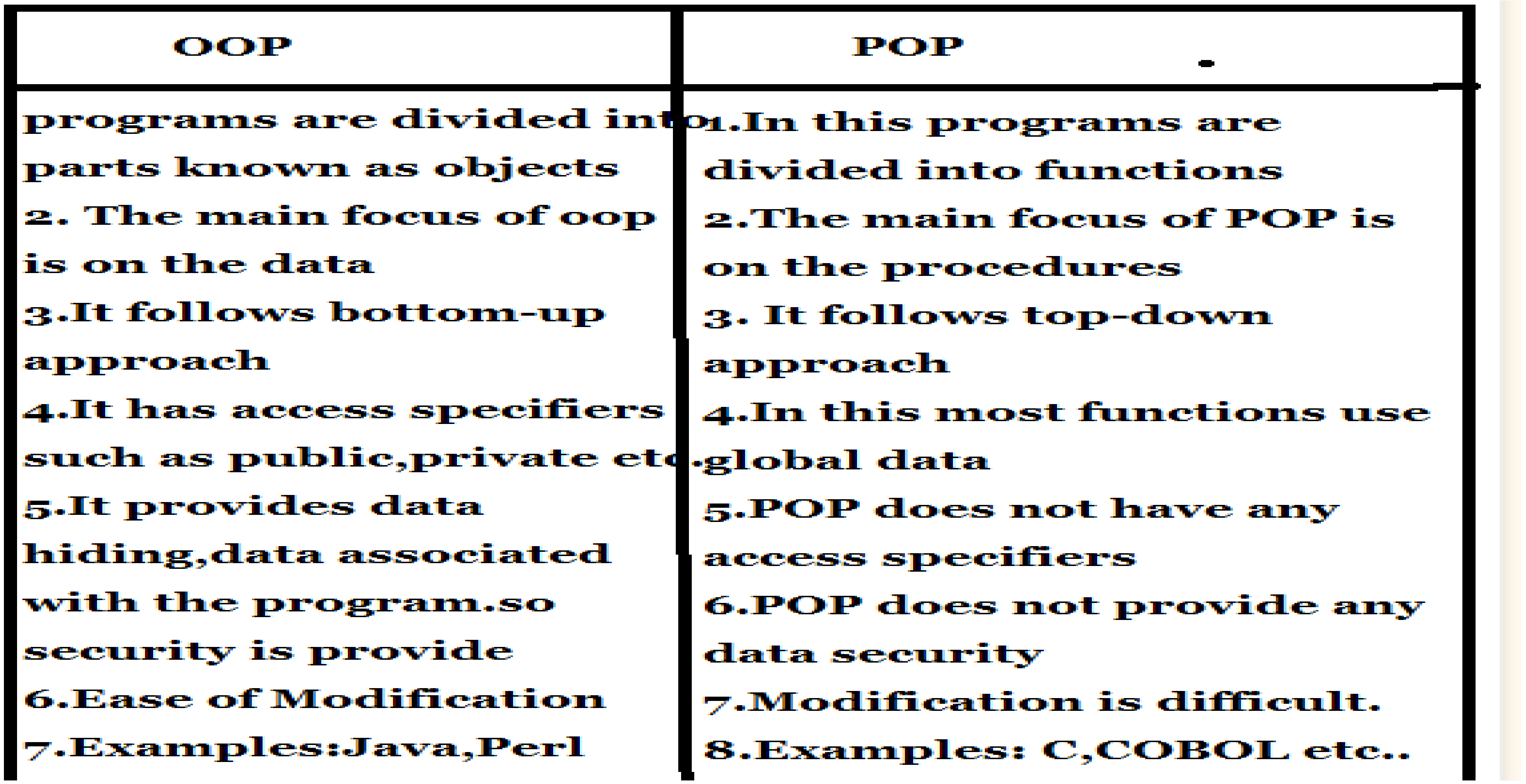
Unit 1: Fundamentals of Java Programming

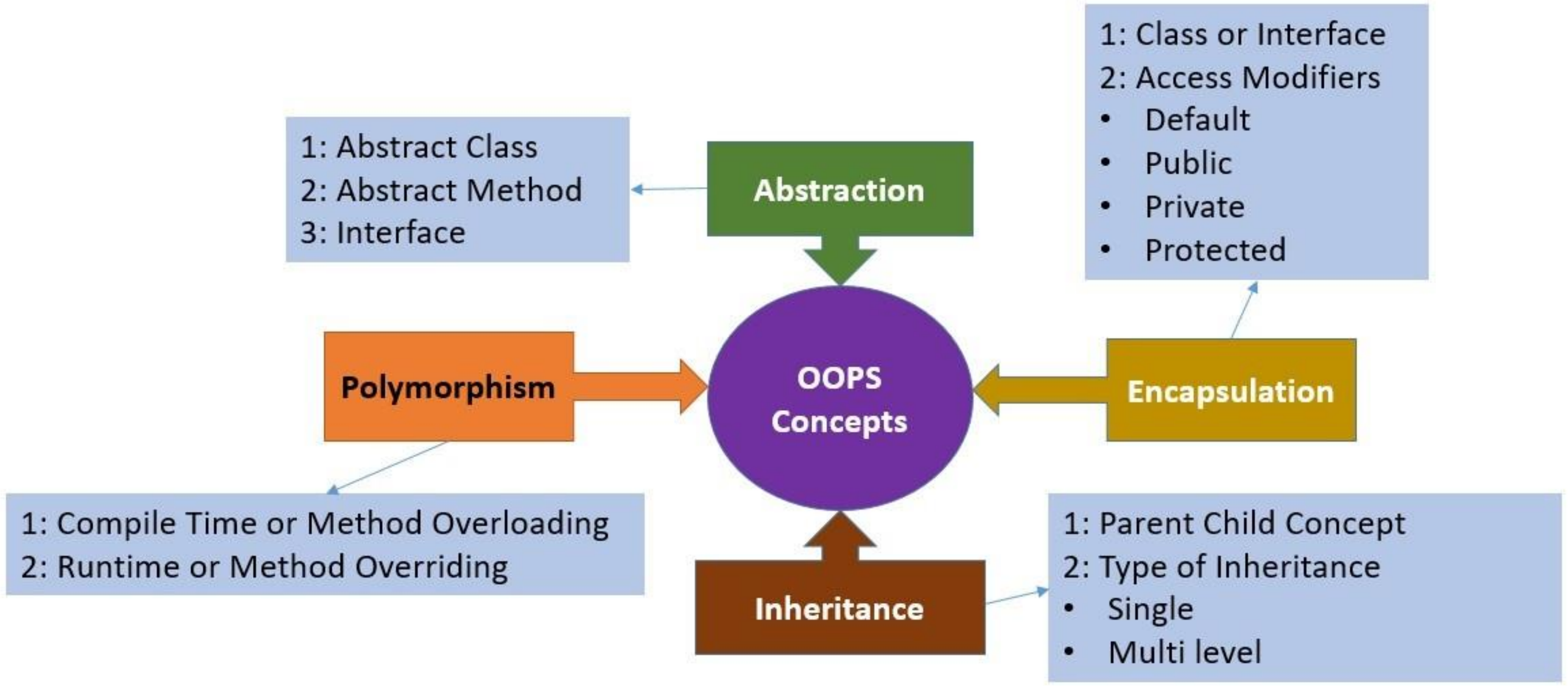
Classes, Instance variables, Methods, Constructors, Access Specifiers, Abstract Classes and Wrapper Classes, Inheritance, Polymorphism Method Overriding, final, super and this keyword, Creating user defined package, Access control protection, Defining interface, Implementing interface

