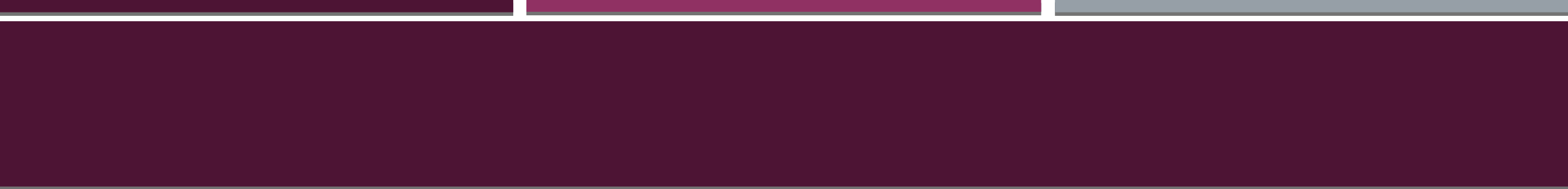


# SOFTWARE DESIGN & ARCHITECTURE

USAMA MUSHARAF

*LECTURER (Department of Computer Science)*


*FAST-NUCES PESHAWAR*

- 
- Introduction to Java Programming
  - Environment Setup
  - Syntax and language constructs
  - Some Basic Programming
    - Input / Output
    - GUI (Input Output in Dialog Box)
    - Classes and Objects creation
    - Programming with Multiple Classes
    - Functions
    - Constructors

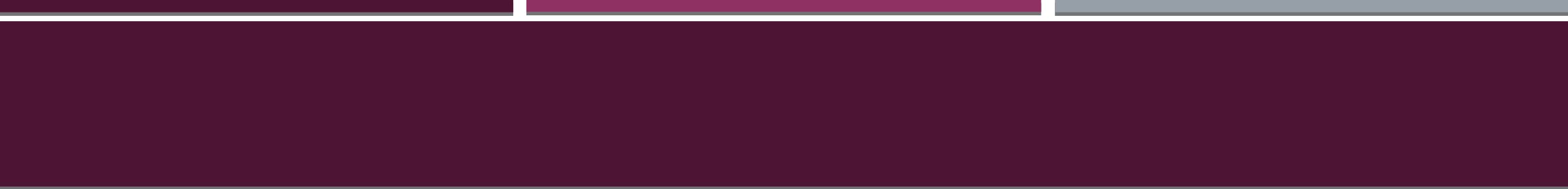


# INTRO TO JAVA PROGRAMMING





One of the most widely used computer programming language.

- 
- Java developed by James Gosling at sun micro-systems.
  - On January 27, 2010, Sun was acquired by Oracle Corporation for US \$7.4 billion, based on an agreement signed on April 20, 2009.
  - Pure object oriented language.
  - Java is based on the OOP notions of classes and objects.
  - Java came on the scene in 1995.

- 
- 
- Before that C and C++ dominated the software development.

#### Disadvantages of C/C++:

- No garbage collector causes memory leakages.
- Not great support of built in libraries.

## JAVA:

- Simple.  
(Very simple syntax similar to c/c++.)
- No operator overloading.
- No multiple Inheritance.
- No pointers.
- Great support of built-in libraries.



- Garbage Collector.

- Programmer Efficiency

(Building an application in java takes less time than in C/C++)

- Support for Web Application.

- Multi-Threaded.

- Robust/Secure.





- Network Oriented

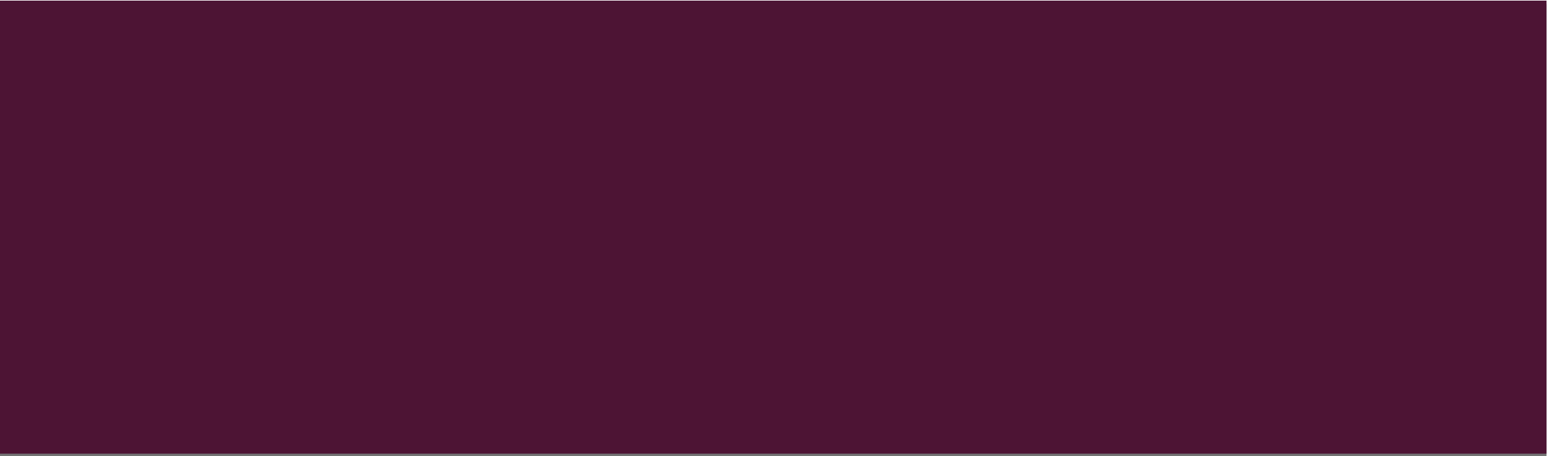
- Portable/Machine Independent (Write Once Run Anywhere).

