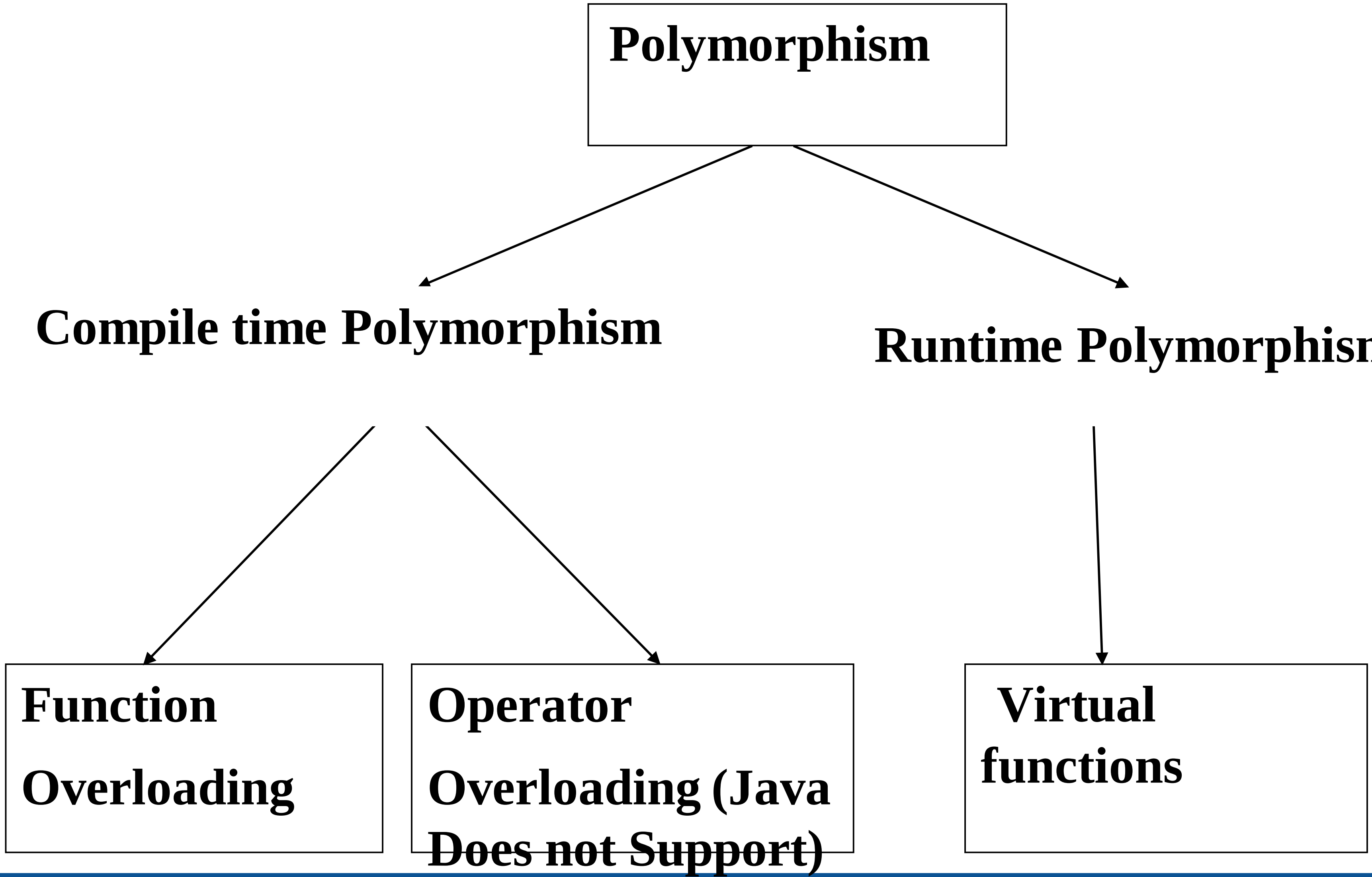
- This is done by deriving a new class from the existing one.

- Ex: Creating menu items

Polymorphism

- Polymorphism indicates the ability to take more than one form.
- Using operators or functions in different ways depending on what they are operating on is called polymorphism.
- When existing operators are used other than its original purpose it is called operator overloading.
- Same function name is used for different purpose it is called function overloading.
- An operation may exhibit different behaviours indifferent instances. The behaviour depends upon the types of data used in the operation.

Types of polymorphism


Polymorphism
Compile time Polymorphism
Runtime Polymorphism
Function Overloading
Operator Overloading (Java Does not Support)
Virtual functions

- Java uses **dynamic method dispatch** to decide which method to execute based on the **actual object**, not the reference type.

- **All non-static, non-final, and non-private methods in Java are virtual**.

Virtual functions support run-time polymorphism, allowing:

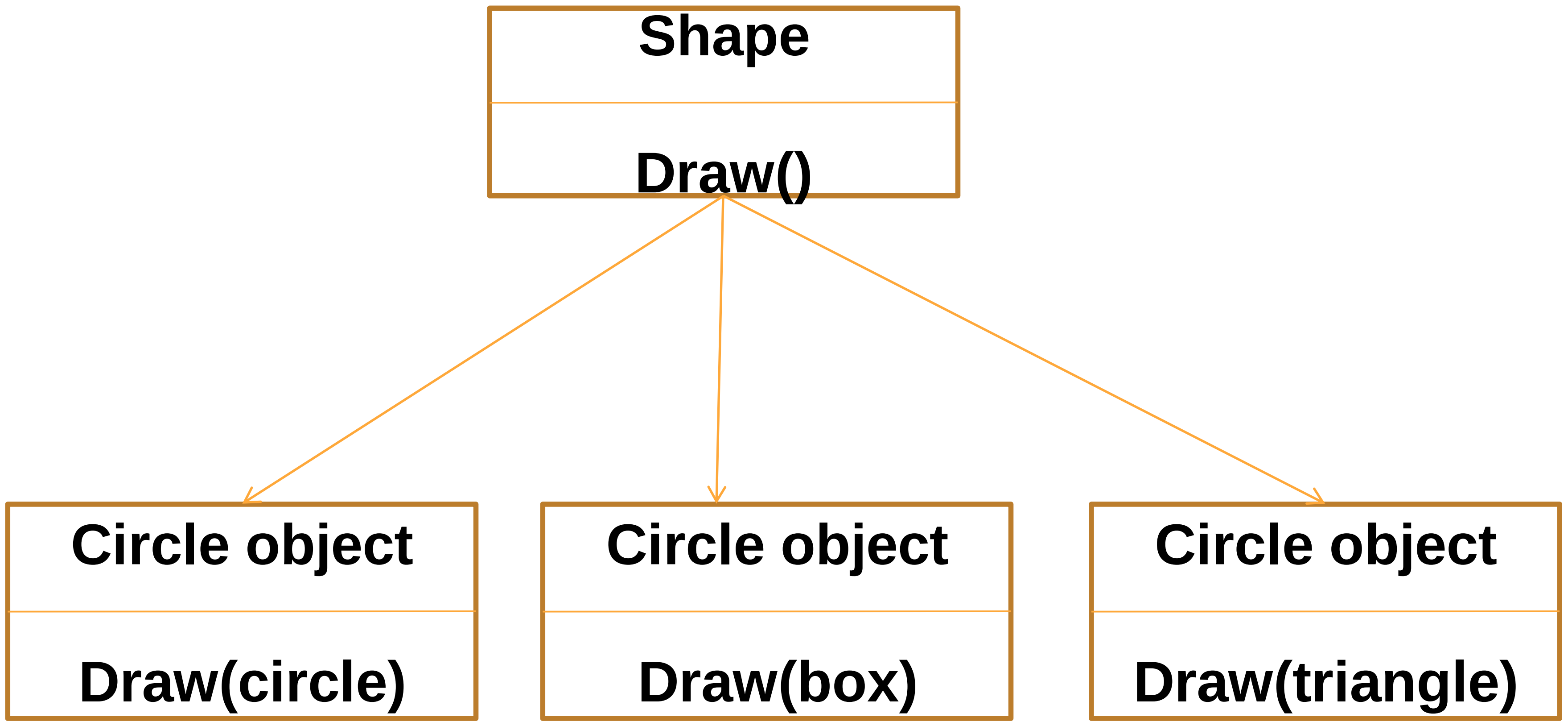
- Method overriding
- Flexible and extensible code
- Correct method execution for subclass objects

Methods that are **NOT** virtual in Java

- static methods, final methods, private methods and Constructors

- These methods are resolved at **compile time**, not run time.

Polymorphism



Dynamic Binding (or) Late Binding

- Linking of procedure call to the code to be executed in response to the call at run time.

- The code associated with given procedure is not known until runtime.

- It is associated with polymorphism and inheritance.

- In the above figure the procedure draw() is linked at the runtime depending on the parameter.

Message Passing

- ❖ One object can communicate with another object by sending or receiving message.
- ❖ The concept of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterparts.
- ❖ A message for an object is a request for execution of a procedure.
- ❖ It will invoke a function(procedure)in the receiving object that generates the desired result.

❖*Message passing* involves specifying the name of the object, the name of the function (message) and the information to be sent.

❖Example:

S1.getdata(parameters)
S1 is an object for a particular class, getdata is a message and parameters is an information to the objects

Object based programming language

❖ Language that support programming with objects and will not support inheritance and dynamic binding.

❖Ex: Ada

Object oriented programming

- Object based features + Inheritance + Dynamic binding.

Pure object-oriented languages:

Simula, small talk80, Eiffel Ruby, scala

Extended conventional languages

Objective C, C++, objective Pascal, turbo pascal5.5

Java is **object-oriented but not purely object-oriented** because:

- It supports **primitive data types (**int, char, float, boolean, etc) these are not objects.
- It allows **static members (It belong to the class but not an object)**
- Programs can execute without creating objects
- Java does not support multiple inheritance using classes (only via interfaces),

Benefits of object-oriented Programming

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.

- We can build programs from the standard working modules that communicate with one another, rather than writing the code from scratch.

- This leads to saving *of* development time and higher productivity.

- The principle of data hiding helps the programmer to build a secured programs that cannot be invoked by other parts of the program.

.

- It is possible to have multiple instances of an object to co-exist without any interfaces.

- It is possible to map objects in the problem domain to those in the program.

- It is easy to partition the work in a project based on objects.

- The data-centered design approach enables us to capture more details of a model in implementable form.

- Object-oriented systems can be easily upgraded from small to large systems.

- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.

- Software complexity can be easily managed.

Problems in object-oriented Programming

•Object libraries must be available for reuse.

•Strict controls and protocols need to be developed **if reuse is not to be compromised.**

•Developing a software that is easy to use makes it bard to build (Easy for the user ≠ easy for the developer) more effort, time, and skill to develop

•More planning and design required

•Higher memory usage

•Complexity in large systems

Application of OOP

- Real time systems
- Simulation and modeling
- Object oriented database(**ObjectDB, db4o, Versant, GemStone/S, and ObjectStore**.)
- Hyper text, Hyper media
- AI and Expert system
- Neural networks and parallel computing
- Decision support and office automation system.
- CAD/CAM

JAVA Evolution : JAVA History

- Java is a general-purpose object-oriented programming language developed by SUN micro system of USA in the year 1991.
- Java was developed by James Gosling, Patrick Noughton, Chris Wrath, Ed Frank and Mike Sheridan at Sun Microsystems in 1991.
- The language was initially called **Oak** but was renamed Java in 1995.
- The primary motivation was the need for a platform-independent language.
- It is used to develop a software for various consumer electronic devices, such as microwave ovens, TVs, VCRs etc.