- Difference between POP and OOP



OOP	POP
programs are divided into parts known as objects	1.In this programs are divided into functions
2. The main focus of oop is on the data	2.The main focus of POP is on the procedures
3.It follows bottom-up approach	3. It follows top-down approach
4.It has access specifiers such as public,private etc.	4.In this most functions use global data
5.It provides data hiding,data associated with the program.so security is provide	5.POP does not have any access specifiers
	6.POP does not provide any data security
6.Ease of Modification	7.Modification is difficult.
7.Examples:Java,Perl	8.Examples: C,COBOL etc..

Need of OOP



OOPS Concepts

The Bytecode

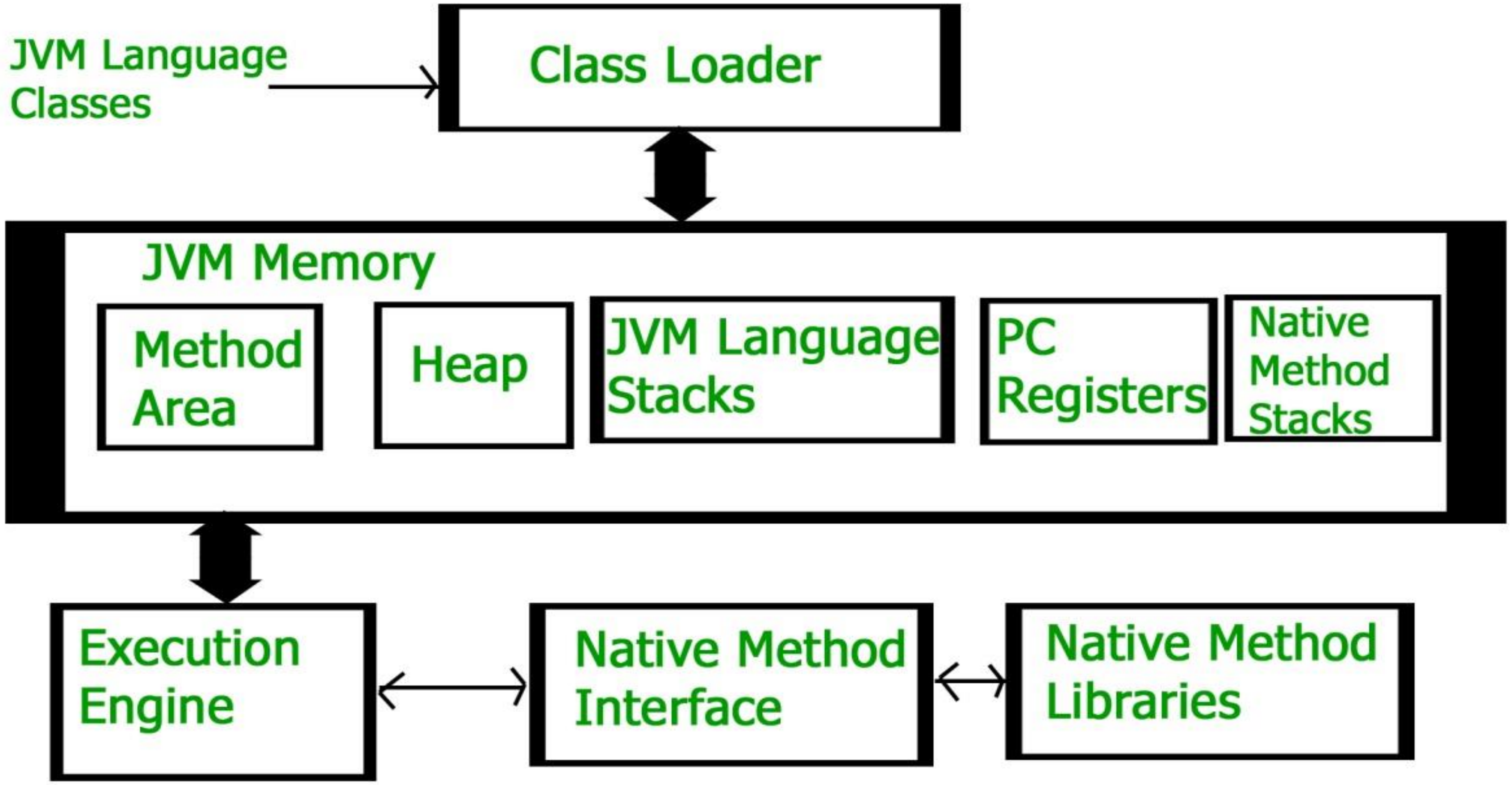
- highly optimized set of instructions designed to be executed by the Java run-time
- Java applications are called WORA (Write Once Run Everywhere).