Disadvantages:

- C++ is faster than java because java creates intermediate code first and then compile it into target code.



# JAVA VIRTUAL MACHINE







When you write a program in C++ it is known as source code.

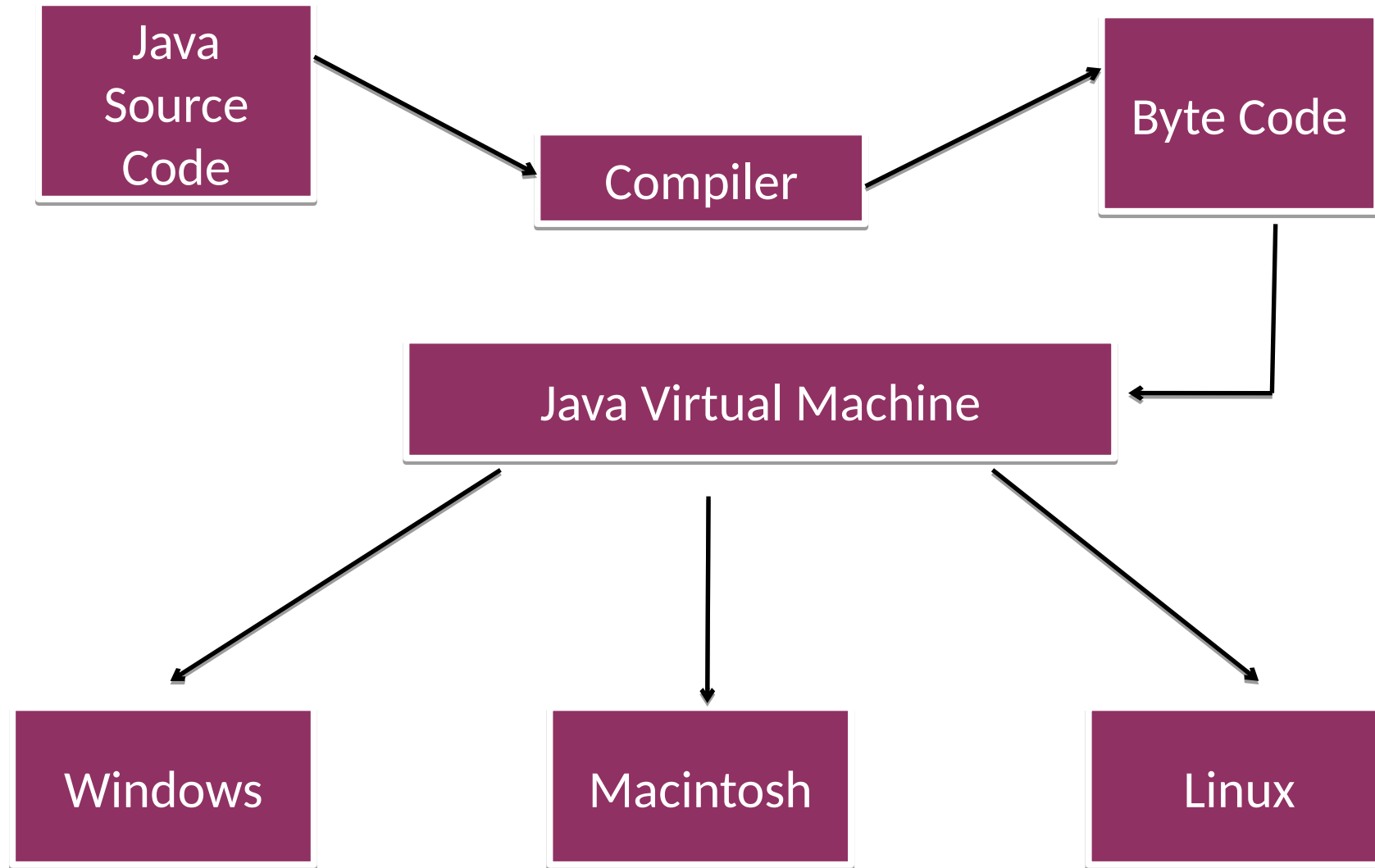
The C++ compiler converts this source code into the machine code of underlying system (e.g. Windows).

If you want to run that code on Linux or on some other operating system you need to recompile it based on the desired compiler.

Due to the difference in compilers, sometimes you need to modify your code.

- 
- Java has the concept of WORA(Write Once Run Anywhere).
  - Java compiler does not compile source code for underlying hardware.
  - Java compiler compiles a code for system software JVM known as byte code.

- 
- The difference with Java is that it uses byte code - a special type of machine code.
  - The JVM executes Java byte codes, so Java byte codes can be thought of as the machine language.
  - JVM are available for almost all operating systems.
  - Java byte code is executed by using any operating system's JVM. Thus achieve portability.





## Windows Installation

- [www.oracle.com](http://www.oracle.com) ---> downloads ---> Java SE (standard edition)
- Download JDK ( Java development kit ) Virtual Machine
- Set path and variables.
- Download Eclipse IDE.

## Linux Installation

### Download Jdk

```
$ sudo dpkg -i jdk-15_linux-x64_bin.deb
```

```
$ sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/jdk-19/bin/java 1
```

```
$ sudo update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/jdk-19/bin/javac 1
```

```
$ sudo update-alternatives --config java
```

```
$ sudo gedit /etc/profile
```

```
Type: JAVA_HOME=/usr/lib/jvm/jdk-19
```

```
$ source /etc/profile
```

Download Eclipse IDE.



## First Simple Java Program:

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

OUTPUT:

Hello World



```
public static void main (string args [])
```

main() is the function from which your program starts.

“void” indicates that main() function does not return anything.