- Java is related to C++.
- Much of the character of Java is inherited from c, and c++.
- Java derives its syntax from c and many of its object-oriented features from c++.
- Java was influenced by C++ and not an enhanced version of C++.
- Java is Simple, reliable, portable and powerful language.

- 1990 –special software that could be to manipulate consumer electronic devices.
- 1991 - The team announced a new language 'OAK'
- 1992 – Home applications with touch sensitive screen
- 1993 –WWW -text based internet into GUI
- 1994- Hot Java (web browser to locate and run the applets on internet)
- 1995 –oak is renamed as java – Netscape and Microsoft announced support to java.
- 1996 – jdk1.0 (Java Development Kit 1.0) released by SUN micro systems.

- 1997-jdk1.1
- 1998-java 2 with version 1.2 of the software development kit (SDK 1.2)
- 1999- SUN releases Java 2 platform, standard edition (J2SE)and enterprise edition (J2EE)
- 2000- J2SE with SDK1.3 was released.
- 2002-J2SE with SDK1.4 was released.
- 2004- J2SE with JDK5.0 (Instead of JDK1.5)was released. This is known as J2SE5.0

Java SE 6- released on December 11, 2006, Sun replaced the name "J2SE" with Java SE.

Scripting Language Support, Improved Web Service support, JDBC 4.0 support, Many GUI improvements.

Java SE 7- launched on July 7, 2011- JVM support for dynamic languages, Concurrency utilities under JSR, Enhanced library-level support.

Java SE 8-released on March 18, 2014, included some features that were planned for Java 7.

Java SE 9-released in 2016, better support for multi-gigabyte heaps, better native code integration, a different default garbage collector and a self-tuning JVM.

Java SE 10- released on March 20, 2018, with new features such as Local-variable type inference, Application class-data sharing, Garbage-collector interface, Thread-local handshakes.

Java SE 11- released on September 25, 2018, version is open for bug fixes.

Java SE 12-released on March 19, it includes the new features Micro benchmark Suite, Switch Expressions, JVM Constants API, Promptly Return Unused Committed Memory from G1.

Java SE 13-(2019)-Improved switch expressions, Better memory management

Java SE 14 (2020)-Switch expressions (final),Records (preview) → simple data classes, Helpful NullPointer, Exception messages, Pattern matching (preview)

Java SE 15 (2020)

Text Blocks (final)

Sealed classes (preview)

Hidden classes (for frameworks)

Removed Nashorn JavaScript engine

Java SE 16 (2021)

Records (final)

Pattern matching for instanceof (final)

Sealed classes (second preview)

Strong encapsulation of JDK internals

Java SE 17 (2021) – LTS(Long term support)

Sealed classes (final)

Enhanced pseudo-random generators

New macOS rendering pipeline

Very stable → widely used in industry

Java SE 22 (2024)

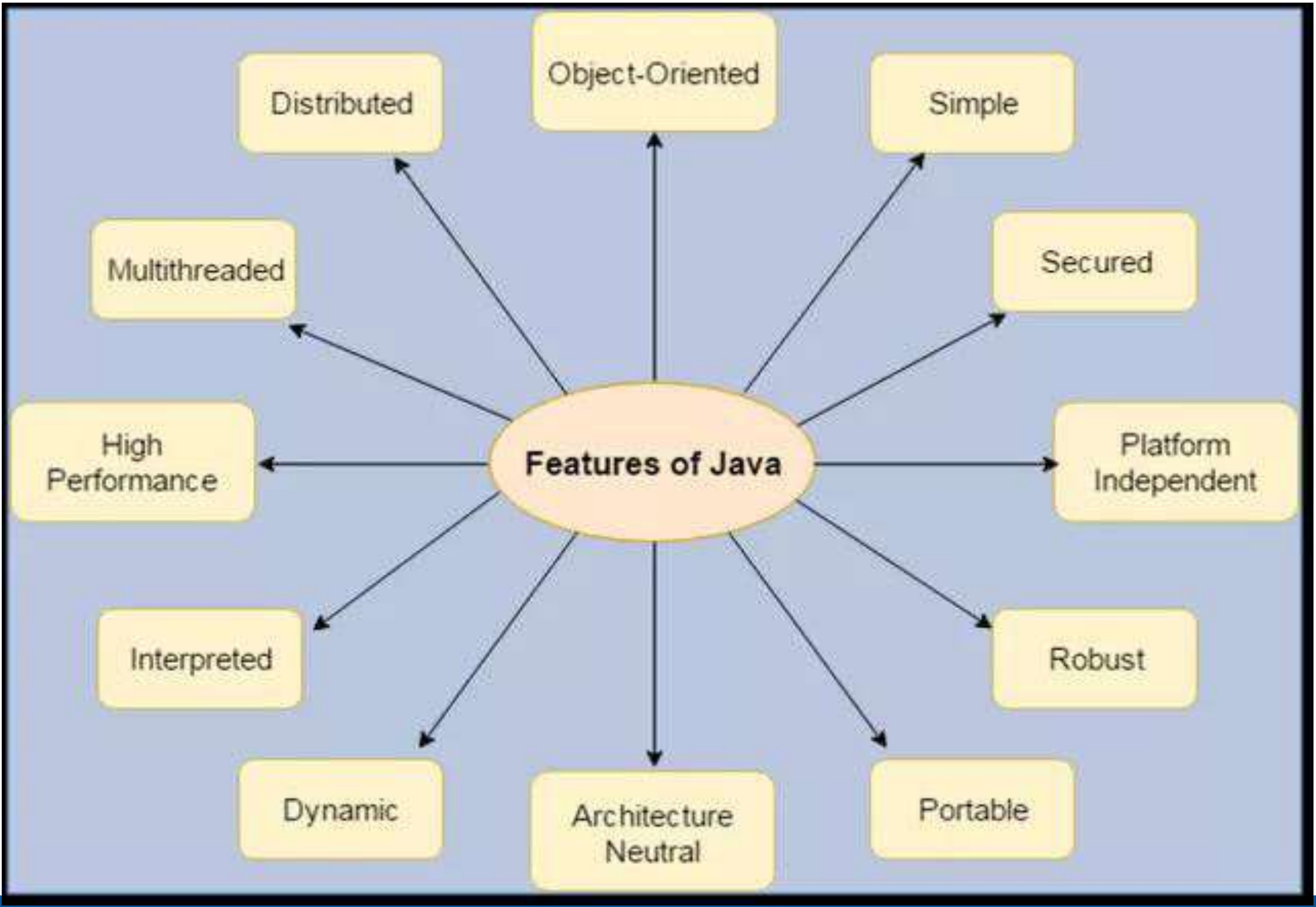
Foreign Function & Memory API (final)
Stream gatherers (preview)
Improved startup and performance

Java SE 23 (2024)

Language syntax refinements
Pattern matching enhancements
Performance and JVM tuning

Java SE 24 (2025) – LTS

Latest LTS version
Performance & security improvements
Better AI / cloud-ready JVM support
Recommended for future projects


Features of Java
Object-Oriented
Simple
Secured
Platform Independent
Robust
Portable
Architecture Neutral
Dynamic
Interpreted
High Performance
Multithreaded
Distributed

JAVA FEATURES

1. Platform-Independent and Portable

- Java programs can be run able at any platform (Any operating system).

- Java programs can be easily moved from one computer system to another, anywhere and anytime.

- Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs.

- Java ensures portability in two ways. First, Java compiler generates byte code instructions that can be implemented on any machine.

- Secondly, the size of the primitive data types are machine- independent.

Object-Oriented

▪Java is a true object-oriented language, Almost everything in Java is an abject.

▪ All program code and data reside within objects and classes.

▪Java comes with an extensive set of classes, arranged in packages that we can use in our programs by inheritance.

Robust and Secure

▪Java provides many safeguards to ensure reliable code.

▪It has strict compile time and run time checking for data types.

▪It is designed as a garbage-collected language because it has built in automatic memory management system that finds and delete objects thar are no longer being used.

- Java also incorporates the concept of exception handling which captures series of errors and eliminates any risk of crashing the system.

- Security becomes an important issue for a language that is used for programming on Internet.

- Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet.

- The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

Distributed

Java is designed as a distributed language for creating applications on networks.

▪It has the ability to share both data and programs.

▪Java applications can open and access remote objects on Internet as easily as they can do in a local system.

▪ This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Simple, Small and Familiar

▪Java is a small and simple language.

▪Many features o f C and C + + that are either redundant or sources of unreliable code are not part of Java.

▪For example, Java does not use pointers, pre-processor header files, goto statement and many others.

- It also eliminates operator overloading and multiple inheritance.

- Familiarity is another striking feature of Java.

- To make the language look familiar to the existing programmers, it was modelled on C and C + + languages.

- Java code looks like a C++ code. Java is a simplified version of C++ .

Multithreaded and Interactive

Multithreaded means handling multiple tasks simultaneously.

Java supports multithreaded programs.

This means that we need not wait for the application to finish one task before beginning another.

High Performance

Java performance is impressive for an interpreted language due to the intermediate byte code.

Java architecture is also designed to reduce overheads during runtime.

The incorporation of multithreading enhances the overall execution speed of Java programs.

Dynamic and Extensible

Java is a dynamic language.

Java is capable of dynamically linking in new class libraries, methods, and objects.

▪Java programs support functions written in other languages such as C and C++.

▪These functions are known as native methods.

▪Native methods arc linked dynamically at runtime.

Ease of Development

▪Java 2 Standard Edition (J2SE)5.0 supports features, such as Generics, Enhanced for Loop, Auto boxing or un boxing.

▪Type safe Enums, Varargs, Static import and Annotation.

▪These features reduce the work of the programmer by shifting the responsibility of creating the reusable code to the compiler.

Scalability and Performance

J2SE 5.0 assures a significant increase in scalability and performance by improving start up time and reducing the amount of memory used in Java 2 runtime environment.

Data sharing in the Hotspot Java Virtual Machine (JVM) improves the start up time by loading the core classes from the jar files into a shared archive.

Memory- utilization is reduced by sharing data in the shared archive among multiple JVM processes.

Desktop Client

J2SE 5.0 provides enhanced features to meet the requirements and challenges of the Java desktop users.

- It provides an improved Swing look and feel called Ocean.

- This feature is mainly used for developing graphics applications that require OpenGL hardware acceleration.

How Java Differs from C and C+ +

Java and C

- Java is an object-oriented language and has mechanism to define classes and objects.

- Java does not include the C unique statement keywords sizeof operator (display amount of memory occupied by the data type) and typedef.(used to create alternate name for existing data type)

- Java does not contain the data types struct and union.

Java does not define the type modifiers keywords auto, extern, register, signed, and unsigned.

- Java docs not support an explicit pointer type.

- Java does not have a pre-processor and therefore we cannot use #define, #include, and #ifdef statements.

- Java requires that the functions with no arguments must be declared with empty parenthesis and not with the void keyword as done in C.

- Java adds new operators such as instanceof (check object belong to specific class or not) and >>> (shifts the bits of a number to the right and fills the leftmost bits with 0) .

- Java adds labelled break and continue statements.

- Java adds many features required for object-oriented programming.

Java and C++

- Java is a true object-oriented language while C++ is basically C with object-oriented extension.

- Java does not support operator overloading.

- Java does not have template classes as in C++.(create type-independent classes, allowing the same class code to work with different data types without rewriting it).

- Java does not support multiple inheritance of classes. This is accomplished using a new feature called "interface".

- Java does not support global variables. Every variable and method is declared within a class and forms part of that class.
- Java does not use pointers.

- Java has replaced the destructor function with a finalize() function.

- There arc no header files in Java.

Java Environment

- Java environment includes a large number of development tools and hundreds of classes and methods.

- The development tools are part of the system known as Java Development Kit (JDK) and

- The classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

Java Development Kit

- The Java Development Kit comes with a collection of tools that are used for developing and running Java programs.

They include:

•appletvicwer -for viewing Java applets

•Javac- Java compiler which translate java soucrce code into byte code.

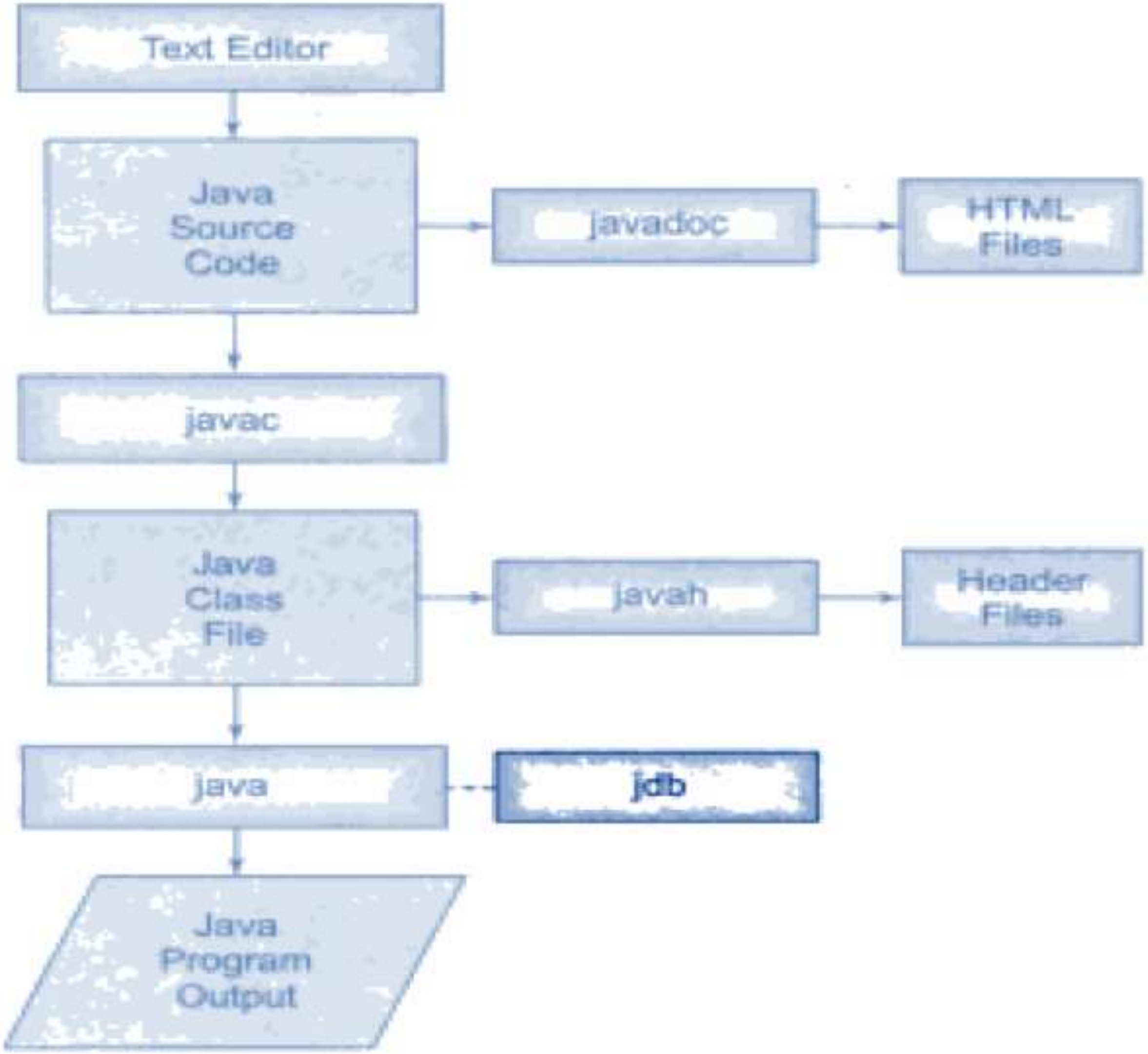
•java -Java interpreter which runs java applets and application by reading and interpreting byte code files.

•Javap -Java disassembler–To convert byte code files into program description.

•javah – produce header files for use with native methods.

•javadoc -for creating HTML documents

•Jdb-Java debugger which helps to find errors in our program



Process of building and running Java application programs

Application Programming Interface

The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages.

Most commonly used packages are:

•Language Support Package: A collection of classes and methods required for implementing basic features o f Java. (import java.lang.*) (String, thread, Exception, process)

•Utilities Package: A collection of classes to provide utility functions such as date and time functions.(import java.util.*)

•Input /Output Package: A collection of classes required for input/output manipulation.(import java.io.*)

•Networking Package: A collection of classes for communicating with other computers via Internet.(import jva.net.*)

•AWT Package: The Abstract Window Tool Kit package contains classes that implements platform-independent graphical user interface (import java.awt.*)

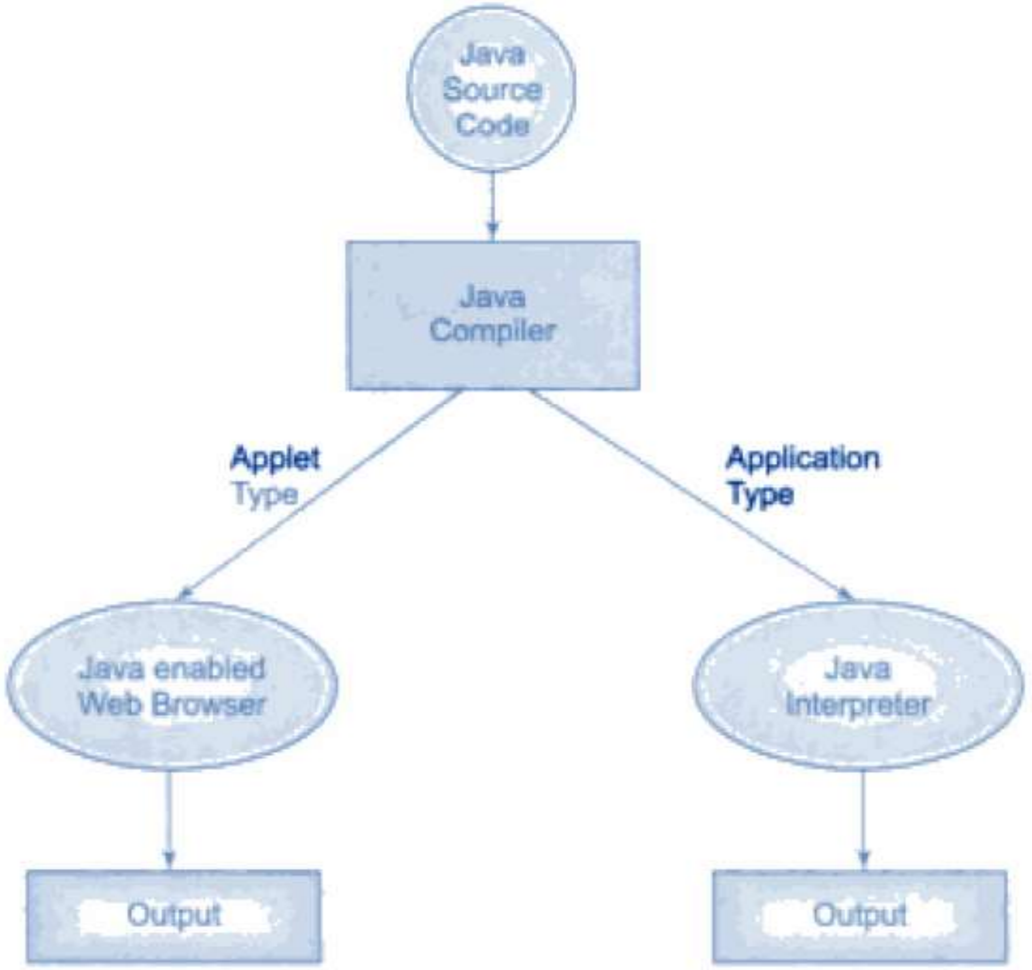
• Applet Package: This includes a set o f classes that allow s us to create Java applets.(import java.applet.*)

Overview of Java Language

Java is a general-purpose, object-oriented programming language. We can develop two types of Java programs:

• Stand-alone applications

• Web applets



Two ways of using Java

Simple Java Program

```
class SampleOne
{
public static void main (String args[ ])
{
System.out .println (“Java is better than C++”);
}
}
```

class is a keyword and declares that a new class

public static void main (String args[ ])-Defines a method named main. This is similar to the main( ) function in C ’C++.

Every Java application program must include the main() method This is the starting point for the interpreter to begin the execution of the program.

A Java application can have any number o f classes but *only one* of them must include a main method to initiate the execution.

Note that Java applets will not use the main method at all.

public: It is an access specifier that declares the main method is unprotected and accessible to all. This is similar to public access specifier in C++.

static: It allows the JVM to call the method without creating an object of the class.
Program execution starts before any object is created.

void: It states that the main does not return any value (But simply print some text to the screen)

Program to find square root of a number

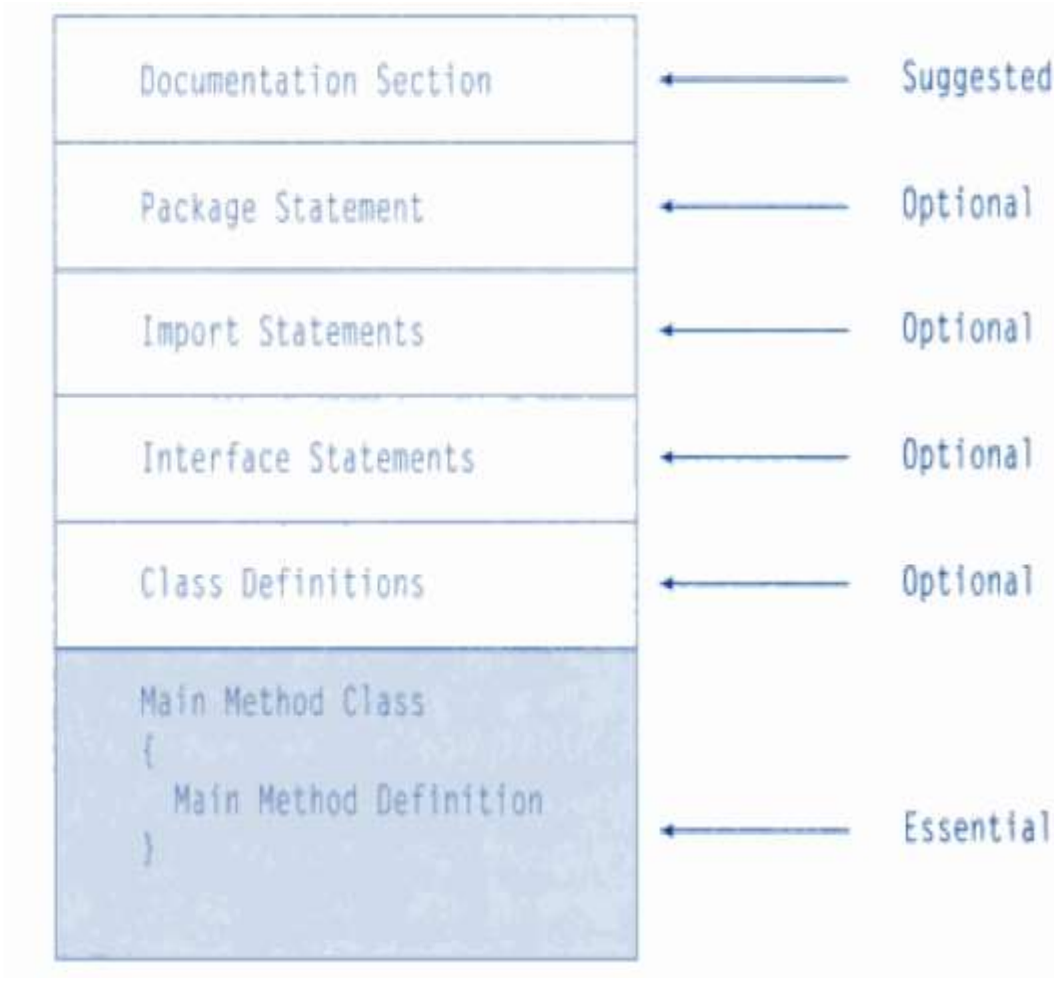
```
/*  More Java statements
This code computes square root */
import java.lang.Math;
class squareroot
{public static void main(String args[])
{
double x=5;    //declaration and initialization
double y;
y=Math.sqrt(x); //The math class is loaded from the package lang.
System.out.println("y=" +y);
}}
```

Comments–It is a non-executable statements used for documentation.

Java permits both the single line comments and multi-line comments.

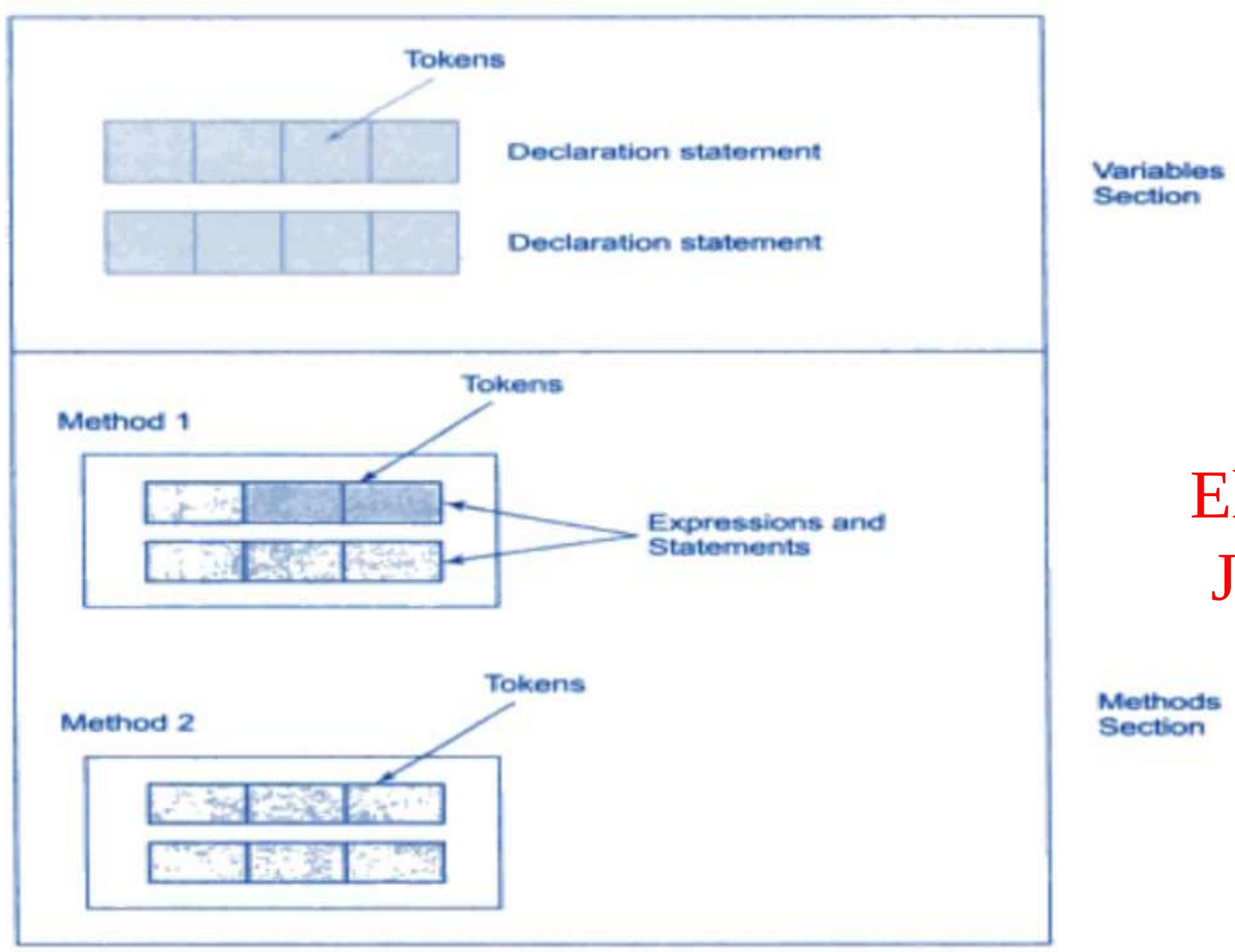
Single line comment start with // and multi line comment start with /* and end with */

Java Program Structure



Java Tokens

Smallest individual units in a program are known as tokens.



Elements of Java Class

In simple terms, a Java program is a collection of tokens, comments and white spaces.

Java language includes five types of tokens. They are:

1. Reserved Keywords
2. Identifiers
3. Literals
4. Operators
5. Separators

Java Character Set

▪The smallest units of Java language are the characters used to write Java tokens.

▪ These characters are defined by the Unicode character set

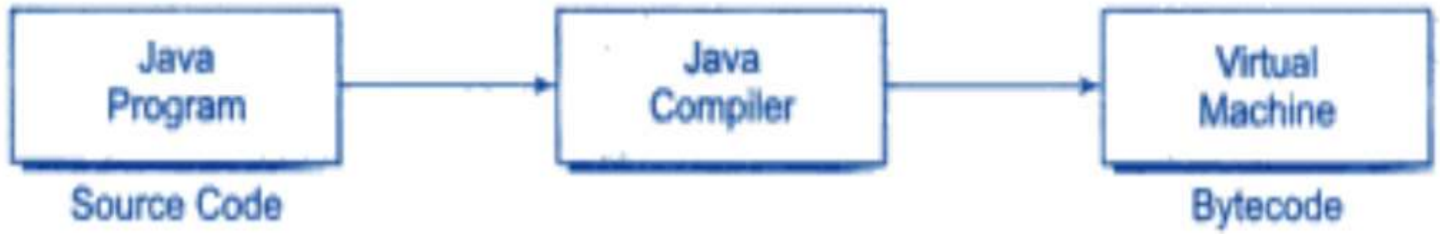
Java Virtual Machine

- Java compiler produce an intermediate code called byte code for machine that does not exist.

- This machine is called java virtual machine and it exists only inside computer memory.

- It is a simulated computer within the computer and does all major functions of real computer.

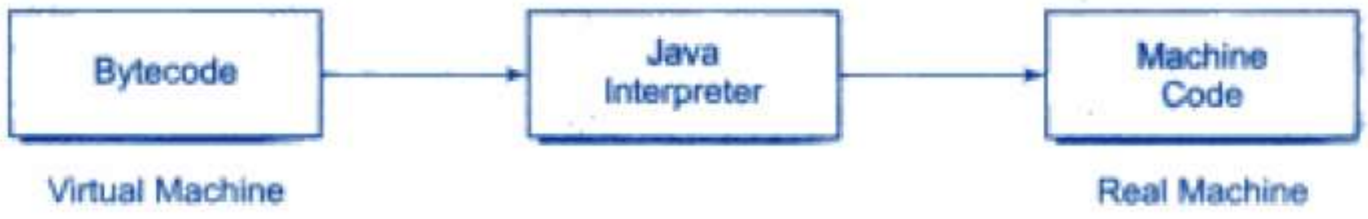
Process of compilation



The virtual machine code is not machine specific.

- The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine (refer fig)

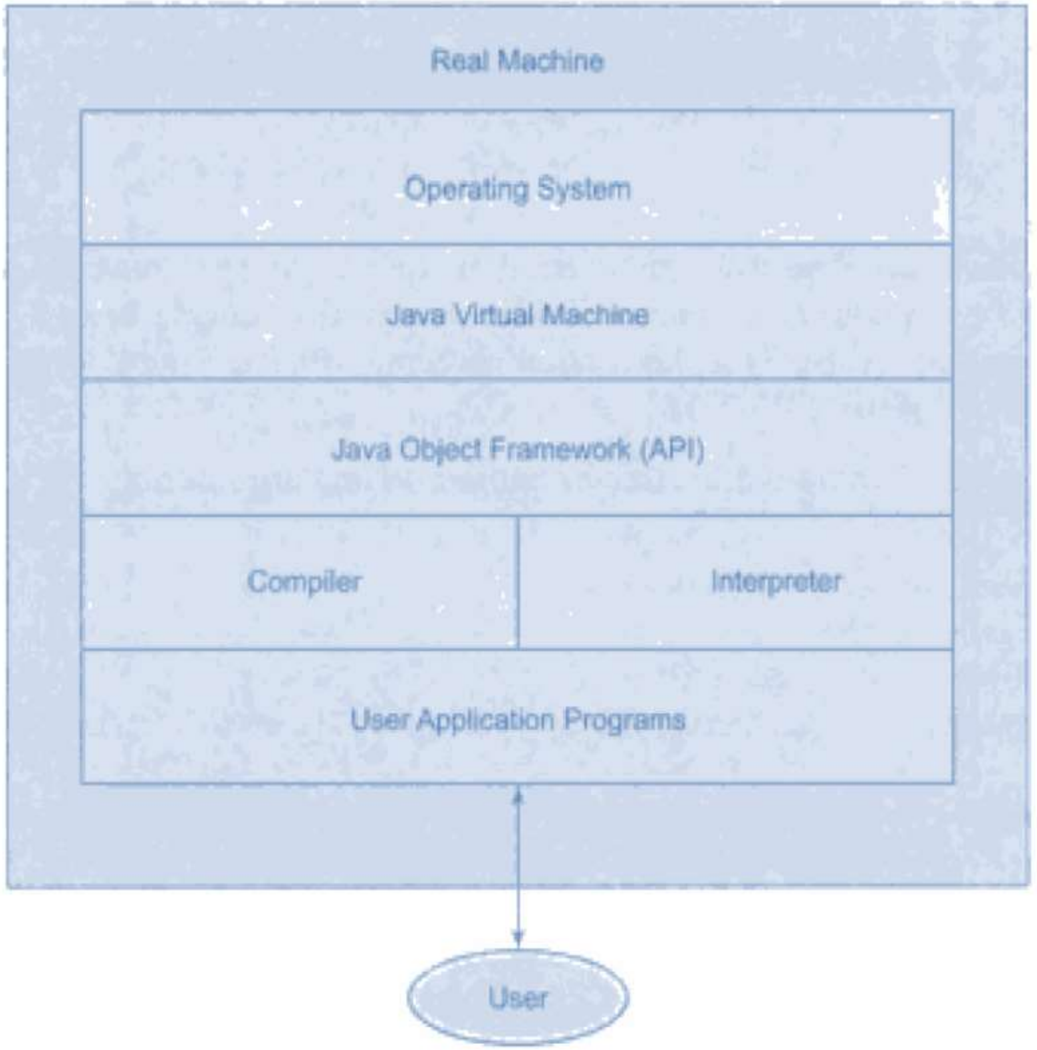
- Remember that the interpreter is different for different machines.