Out of syllabus

JVM Architecture

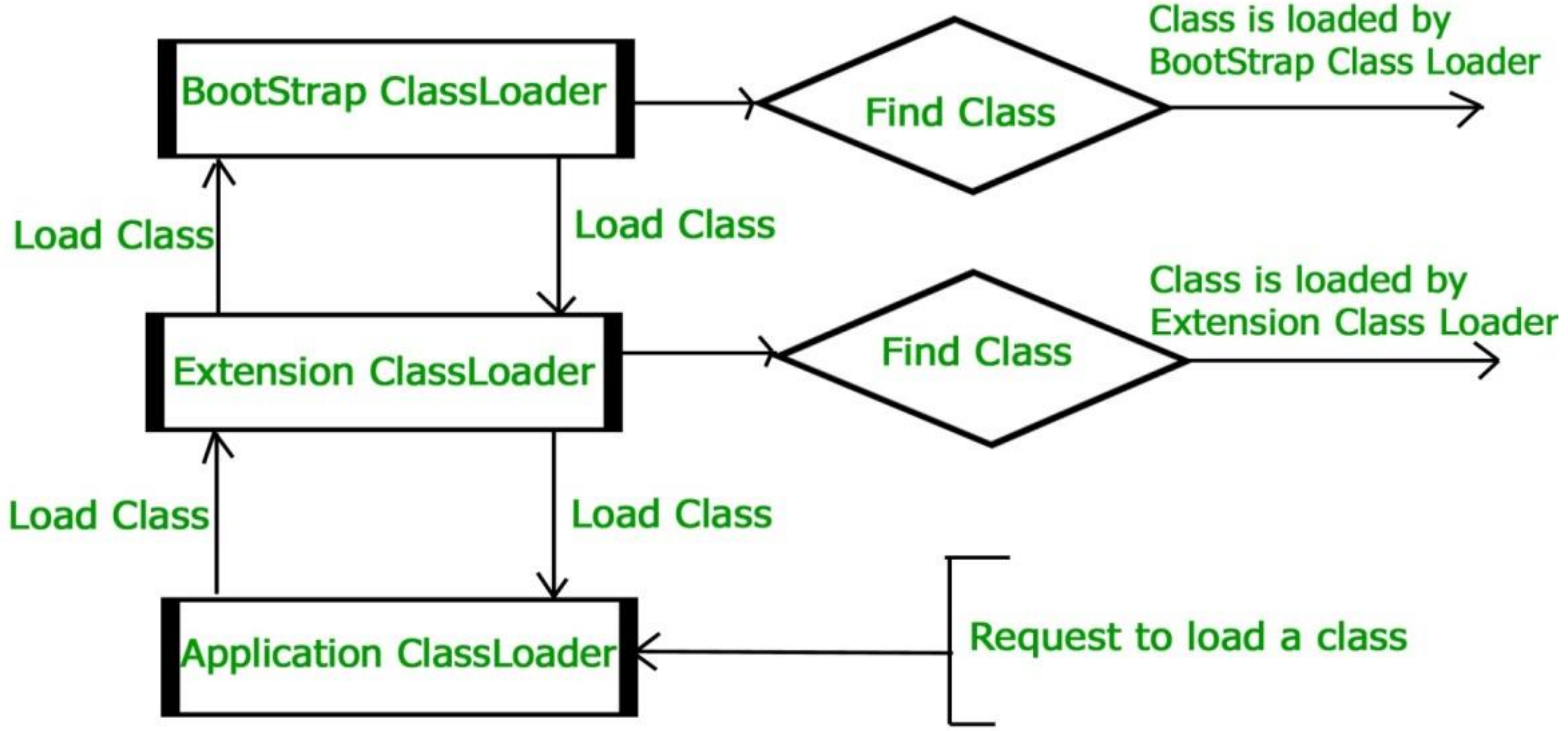
three activities

- Loading
- Linking
- Initialization

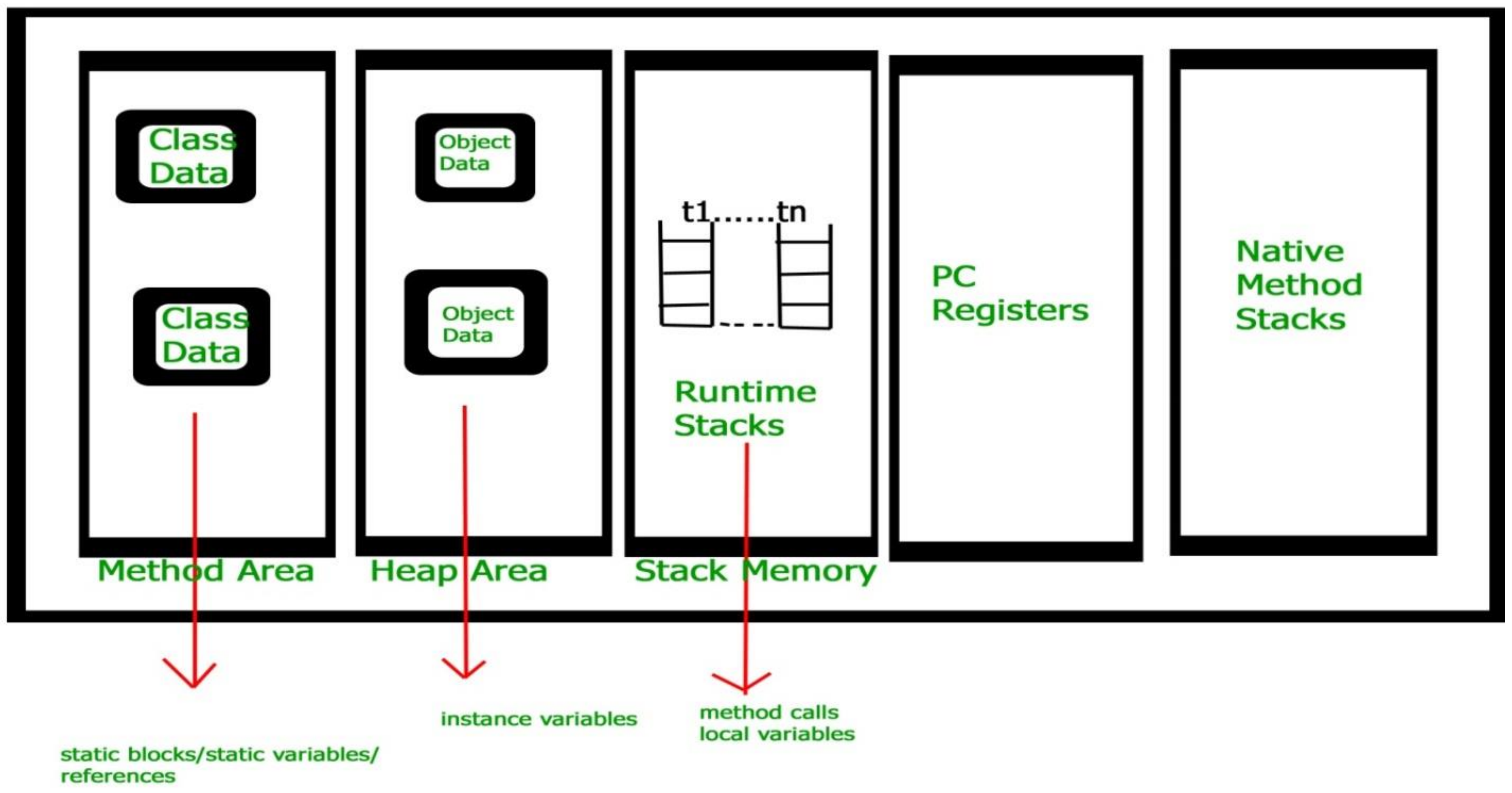


Out of syllabus

Delegation-Hierarchy principle to load classes



JVM Memory


Class Data
Class Data
Object Data
Object Data
t1......tn
Runtime Stacks
PC Registers
Native Method Stacks
Method Area
Heap Area
Stack Memory
instance variables
method calls
local variables
static blocks/static variables/
references

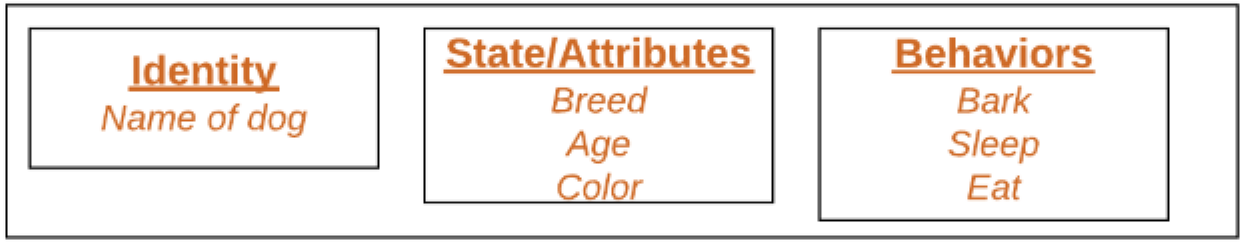
Out of syllabus

Why java programs are secured

Class

- **Modifiers** : A class can be public or has default access
- **Class name:** The name should begin with a initial letter
- **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- **Body:** The class body surrounded by braces, { }.

Object



- **State** : It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