Way of specifying input (often called command-line arguments) at startup of application.

Why public?

It is kept public so that it is accessible from outside.

Remember private methods are only accessible from within the class.



## Why static?

Every Java program starts when the JRE (Java Run Time Environment) calls the main method of that program.

If main is not static then the JRE have to create an object of the class in which main method is present and call the main method on that object (In OOP based languages method are called using the name of object if they are not static).

It is made static so that the JRE can call it without creating an object. Also to ensure that there is only one copy of the main method per class.



# **JAVA BASIC SYNTAX**

---

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods.

- **Object** – Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.
- **Class** – A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

- 
- **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
  - **Instance Variables** – Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

- 
- **Case Sensitivity** – Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
  - **Class Names** – For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
  - **Example:** *class MyFirstJavaClass*
  - **Method Names** – All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
  - **Example:** *public void myMethodName()*

- 
- **Program File Name** – Name of the program file should exactly match the class name.
  - When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).
  - **Example:** Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as '*MyFirstJavaProgram.java*'
  - **public static void main(String args[])** – Java program processing starts from the main() method which is a mandatory part of every Java program.





# **JAVA IDENTIFIERS**



## Java Identifiers:

- All Java components require names. Names used for classes, variables, and methods are called **identifiers**.
- In Java, there are several points to remember about identifiers. They are as follows.
- All identifiers should begin with a letter (A to Z or a to z), currency character (\$) or an underscore (\_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, \$salary, \_value, \_\_1\_value.
- Examples of illegal identifiers: 123abc, -salary.



# **JAVA BASIC DATA TYPES**



Primitive data types are not objects in java.

## Types of Numeric Variables

- Integer Variables
- Floating-Point Variables

### Integer Data types:

The four integer types in java are

- Byte
- Short
- int
- long

---

**Note:** When variable of type ***long*** is declared, then the value assigned to the variable will be appended by L.

e.g. long value=243567L;

## **Floating-Point Variables:**

float

double

Note: The value of type ***float*** will be appended by f.

## **Character Variable:**

Variable of type character stores a single character. 2 bytes space is reserved in memory because all characters in java are stored in Unicode.

e.g `char ch = '#';`

## **Boolean Variable:**

Variables that can have one of two values 'true' or 'false'.

e.g `boolean flag = false;`

Note: Boolean variables cannot be cast to any other type of variable and vice versa.



## Casting in Data Types:

- We can cast any of the basic type to any other.
- Casting from the larger integer type to a smaller causes loose information.
- Casting from float to integer loose some information.
- Casting from double to float also loose information.



Byte --> short --> int --> long --> float --> double

Left to right : implicitly

Right to left : explicitly

If  $x = 3$  and  $y = 2$ , then

$z = 1 + x/y;$       Result  $z = 2$

$z = 1 + (\text{float})x/y$       Result  $z = 2.5$



## Assignment Operator ( = )

Assignment operator is used to assign literal or value of variable or expression to a variable that comes on its left side.

e.g. `c = a*b;`

Multiple assignment in java is valid i.e.

`a = b = c = 100;`

## Arithmetic Operators

`+` , `-` , `*` , `/` , `%`

e.g. if `a = 15` and `b = 6` then

`a % b = 3`

## Increment and Decrement Operator

++ and --

e.g. If  $x = 10$ ,  $y = 5$ , then

$Z = x++ + y = 15$  and after evaluation,  $x = 11$

but in

$Z = ++x + y = 16$  and after evaluation,  $x = 11$

Both can be used as prefix and postfix.



## Relational Operators

`>`, `<`, `<=`, `>=`, `==`, `!=`

## Relational Expression

If relational operator is applied on operands, then relational expression is formed.  
The value of relational expression is either true or false.

## Logical Operators:

&&	Logical AND
	Logical OR
!	Logical NOT

## Compound Assignment operator:

+=	c+=1	=	c=c+1
*=	c*=1	=	c=c*1
-=	.....	=	.....
/=	.....	=	.....
%=	.....	=	.....

## Decision Control Structures:

### if - Statement

General syntax is

```
If (expression)
{
    Statement(s);
}
```

For example

```
If (numbr%2 == 0)
    System.out.println("The number is even");
```

## if-else Statement

### General Syntax

```
if (expression)
{
    statement(s);
}
else
{
    statement(s);
}
```

## If-else-if statement

```
if (expression)
{ statements; }
```

```
else if (expression)
{ statements; }
```

```
else if (expression)
{ statements; }
```

```
!
```

```
!
```

```
else
{ Statements; }
```

## Switch Statement

```
switch(variable/expression)
```

```
{
```

```
    case value1:
```

```
        Statements;
```

```
        break;
```

```
    case value2:
```

```
        Statements;
```

```
        break;
```

```
    case value3:
```

```
        Statements;
```

```
        break;
```

```
    !
```

```
    !
```

```
    !
```

```
    default:
```

```
        Statements;
```

```
        break;
```

```
}
```





# STRINGS

---

A string is commonly considered to be a sequence of characters stored in memory and accessible as a unit.

### String Concatenation:

“+” operator is used to concatenate strings.

`System.out.println(“Hello” + “World”)` will print Hello World.

```
int i = 4; int j = 5;
```

`System.out.println (“Hello” + i)` will print Hello 4 on screen.


However

```
System.out.println( i+j);
```

will print 9 on the console because both i and j are of type int.

