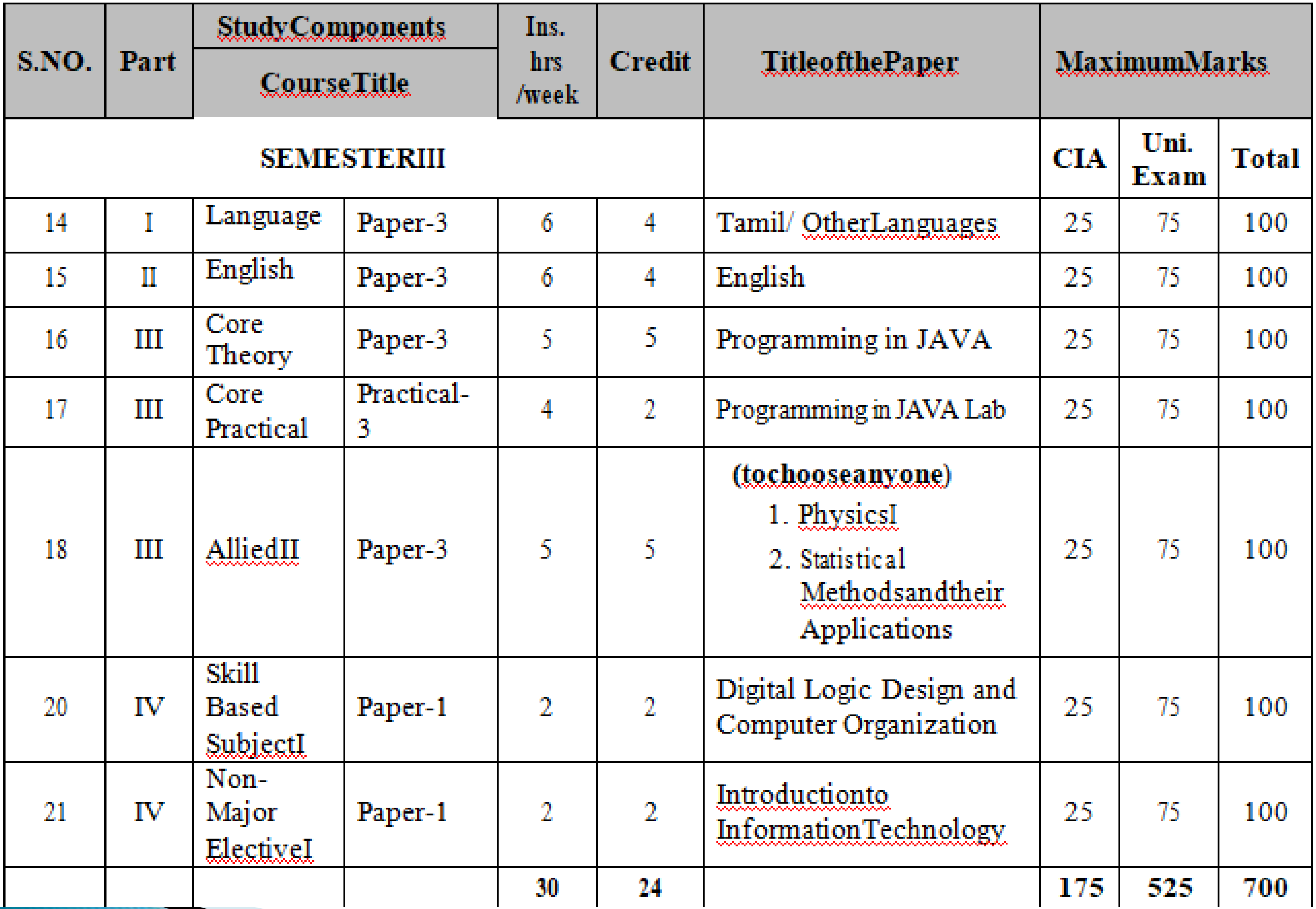
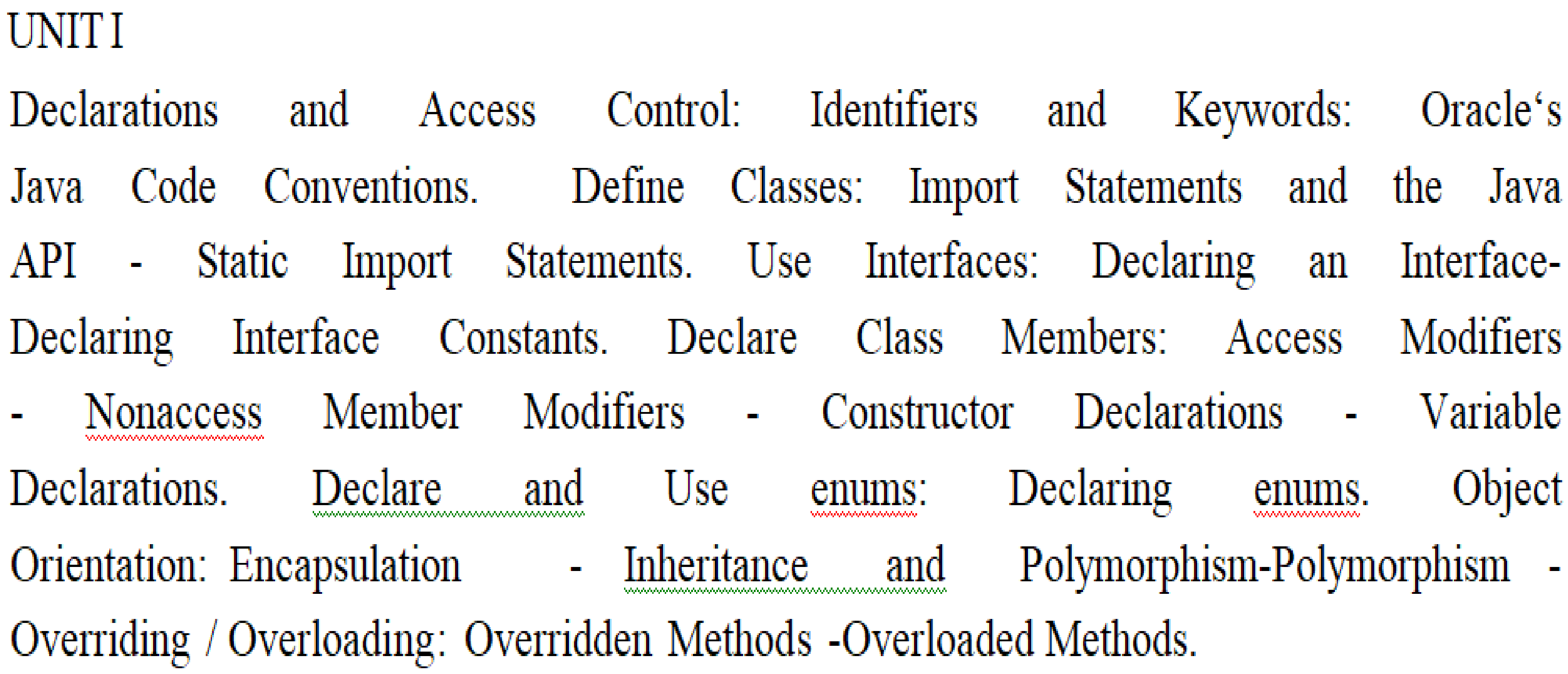
JAVA PROGRAMMING LANGUAGE
UNIT -I

R.Venkatesan, MCA.,M.Phil.,B.Ed.,
Assistant Professor,
Shanmuga Industries Arts & Science College,
Tiruvannamalai



S.NO.	Part	StudyComponents / CourseTitle		Ins. hrs /week	Credit	TitleofthePaper	MaximumMarks		
SEMESTERIII							CIA	Uni. Exam	Total
14	I	Language	Paper-3	6	4	Tamil/ OtherLanguages	25	75	100
15	II	English	Paper-3	6	4	English	25	75	100
16	III	Core Theory	Paper-3	5	5	Programming in JAVA	25	75	100
17	III	Core Practical	Practical-3	4	2	Programming in JAVA Lab	25	75	100
18	III	AlliedII	Paper-3	5	5	**(tochooseanyone)** 1. PhysicsI 2. Statistical Methodsandtheir Applications	25	75	100
20	IV	Skill Based SubjectI	Paper-1	2	2	Digital Logic Design and Computer Organization	25	75	100
21	IV	Non-Major ElectiveI	Paper-1	2	2	Introductionto InformationTechnology	25	75	100
				30	**24**		**175**	**525**	**700**

Unit – I



UNIT I

Declarations and Access Control: Identifiers and Keywords: Oracle's Java Code Conventions. Define Classes: Import Statements and the Java API - Static Import Statements. Use Interfaces: Declaring an Interface- Declaring Interface Constants. Declare Class Members: Access Modifiers - Nonaccess Member Modifiers - Constructor Declarations - Variable Declarations. Declare and Use enums: Declaring enums. Object Orientation: Encapsulation - Inheritance and Polymorphism-Polymorphism - Overriding / Overloading: Overridden Methods -Overloaded Methods.

HISTORY OF JAVA

- Father of java : **James Gasoline**
- EXPANSIONS:
- • **API-**Application Programming Interface
- • **SDK-**Software Development Kit
- • **JVM-**java virtual machine

JAVA HISTORY

- James Gosling, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991.
- The small team of sun engineers called **Green Team**.
- Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.

- Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.
- After that, it was called **Oak** and was developed as a part of the Green project.

Why Java was named as "Oak"?

- **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
- In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

Why Java Programming named "Java"?

- Why had they chose the name Java for Java language? The team gathered to choose a new name.
- The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc.
- They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say.
- According to James Gosling, "Java was one of the top choices along with **Silk**".
- Since Java was so unique, most of the team members preferred Java than other names.

- java is an island in Indonesia where the first coffee was produced (called Java coffee).
- It is a kind of espresso bean.
- Java name was chosen by James Gosling while having a cup of coffee nearby his office.
- Notice that Java is just a name, not an acronym.
- Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995

- In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.
- JDK 1.0 was released on January 23, 1996.
- After the first release of Java, there have been many additional features added to the language.
- Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc.
- Each new version adds new features in Java.

Java Version History

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)
- Java SE 8 (18th Mar 2014)
- Java SE 9 (21st Sep 2017)

- Java SE 10 (20th Mar 2018)
- Java SE 11 (September 2018)
- Java SE 12 (March 2019)
- Java SE 13 (September 2019)
- Java SE 14 (Mar 2020)
- Java SE 15 (September 2020)
- Java SE 16 (Mar 2021)
- Java SE 17 (September 2021)
- Java SE 18 (to be released by March 2022)

- Since Java SE 8 release, the Oracle corporation follows a pattern in which every even version is release in March month and an odd version released in September month.

- Java is a **programming language** and a **platform**.
- Java is a **high level, robust, object-oriented** and secure programming language
- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995.
- **Platform**: Any hardware or software environment in which a program runs, is known as a platform.
- Since Java has a runtime environment (JRE) and API, it is called a platform.

Features of Java

- The primary objective of Java programming language creation was to make it portable, simple and secure programming language.
- Apart from this, there are also some excellent features which play an important role in the popularity of this language.
- The features of Java are also known as Java buzzwords.

What is Java?

- **It is owned by Oracle, and more than 3 billion devices run Java.**
- Mobile applications (specially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!
-

Why Use Java?

- Java works on different platforms **(Windows, Mac, Linux, Raspberry Pi, etc.)**
- It is one of the most popular programming language in the world
- It is easy to learn and simple to use
- It is open-source and free
- It is secure, fast and powerful
- It has a huge community support **(tens of millions of developers)**
- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

OBJECT ORIENTED:

- no coding outside of class definitions including main()
- – An extensive class library available in the core language packages

COMPILER AND INTERPRETER

- Code is compiled to byte codes that are
- interpreter by a JVM
- – This provides portability to any machine for which a virtual machine has been written
- – The two steps of compilation and interpretation allow for extensive code checking and improved security

- **ROBUST**
- Exception handling built-in strong type checking
- **Several dangerous features of c & c++**
- No memory pointers
- – No pre processor
- – Garbage collector
- **AUTOMATIC MEMORY MANAGEMENT**
- Automatic garbage collection memory management handled by JVM
- **SECURITY**
- No memory pointers
- Programs runs inside the virtual machine sandbox

Byte code verifier:

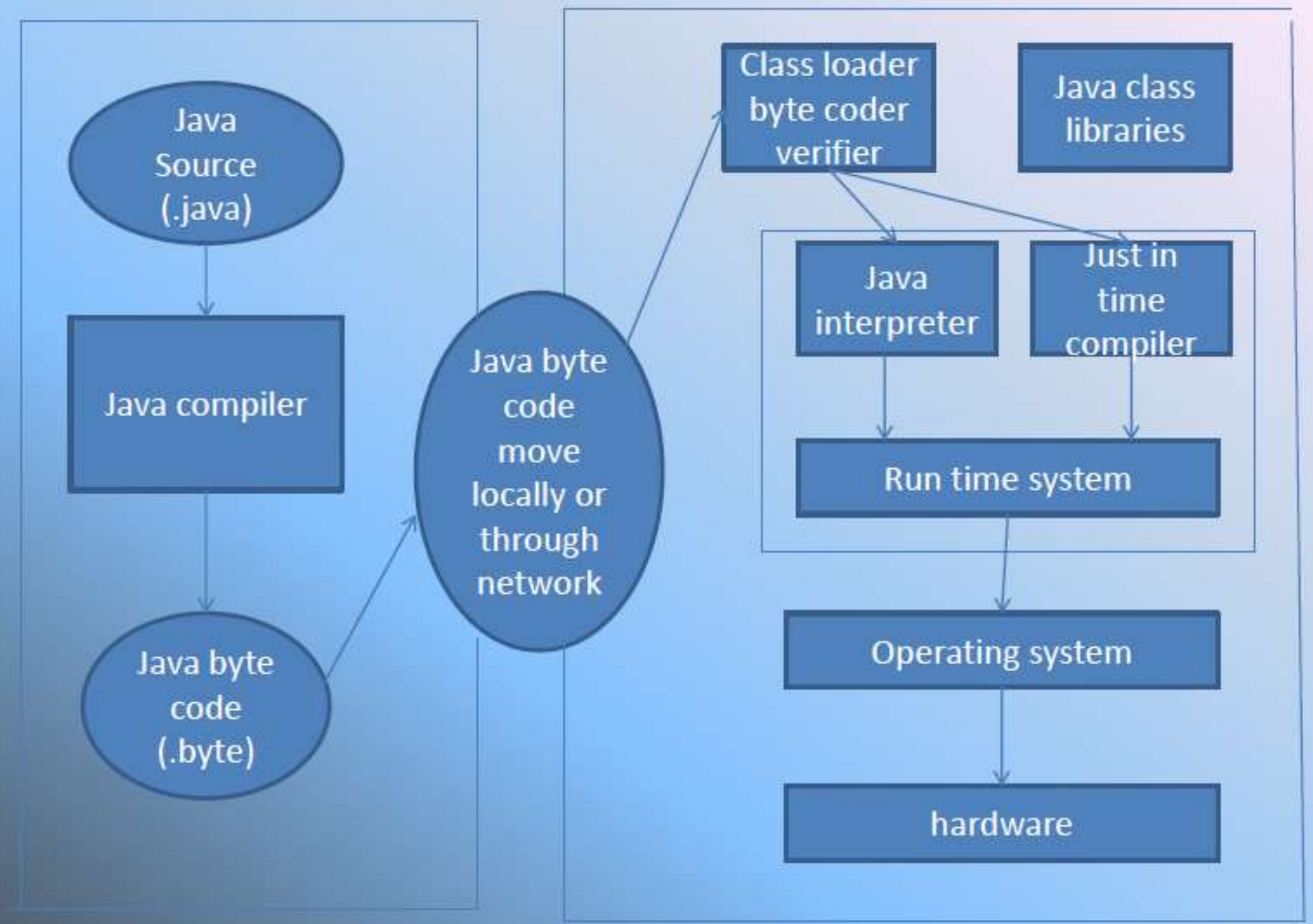
- Checks classes after loading

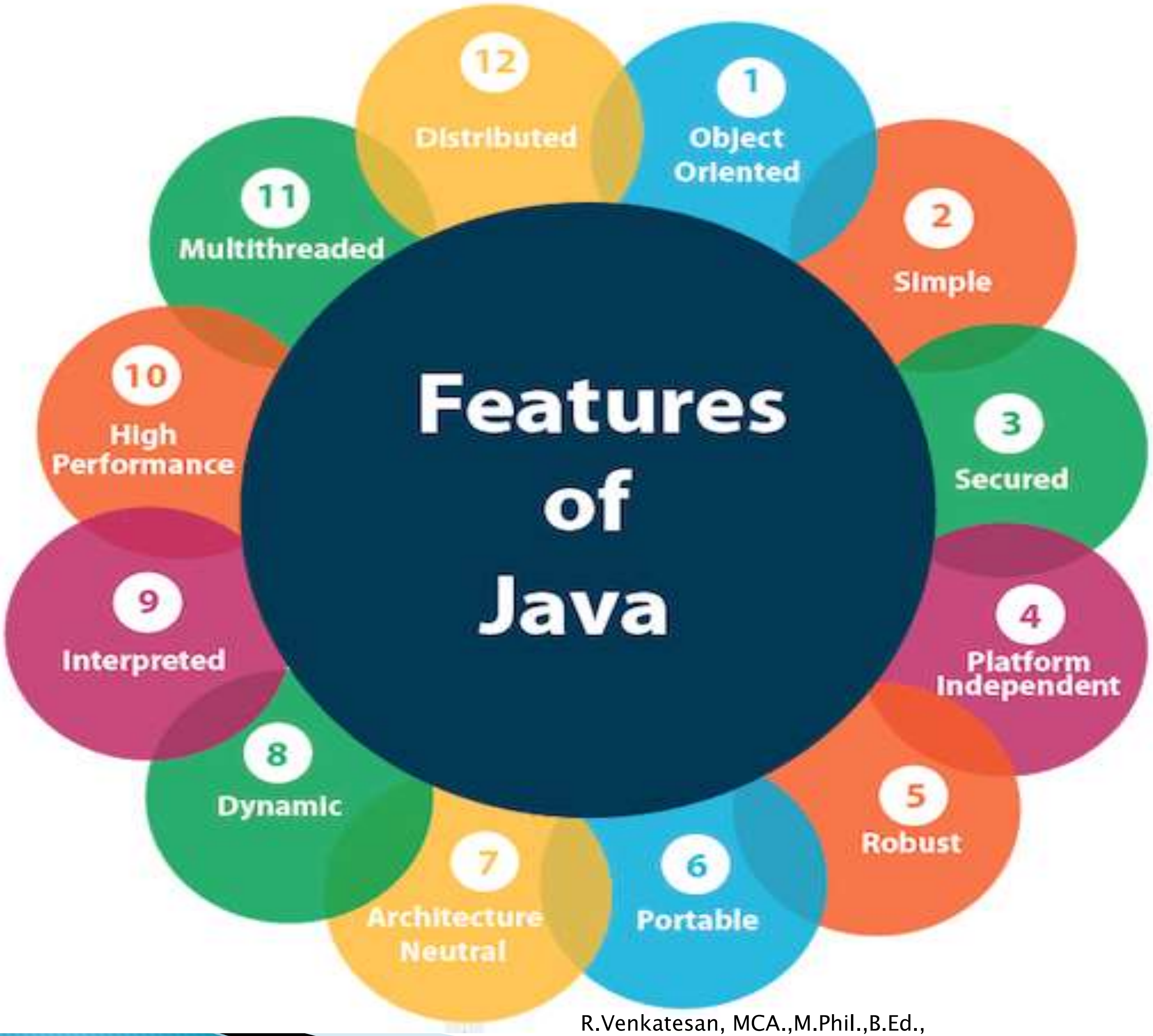
Class loaders

- Confines object to unique name space

Security manager:

- Determines what resources a class can access


Java Source (.java)
Java compiler
Java byte code (.byte)
Java byte code move locally or through network
Class loader byte coder verifier
Java class libraries
Java interpreter
Just in time compiler
Run time system
Operating system
hardware


Features of Java
1 Object Oriented
2 Simple
3 Secured
4 Platform Independent
5 Robust
6 Portable
7 Architecture Neutral
8 Dynamic
9 Interpreted
10 High Performance
11 Multithreaded
12 Distributed

Simple

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

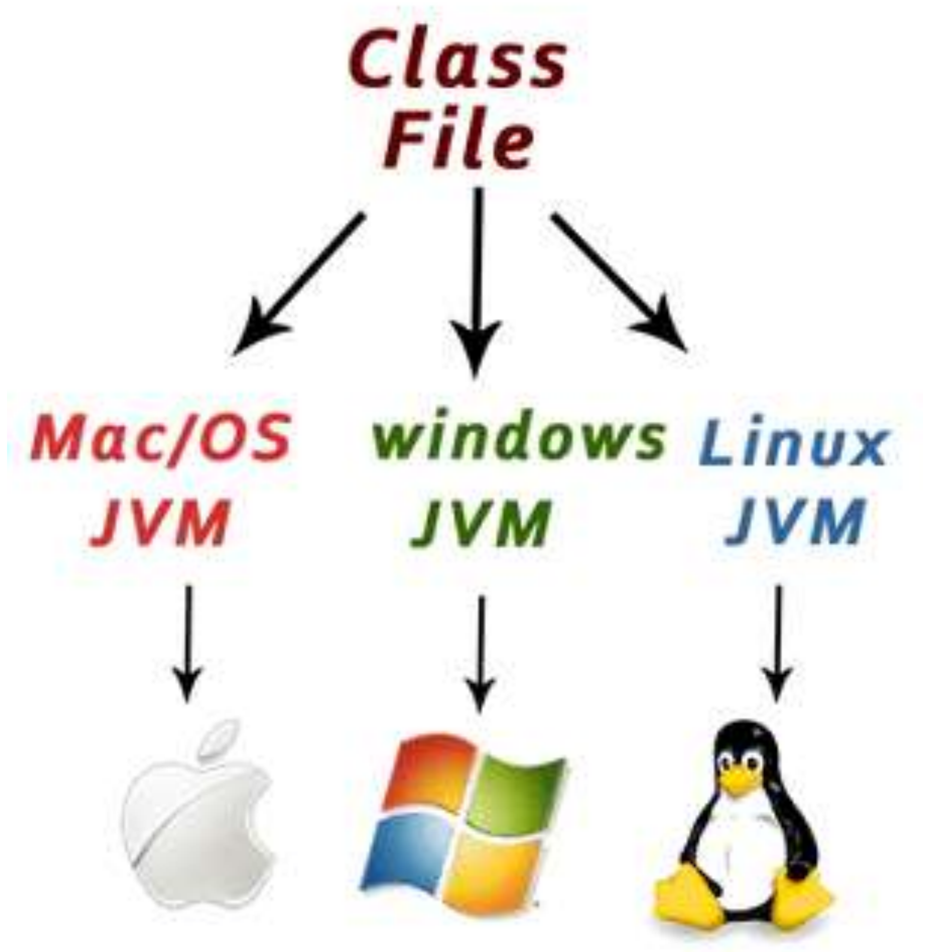
Object-oriented

- Java is an object-oriented programming language.
- Everything in Java is an object.
- Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.
- Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

- Basic concepts of OOPs are:
- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

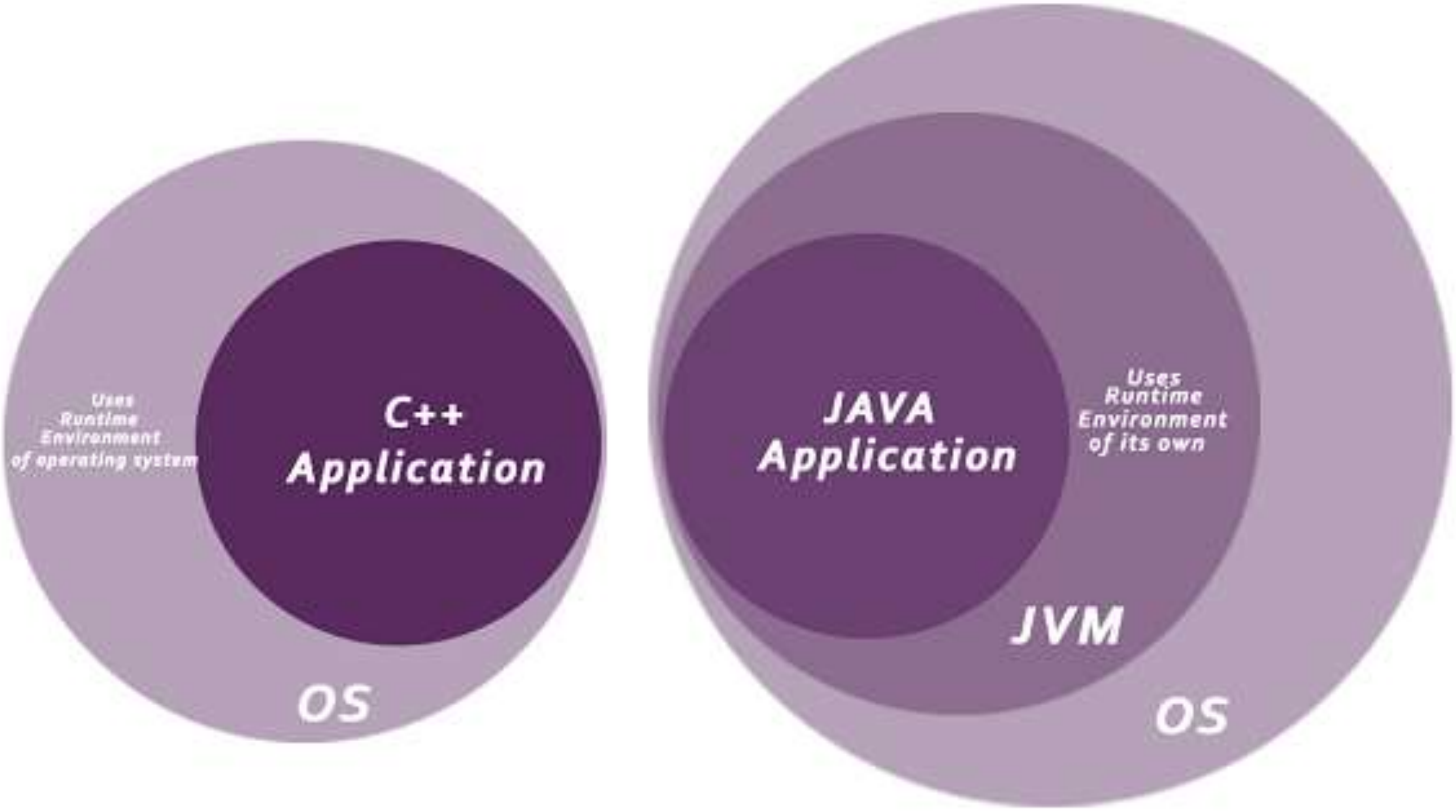
Platform Independent

- Java is platform independent because it is different from other languages like C, C++, etc.
- which are compiled into platform specific machines while Java is a write once, run anywhere language.
- A platform is the hardware or software environment in which a program runs.
- There are two types of platforms software-based and hardware-based.
- Java provides a software-based platform.
- The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms.