Explain Main methods

```
/*
This is a simple Java program. Call this file "Example.java".
*/
class Example {
// Your program begins with a call to main().
public static void main(String args[]) {
System.out.println("This is a simple Java program.");
}
}
```

can you run a Java program without a main method,

- Applets
- Servlet,
- MIDlet,

- **Can you run a Java program without main method in Core Java**, not on managed with above environment

public class Java without Main

```
public class JavaAppWithoutMain {
static
{
System.out.println("HelloWorld, Java
progarm without main method");
int x = 20;
// Can initialize static variables
System.out.println("Variable x : " + x);
}}
```

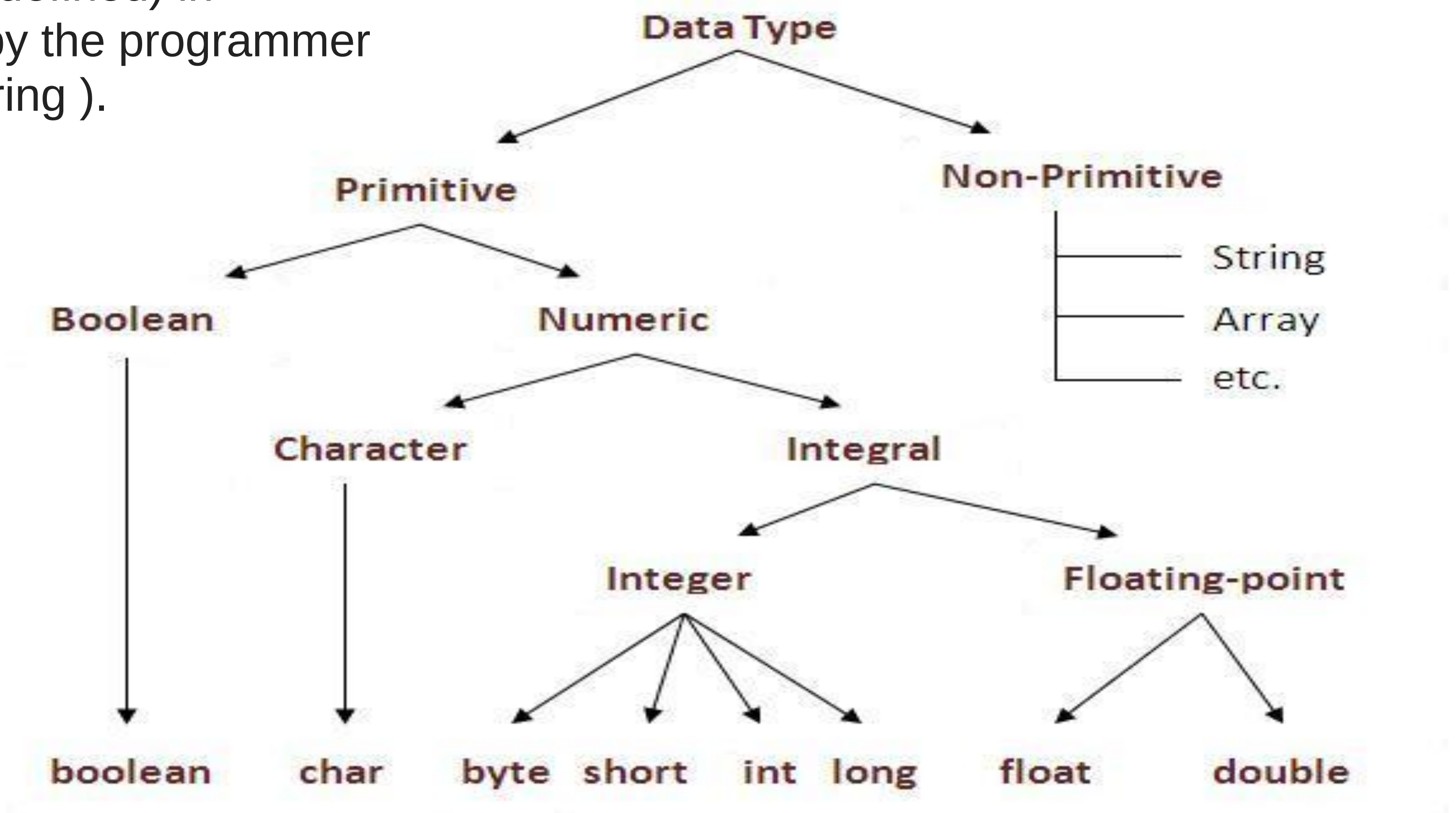
```
abstract class Test extends
javafx.application.Application
{
static
{
System.out.println("hello");
System.exit(0);
}}
```