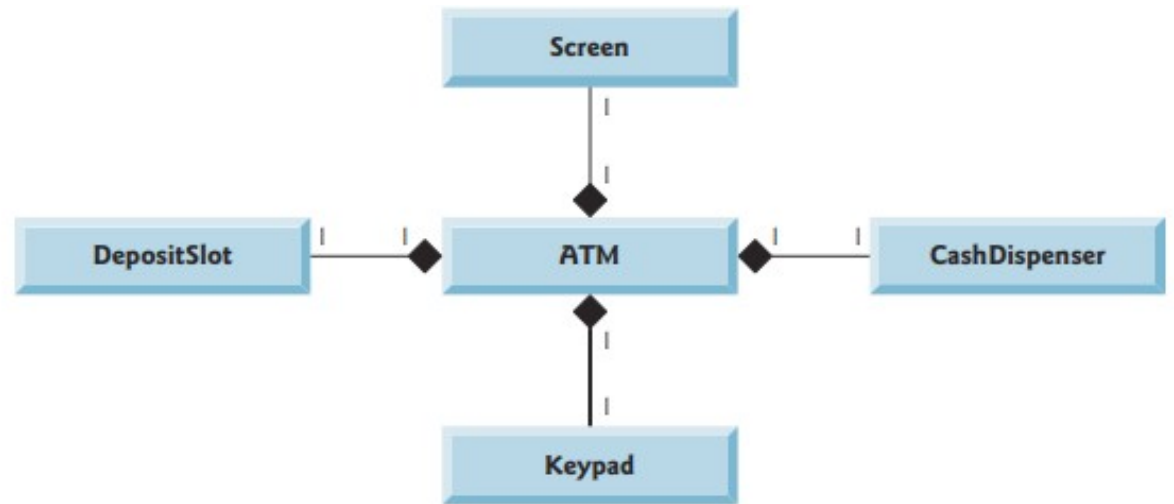


**Lets Code**

- 
- Packages
  - Constructors
  - Arrays,
  - Multi-Dimensional Arrays
  - Passing Array to Functions
  - Composition
  - Inheritance

# Composition

A class can have references to objects of other classes as members.  
This is called composition.





**Lets Code!**

# Inheritance

- In general inheritance is transfer of characteristics from parent to offspring.
- A child inherit characteristics of its parents, besides inherited characteristics a child may have its own characteristics.
- Inheritance is the process of creating new classes called derived classes from existing classes.
- The class which inherits the properties of other is known as subclass derived class, child class, and the class whose properties are inherited is known as super class, base class, or parent class.
- Keyword “EXTENDS” is used to inherit the properties of base class.

## Syntax:

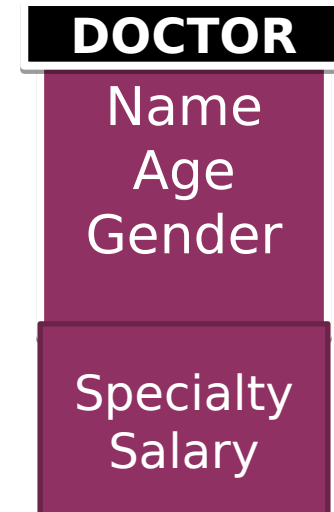
```
class Super
{
  data members
  member functions
}
```

```
class Sub extends Super
{
  data members
  member functions
}
```