Process of converting byte code into machine code

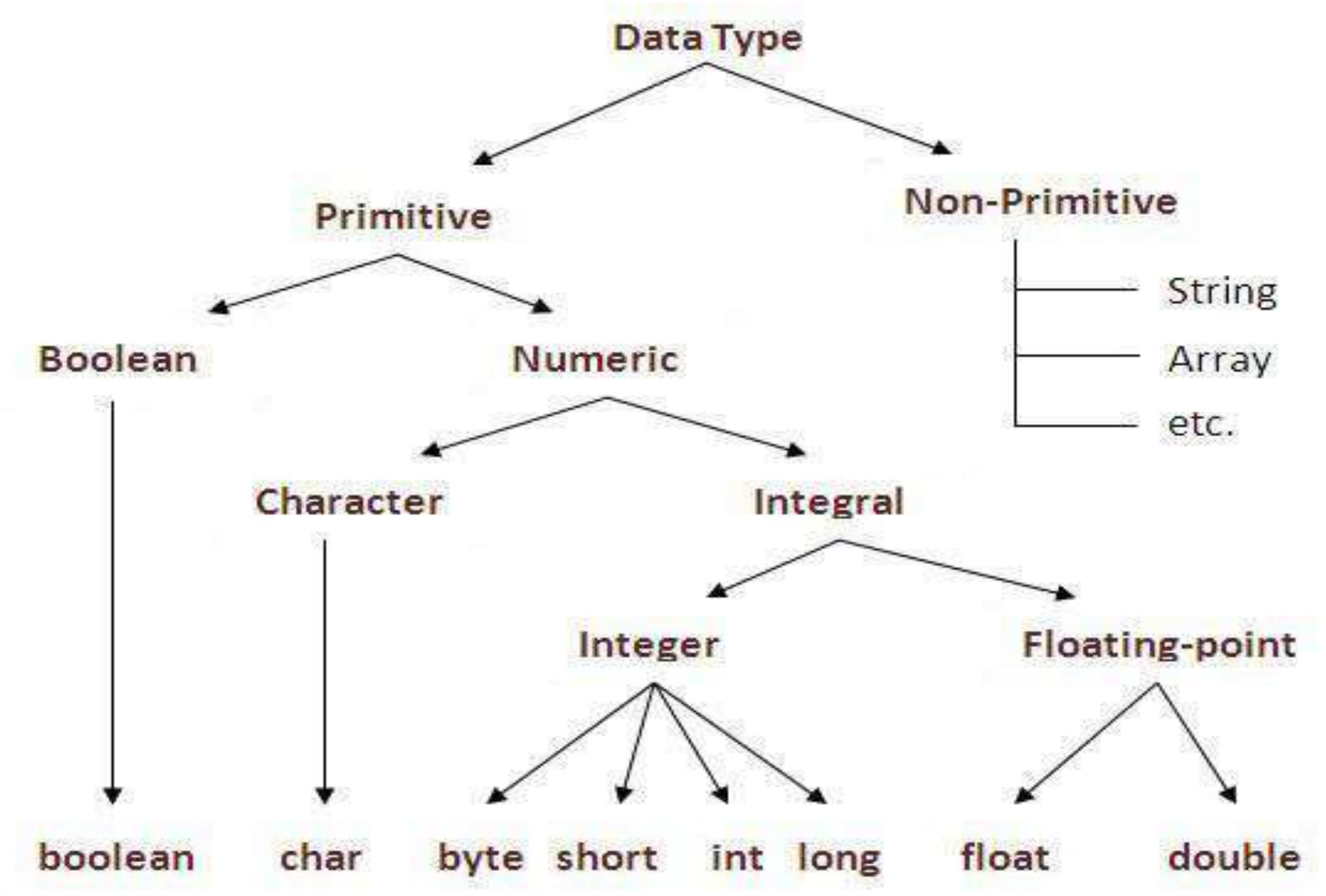
- The Java object framework (Java API) acts as the intermediary between the user programs and the virtual machine

- which in turn acts as the intermediary' between the operating system and the Java object framework.

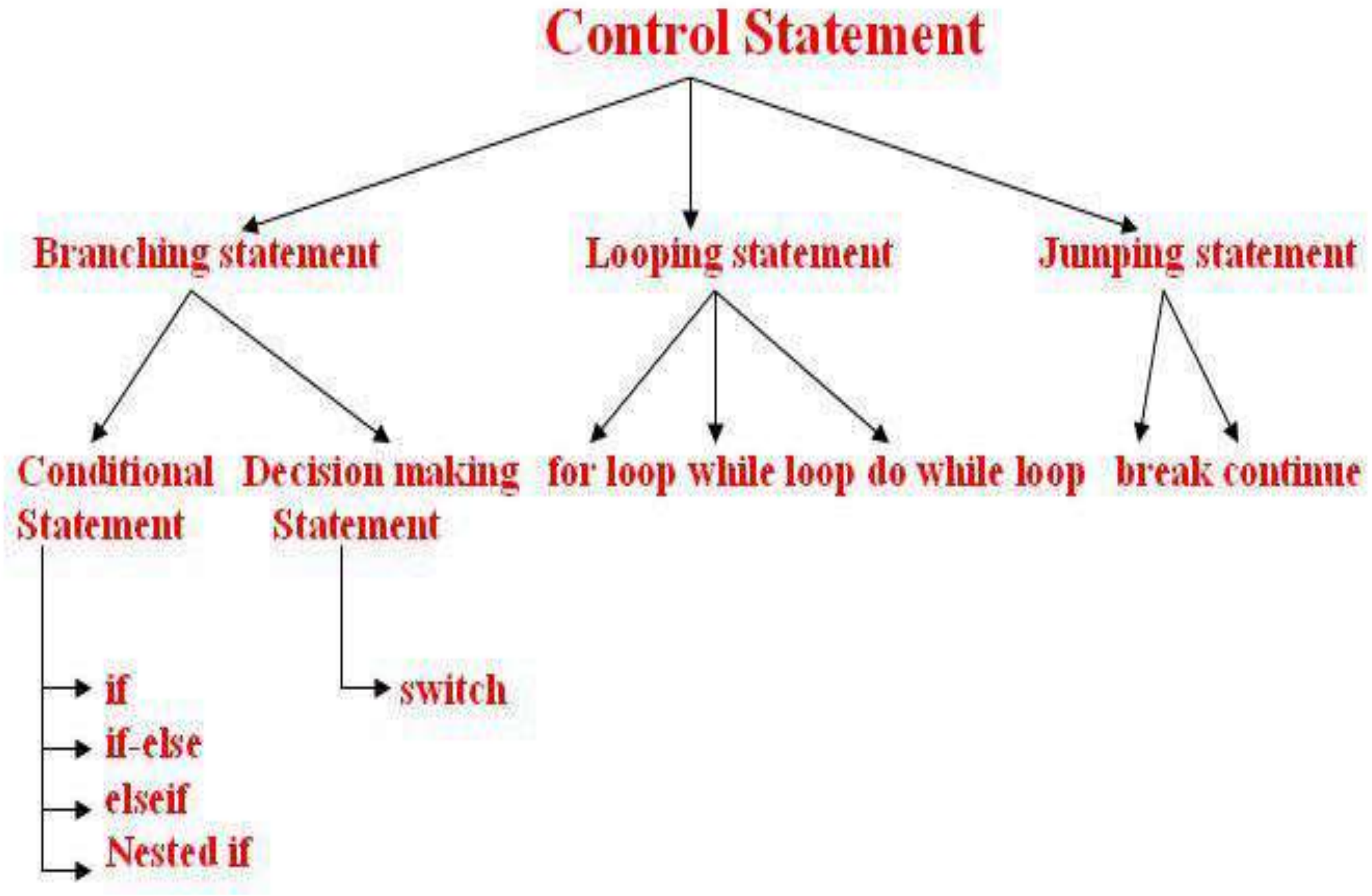


Layers of interactions for Java programs

Data types in Java



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte


Control Statement
Branching statement
Looping statement
Jumping statement
Conditional Statement
Decision making Statement
for loop
while loop
do while loop
break
continue
if
if-else
elseif
Nested if
switch

Classes, Objects and Methods

- Java is a true object-oriented language and therefore the underlying structure of all Java programs is classes.

- Classes create objects and objects use methods to communicate between them.

- Classes provide a convenient way for packing a group of logically related data items and functions.

- In Java, the data items are called fields and the functions are called methods.

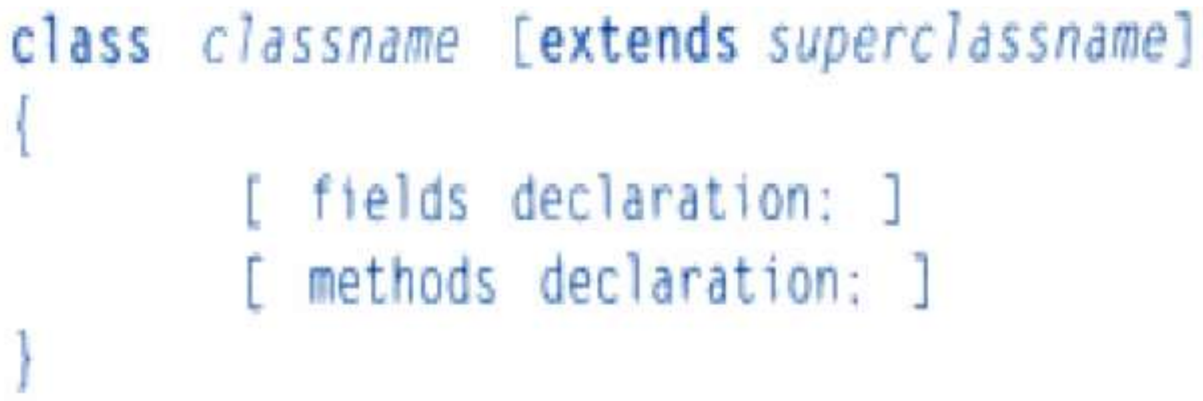
Defining a Class

A class is a user-defined data type with a template that serves to define its properties.

▪Once the class type has been defined, we can create “ variables” of that type using declarations that are similar to the basic type declarations.

▪In Java, these variables are termed as instances of classes, which are the actual objects.

The basic form of a class definition is



```
class classname [extends superclassname]
{
        [ fields declaration; ]
        [ methods declaration; ]
}
```

In C++ there is a semicolon after closing brace

Fields Declaration

- Data is encapsulated in a class by placing data fields inside the body of the class definition.

- These variables are called instance variables because they are created whenever an object of the class is instantiated.

- We can declare the instance variables exactly the same way as we declare local variables.

```
class Rectangle
{
int length;
int width;
}
```

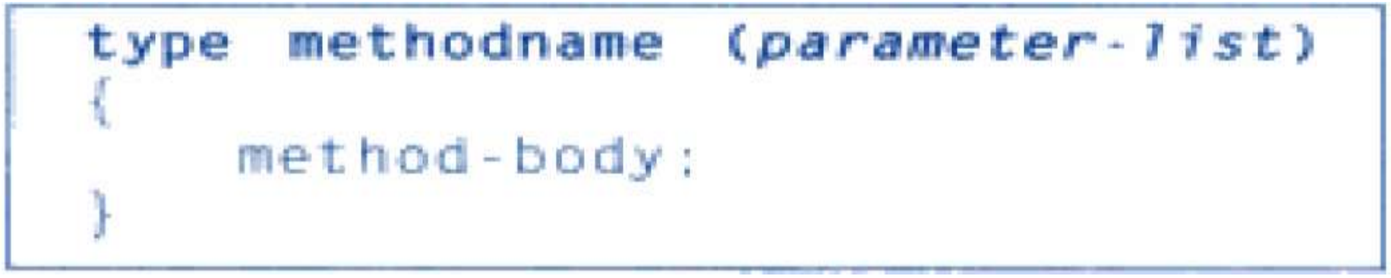
*Remember these variables are only declared and therefore no storage space has been created in the memory.

Methods Declaration

▪Methods are necessary- for manipulating the data contained in the class.

▪Methods are declared inside the body of the class but immediately after the declaration of instance variables.

▪The general form o f a method declaration is



```
type methodname (parameter-list)
{
        method-body;
}
```

Method declarations have four basic parts:

1.The name of the method *(method name)*

2.The type of the value the method returns (type)

3.A list of parameters (parameter-list)

4.The body o f the method (Method body)

```
class Access
{
int  x ;
void  method1()
{
int  y ;
x = 10 ; / / legal
y =x ; legal
}
void method2()
 {
int z ;
x = 5 ; // legal
z=10 : //legal
y =1; / / illegal
}
}// A method cannot access the variables which is declared in other methods
```

Creating Objects

▪An object in Java is essentially a block of memory that contains space to store all the instance variables.

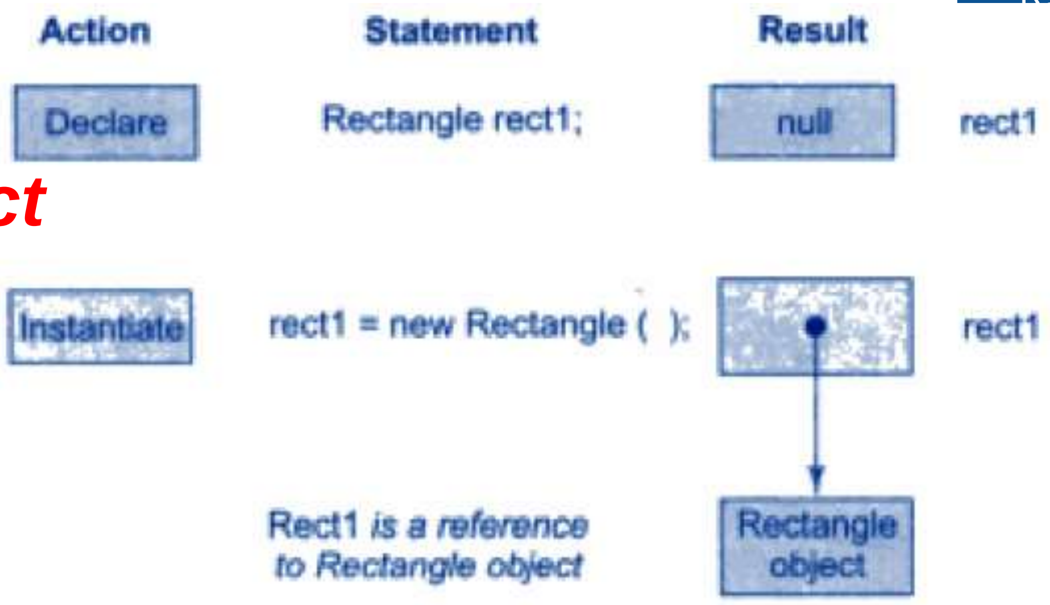
▪Creating an object is also referred to as *instantiating an object.*

```
Rectangle rect1;                // declare the object  Or object reference
rect1 = new Rectangle( ); // instantiate the object  Or object
```

▪The first statement declares a variable to hold the object reference and the second one actually assigns the object reference to the variable.

▪The variable rect1 is now an object of the Rectangle class.

Creating object references



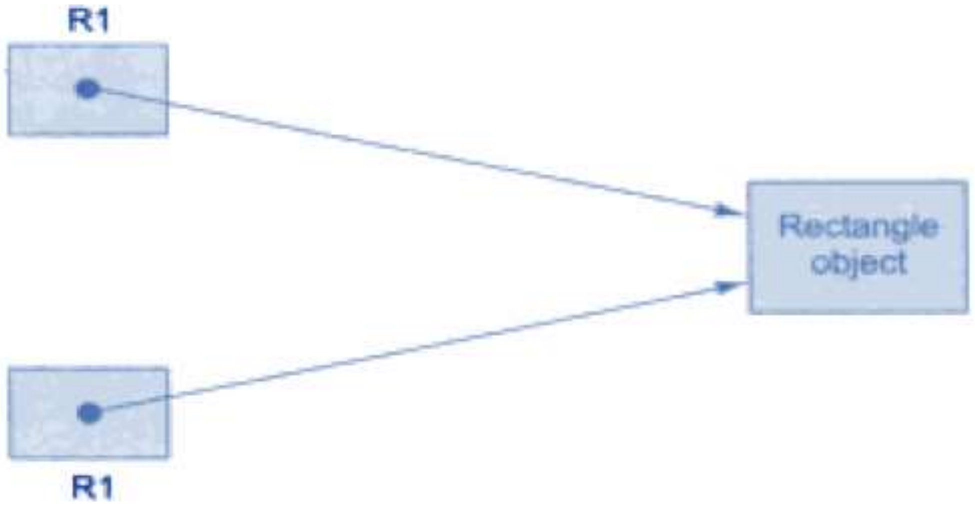
Both statements, can be combined into one as shown below:
Rectangle rect1= new Rectangle( )

The method Rectangle() is the default constructor of the class.

We can create any number of objects of the class Rectangle
Rectangle rect1 =new Rectangle();
Rectangle rect2 =new Rectangle(); and so on

Assigning one object reference variable to another

```
Rectangle R1 =new Rectangle ( );
 Rectangle R2 = R1;
```



Both R1 and R2 refer to the same object

Accessing Class Members

❖Now we have created objects, each containing its own set of variables.

❖we should assign values to these variables in order to use them in our program.

▪To do this, we must use the concerned object and the *dot operator as shown below:*

Objectname.variablename=value;
Objectname.methodname(parameter_list);

▪Here object name is the name o f the object.

▪variable name is the name of the instance variable inside the object that we wish to access.

▪Method name is the method that we wish to call, and parameter-list is a comma separated list.

rect1.length=15;
rect1.width=10;
rect2.length=20; rect2.width=12; Exprogram to access fields and methods

Constructors

❖All objects that are created must be given initial values.

❖We have done this earlier using two approaches.

❖Using dot operator to access the instance variables and then assigns values to them individually.(Rect1.length=20)

❖The second approach takes the help of a method like getdata to initialize each object individually using statements like,

Rect1.getdata(15,10);

❖Java supports a special type of method, called a constructor, that enables an object to initialize itself when it is created.

❖Constructors have the same name as the class itself.

Constructor in java is a *special type of method* that is used to initialize the object.

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

There are basically two rules defined for the constructor.
1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:
1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

```
<class name>()
{  }
```

Example of default constructor

```
class Bike1{
Bike1()
{System.out.println("Bike is created");
}
public static void main(String args[]){
Bike1 b=new Bike1();
} }
```

Output: Bike is created

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student{
int id;
String name;
Student(int i,String n){
id = i;
name = n;
}
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
Student s1 = new Student(111,"Karan");
Student s2 = new Student(222,"Aryan");
s1.display();
s2.display();
} }
```

Constructor Overloading in Java

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.
- The compiler differentiates these constructors by means of the number of parameters and their type.

Example program

Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor

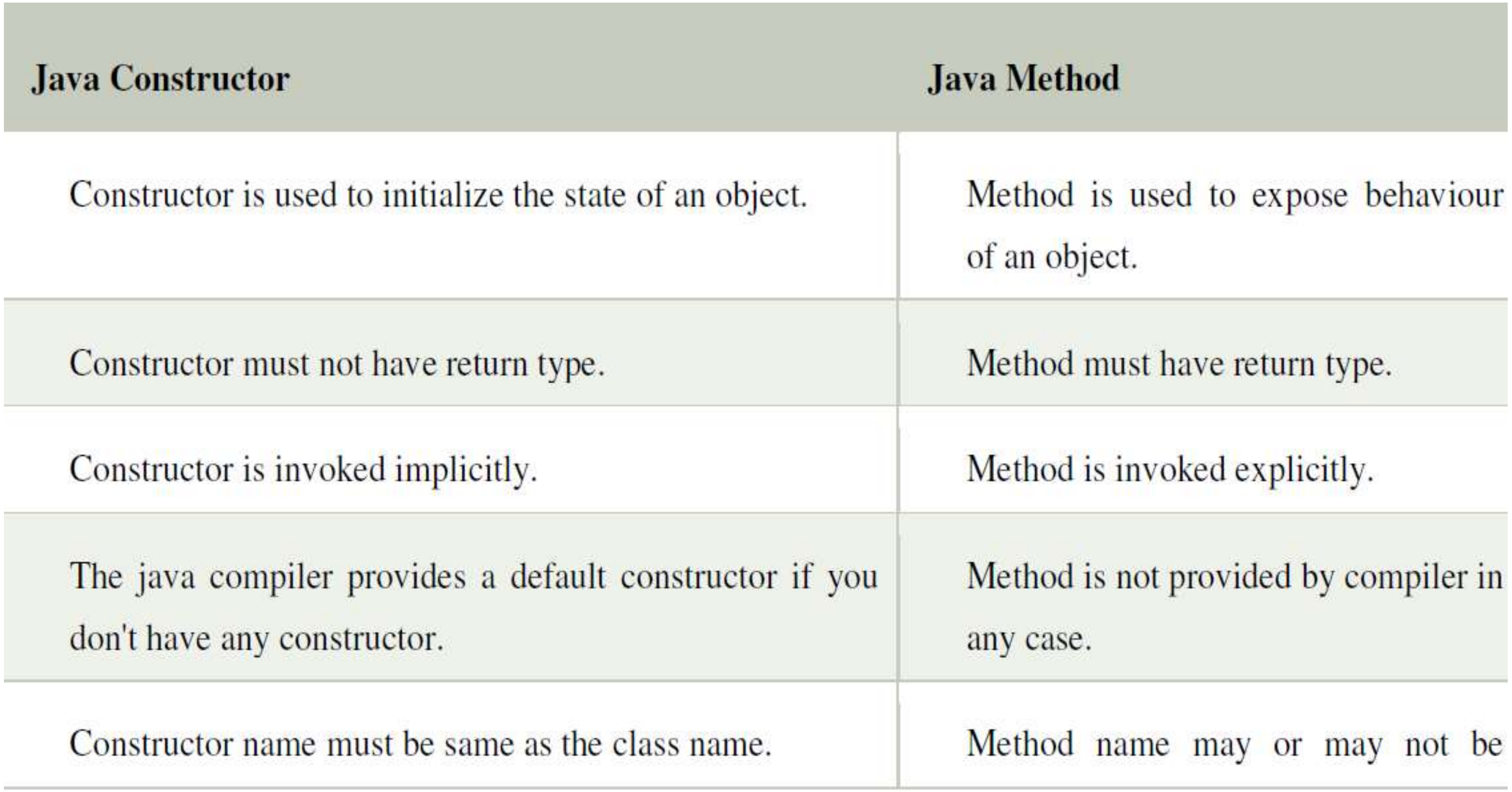
- By assigning the values of one object into another

- By clone() method of Object class

- clone() creates a copy of an object

- Performs shallow copy by default-Primitive fields → values are copied
  Object references → references are copied (not the actual objects)
  So both original and cloned objects may share referenced objects.

- Requires Cloneable interface   program



Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be

this keyword

1. this.variable=Differentiate instance variables from local variables
2. this()=Calls current class constructor
3. this.method()=Calls current object method
4. this as arguments= Passes current object
5. return this=Returns current object

1. To differentiate instance variables from local variables
When method/constructor parameters have the same name as instance variables, this keyword will avoid confusion.

```
class Student
 {
 int id;    String name; // instance variable
//instance variables are declared inside the class but outside the method
// Accessible to all methods of the object
//Default values are provided
// Access modifiers allowed
//initialization is optional
Student(int id, String name)
 {
// Local variables are declared inside the method, constructor or block
//Accessibility is limited to method or block where it is declared
//No default values are provided
// Access modifiers are not allowed//initialization is mandatory
 this.id = id;
this.name = name;    }}
 //Without this, Java would refer to the local variables only.
```

2. To refer to the current class object

```
class Testthis1 {
 void show() {        System.out.println(this);    }
public static void main(String[] args) {
Testthis1 t = new Testthis1();
System.out.println(t);
t.show();    }}
//Both print the same reference, proving this refers to the current object.
```

3. To call another constructor in the same class

```
//This  is used to invoke another constructor.
class Demo {
 int x, y;
Demo() {        this(10, 20);   // calls parameterized constructor    }
Demo(int x, int y) {        this.x = x;        this.y = y;    }}//Must be the first statement in the constructor.
```

4. To pass current object as a method argument

```
class Example {
void display(Example obj) {
 System.out.println("Method called");    }
 void call() {
 display(this);  // passing current object    }}
```

5. To return the current object

```
//Useful in method chaining.
class Box {
 int length;
Box setLength(int length)
{
this.length = length;
return this;
}}
```

Method Overloading

- ❖In Java, it is possible to create methods that have the same name, but different parameter lists and different definitions.

- ❖This is called method overloading.

- ❖Method overloading is used when objects are required to perform similar tasks but using different input parameters.

- ❖When we call a method in an object, Java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute.

- ❖This process is known as polymorphism.
- ❖ program

In java, method overloading is not possible by changing the return type of the method only because of ambiguity.

```
class Adder
{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading
{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Output

```
Compile Time Error: method add(int,int) is already defined in class Adder
```

Overloading main method in java

We can overload main method in java.

We can have any number of main methods in a class. But JVM calls main() method which receives string array as arguments only.

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

public static void main(){System.out.println("main without args");}
}
```

Output: main with String[]

Static Members

❖We have seen that a class basically contains two sections. One declares variables and the other declares methods.

❖These variables and methods are called instance variables and instance methods.

❖This is because every time the class is instantiated, a new copy of each of them is created.

❖They are accessed using the objects (with dot operator).

❖we want to define a member that is common to all the objects and accessed without using a particular object then we use static.
❖The member belongs to the class as a whole rather than the objects created from the class.

```
static int count;
static  int  max(int x, int y ) ;
```

- The static variables and static methods are often referred to as class variables and class methods in order to distinguish them from instance variables and instance methods.

- Like static variables, static methods can be called without using the objects.

- A static method belongs to the class rather than object of a class

- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it. program

A method can he called by using only its name by another method of the same class. This is known as ***nesting of methods***

Nesting of Methods

```
class Nesting
{
int m,n;
Nesting( int x, int  y) / / constructor method
{
m=x; n=y;
}
int largest()
{
i f (m >= n) re tu rn (m );
e l se r e tu r n ( n ) ;
}
void display()
{
int large = largest(); / / calling a method
System.out.println("Largest value -=" + large);
}}
class NestingTest
{public static void main(String args[]){Nesting nest=new Nesting(50,40);nest.display();}}
```

Final Variables and Methods

- All methods and variables can be overridden by default in subclasses.

- If we wish to prevent the subclasses from overriding the members of the super class, we can declare them as final using the keyword final as a modifier

Example

```
final int SIZE=100;  // it cannot be redefined in sub class and value never
//be changed

final void showstatus( ) { ......................}
```

- Making a method final ensures that the functionality defined in this method will never be altered in any way.

Similarly, the value o f a final variable can never be changed.

Final variables, behave like class variables and they do not take any space on individual objects of the class.
program for final variable
program for final method

Final Class

❖Sometimes we may like to prevent a class being further subclasses for security reasons.

❖ A class that cannot be sub classed is called a final class.

❖This is achieved in Java using the keyword final as follows:

final class Aclass { .... }
final class Bclass extends Someclass { ....}

Declaring a class final prevents any unwanted extensions to the class.

It also allows the compiler to perform some optimizations when a method of a final class is invoked. program

Finalizer Methods

❖Constructor method is used to initialize an object when it is declared.

❖This process is know n as initialization. Similarly, Java supports a concept called finalization. which is just opposite to initialization.

- ❖Java run-time is an automatic garbage collecting system.
- ❖It automatically frees up the memory resources used by the objects.
- ❖But objects may hold other non-object resources such as file descriptors or window system fonts.
- ❖The garbage collector cannot free these resources In order to free these resources we must use a *finalizer method.*
- ❖*This is similar to destructors in C++.*
- ❖The finalizer method is simply finalize and can be added to any class.

❖Java calls that method whenever it is about to reclaim the space lot that object.

❖The finalize method should explicitly define the tasks to be performed.

Abstract Methods and Classes

❖By making a method final we ensure that the method is not redefined in a subclass.

❖That is, the method can never be sub classed. Java allows us to do something that is exactly opposite to this.

❖That is we can indicate that a method must always be redefined in a subclass, thus making overriding compulsory.

❖This is done using the modifier keyword abstract m the method definition.

Example:

```
abstract class Shape
{
abstract void draw( );
}
```

When a class contains one or more abstract methods, it should also be declared abstract as shown in the example above.

While using abstract classes, we must satisfy' the following conditions:

We cannot use abstract classes to instantiate objects directly. For example Shape s = new Shape()

is illegal because shape is an abstract class.

- The abstract methods of an abstract class must be defined in its subclass.

- We cannot declare abstract constructors or abstract static methods.

program for abstract class and method

Strings

Strings represent a sequence of characters..

The easiest way to represent a sequence o f characters in Java is by using a **character array.**

Ex:

char charArray[ ]=new Char[4];

Char Array[0] = ’J ’;

Char Array[1] = ‘a';

Char Array[2]=‘v’;

Char Array[3] = ’a’;

Character array in not good enough to support the range of operations.

Hence, we go for strings. Java, strings are class objects and implemented using three classes, namely String, StringBuffer and StringBuilder class

- String class-(immutable(content cannot be changed), Strings are fixed length.

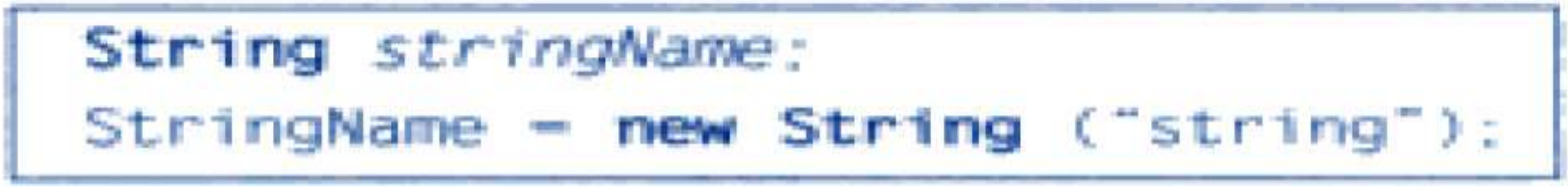
- StringBuffer class- (mutable (content cannot be changed),Flexible length,modified frequently, thread-safe)

- StringBuilder class- (mutable, not thread-safe)

- Package: java.lang

- Why StringBuffer?
- String → inefficient for frequent modifications
- StringBuilder → faster but not thread-safe
- StringBuffer →Efficient for frequent modification and safe for multithreaded environments

string is not a character array and is not N U LL terminated. Strings may be declared and created as follows:



```
String stringName;
StringName = new String ("string");
```

Example:

String firstName;

firstName = new String("Anil");

These two statements may be combined as follows:

String firstName=new String ("Anil");

Like arrays, it is possible to get the length of string using the length method of the String class.

int m =firstName.length();

Java strings can be concatenated using the + operator
String fullname =namel + name2;
String city=“new” + “Delhi”;

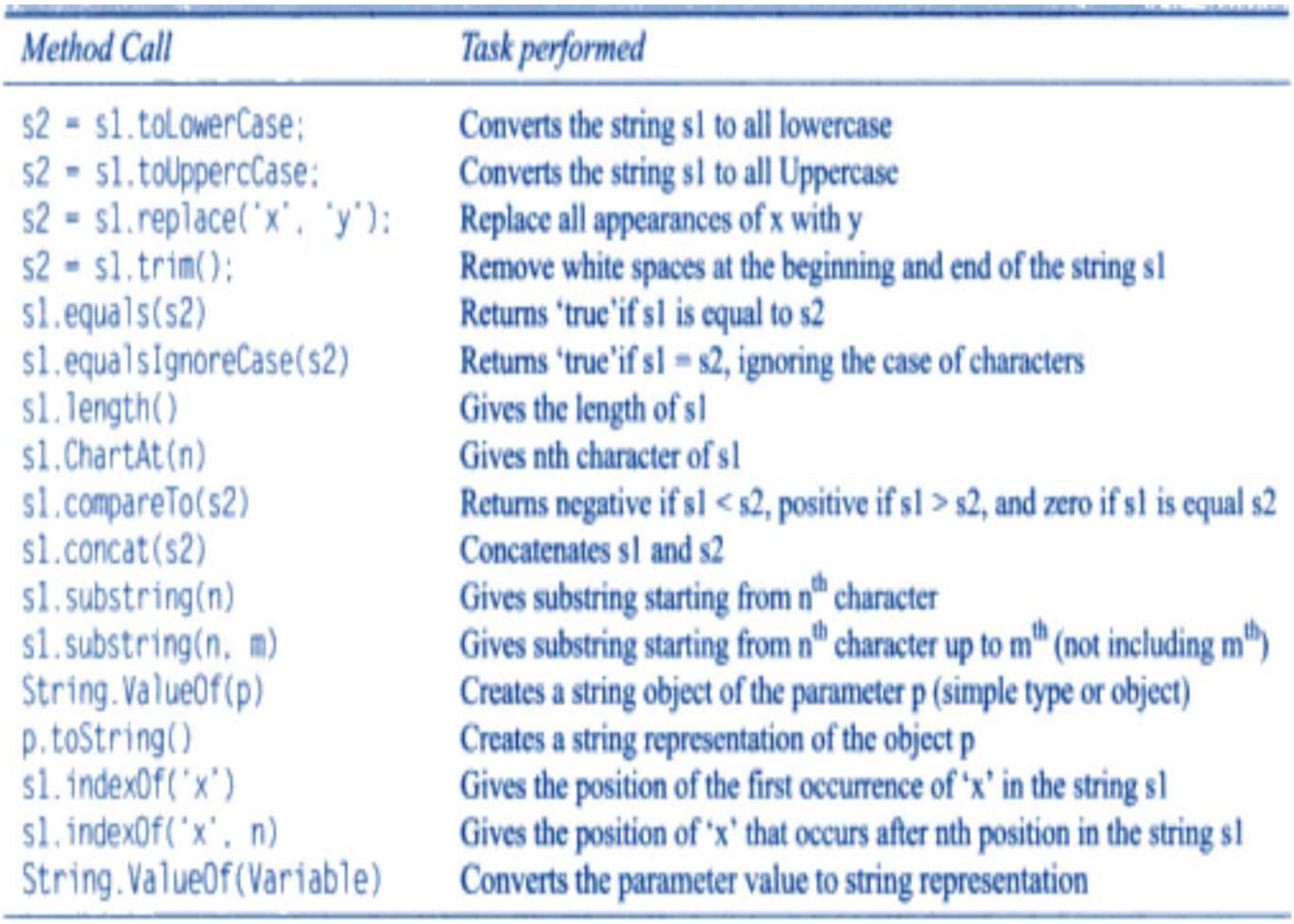
String Arrays

We can also create and use arrays that contain strings.
String itemArray[]=new String[3];

will create an item Array of size 3 to hold three string constants.

The string class defines a number of methods that allows us to accomplish variety of string manipulation task.

Most commonly used string methods

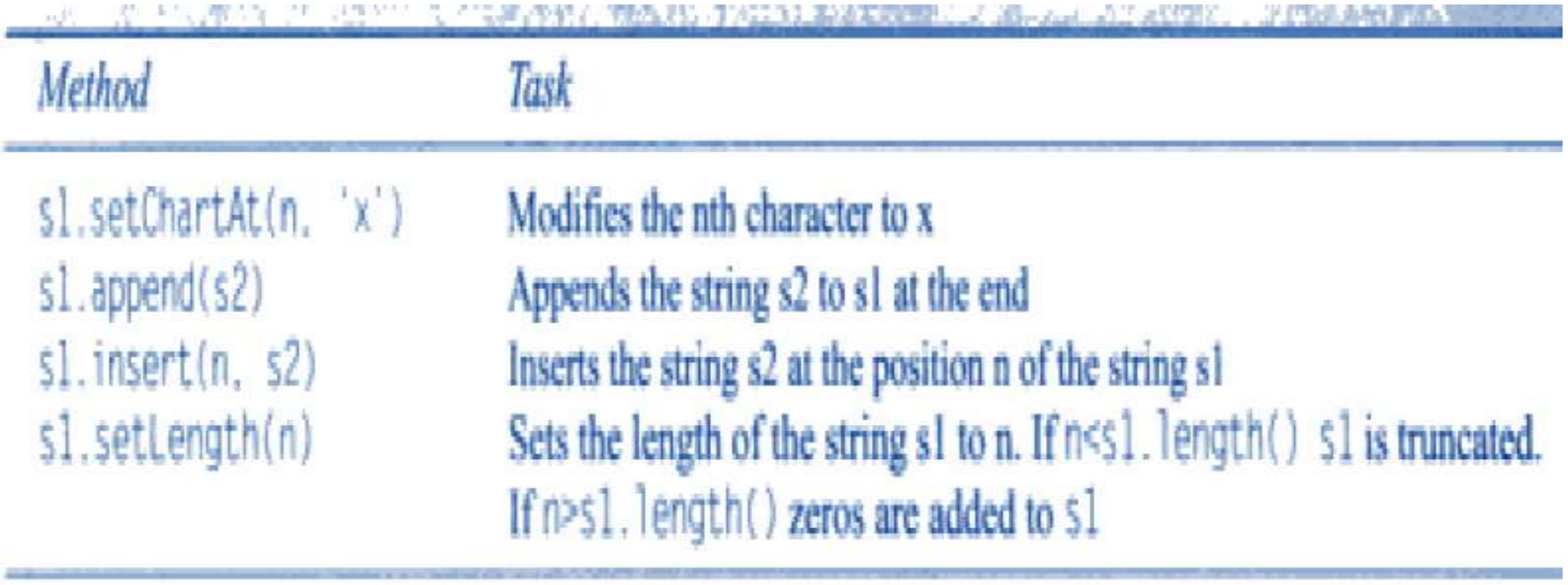


Method Call	*Task performed*
`s2 = s1.toLowerCase;`	Converts the string s1 to all lowercase
`s2 = s1.toUppercCase;`	Converts the string s1 to all Uppercase
`s2 = s1.replace('x', 'y');`	Replace all appearances of x with y
`s2 = s1.trim();`	Remove white spaces at the beginning and end of the string s1
`s1.equals(s2)`	Returns 'true'if s1 is equal to s2
`s1.equalsIgnoreCase(s2)`	Returns 'true'if s1 = s2, ignoring the case of characters
`s1.length()`	Gives the length of s1
`s1.ChartAt(n)`	Gives nth character of s1
`s1.compareTo(s2)`	Returns negative if s1 < s2, positive if s1 > s2, and zero if s1 is equal s2
`s1.concat(s2)`	Concatenates s1 and s2
`s1.substring(n)`	Gives substring starting from nth character
`s1.substring(n, m)`	Gives substring starting from nth character up to mth (not including mth)
`String.ValueOf(p)`	Creates a string object of the parameter p (simple type or object)
`p.toString()`	Creates a string representation of the object p
`s1.indexOf('x')`	Gives the position of the first occurrence of 'x' in the string s1
`s1.indexOf('x', n)`	Gives the position of 'x' that occurs after nth position in the string s1
`String.ValueOf(Variable)`	Converts the parameter value to string representation

StringBuffer Class

String creates strings of fixed length, StringBuffer creates strings of flexible length that can be modified in terms of both length and content.

Frequently used in String manipulation methods



Method	Task
s1.setChartAt(n, 'x')	Modifies the nth character to x
s1.append(s2)	Appends the string s2 to s1 at the end
s1.insert(n, s2)	Inserts the string s2 at the position n of the string s1
s1.setLength(n)	Sets the length of the string s1 to n. If n<s1.length() s1 is truncated. If n>s1.length() zeros are added to s1

program for string manipulation

Methods for Deleting Content

delete()

delete(int start, int end)

deleteCharAt()

Deletes a character at a specific index.

deleteCharAt(int index)

capacity()

Returns current capacity-default capacity is 16 (16+string length).

capacity()

New Capacity = (Old Capacity × 2) + 2

getChars()

Copies characters into an array.

getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

Command Line Arguments

Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.

EX:

public static void main(String args[])

As pointed out earlier, args is declared as an array of strings (known as string objects).

Any arguments provided in the command line (at the time of execution) are passed to the array args as its elements.

We can simply access the array elements and use them in the program as we wish.

For example, consider the command line

```
Java Test BASIC FORTRAN C++ Java
```

This command line contains four arguments.

These are assigned to the array args as follows:

BASIC -------- >>args[0]

FORTRAN -------- >> args [1]

C+ + ---- >> args [2]

Java -------- >> args [3]

Example program

Java Garbage Collection

In java, garbage means unreferenced objects.

- Garbage Collection is process of reclaiming the runtime unused memory automatically.
- In other words, it is a way to destroy the unused objects
- To do so, we were using free() function in C language and delete() in C++.
- But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

gc() method

- The gc() method is used to invoke the garbage collector to perform cleanup processing.

- The gc() is found in System and Runtime classes.

- **public static void** gc(){}

```
//Simple Example of garbage collection in java
public class TestGarbage1// public class can be accessed from any other class
//or packages
{
public void finalize()
{
System.out.println("object is garbage collected");
}
public static void main(String args[]){
TestGarbage1 s1=new TestGarbage1();
TestGarbage1 s2=new TestGarbage1();
s1=null;
s2=null;
System.gc();
} }
```

- object is garbage collected
- object is garbage collected

Access Control
Access Modifiers in java

- We have also seen that the variables and methods of a class are visible everywhere in the program.
- However, it may be necessary in some situations to restrict the access to certain variables and methods from outside the class.
- For example, we may not like the objects of a class directly alter the value of a variable or access a method.
- We can achieve this in Java by applying visibility modifiers to the instance variables and methods.
- The visibility modifiers are also known as access modifiers.

- There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

- The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

- There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

private access modifier

- The private access modifier is accessible only within class.

Simple example of private access modifier

- In this example, we have created two classes A and Simple. A class contains private data member and private method.

- We are accessing these private members from outside the class, so there is compile time error.

```
class A{
private int data=40;
private void msg()
{
System.out.println("Hello java");
} }
public class Simple
{
public static void main(String args[])
{
A obj=new A();
System.out.println(obj.data);//Compile Time Error
obj.msg();//Compile Time Error
} }
```

default access modifier

- If you don't use any modifier, it is treated as **default** bydefault.
- The default modifier is accessible only within package.
- Example of default access modifier
- In this example, we have created two packages pack and mypack.
- We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.
- Program

protected access modifier

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.

- The protected access modifier can be applied on the data member, method and constructor.

- It can’t be applied on the class.

- Example of protected access modifier

public access modifier

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
//Example of public access modifier
//save by A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");} }
//save by B.java
package mypack;
import pack.*;
class B{
public static void main(String args[]){
A obj = new A();
obj.msg();
} }
Output:Hello
```

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Nested Classes

- Java allows a class to be defined **inside another class.** Such classes are called **Nested Classes**.

Benefits of nested classes

- Logical grouping of classes
- Improved encapsulation
- Better readability & maintainability
- Can access private members of outer class

Types of Nested Classes in Java

- Static Nested Classes

- Inner Classes (Non-static Nested Classes)

Static Nested Class

- A class declared **static inside another class.**

```
class Outer
{
 static class StaticNested
{
void display()
{   System.out.println("Static Nested Class");
 } }}
Outer.StaticNested obj = new Outer.StaticNested();obj.display();
```

Characteristics

- Can access **only static members** of outer class
- Does **not** require outer class object
- Behaves like a regular class, but logically grouped

Inner Classes (Non-Static Nested Classes)

An **inner class** is a non-static class defined inside another class.

Types of Inner Classes

- Member Inner Class
- Local Inner Class
- Anonymous Inner Class

Member Inner Class

- A class defined **inside another class without static keyword**.

```
class Outer {
 private int x = 10;
class Inner {        void show() {
  System.out.println(x);        }    }}
```

Characteristics

- Can access **all members (including private)** of outer class
- Requires **outer class object** for instantiation

Object Creation

```
Outer obj = new Outer();
Outer.Inner in = obj.new Inner();
in.show();
```

Advantages

- Strong encapsulation

- Useful when inner class is tightly coupled with outer class

Local Inner Class

- A class defined inside a method, constructor, or block.

```
class Outer {
 void display()
 {
class LocalInner
 {
void msg()
{
System.out.println("Local Inner Class");
  }       }
LocalInner li = new LocalInner();
li.msg();
}}
```

Characteristics

- Scope limited to the method
- Can access **final or effectively final variables**
- Cannot use access modifiers

Anonymous Inner Class

- An inner class **without a name**, used to implement:
- Interfaces
- Abstract classes
- Override methods

```
interface Test {
  void show();}
class Demo {    public static void main(String[] args) {
 Test t = new Test() {
public void show() {
 System.out.println("Anonymous Inner Class");          }      };       t.show();   }}
```

Characteristics

- No class name
- Used for **one-time use**
- Reduces code length

Limitations

- Cannot define constructors
- Cannot extend multiple classes

Feature	Static Nested	Member Inner	Local Inner	Anonymous
Static allowed	Yes	No	No	No
Access outer members	Static only	All	Final only	Final only
Object creation	Direct	Via outer object	Inside method	Inline
Named class	Yes	Yes	Yes	No

- Java program to read int and float
- Java program to read string using scanner
- Java program to read string using buffered reader