Constructor

- Simple constructor
- Parameterized constructor
- Constructor overloading
- Constructor chaining
- Copy constructor

Data type

Primitive types are predefined (already defined) in Java. **Non-primitive types** are created by the programmer and is not defined by Java (except for String).



Ways to create object of a class

- **Using new keyword: Test t1; t1= new Test();**

 Test t = new Test();
- **Using Class.forName(String className) method** :

Test Obj =(Test)Class.forName("com.p1.test").newInstance();
- **Using clone() method**: Test t1= (Test) t.clone();
- **Deserialization** :
- String filename="c:/sdfs";

FileInputStream file = new FileInputStream(filename);

ObjectInputStream in = new ObjectInputStream(file);

Object obj = in.readObject();

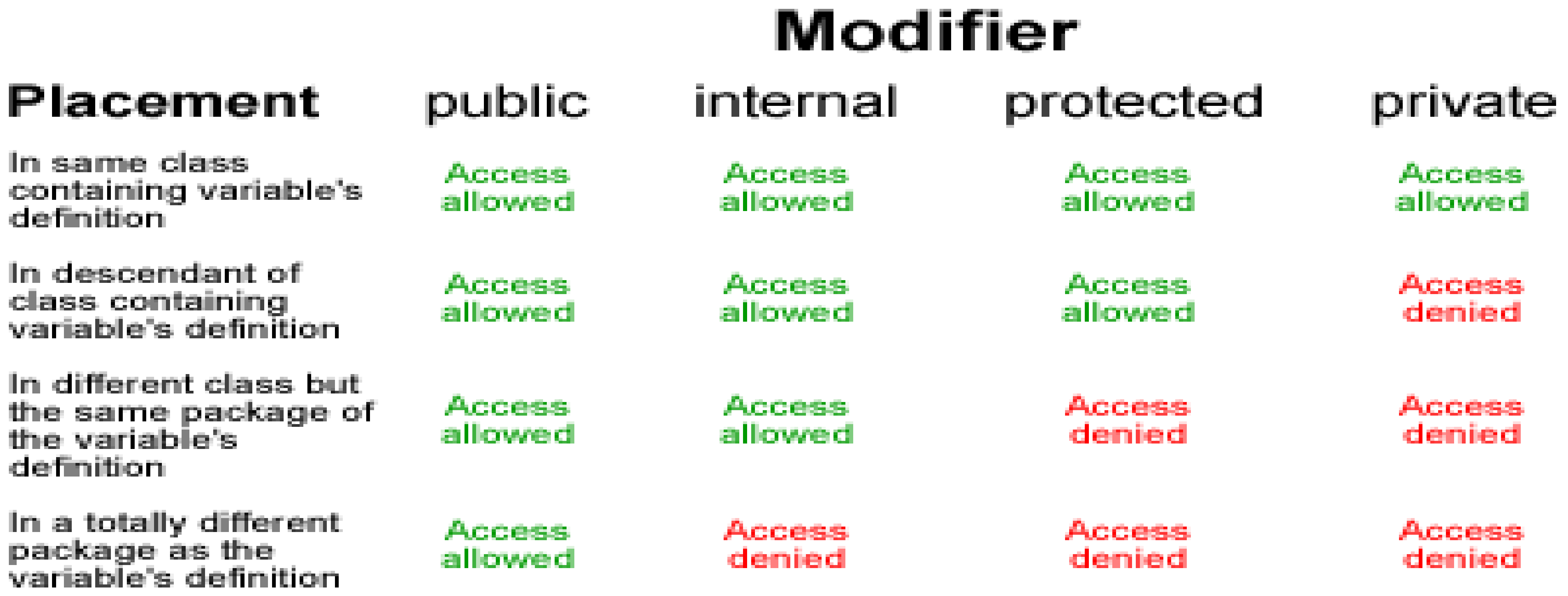
Variables

- Instance Variable
- Static variable:
  - They can only call other **static** methods.
  - They must only access **static** data.
  - They cannot refer to **this** or **super** in any way.