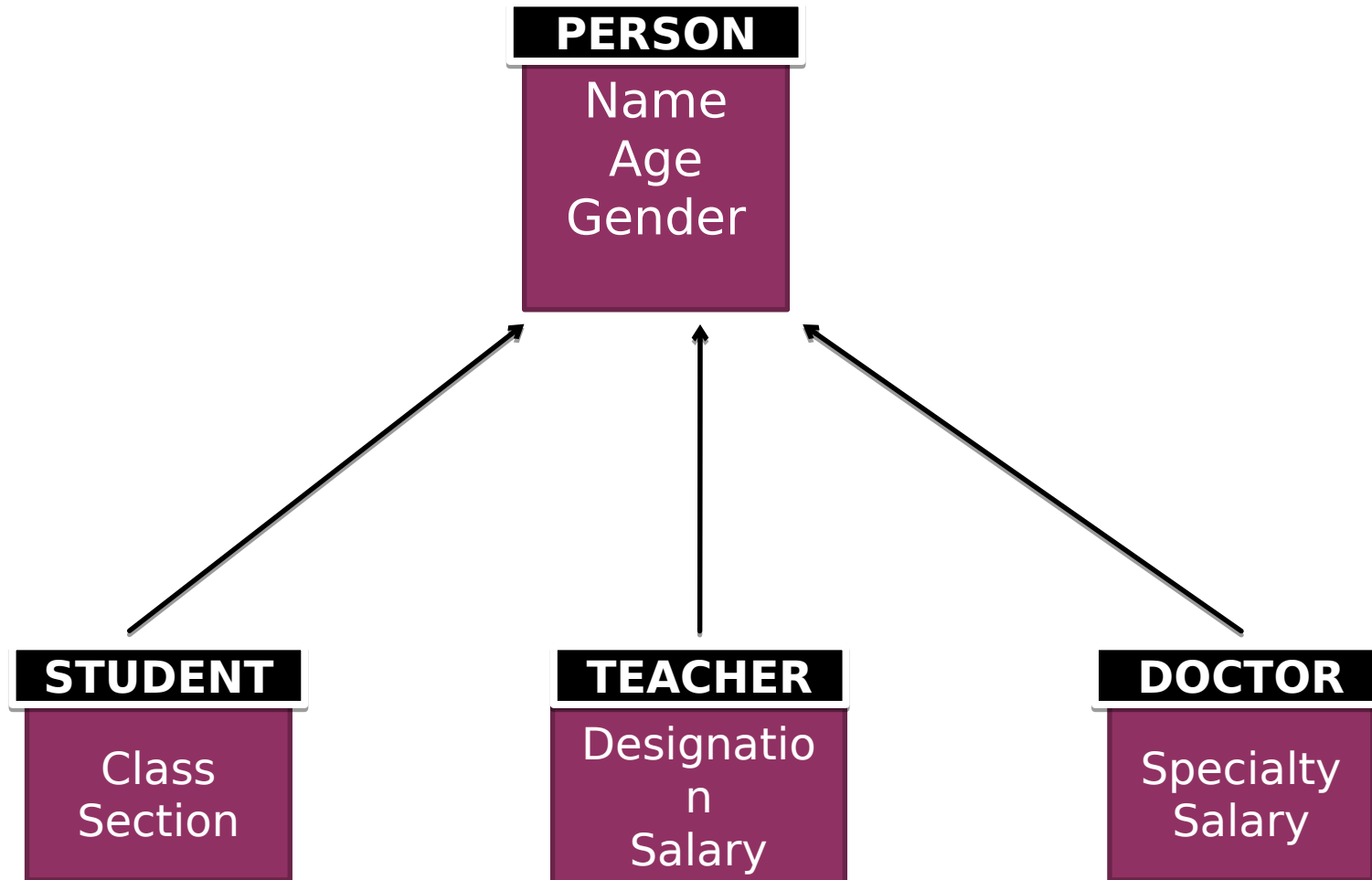


## Example:

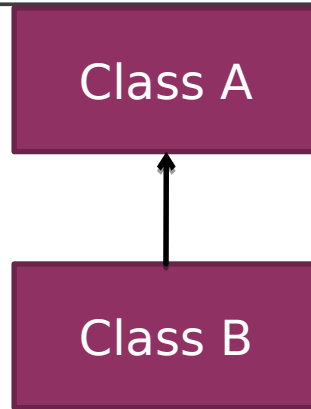
- Suppose we want to make three classes of teacher student and doctor,
- Each one of them has its own characteristics like.



- 
- *Now we have observe that there are some common characteristics in above three classes like (Name, Age, Gender).*
  - *So we made a base class named (Person) with common characteristics (name, age, gender), and created some other classes called derived classes (student, teacher , doctor) inherited from the person class.*

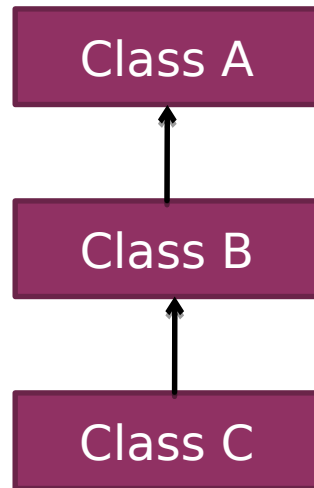


## ■ Single Inheritance:



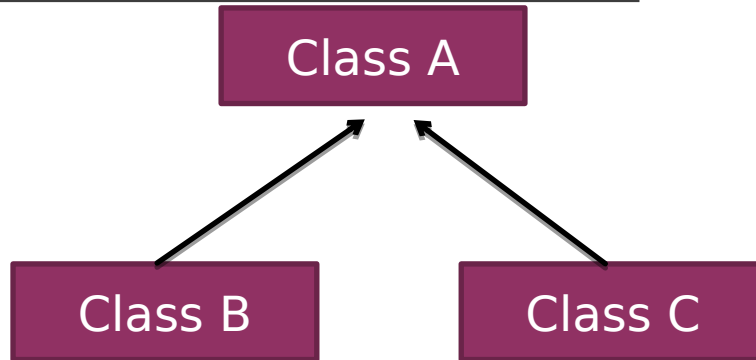
```
Public class A{  
.....  
}  
Public class B extends A{  
.....  
}
```

## ■ Multi level inheritance:



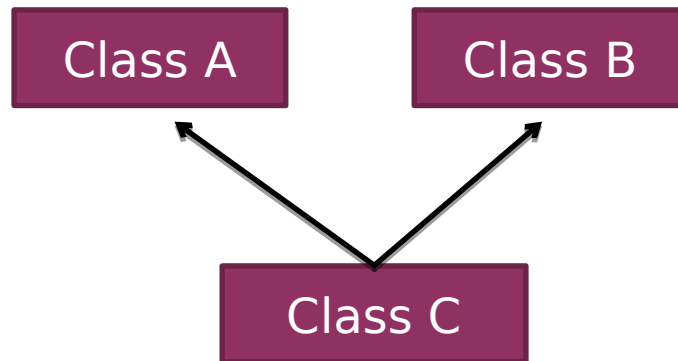
```
Public class A{  
.....  
}  
Public class B extends A{  
.....  
}  
Public class C extends B{  
.....  
}
```

## Hierarchical Inheritance:



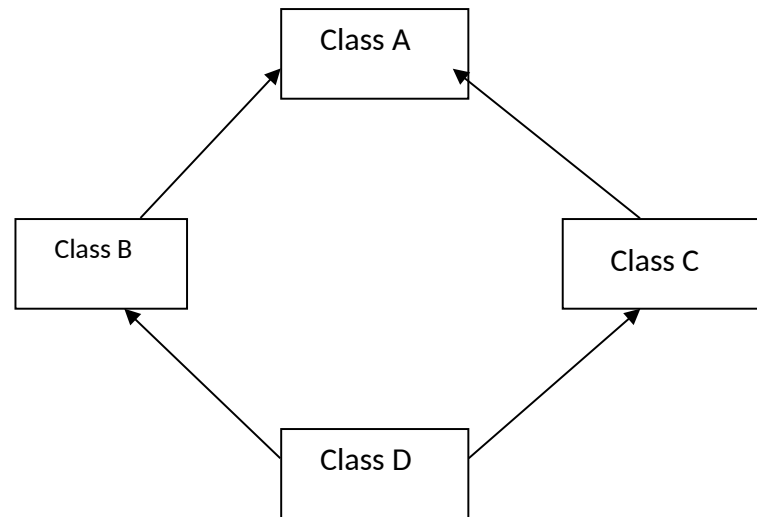
```
Public class A{  
.....  
}  
Public class B extends A{  
.....  
}  
Public class C extends A{  
.....  
}
```