Class File
Mac/OS JVM
windows JVM
Linux JVM

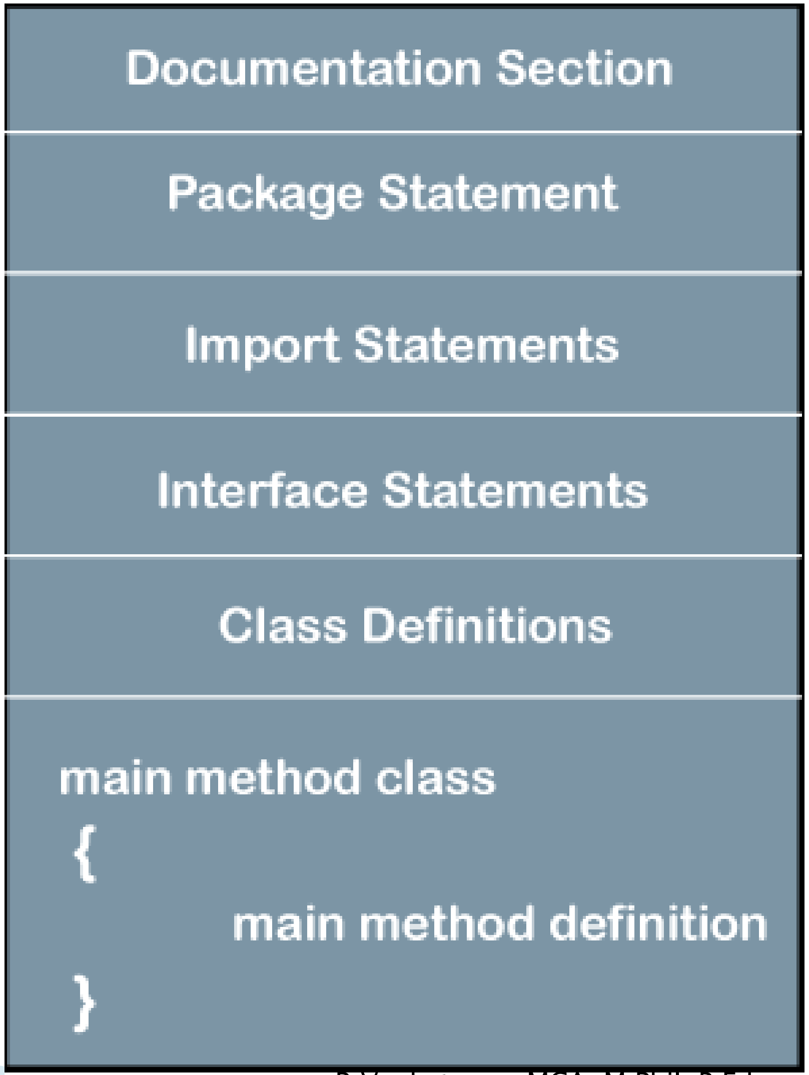
Secured

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


Uses
Runtime
Environment
of operating system
C++
Application
OS
JAVA
Application
Uses
Runtime
Environment
of its own
JVM
OS

- **Class loader:**
- Class loader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically.
- It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Byte code Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Structure of Java Program



Documentation Section

- The documentation section is an important section but optional for a Java program.
- It includes **basic information** about a Java program.
- The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program.
- It improves the readability of the program.
- Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program.
- To write the statements in the documentation section, we use **comments**.
- The comments may be **single-line, multi-line,** and **documentation** comments.
- **Single-line Comment:** It starts with a pair of forwarding slash **(//)**. For example:
- **//First Java Program**

Package Declaration

- The package declaration is optional. It is placed just after the documentation section.
- In this section, we declare the **package name** in which the class is placed.
- Note that there can be **only one package** statement in a Java program.
- It must be defined before any class and interface declaration.
- It is necessary because a Java class can be placed in different packages and directories based on the module they are used.
- For all these classes package belongs to a single parent directory.
- We use the keyword **package** to declare the package name.

- **package** javatpoint; //where javatpoint is the packa ge name
- **package** com.javatpoint; //where com is the root di rectory and javatpoint is the subdirectory

Import Statements

- The package contains the many predefined classes and interfaces.
- If we want to use any class of a particular package, we need to import that class.
- The import statement represents the class stored in the other package.
- We use the **import** keyword to import the class. It is written before the class declaration and after the package statement.
- We use the import statement in two ways, either import a specific class or import all classes of a particular package.
- In a Java program, we can use multiple import statements.

For example:

- **import** java.util.Scanner; //it imports the Scanner class only
- **import** java.util.*; //it imports all the class of the java.util package

Interface Section

- It is an optional section. We can create an **interface** in this section if required.
- We use the **interface** keyword to create an interface.
- An interface is a slightly different from the class. It contains only **constants** and **method** declarations.
- Another difference is that it cannot be instantiated. We can use interface in classes by using the **implements** keyword.
- An interface can also be used with other interfaces by using the **extends** keyword

- **interface** car
- {
- **void** start();
- **void** stop();
- }

Class Definition

- In this section, we define the class.
- It is **vital** part of a Java program. Without the class, we cannot create any Java program.
- A Java program may conation more than one class definition.
- We use the **class** keyword to define the class. The class is a blueprint of a Java program.
- It contains information about user-defined methods, variables, and constants.
- Every Java program has at least one class that contains the main() method.

For example:

- **class** Student //class definition
- {
- }

Class Variables and Constants

- In this section, we define variables and **constants** that are to be used later in the program.
- In a Java program, the variables and constants are defined just after the class definition.
- The variables and constants store values of the parameters.
- It is used during the execution of the program. We can also decide and define the scope of variables by using the modifiers.
- It defines the life of the variables.

For example:

```
class Student //class definition
{
String sname;  //variable
int id;
double percentage;
}
```

Main Method Class

- In this section, we define the **main() method.** It is essential for all Java programs.
- Because the execution of all Java programs starts from the main() method.
- In other words, it is an entry point of the class. It must be inside the class.
- Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:

- **public static void** main(String args[])
- {
- }

For example:

- **public class** Student //class definition
- {
- **public static void** main(String args[])
- {
- //statements
- }
- }

- **public class** Demo //class definition
- {
- **public static void** main(String args[])
- {
- **void** display()
- {
- System.out.println("Welcome to javatpoint");
- }
- //statements
- }
- }

Declarations & Access Control in Java

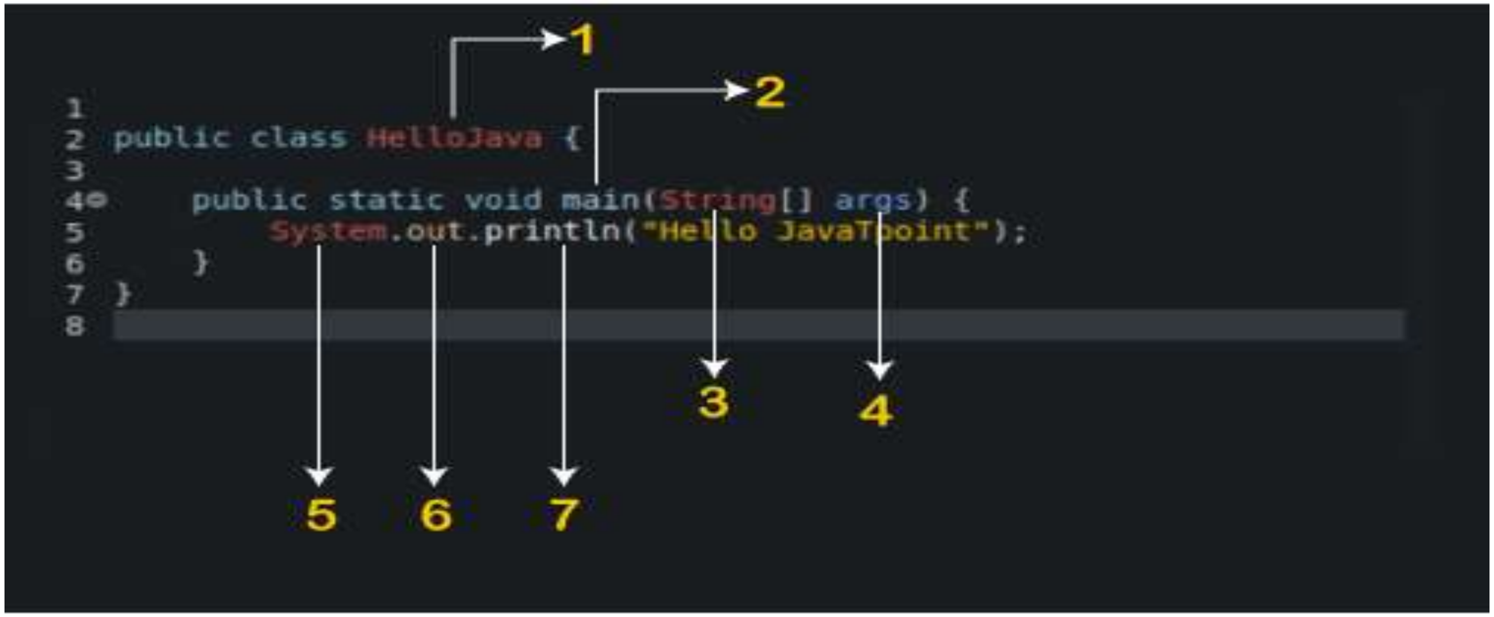
- Source File Declaration
- Identifiers Declaration
- Local Variables
- Instance Variables
- Constructors
- Static
- ENUM
- Class Modifiers (non-access)
- Class Access Modifiers – public- protected- private

Declarations & Access Control in Java

- **Identifiers in Java**
- Identifiers in Java are symbolic names used for identification.
- They can be a class name, variable name, method name, package name, constant name, and more.
- However, In Java, There are some reserved words that can not be used as an identifier.
- For every identifier there are some conventions that should be used before declaring them.
-

Let's understand it with a simple Java program

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



```java

public class HelloJava {

    public static void main(String[] args) {
        System.out.println("Hello JavaTpoint");
    }
}

```

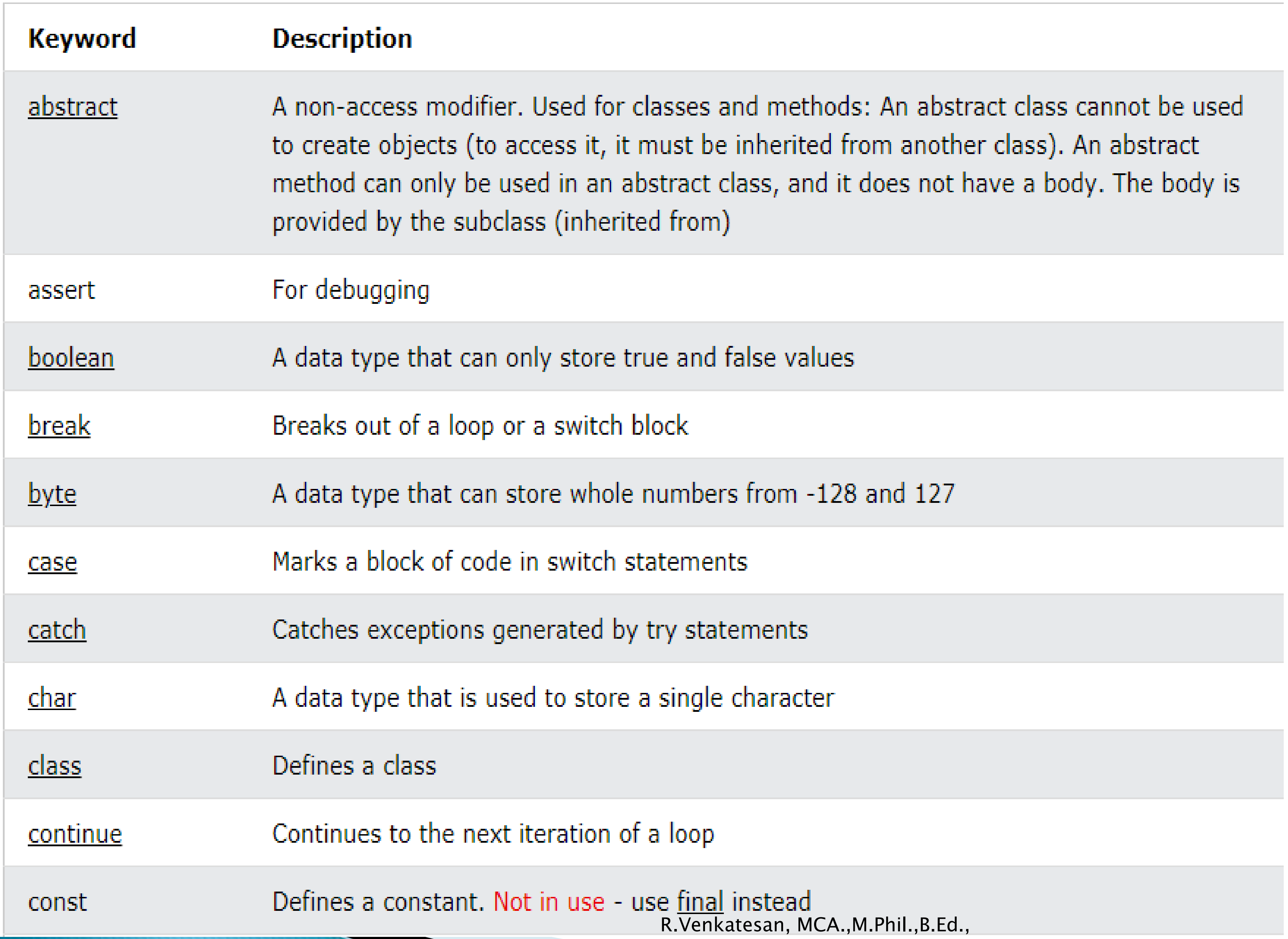
- HelloJava (Class name)
- main (main method)
- String (Predefined Class name)
- args (String variables)
- System (Predefined class)
- out (Variable name)
- println (method)

Rules for Identifiers in Java

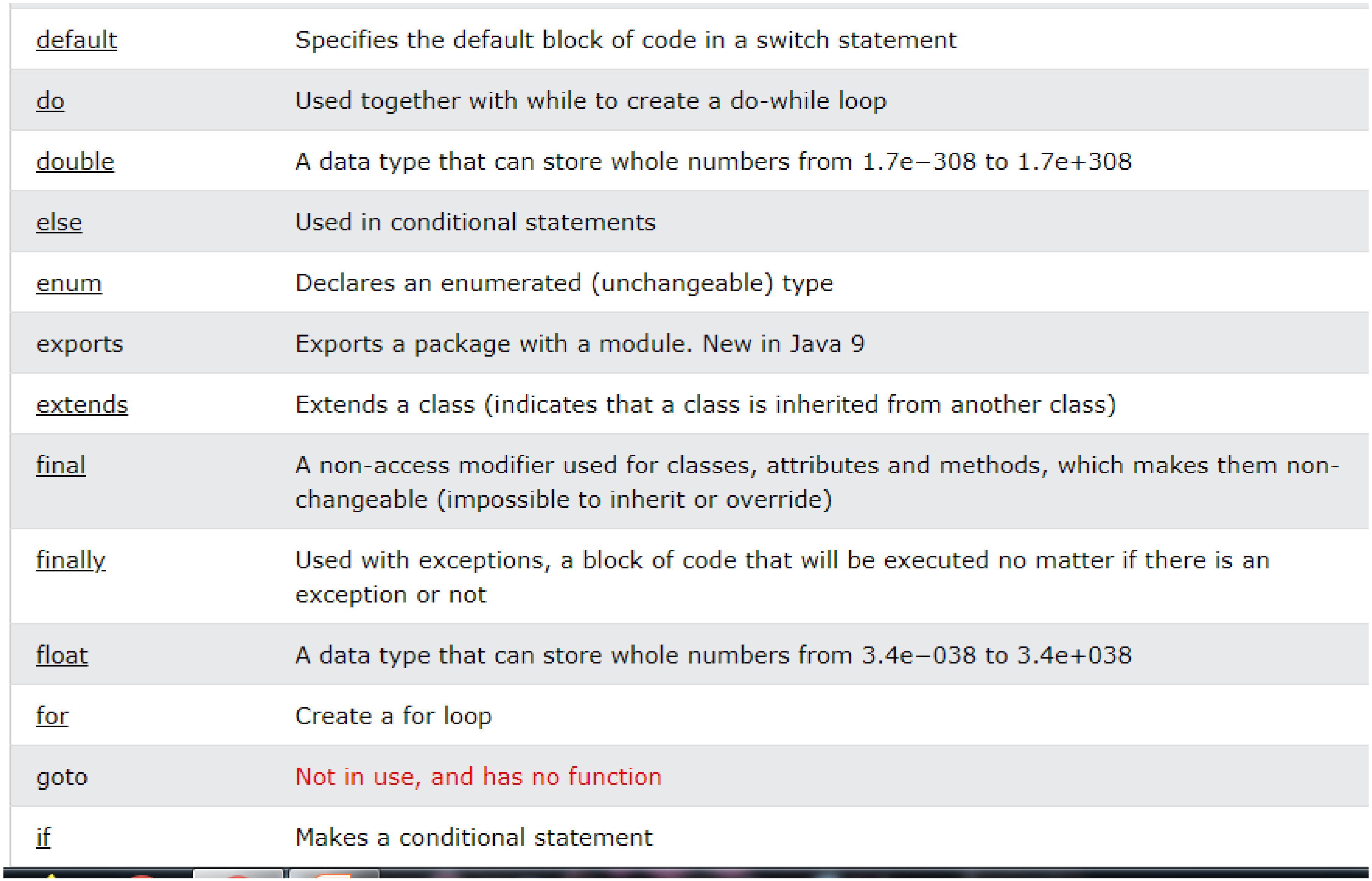
- A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(_) or a dollar sign($).
- There should not be any space in an identifies.
- An identifier should not contain a number at the starting
- We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.
- An identifier should not be any query language keywords such as SELECT, FROM, COUNT, DELETE, etc.

Java Keywords

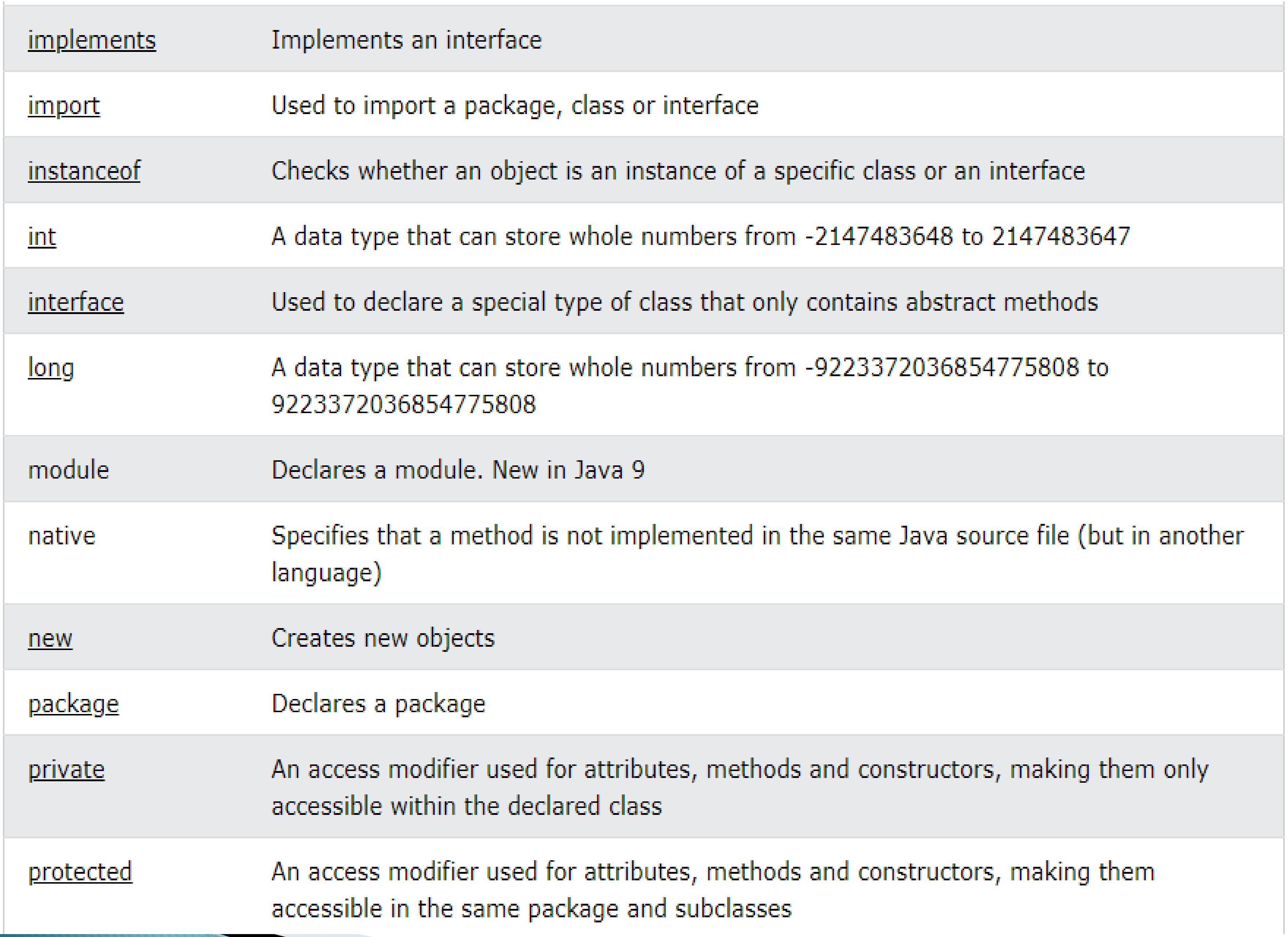
- Java keywords are also known as reserved words.
- Keywords are particular words that act as a key to a code.
- These are predefined words by Java so they cannot be used as a variable or object name or class name.



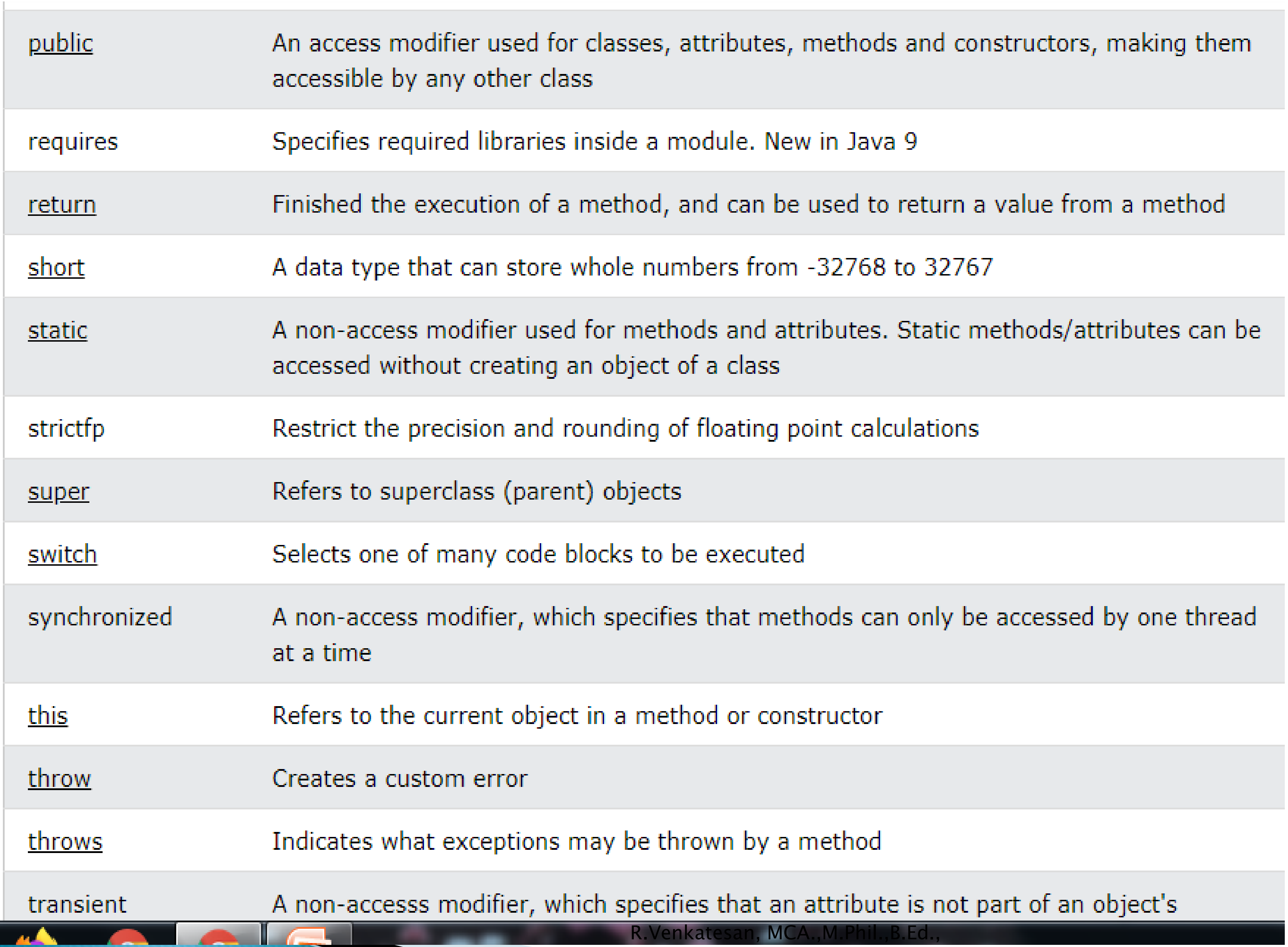
Keyword	Description
abstract	A non-access modifier. Used for classes and methods: An abstract class cannot be used to create objects (to access it, it must be inherited from another class). An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from)
assert	For debugging
boolean	A data type that can only store true and false values
break	Breaks out of a loop or a switch block
byte	A data type that can store whole numbers from -128 and 127
case	Marks a block of code in switch statements
catch	Catches exceptions generated by try statements
char	A data type that is used to store a single character
class	Defines a class
continue	Continues to the next iteration of a loop
const	Defines a constant. Not in use - use final instead



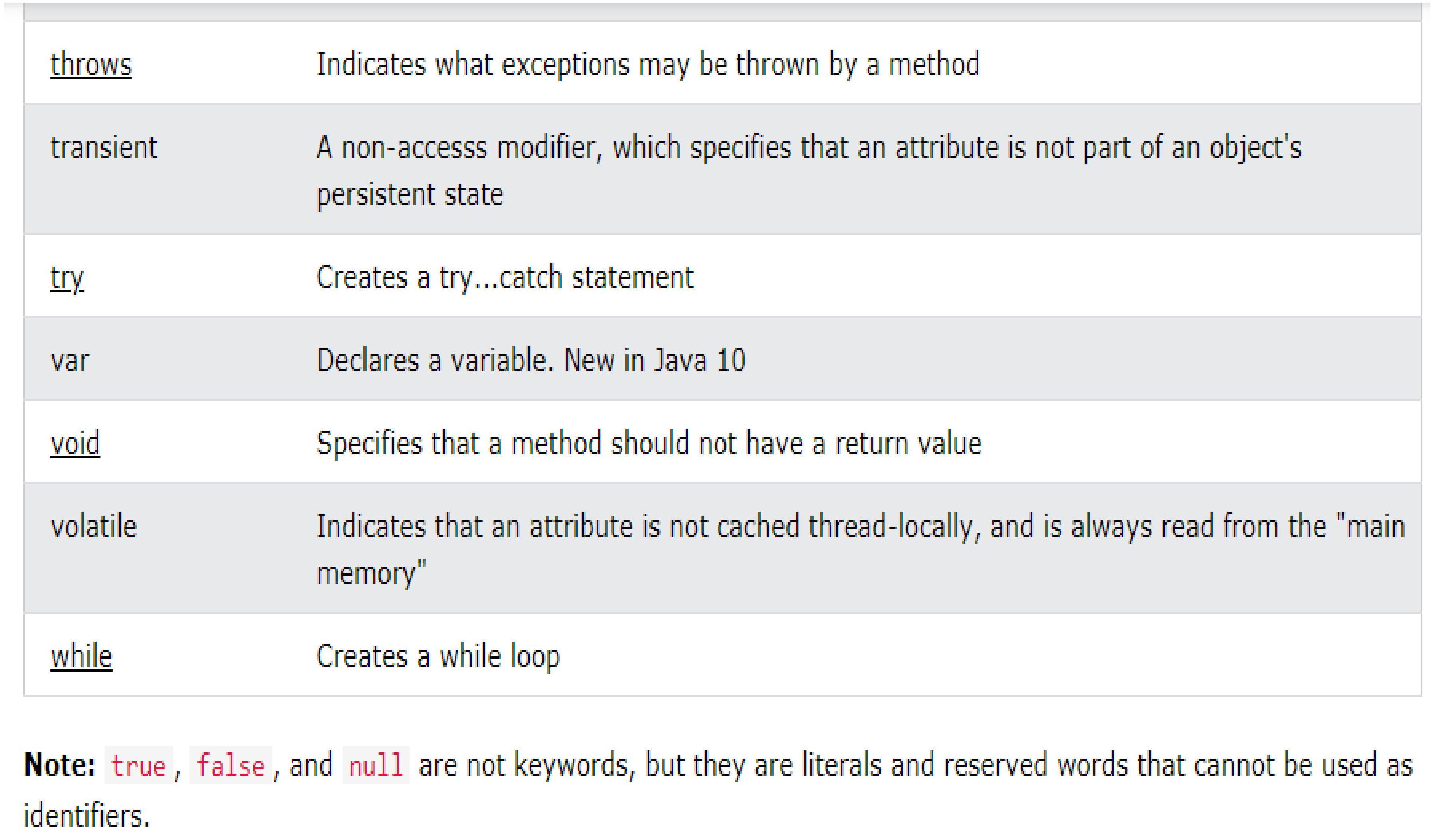
default	Specifies the default block of code in a switch statement
do	Used together with while to create a do-while loop
double	A data type that can store whole numbers from 1.7e−308 to 1.7e+308
else	Used in conditional statements
enum	Declares an enumerated (unchangeable) type
exports	Exports a package with a module. New in Java 9
extends	Extends a class (indicates that a class is inherited from another class)
final	A non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override)
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
float	A data type that can store whole numbers from 3.4e−038 to 3.4e+038
for	Create a for loop
goto	Not in use, and has no function
if	Makes a conditional statement



implements	Implements an interface
import	Used to import a package, class or interface
instanceof	Checks whether an object is an instance of a specific class or an interface
int	A data type that can store whole numbers from -2147483648 to 2147483647
interface	Used to declare a special type of class that only contains abstract methods
long	A data type that can store whole numbers from -9223372036854775808 to 9223372036854775808
module	Declares a module. New in Java 9
native	Specifies that a method is not implemented in the same Java source file (but in another language)
new	Creates new objects
package	Declares a package
private	An access modifier used for attributes, methods and constructors, making them only accessible within the declared class
protected	An access modifier used for attributes, methods and constructors, making them accessible in the same package and subclasses



public	An access modifier used for classes, attributes, methods and constructors, making them accessible by any other class
requires	Specifies required libraries inside a module. New in Java 9
return	Finished the execution of a method, and can be used to return a value from a method
short	A data type that can store whole numbers from -32768 to 32767
static	A non-access modifier used for methods and attributes. Static methods/attributes can be accessed without creating an object of a class
strictfp	Restrict the precision and rounding of floating point calculations
super	Refers to superclass (parent) objects
switch	Selects one of many code blocks to be executed
synchronized	A non-access modifier, which specifies that methods can only be accessed by one thread at a time
this	Refers to the current object in a method or constructor
throw	Creates a custom error
throws	Indicates what exceptions may be thrown by a method
transient	A non-accesss modifier, which specifies that an attribute is not part of an object's

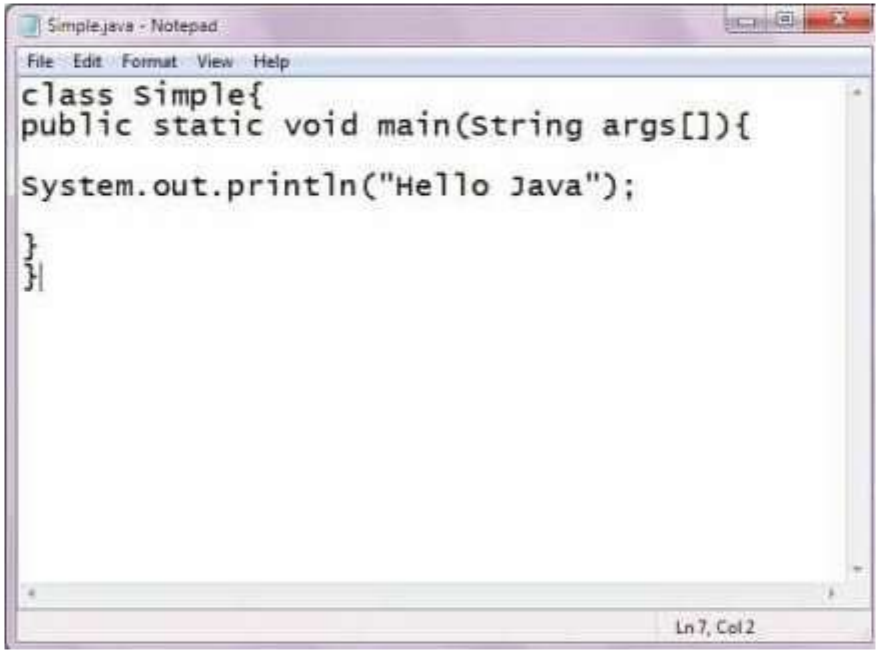


throws	Indicates what exceptions may be thrown by a method
transient	A non-accesss modifier, which specifies that an attribute is not part of an object's persistent state
try	Creates a try...catch statement
var	Declares a variable. New in Java 10
void	Specifies that a method should not have a return value
volatile	Indicates that an attribute is not cached thread-locally, and is always read from the "main memory"
while	Creates a while loop

Note: `true`, `false`, and `null` are not keywords, but they are literals and reserved words that cannot be used as identifiers.

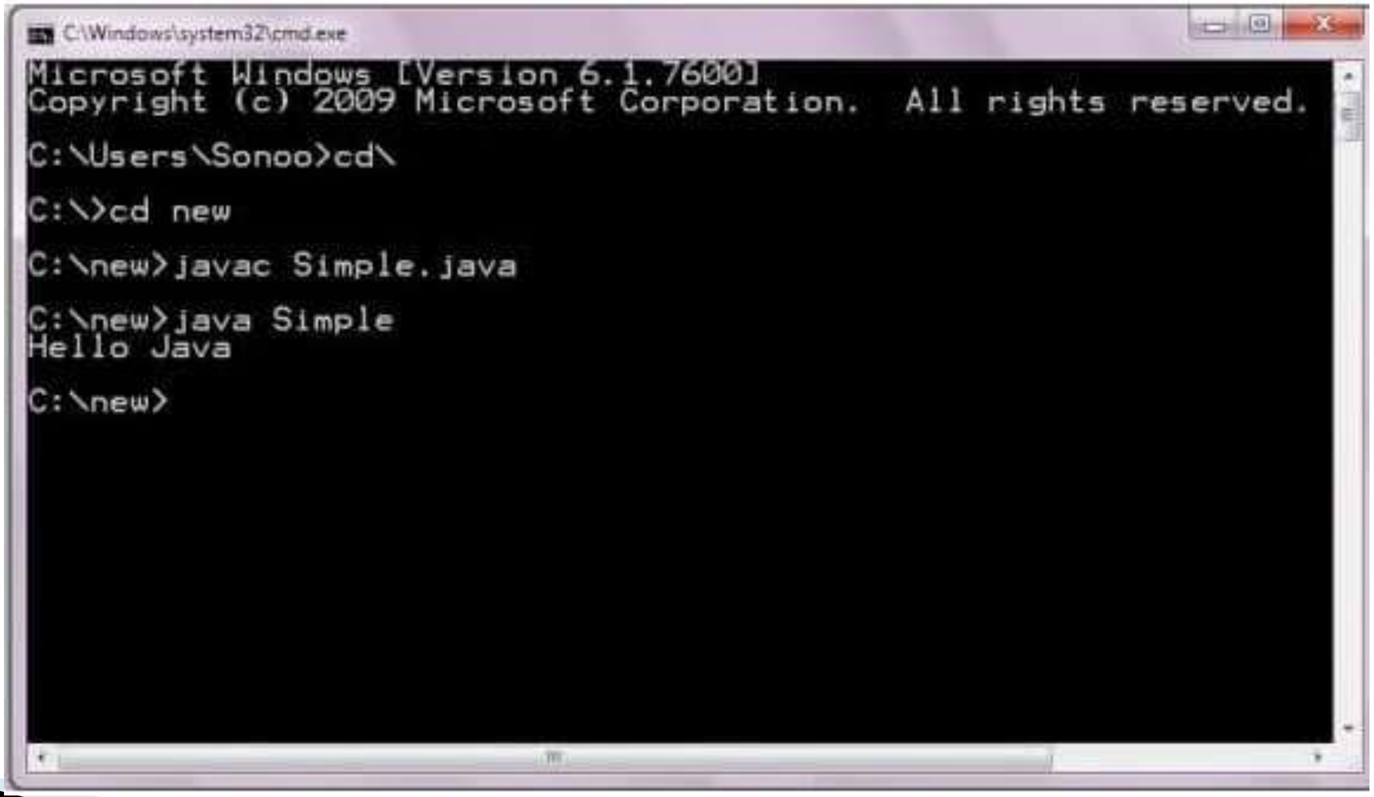
How to Create Program

- To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> Notepad**



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

System.out.println("Hello Java");

}
}
```

- In order to compile and run the above program, you need to open the command prompt by **start menu -> All Programs -> Accessories -> command prompt**.



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sonoo>cd\

C:\>cd new

C:\new>javac Simple.java

C:\new>java Simple
Hello Java

C:\new>
```

Sample Program

- **The requirement for Java Hello World Example**
- For executing any Java program, the following software or application must be properly installed.
- Install the JDK if you don't have installed it, download the JDK and install it.
- Set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- Create the Java program
- Compile and run the Java program

Creating Hello World Example

- **class** Simple
- {
-     **public static void** main(String args[])
- {
-      System.out.println("Hello Java");
-     }
- }

- **Save Program:**
- Save the above file as Simple.java

- **To compile:** javac Simple.java

- **To execute:** java Simple

- **Output:**
- Hello Java

Java class

- A Java class keyword is the most common keyword which is used to declare a new Java class.
- A class is a container that contains the block of code that includes field, method, constructor, etc.,

  A class is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

Points to remember

- Every object is an instance of a class.
- A class can contain one or more classes.
- This concept can be called a nested class.
- We can assign only public, abstract, strictfp and final modifier to class.
- However, we can assign other modifiers like private, protected and static to the inner java class.
- A class name must be unique within a package.
- We can use a class keyword as Class.
- class to get a Class object without needing an instance of that class.

Let's see a simple example of a class keyword.

```
public class ClassExample {

    public static void main(String[] args) {

        System.out.println("Hello World");
    }

}
```

o/P

```
Hello World
```

In this example, we declare two classes

```
class Employee
{
   int id;
   String name;
   long salary;
   public Employee(int id, String name, long salary)
 {
      super();
      this.id = id;
      this.name = name;
      this.salary = salary;
   }

}
```

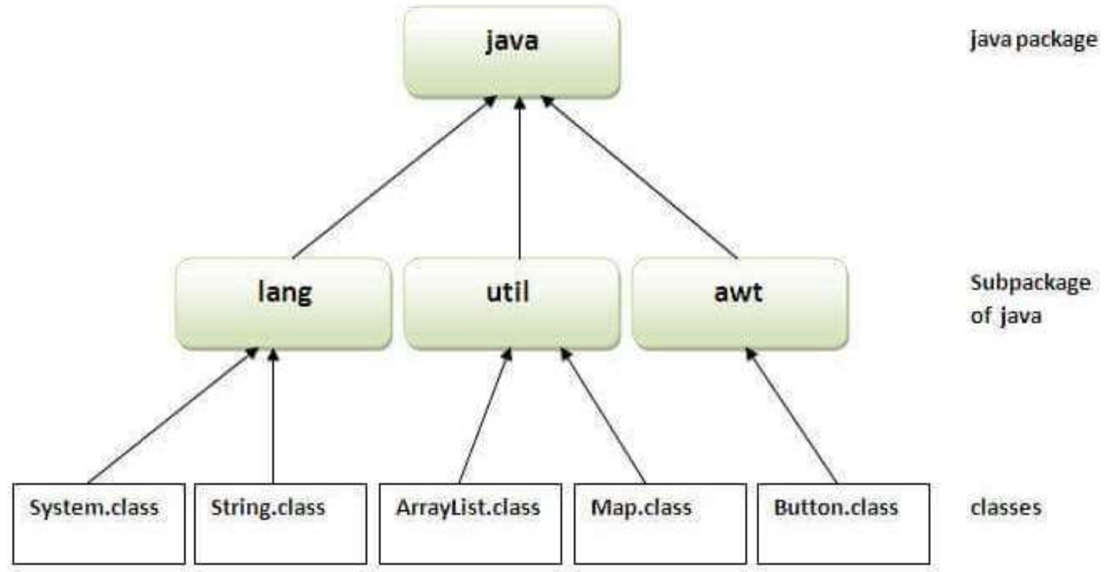
```
class ClassExample
{

    public static void main(String[] args) {

        Employee e=new Employee(101,"John William",25000);

    System.out.println(e.id+" "+e.name+" "+e.salary);



    }
}
```

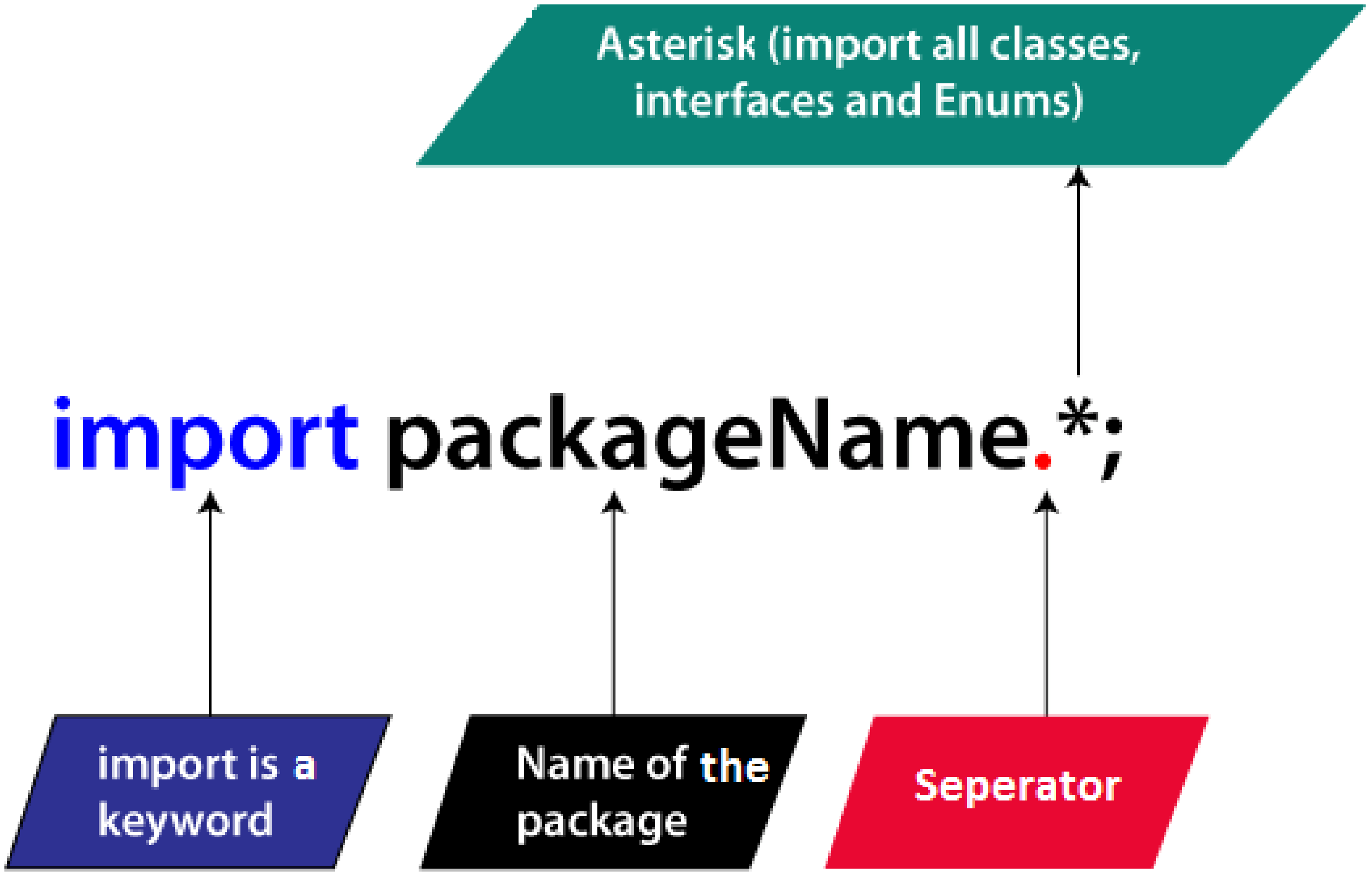
O/P

```
101 John William 25000
```

Java Package import

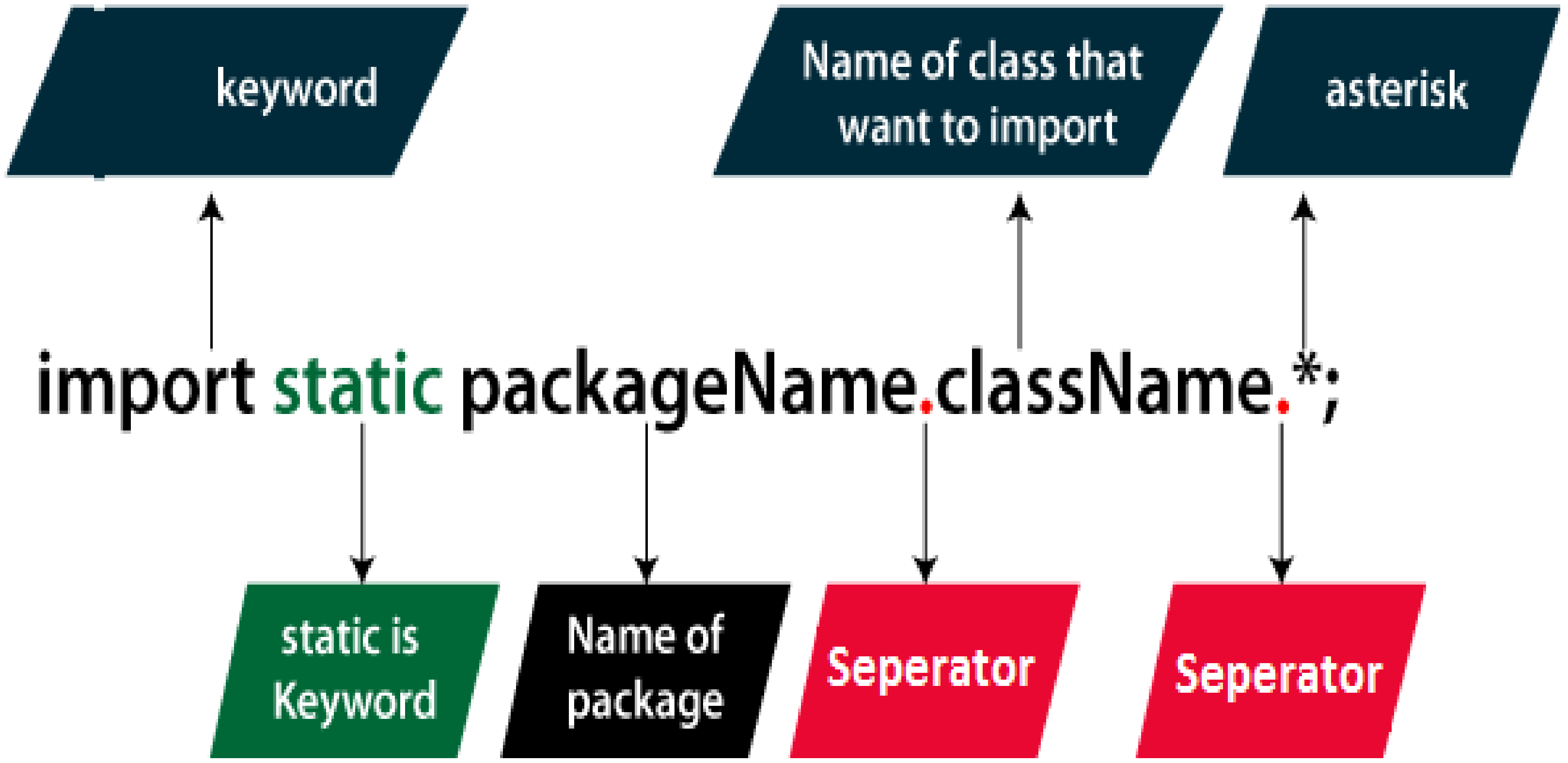
- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.




Asterisk (import all classes, interfaces and Enums)
import packageName.*;
import is a keyword
Name of the package
Seperator

Static Import Statements

- The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.
- **Advantage of static import:**
- Less coding is required if you have access any static member of a class oftenly.
- **Disadvantage of static import:**
- If you overuse the static import feature, it makes the program unreadable and unmaintainable


keyword
Name of class that want to import
asterisk
import static packageName.className.*;
static is Keyword
Name of package
Seperator
Seperator

Simple Example of static import

- **import static** java.lang.System.*;
- **class** StaticImportExample
- {
- **public static void** main(String args[])
- {
-
-    out.println("Hello");//Now no need of System.out
-    out.println("Java");
-
- }
- }
- **O/P**
- Hello
- Java

Interface in Java

- An **interface in Java** is a blueprint of a class. It has **static constants** and **abstract methods**.
- The interface in Java is *a mechanism to achieve abstraction*.
- There can be only abstract methods in the Java interface, not method body.
- It is used to achieve abstraction and multiple inheritance in Java.
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Why use Java interface?

- here are mainly three reasons to use interface.
- They are given below.
- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

It is used to achieve abstraction.
1
2
By interface, we can support the functionality of multiple inheritance.
It can be used to achieve loose coupling.
3

How to declare an interface?

- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

Syntax:

- **interface** <interface_name>
- {
-
-     // declare constant fields
-     // declare methods that abstract
-     // by default.
- }

Interface Constants

- **What is constant?**
- **Constant** is a value that cannot be changed after assigning it.
- Java does not directly support the constants.
- There is an alternative way to define the constants in Java by using the non-access modifiers static and final.

How to declare constant in Java?

- In Java, to declare any variable as constant, we use static and final modifiers.
- It is also known as **non-access** modifiers.
- According to the Java naming convention the identifier name must be in **capital letters**.

Class and Interface in Java

- **Class in Java**
- A Class can be defined as the collection of objects that have a similar type of properties.
- It is a logical entity that can be seen as a blueprint for creating the objects.
- A Class can have many objects where each has the attributes and behavior defined by the class itself.
- However, we can also create a singleton class that has only a single instance.

In Java, a class can contain the following entities:

- Attributes
- Behavior(methods)
- Constructors
- Blocks
- Nested class and interface

- **Consider the following syntax to declare a class in java.**

- **public class** Main
- {
- **int** attr = 10; //instance variable
- }
- The 'class' keyword is used to declare a class.
- The class declaration may contain the following components defined in the sequence.

Access Modifiers in Java

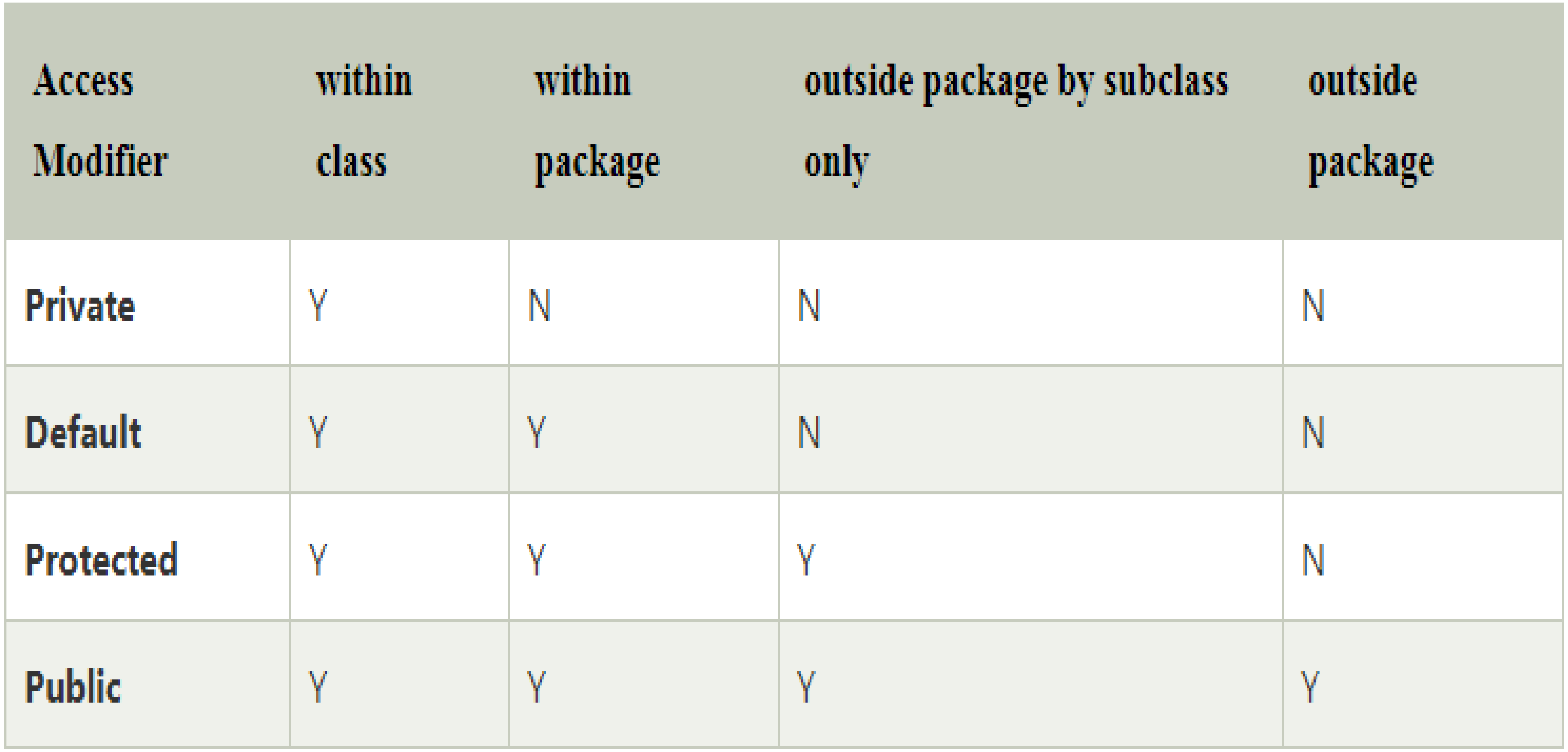
- There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.
- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

- **Private**: The access level of a private modifier is only within the class.
- It cannot be accessed from outside the class.
- **Default**: The access level of a default modifier is only within the package.
- It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

- **Protected**: The access level of a protected modifier is within the package and outside the package through child class.
- If you do not make the child class, it cannot be accessed from outside the package.
- **Public**: The access level of a public modifier is everywhere.
- It can be accessed from within the class, outside the class, within the package and outside the package.

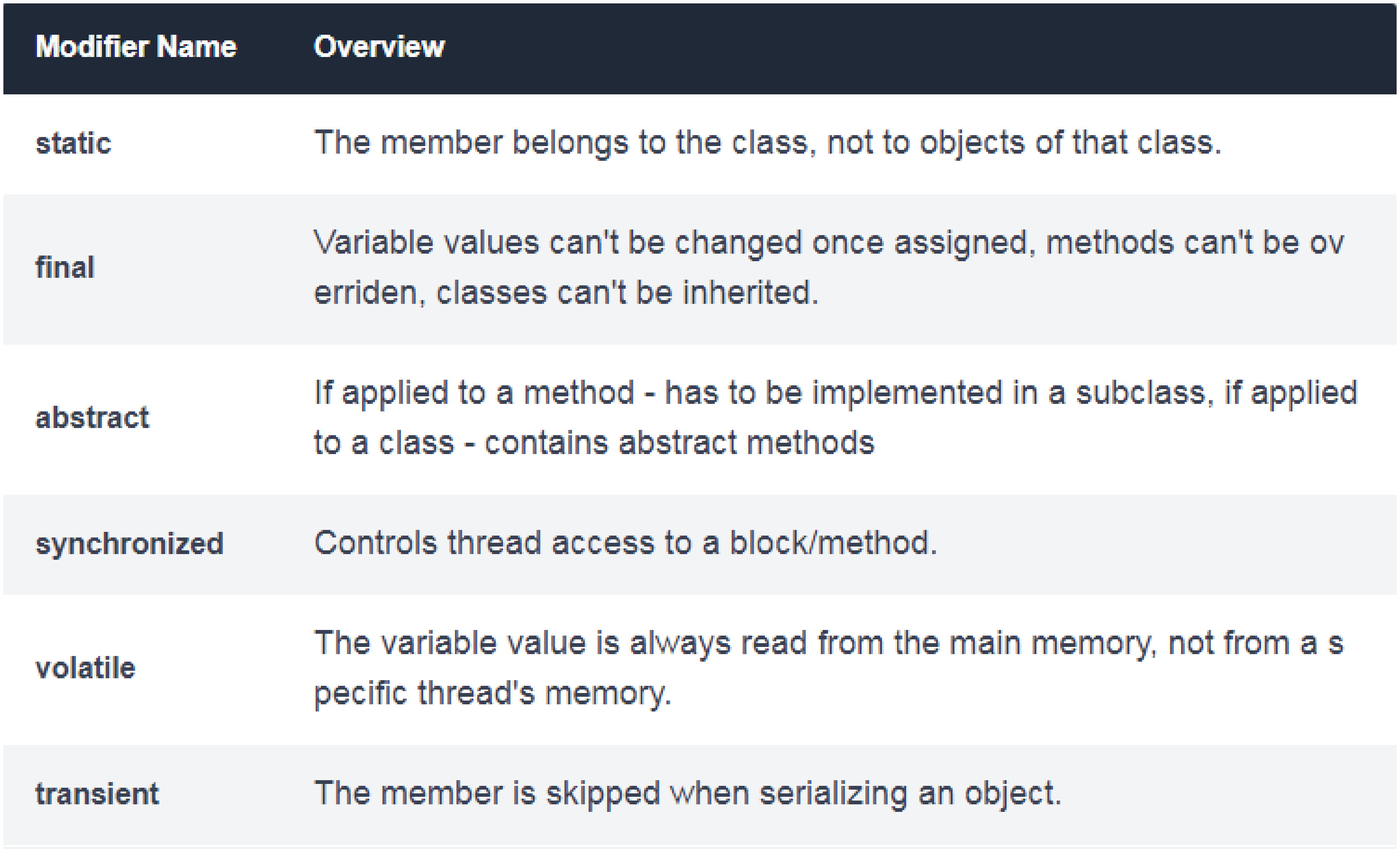
Understanding Java Access Modifiers



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

Non-Access Modifiers

- These types of modifiers are used to control a variety of things, such as inheritance capabilities, whether all objects of our class share the same member value or have their own values of those members, whether a method can be overridden in a subclass, etc.



Modifier Name	Overview
static	The member belongs to the class, not to objects of that class.
final	Variable values can't be changed once assigned, methods can't be overriden, classes can't be inherited.
abstract	If applied to a method - has to be implemented in a subclass, if applied to a class - contains abstract methods
synchronized	Controls thread access to a block/method.
volatile	The variable value is always read from the main memory, not from a specific thread's memory.
transient	The member is skipped when serializing an object.

Constructors in Java

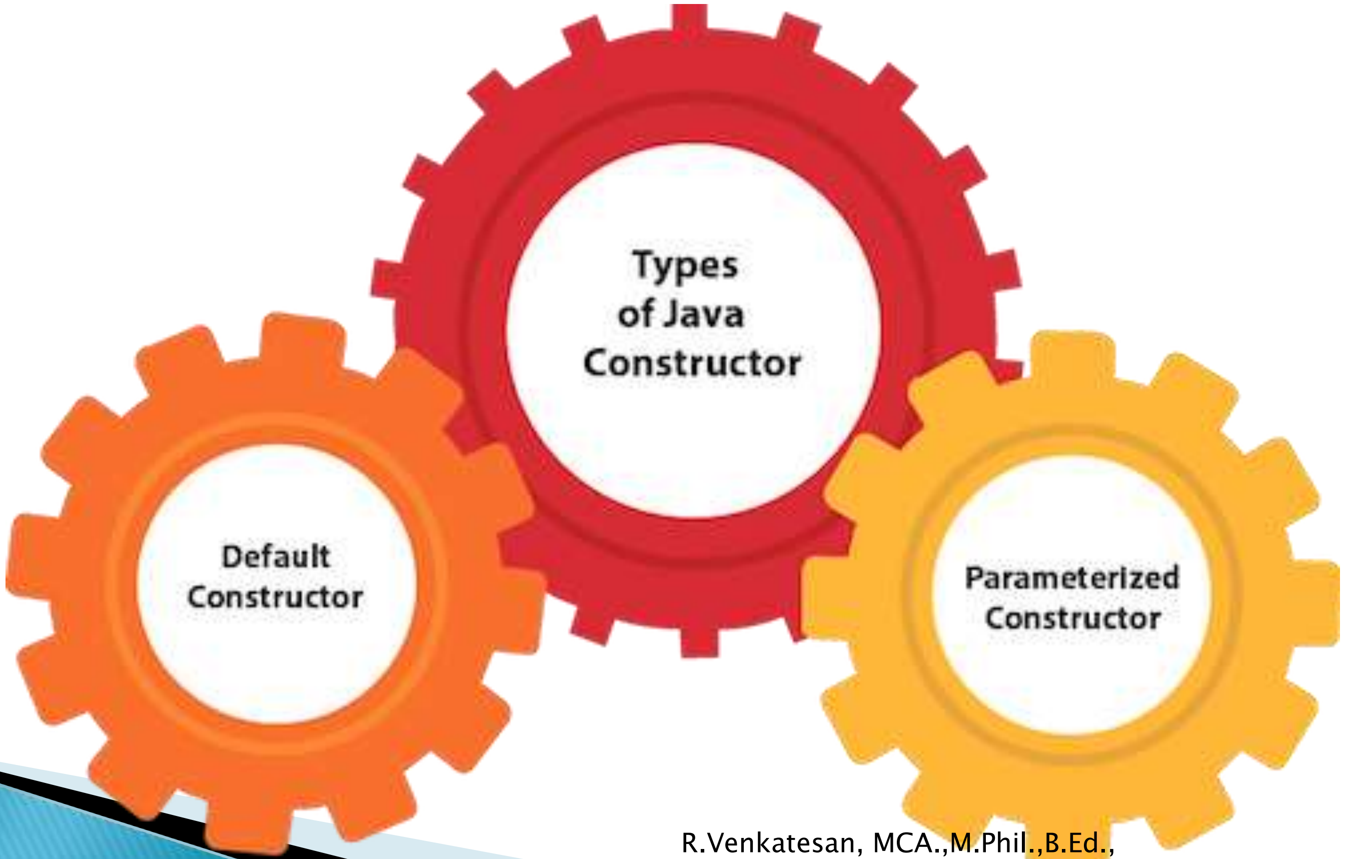
- In Java, a constructor is a block of codes similar to the method.
- It is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class.
- In such case, Java compiler provides a default constructor by default.

- **Note:** It is called constructor because it constructs the values at the time of object creation.
- It is not necessary to write a constructor for a class.
- It is because java compiler creates a default constructor if your class doesn't have any.

- **Rules for creating Java constructor**
  There are two rules defined for the constructor.
- Constructor name must be the same as its class name
- A Constructor must have no explicit return type.
- A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

- Default constructor (no-arg constructor)
- Parameterized constructor



Java Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- **Syntax of default constructor:**
- <class_name>()
- {
- }

Example of default constructor

- In this this example, we are creating the **no-arg constructor** in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1
{
//creating a default constructor
Bike1(){
System.out.println("Bike is created");
}
//main method
public static void main(String args[])
{
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

o/p

```
Bike is created
```

Java Parameterized Constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.

- **Why use the parameterized constructor?**
- The parameterized constructor is used to provide different values to distinct objects.
- However, you can provide the same values also.

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4
{
   int id;
   String name;
   //creating a parameterized constructor
   Student4(int i,String n){
   id = i;
   name = n;
   }
   //method to display the values
   void display()
{
System.out.println(id+" "+name);
}

   public static void main(String args[])
{
   //creating objects and passing values
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   //calling method to display the values of object
   s1.display();
   s2.display();
  }
}
```

o/p

```
111 Karan
 222 Aryan
```

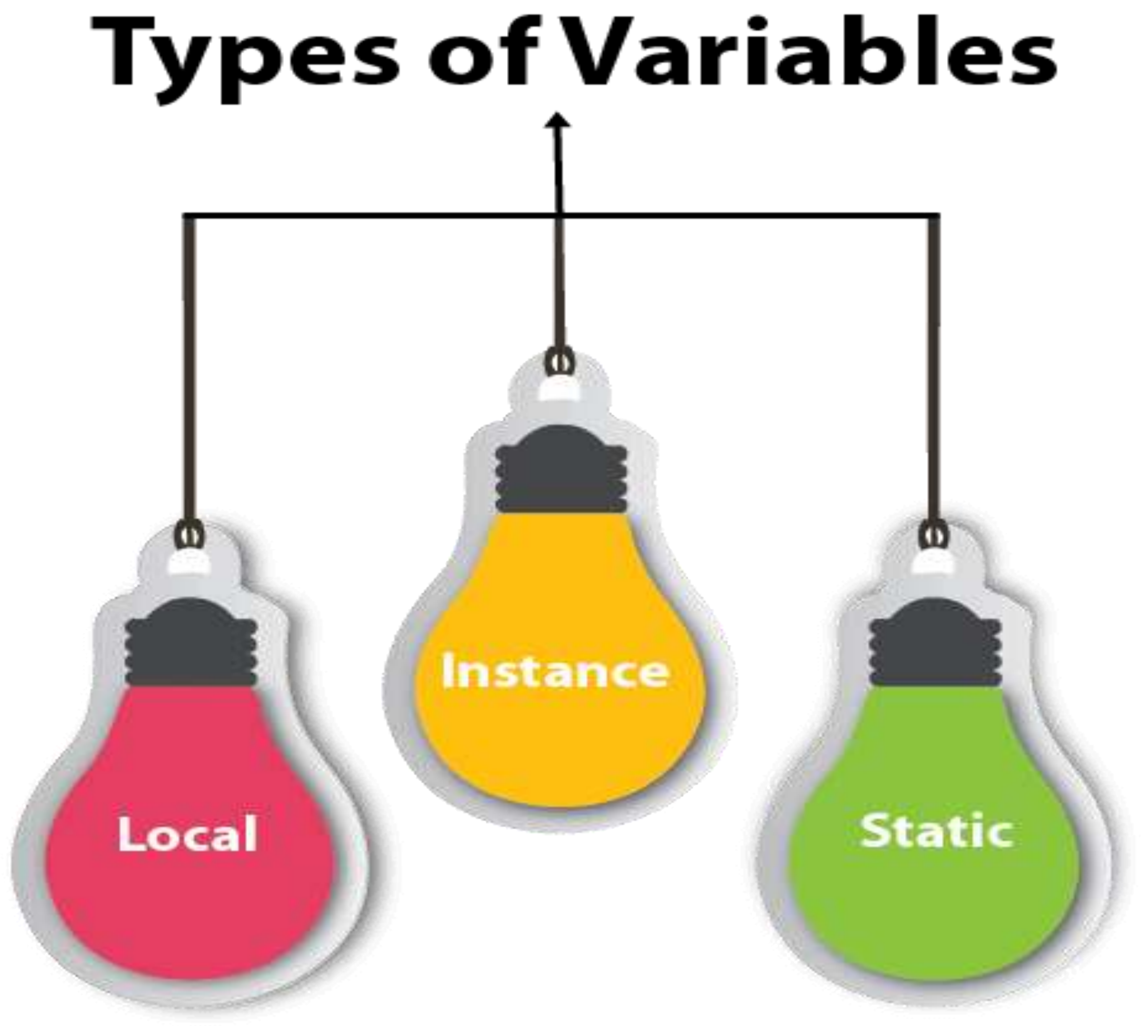
Java Variables

- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.
- Variable is a name of memory location.
- There are three types of variables in java: **local, instance and static.**
- There are two types of data types in Java: **primitive and non-primitive.**

Variable

- A variable is the name of a reserved area allocated in memory.
- In other words, it is a name of the memory location.
- It is a combination of "vary + able" which means its value can be changed.

- **int** data=50;//Here data is variable


Types of Variables
Instance
Local
Static

Local Variable

- A variable declared inside the body of the method is called local variable.
- You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.

Instance Variable

- A variable declared inside the class but outside the body of the method, is called an instance variable.
- It is not declared as static.
- It is called an instance variable because its value is instance-specific and is not shared among instances.

Static variable

- A variable that is declared as static is called a static variable.
- It cannot be local.
- You can create a single copy of the static variable and share it among all the instances of the class.
- Memory allocation for static variables happens only once when the class is loaded in the memory.

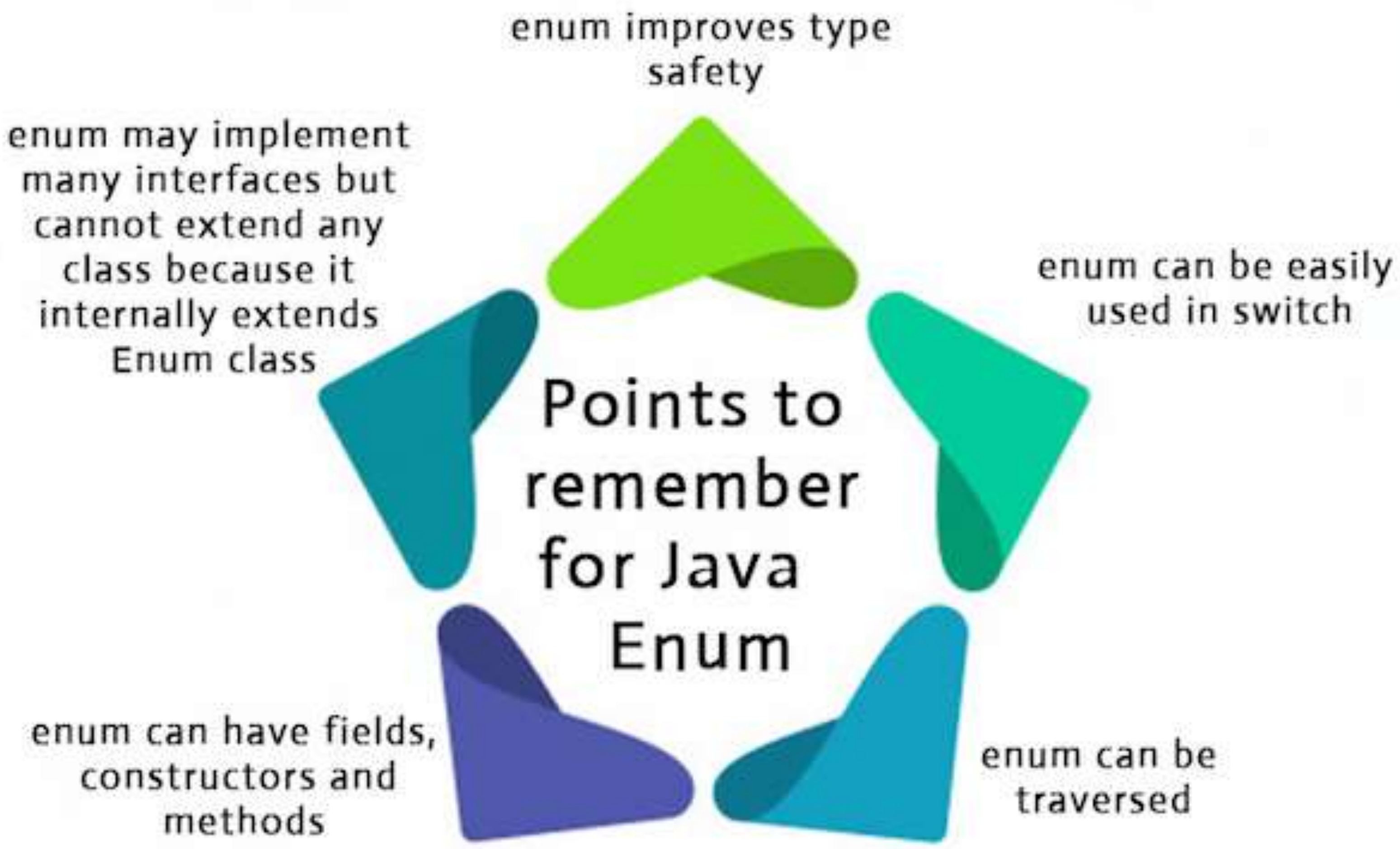
```
public class A
{
    static int m=100;//static variable
    void method()
    {
        int n=90;//local variable
    }
    public static void main(String args[])
    {
        int data=50;//instance variable
    }
}  //end of class
```

Java Enums

- The **Enum in Java** is a data type which contains a fixed set of constants.
- It can be used for days of the **week** (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc.
- According to the Java naming conventions, we should have all constants in capital letters.
- So, we have enum constants in capital letters.

Points to remember for Java Enum

- Enum improves type safety
- Enum can be easily used in switch
- Enum can be traversed
- Enum can have fields, constructors and methods
- Enum may implement many interfaces but cannot extend any class because it internally extends Enum class


enum improves type safety
enum can be easily used in switch
enum can be traversed
enum can have fields, constructors and methods
enum may implement many interfaces but cannot extend any class because it internally extends Enum class
Points to remember for Java Enum

Simple Example of Java Enum

```
class EnumExample1
{
//defining the enum inside the class
public enum Season
{
 WINTER, SPRING, SUMMER, FALL
 }
//main method
public static void main(String[] args)
 {
//traversing the enum
for (Season s : Season.values())
System.out.println(s);
}
}
```

Output:

```
WINTER
 SPRING
 SUMMER
FALL
```

OOPs in Java: Encapsulation, Inheritance, Polymorphism

- **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc.
- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects.
- It simplifies software development and maintenance by providing some concepts:
- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Encapsulation

- It is also a feature of OOP.
- It is used to bind up the data into a single unit called class.
- It provides the mechanism which is known as **data hiding**.
- It is an important feature of OOPs.
- It prevents to access data members from the outside of the class.
- It is also necessary from the security point of view.

Inheritance in Java

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a **parent object.**
- It is an important part of OOPs (Object Oriented programming system).
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and fields of the **parent class**.
- Moreover, you can add new methods and fields in your current class also.
- Inheritance represents the **IS-A relationship** which is also known as a ***parent-child* relationship**.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

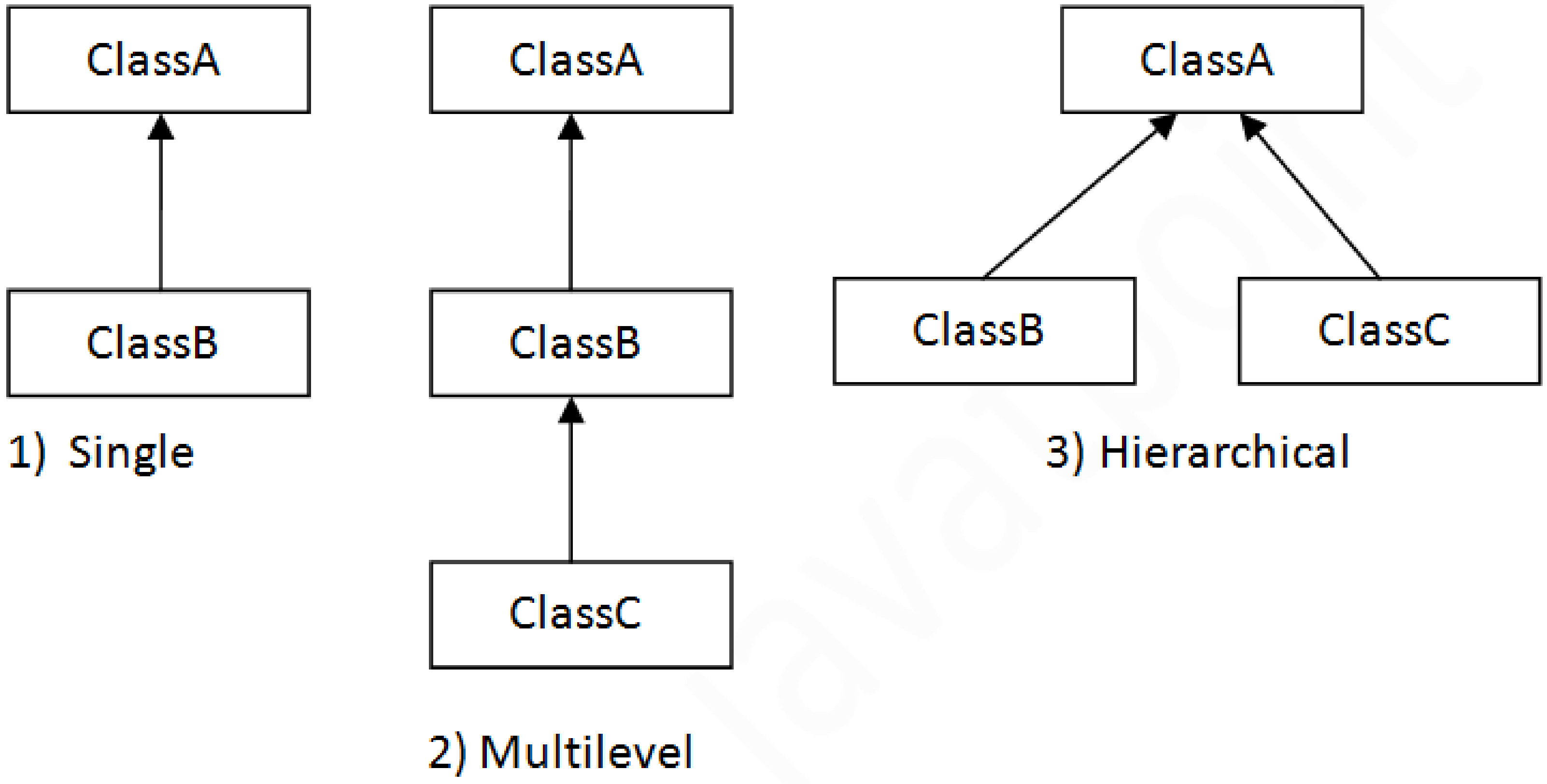
- **Class:** A class is a group of objects which have common properties.
- It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class.
- It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Super class is the class from where a subclass inherits the features.
- It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class.
- You can use the same fields and methods already defined in the previous class.

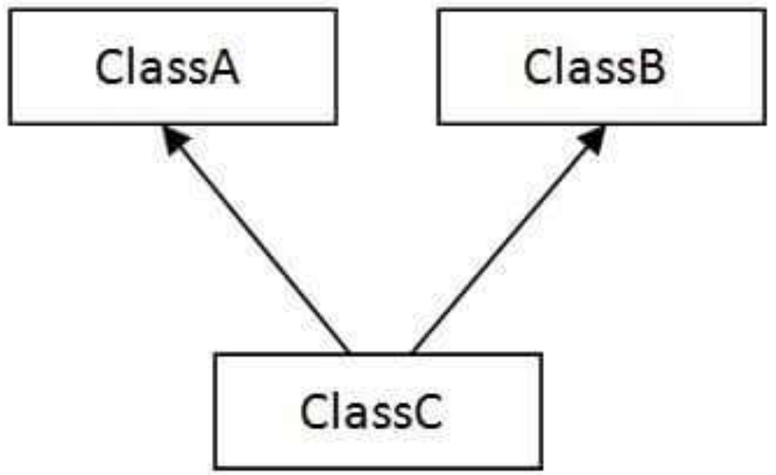
The syntax of Java Inheritance

- **class** Subclass-name **extends** Superclass-name
- {
-   //methods and fields
- }

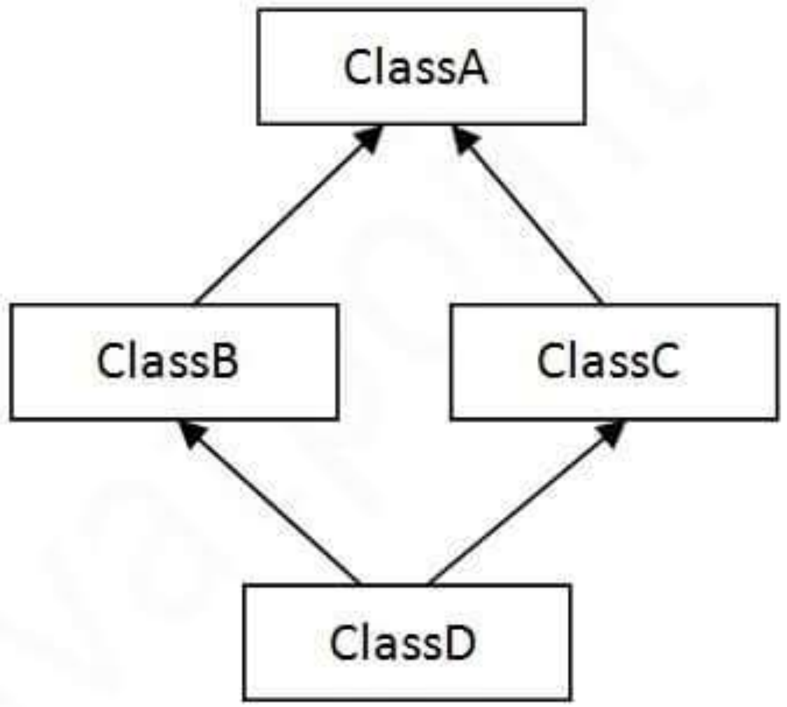
- The **extends keyword** indicates that you are making a **new class** that derives from an existing class.
- The meaning of "extends" is to increase the functionality.

Types of inheritance in java





4) Multiple



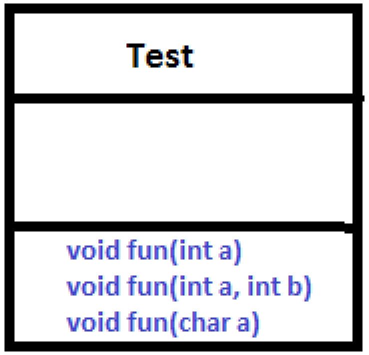
5) Hybrid

Polymorphism in Java

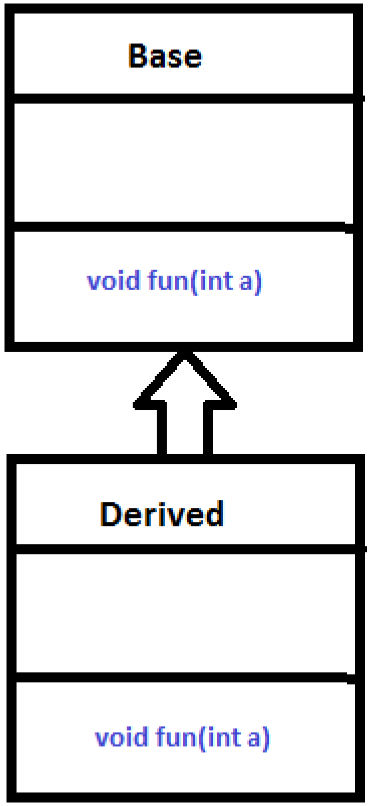
- **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*.
- Polymorphism is derived from 2 Greek words: poly and morphs.
- The word "poly" means **many** and "morphs" means **forms**.
- So polymorphism means **many forms**.
- There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.
- We can perform polymorphism in java by method overloading and method overriding.

Compile time Polymorphism

- **Compile-time polymorphism**: It is also known as static polymorphism.
- This type of polymorphism is achieved by function overloading or operator overloading.
- But **Java doesn’t support the Operator Overloading**.



Overloading



Overriding

- **Method Overloading**: When there are multiple functions with same name but different parameters then these functions are said to be **overloaded.**
- Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments.**

Runtime polymorphism

- It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime.
- This type of polymorphism is achieved by Method Overriding.
- Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class.
- That base function is said to be **overridden**.