Difference between scanner and bufferreader

•Scanner is used for parsing tokens from the contents of the stream while BufferedReader just reads the stream and does not do any special parsing.
•BufferedReader is synchronized and Scanner is not, Use BufferedReader if you're working with multiple threads.
•Scanner hides IOException while BufferedReader throws it immediately.
•the Scanner has a smaller buffer (1024 chars) as opposed to the BufferedReader (8192 chars), but it's more than sufficient.
A scanner however is not thread safe, it has to be externally synchronized

Access specifier



	Modifier			
Placement	public	internal	protected	private
In same class containing variable's definition	Access allowed	Access allowed	Access allowed	Access allowed
In descendant of class containing variable's definition	Access allowed	Access allowed	Access allowed	Access denied
In different class but the same package of the variable's definition	Access allowed	Access allowed	Access denied	Access denied
In a totally different package as the variable's definition	Access allowed	Access denied	Access denied	Access denied

Abstract Classes and Wrapper Classes

Abstract class

There are two ways to achieve abstraction in java

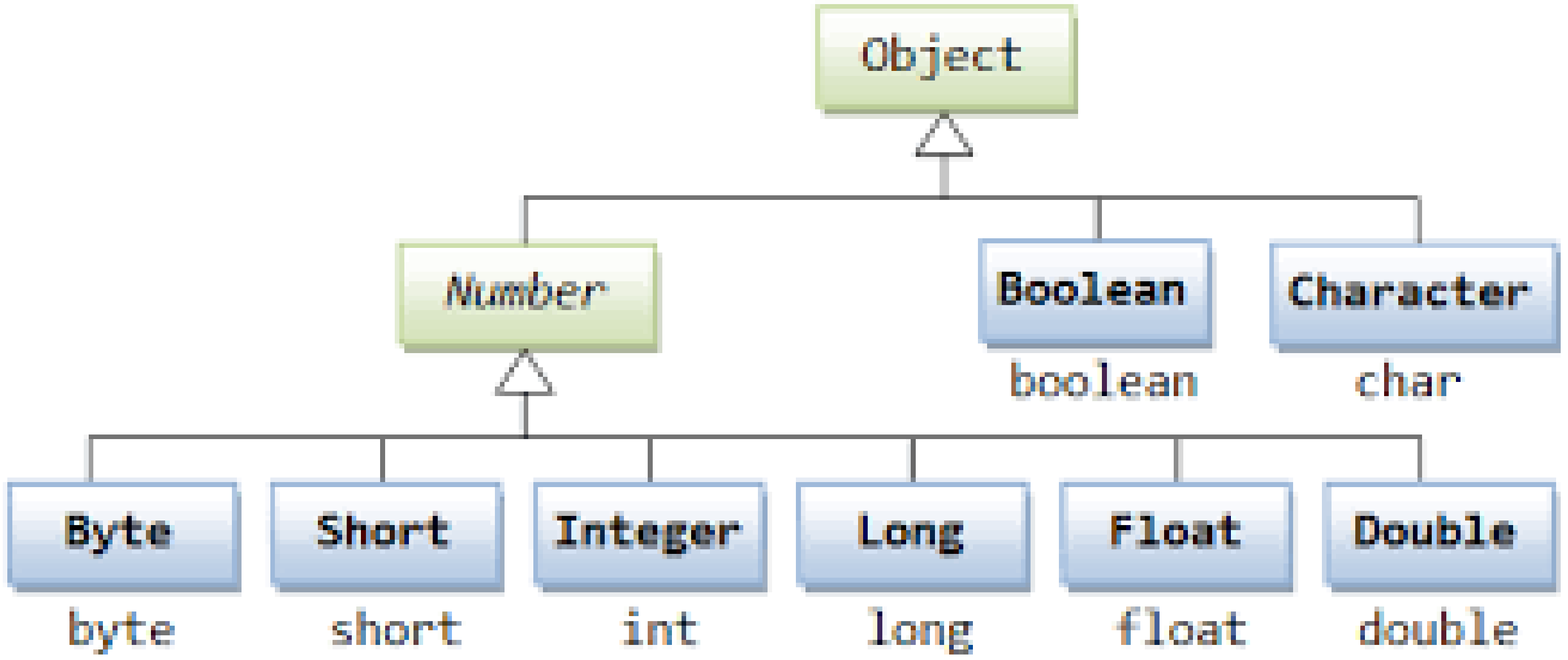
- Abstract class (0 to 100%)
- Interface (100%)

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

Rules for abstract class

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Wrapper class



Inheritance

- Inheritance in Java is the method to create a hierarchy between classes by inheriting from other classes.
- Simple inheritance
- Single inheritance
- Multilevel inheritance
- Hierarchical inheritance
- Has- a relationship
- aggregation

Polymorphism

- **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*.
- 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.
- If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