## Multiple inheritance:



JAVA DOES NOT SUPPORT  
MULTIPLE INHERITANCE

*Why JAVA does not support Multiple Inheritance ?*



### ***Diamond Problem***

*Classes B and C inheriting from class A.*

*Class D is inheriting from both B and C implementing multiple inheritance.*

*Assume both B and C class have a method with the same name & signature then when we invoke that method so compiler will get confused to which method has to process.*

*This is called diamond problem.*

*Due to this problem Java designer ignores this ambiguity.*

*The same feature can be achieved in Java through implementing interfaces.*

## Super Keyword:

- *It is used to differentiate the members of super class from the members of base class.*
- *It is used to invoke the super class constructor or function from subclass.*

```
public class Dog extends Animal {  
    public void walk(){  
        super.move();  
        System.out.println("Dogs can run and walk");  
    }  
    public static void main(String args[]){  
        Dog obj = new Dog();  
        obj.walk ();  
    }  
}
```

```
public class Animal {  
    public void move(){  
        System.out.println("Animals  
        Can Move");  
    }  
}
```

## FUNCTION/METHOD OVERRIDING:

- The existence of a method in a derived class having the same name and signature as the method of base class, then the derived class method overrides the base class method, this is called method overriding.




## Function/Method Overriding:

```
public class Dog extends Animal {  
public void move(){  
System.out.println("Dogs can also Move");  
}  
public static void main(String args[]){  
Animal obj1 = new Animal();  
Animal obj2 = new Dog();  
obj1.move();  
obj2.move();  
    }  
}
```

```
public class Animal {  
public void move(){  
System.out.println("Anim  
als Can Move");  
    }  
    }
```

## DIFFERENCE BETWEEN METHOD OVERLOADING & OVERRIDING:

- Method overloading is the existence of two or more methods within a same class having same names but different signature.
- While Methods Overriding is the existence of methods in sub class having same name and signature as the base class methods as a result of which the base class methods are overridden.
- Methods Overloading is resolved using static binding in JAVA at compile time,
- While methods overriding is resolved using dynamic binding in java at run time.

- 
- Polymorphism
  - Abstract Class VS Interface

# ***POLYMORPHISM***

- *Polymorphism in java is a concept by which we can perform a single action by 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.*

# **POLYMORPHISM**

- *Runtime polymorphism is a process in which a call to an overridden method is resolved at runtime rather than compile-time.*
- *In this process, an overridden method is called through the reference variable of a super class.*
- *The determination of the method to be called is based on the object being referred to by the reference variable.*

```
class Animal{  
void eat() {System.out.println("eating...");}  
}  
class Dog extends Animal{  
void eat() {System.out.println("eating bread...");}  
}  
class Cat extends Animal{  
void eat() {System.out.println("eating rat...");}  
}  
class Lion extends Animal{  
void eat() {System.out.println("eating meat...");}  
}
```

*Output:*

eating bread... eating rat... eating meat...

```
class TestPolymorphism {  
public static void main(String[]  
args)  
{  
    Animal a;  
    a=new Dog();  
    a.eat();  
    a=new Cat();  
    a.eat();  
    a=new Lion();  
    a.eat();  
}  
}
```

# *STATIC AND DYNAMIC BINDING*

*Association of method definition to the method call is known as binding.*

*There are two types of binding:*

*Static binding and dynamic binding.*

*The binding which can be resolved at compile time by compiler is known as static or early binding.*



*All the static, private methods have always been bonded at **compile time** .*

*Compiler knows that all such methods cannot be overridden and will always be accessed by object of local class.*

*Hence compiler doesn't have any difficulty to determine object of class (local class for sure).*

*That's the reason binding for such methods is static.*



*Example:*

```
class Human {  
    ....  
}  
  
class Boy extends Human {  
    public void walk() {  
        System.out.println("Boy walks");  
    }  
  
    public static void main( String args[]) {  
        Boy obj1 = new Boy();  
        obj1.walk();  
    }  
}
```

**Note:**  
Here we have created an object of Boy class and calling the method walk() of the same class. Since nothing is ambiguous here, compiler would be able to resolve this binding during compile-time, such kind of binding is known as static binding.

- 
- *When compiler is not able to resolve the call/binding at compile time, such binding is known as Dynamic or late Binding.*
  - *Overriding is a perfect example of dynamic binding as in overriding both parent and child classes have same method.*
  - *Thus while calling the overridden method, the compiler gets confused between parent and child class method(since both the methods have same name).*