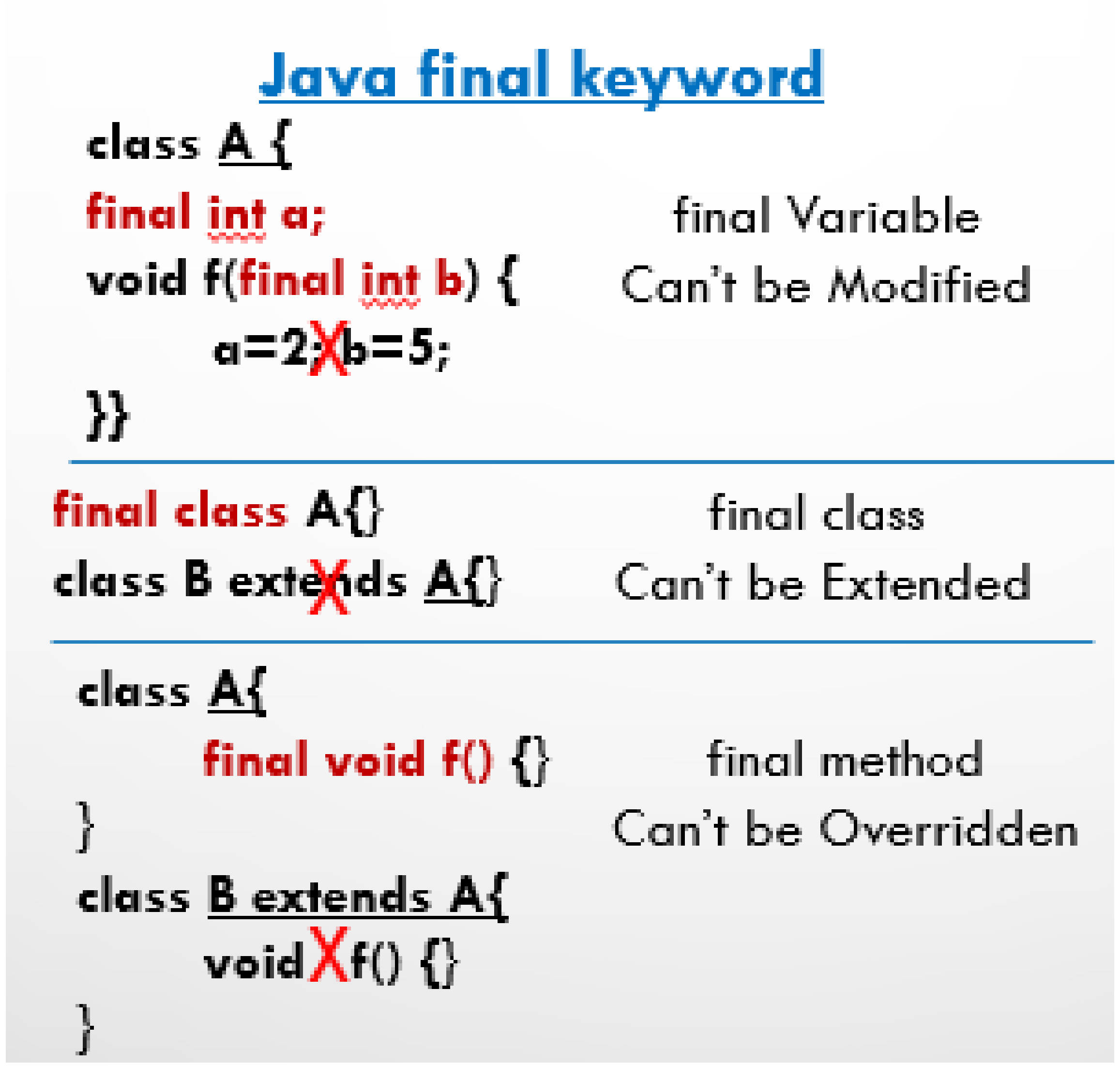
JAVA Method overriding

- Usage of Java Method Overriding
  - Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
  - Method overriding is used for runtime polymorphism

- Rules for Java Method Overriding
  - The method must have the same name as in the parent class
  - The method must have the same parameter as in the parent class.
  - There must be an IS-A relationship (inheritance).

final,

- Stop value change
- Stop inheritance
- Stop method overriding



Java final keyword

```
class A {
final int a;
void f(final int b) {
        a=2; b=5;
}}
```

final Variable
Can't be Modified

```
final class A{}
class B extends A{}
```

final class
Can't be Extended

```
class A{
        final void f() {}
}
class B extends A{
        void f() {}
}
```

final method
Can't be Overridden

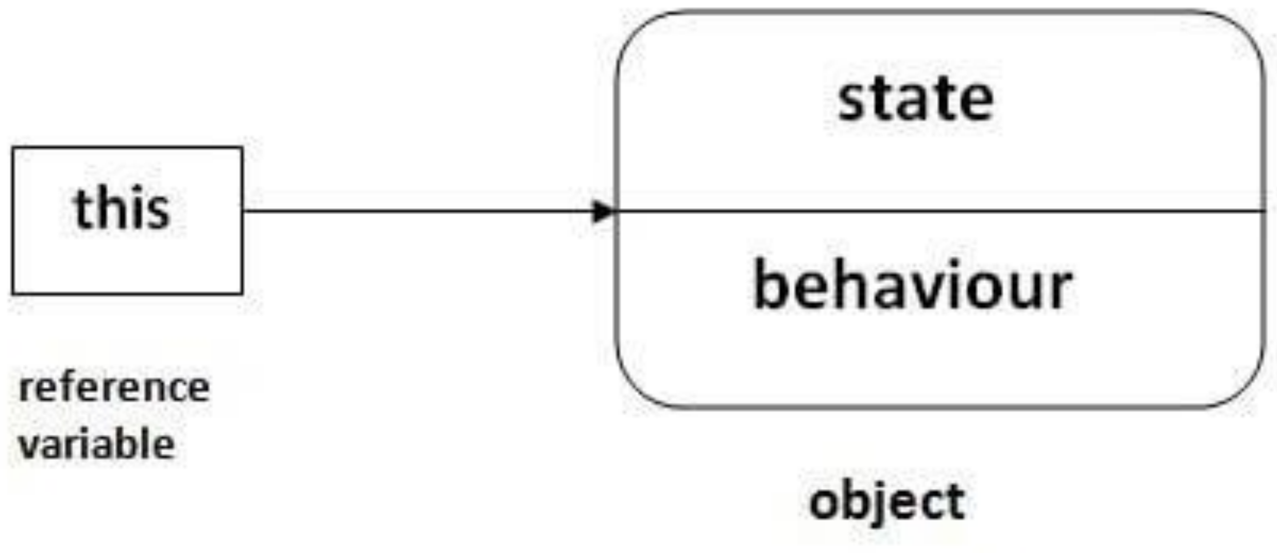
super

- The **super** keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor. The most common use of the **super** keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.
- The **super** keyword in **Java** is a reference variable that is used to refer parent class objects. The **super() in Java** is a reference variable that is used to refer parent class constructors. **super** can be used to call parent class' variables and methods.
- Because if you use this**() and super() together** in a constructor it **will** give compile time error. Because this**() and super()** must be the first executable statement. If you write this**()** first than **super() will** become the second statement and vice-versa. That's why **we can**'t use this**() and super() together**

The use of super keyword

- 1) To access the data members of parent class when both parent and child class have member with same name
  2) To explicitly call the no-arg and parameterized constructor of parent class
  3) To access the method of parent class when child class has overridden that method.

this keyword



Package concept

- Preventing naming conflicts
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- Providing controlled access
- data encapsulation

Built-in Packages

1) **java.lang:** Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
2) **java.io:** Contains classed for supporting input / output operations.
3) **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
4) **java.applet:** Contains classes for creating Applets.
5) **java.awt:** Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).
6) **java.net:** Contain classes for supporting networking operations.