## CASE STUDY: PAYROLL SYSTEM USING POLYMORPHISM

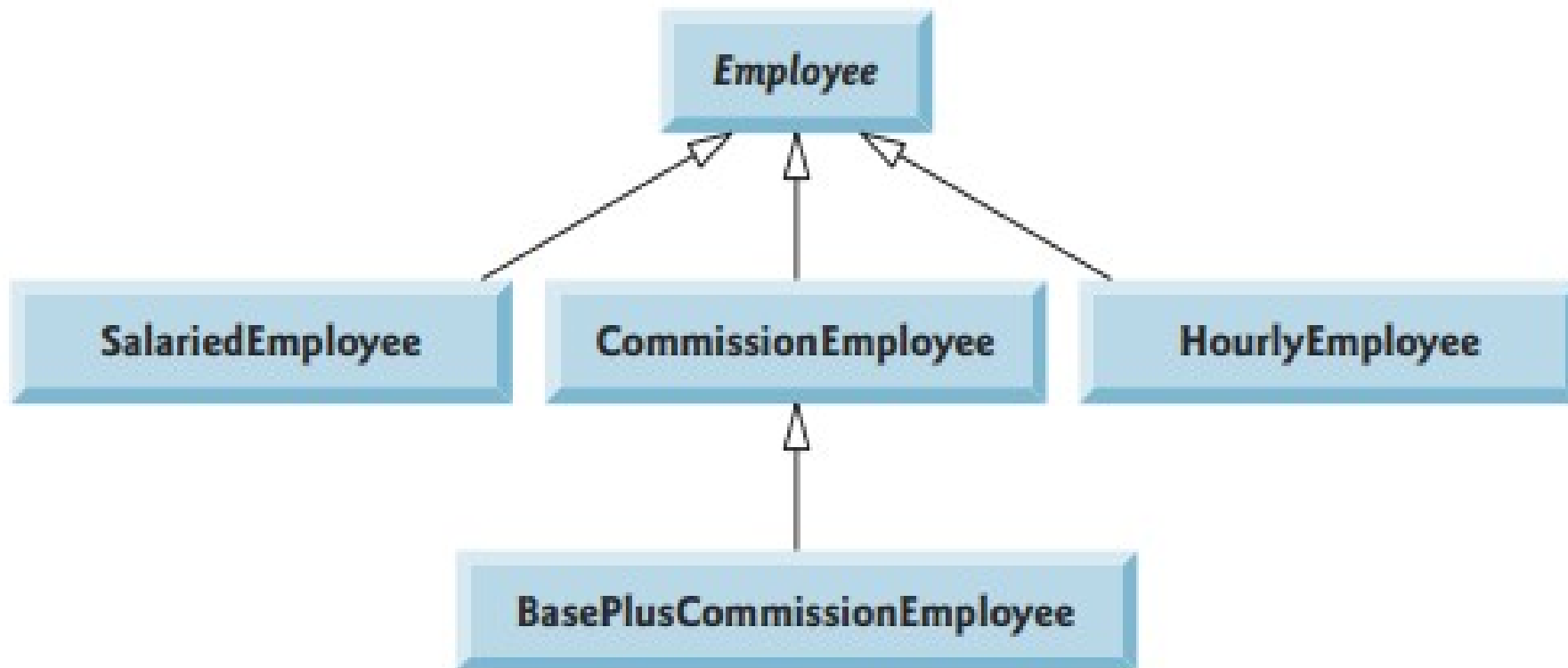
**A company pays its employees on a weekly basis. The employees are of four types:**

- **Salaried employees** are paid a fixed weekly salary regardless of the number of hours worked,
- **Hourly employees** are paid by the hour and receive overtime pay (i.e., 1.5 times their hourly salary rate) for all hours worked in excess of 40 hours,
- **Commission employees** are paid a percentage of their sales and **base-salaried commission employees** receive a base salary plus a percentage of their sales.

For the current pay period, the company has decided to reward salaried-commission employees by adding 10% to their base salaries.


The company wants to write an application that performs its payroll calculations polymorphically.

## CASE STUDY: PAYROLL SYSTEM USING POLYMORPHISM





# **Abstract Class**




A class that is declared using “abstract” keyword is known as abstract class.

Abstract classes may or may not contain abstract methods ie., methods with out body ( public void get(); )

which means in abstract class you can have concrete methods (methods with body) as well along with abstract methods (without body).

Abstract class can extend only one class or one abstract class at a time.

A class can extend only one abstract class.



An abstract class can not be **instantiated** (you are not allowed to create **object** of Abstract class).

Since abstract class allows concrete methods as well, it does not provide 100% abstraction.

You can say that it provides partial abstraction.

Interfaces are used for 100% abstraction.



## **Key Points to Remember**



---

An abstract class has no use until unless it is extended by some other class.

If you declare an **abstract method** in a class then you must declare the class abstract as well. you can't have abstract method in a **non-abstract class**.

Abstract class can have non-abstract method (concrete) as well.

Abstract method has no body.

Always end the declaration with a semi colon.



## **Why Need an Abstract Class?**



---

Suppose there is a class `Animal` and there are few other classes like `Cat`, `Dog` and `Horse`.

These classes extend `Animal` class so basically they are having few common habits(methods in technically) which they are inheriting from `Animal` class.

Now, if you have understood the above example then you would have been able to figure out that **creating object of `Animal` class has no significance** as you can't judge that the **new** object of `Animal` class will represent which animal.

Hence for such kind of scenarios we generally create an **abstract classes**.

```
abstract class Demo1{
    public void disp1() {
        System.out.println("Concrete method of abstract class");
    }
    abstract public void disp2();
}

class Demo2 extends Demo1{
    public void disp2() {
        System.out.println("I'm overriding abstract method");
    }

    public static void main(String args[]) {
        Demo2 obj = new Demo2();
        obj.disp2();
    }
}
```

OUTPUT:

***I'm overriding  
abstract method***




## Abstract Vs Concrete:

*A class which is not abstract is referred as **Concrete class**.*

*In the given example, animal is a abstract class and Cat ,  
Dog and Horse are concrete classes .*



# Interface



Interface are special java type which contains only a set of method prototypes, but does not provide the implementation for these prototypes.

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method names, no body).

Interface is tantamount to a pure abstract class.

Interfaces are used for 100% abstraction.



## **Why Need an Interface?**





As mentioned before they are used for pure abstraction.

Methods in interfaces do not have body, they have to be implemented by the class before you can access them.

The class that implements interface must implement all the methods of that interface.

Java programming language does not support multiple inheritance.

Using interfaces we can achieve this as a class can implement more than one interfaces.

```
public class Abc implements MyInterface {  
    public void method1()  
    {  
        System.out.println ("implementation of method1");  
    }  
    public void method2()  
    {  
        System.out.println("implementation of method2");  
    }  
  
    public static void main(String args []){  
        Abc obj = new Abc();  
        obj. method1();  
    }  
}
```

```
public interface  
MyInterface {  
  
    public void method1();  
    public void method2();  
}
```



# **Multiple Inheritance with Interfaces**




---

```
interface A {  
    public void aaa();  
}  
  
interface B {  
    public void bbb();  
}  
  
class Central implements A,B {  
    public void aaa() {  
        //Any Code here  
    }  
  
    public void bbb() {  
        //Any Code here  
    }  
}
```

```
public static void  
main(String args[]) {  
    //Statements  
}  
}
```



## **Key points about Interface**

- 
- 1- Interface provides complete abstraction as none of its methods can have body. On the other hand, abstract class provides partial abstraction as it can have abstract and concrete methods both.
  - 2- Implements keyword is used by classes to implement an interface.
  - 3- Class implementing any interface must implement all the methods, otherwise the class should be declared as “abstract”.

4- Variable names conflicts can be resolved by interface name e.g:

```
interface A { int x=10; }
```

```
interface B { int x=100; }
```

```
class Hello implements A,B {
```

```
public static void Main(String args[]) {
```

```
System.out.println(x);      // reference to x is ambiguous both variables are x.
```

```
System.out.println(A.x);
```

```
System.out.println(B.x);
```

```
}
```

```
}
```



## **Difference b/w abstract class vs Interface**





1- Abstract class can extend only one class or one abstract class at a time.

Interface can extend any number of interfaces at a time.

2- Abstract class can extend from a class or from an abstract class.

Interface can extend only from an interface.

3- Abstract class can have both abstract and concrete methods.

Interface can have only abstract methods.

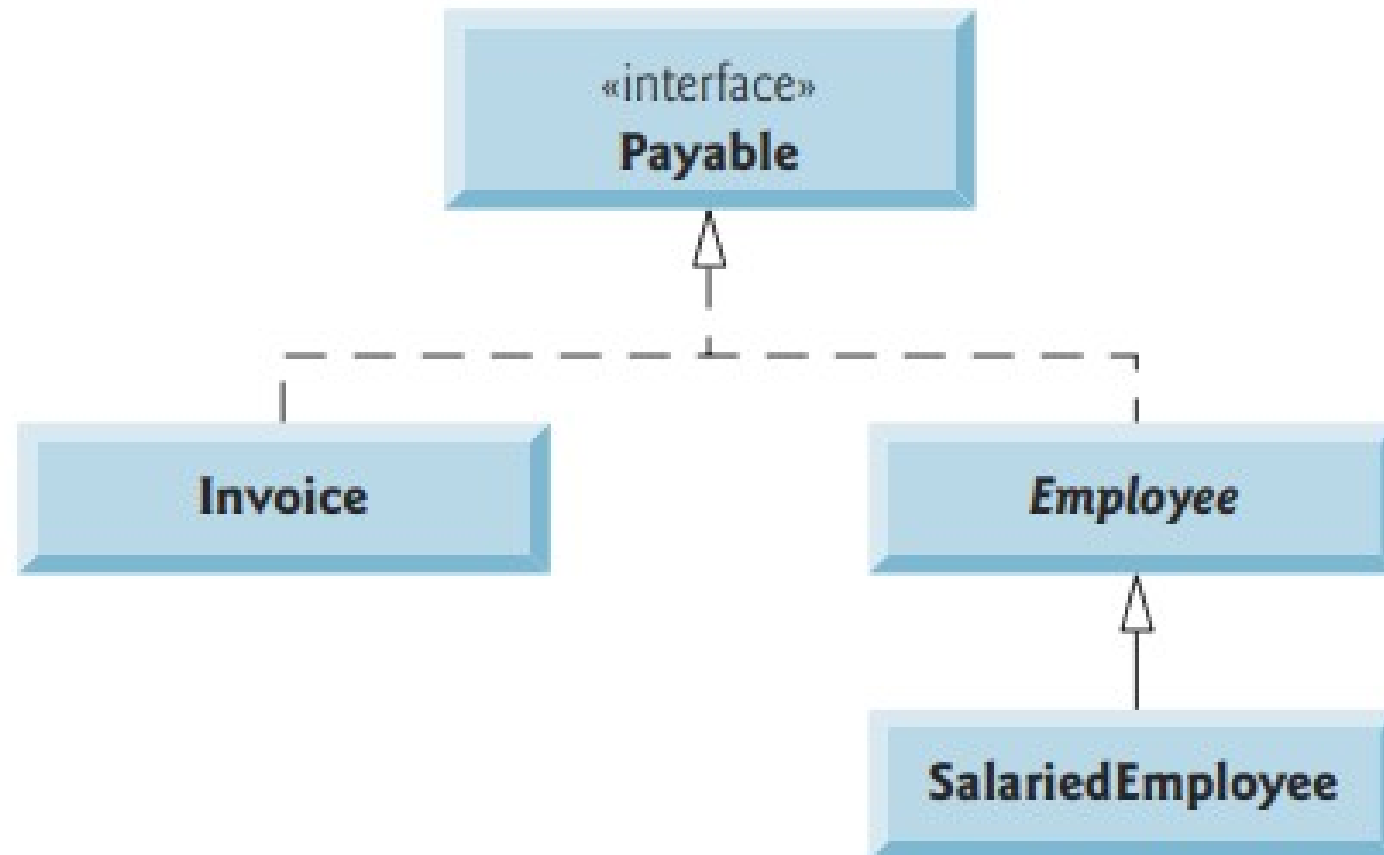


4- A class can extend only one abstract class.

A class can implement any number of interfaces.

5- In abstract class keyword 'abstract' is mandatory to declare a method as an abstract

In an interface keyword 'abstract' is optional to declare a method as an abstract.





**Lets Code!**



**HAVE A GOOD DAY!**