User-defined packages

The Directory Structure of Packages

- when a class is placed in a package −
  - The name of the package becomes a part of the name of the class, as we just discussed in the previous section.
  - The name of the package must match the directory structure where the corresponding bytecode resides.

Static import

- It is a feature introduced in **Java** programming language (versions 5 and above) that allows members (fields and methods) defined in a class as public **static** to be used in Java code without specifying the class in which the field is defined.
- The **import** allows the **java** programmer to access classes of a package without package qualification whereas the **static import** feature allows to access the **static** members of a class without the class qualification.
- The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.
- Adv: Less coding is required if you have access any static member of a class oftenly.
- Dis: If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

ways to access the package from outside the package

- import package.*;
- import package.classname;
- fully qualified name.

Package class

- Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.

Access control protection

- Packages adds another dimension to the access control.
- Java provides many levels of protection to allow fine-grained control over visibility of the variables and methods within classes, subclasses, and packages.
- Classes and packages are means of encapsulating and containing the name space and scope of the variables and methods.
- Packages behaves as containers for classes and other subordinate packages.

Class Members Visibility

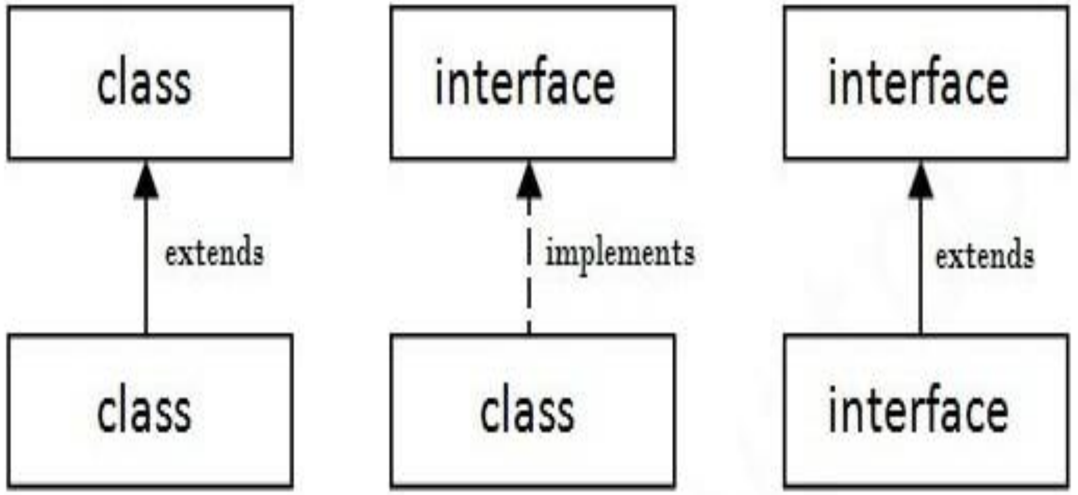
- Anything declared as **public** can be accessed from anywhere.
- Anything declared as **private** can't be seen outside of its class.

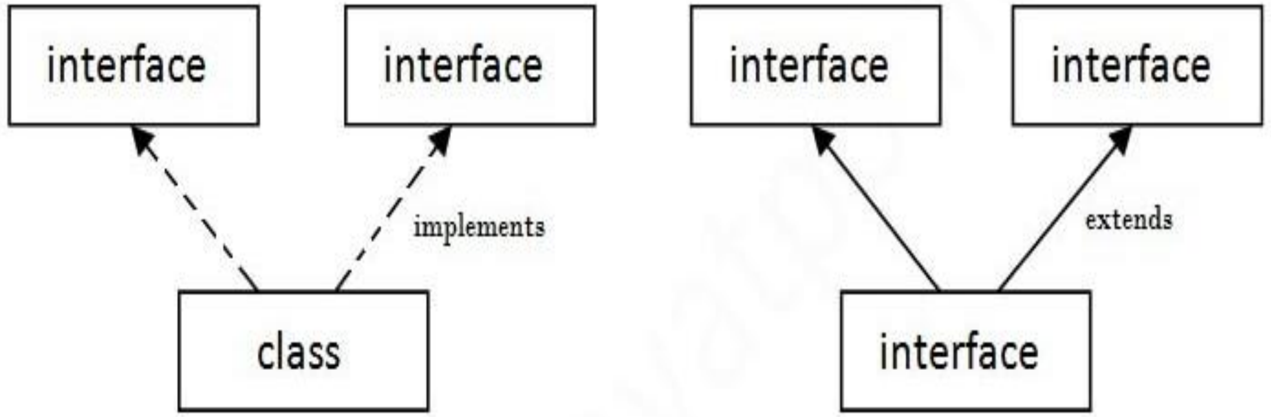
Class Member Access

	Private	Protected	Public	No Modifier
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	Yes	Yes	No
Different package non-subclass	No	No	Yes	No

Interface

- 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.
- Java Interface also **represents the IS-A relationship**. It cannot be instantiated just like the abstract class.
- Since Java 8, we can have **default and static methods** in an interface.
- Since Java 9, we can have **private methods** in an interface.





Multiple Inheritance in Java

Interfaces in Java

- An interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).
- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
- If a class implements this interface, then it can be used to sort a collection.

Why do we use interface ?

- It is used to achieve total abstraction.
- Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance.
- It is also used to achieve loose coupling.
- Interfaces are used to implement abstraction. why use interfaces when we have abstract classes?
- The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

Interface inheritance and Default Method in Interface

- An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc.
- They are used to provide some essential information to the JVM so that JVM may perform some useful operation. marker or tagged interface.java
- List’ or ‘Collection’ interfaces do not have ‘forEach’ method declaration. Thus, adding such method will simply break the collection framework implementations. Java 8 introduces default method so that List/Collection interface can have a default implementation of forEach method, and the class implementing these interfaces need not implement the same
- Static methods in interface

Nested Interface in Java

- Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
- Nested interfaces are declared static implicitely.

nested interface which is declared within the class

a class inside the interface?

- Yes, If we define a class inside the interface, java compiler creates a static nested class. Let's see how can we define a class within the interface: