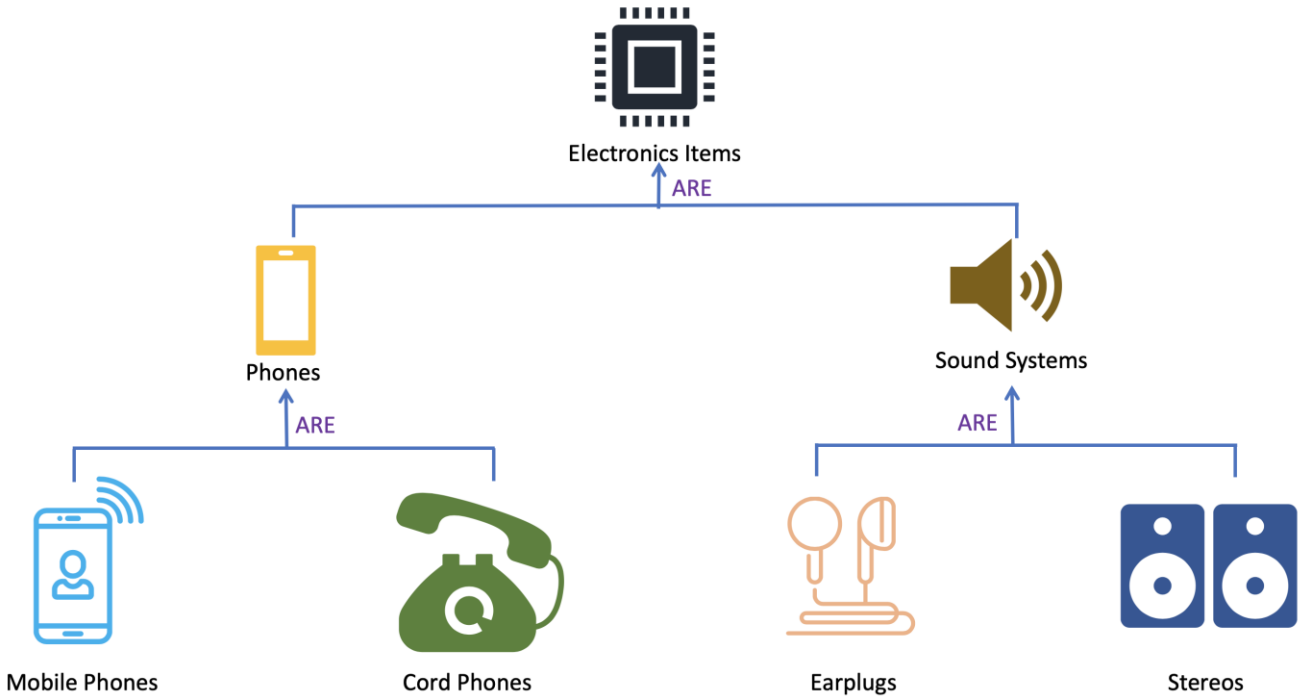
AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

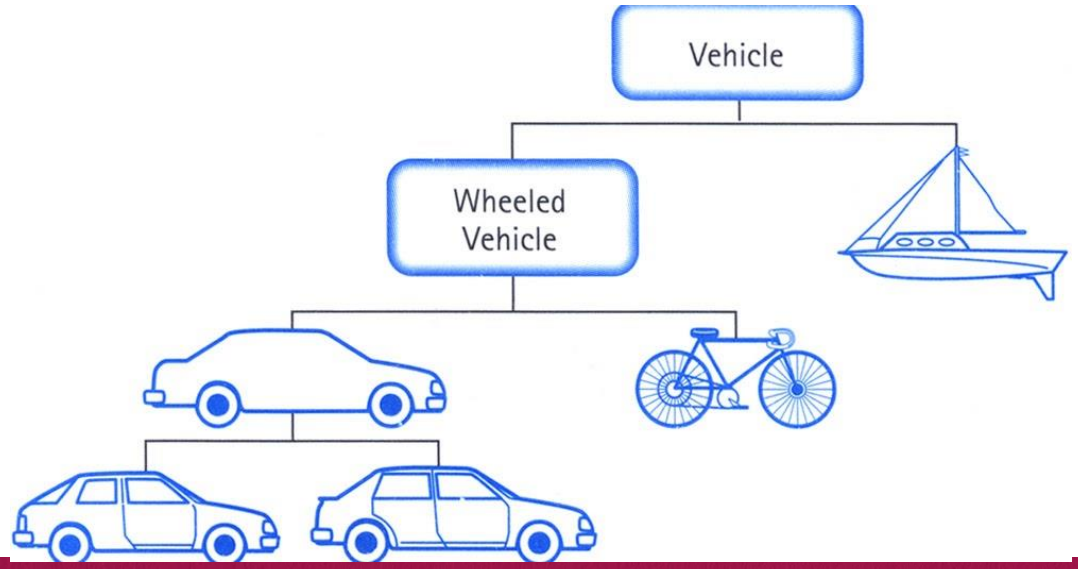
19CSE204
Object Oriented Paradigm
2-0-3-3

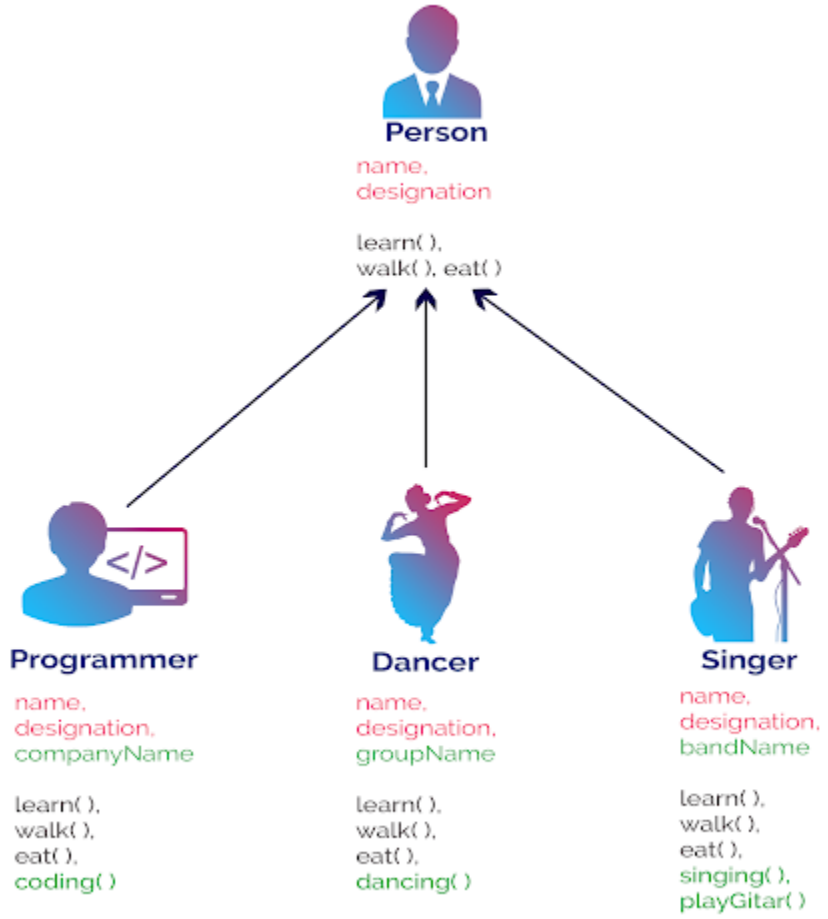
Inheritance in Java

-the ability in **Java** for one class to **inherit** from another class.

Real world examples- Inheritance







Inheritance (Another corner stone of OOPS)

- **Java inheritance** refers to the ability in **Java** for one class to **inherit** from another class.
- In **Java** this is also called extending a class. One class can extend another class and thereby **inherit** from that class.
- When one class **inherits** from another class in **Java**, the two classes take on certain roles.

  - **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**: **Superclass** is the **class** from whereas **subclass** inherits the features. It is also called a **base class** or a **parent class**.

Inheritance promotes code reusability

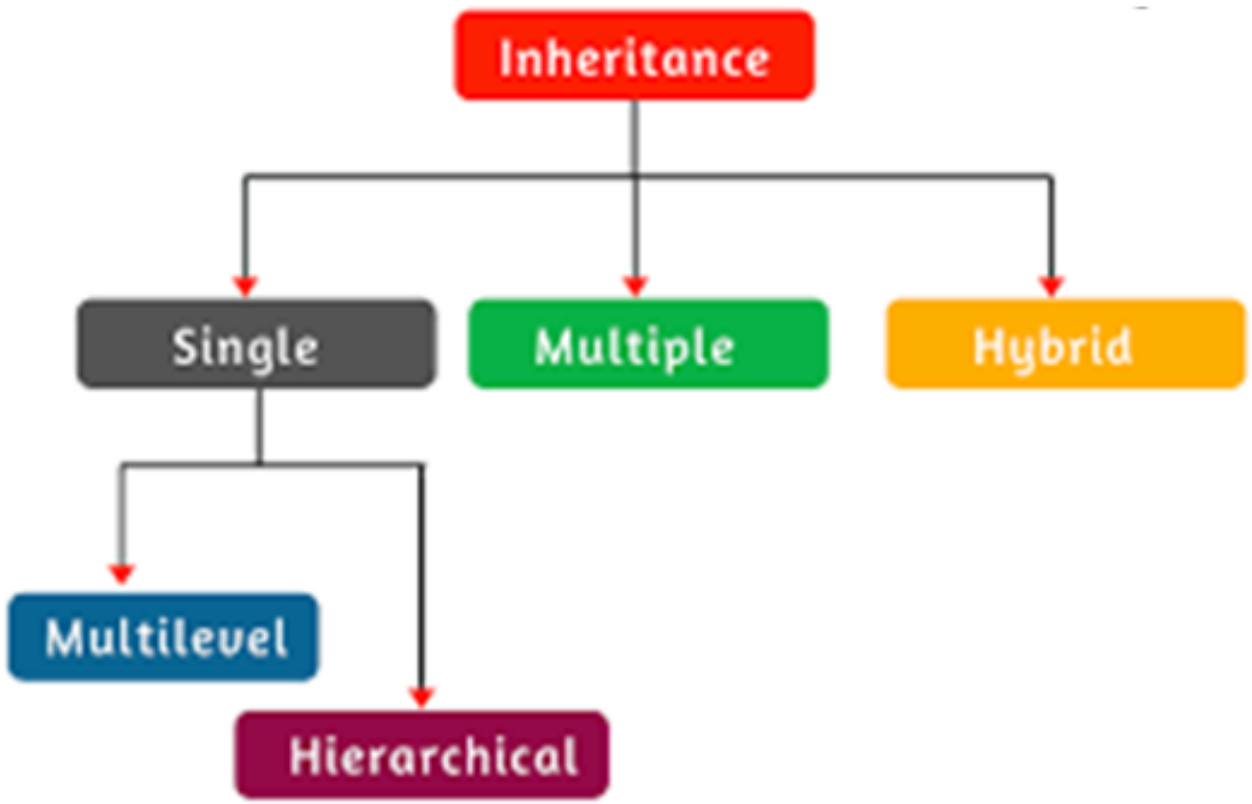


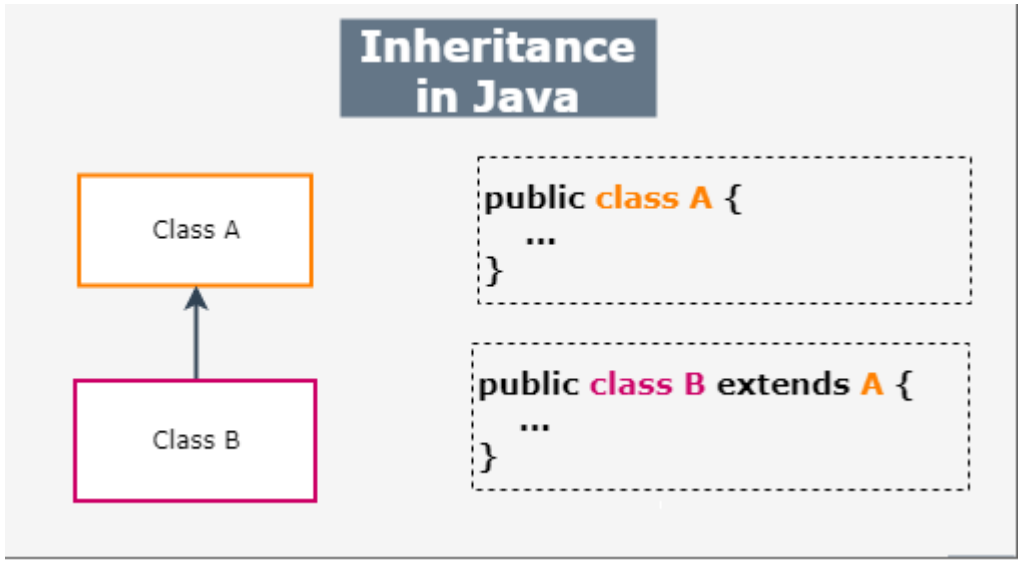
Fig: Classification of Inheritance in Java

Types of Inheritance

A class which is inherited is called a **parent** or **superclass**, and the new class is called **child** or **subclass**.

A simple example of inheritance

class *subclass-name* extends *superclass-name* {
// body of class
}




```java
package inherit;
//Create a superclass.
class A {
int i, j;
void showij() {
System.out.println("i and j: " + i + " " + j);
}
}
//Create a subclass by extending class A.
class B extends A {
int k;
void showk() {
System.out.println("k: " + k);
}
void sum() {
System.out.println("i+j+k: " + (i+j+k));
}
}
```

```java
public class Driver {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        A superOb = new A();
        B subOb = new B();
        // The superclass may be used by itself.
        superOb.i = 10;
        superOb.j = 20;
        System.out.println("Contents of superOb: ");
        superOb.showij();
        System.out.println();
        /* The subclass has access to all public members of
        its superclass. */
        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        System.out.println("Contents of subOb: ");
        subOb.showij();
        subOb.showk();
        System.out.println();
        System.out.println("Sum of i, j and k in subOb:");
        subOb.sum();
        }
```

Output

```
Contents of superOb:
i and j: 10 20

Contents of subOb:
i and j: 7 8
k: 9

Sum of i, j and k in subOb:
i+j+k: 24
```

Member Access and Inheritance

- Although a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared as **private**.

Solution : Give the access modifier protected to j

This gives an error as j is not accessible in class B, as it is declared as private in A

```java
package typesinheritance;
/* In a class hierarchy, private members remain
private to their class. This program contains
an error and will not compile.*/
class A {
int i; // public by default
private int j; // private to A
void setij(int x, int y) {
i = x;
j = y;
}
}
// A's j is not accessible here.
class B extends A {
int total;
void sum() {
total = i + j; // ERROR, j is not accessible here
}
}

public class inheritanceDemo3 {

    public static void main(String[] args) {
        B subOb = new B();
        subOb.setij(10, 12);
        subOb.sum();
        System.out.println("Total is " + subOb.total);

    }
}
```

Access specifiers

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

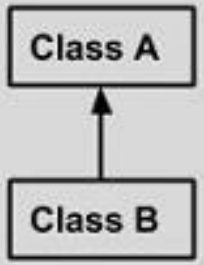
Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

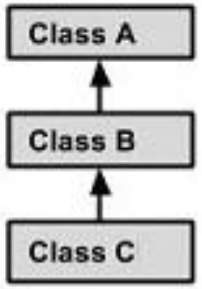
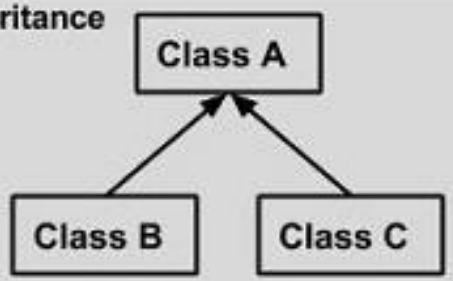
Error rectified

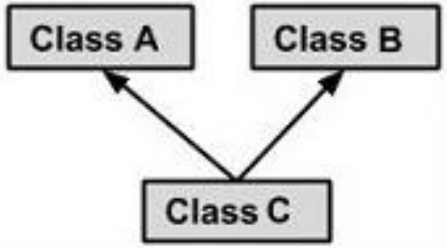
- Modified program in slide 8

- Line 7- j is made protected, hence possible to access within the same package subclass

```
1  package typesinheritance;
2  /* In a class hierarchy, private members remain
3  private to their class. This program contains
4  an error and will not compile.*/
5  class A {
6  int i; // public by default
7  protected int j; // protected to A
8  void setij(int x, int y) {
9  i = x;
10 j = y;
11 }
12
13
14 class B extends A {
15 int total;
16 void sum() {
17 total = i + j;
18 }
19 }
20
21 public class inheritanceDemo3 {
22
23     public static void main(String[] args) {
24         B subOb = new B();
25         subOb.setij(10, 12);
26         subOb.sum();
27         System.out.println("Total is " + subOb.total);
28
29     }
30 }
```

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 	public class A { .....}
	public class B {.....}
	public class C **extends** A,B {
	.....
	} // Java does not support mutiple Inheritance

BoxWeight class Extends Box class – Add weight data member

```
class BoxWeight extends Box {
double weight;
// weight of box
// constructor for BoxWeight

BoxWeight(double w, double h,
double d, double m) {
width = w;
height = h;
depth = d;
weight = m;
}
}
```

```
class DemoBoxWeight
{
public static void main(String args[]) {
BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
double vol;
vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);
System.out.println("Weight of mybox1 is " + mybox1.weight)
System.out.println();
vol = mybox2.volume();
System.out.println("Volume of mybox2 is " + vol);
System.out.println("Weight of mybox2 is " + mybox2.weight)
}
}
```

Super Keyword

- In prev eg: Classes derived from **Box** were not implemented as efficiently or as robustly as they could have been. For example, the constructor for **BoxWeight** explicitly initializes the **width**, **height**, and **depth** fields of **Box( )**.
- Not only does this duplicate code found in its superclass, which is inefficient, but it implies that a subclass must be granted access to these members.

Super keyword

Whenever a subclass needs to refer to its immediate Superclass, it can do so by use of the **keyword super.**
super has two general forms.

- The first calls the superclass' constructor.
- The second is used to access a member of the superclass that has been hidden by a member of a subclass

Using super to Call Superclass Constructors

- A subclass can call a constructor method defined by its superclass by use of the following form of **super**:

  super(*parameter-list*);

- Here, *parameter-list* specifies any parameters needed by the constructor in the superclass.
- **super( )** must always be the first statement executed inside a subclass' constructor.

Improved version of BoxWeight

```
// BoxWeight now uses super to initialize its Box
attributes.
class BoxWeight extends Box {
double weight; // weight of box
// initialize width, height, and depth using super()
BoxWeight(double w, double h, double d, double m) {
super(w, h, d); // call superclass constructor
weight = m;
}
}
```

```java
package typesinheritance;

//Extend BoxWeight to include shipping costs.
//Start with Box.
class Box {
private double width;
private double height;
private double depth;
//construct clone of an object
Box(Box ob) { // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.depth;
}
//constructor used when all dimensions specified
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
// constructor used when no dimensions specified
Box() {
width = -1; // use -1 to indicate
height = -1; // an uninitialized
depth = -1; // box
}
// constructor used when cube is created
Box(double len) {
width = height = depth = len;
}
```

```java
// compute and return volume
double volume() {
return width * height * depth;
}
}
// Add weight.
class BoxWeight extends Box {
double weight; // weight of box
// construct clone of an object
BoxWeight(BoxWeight ob) { // pass object to constructor
super(ob);
weight = ob.weight;
}
// constructor when all parameters are specified
BoxWeight(double w, double h, double d, double m) {
super(w, h, d); // call superclass constructor
weight = m;
}
// default constructor
BoxWeight() {
super();
weight = -1;
}
//constructor used when cube is created
BoxWeight(double len, double m) {
super(len);
weight = m;
}
}
```

```java
//Add shipping costs
class Shipment extends BoxWeight {
double cost;
//construct clone of an object
Shipment(Shipment ob) { // pass object to construct
super(ob);
cost = ob.cost;
}
//constructor when all parameters are specified
Shipment(double w, double h, double d,
double m, double c) {
super(w, h, d, m); // call superclass constructor
cost = c;
}
//default constructor
Shipment() {
super();
cost = -1;
}
//constructor used when cube is created
Shipment(double len, double m, double c) {
super(len, m);
cost = c;
}
}
```

```java
public class superkeyword {

    public static void main(String[] args) {
        Shipment shipment1 =new Shipment(10, 20, 15, 10, 3.41);
        Shipment shipment2 =
        new Shipment(2, 3, 4, 0.76, 1.28);
        double vol;
        vol = shipment1.volume();
        System.out.println("Volume of shipment1 is " + vol);
        System.out.println("Weight of shipment1 is "
        + shipment1.weight);
        System.out.println("Shipping cost: $" + shipment1.cost);
        System.out.println();
        vol = shipment2.volume();
        System.out.println("Volume of shipment2 is " + vol);
        System.out.println("Weight of shipment2 is "
        + shipment2.weight);
        System.out.println("Shipping cost: $" + shipment2.cost);

    }
}
```

```
Volume of shipment1 is 3000.0
Weight of shipment1 is 10.0
Shipping cost: $3.41
Volume of shipment2 is 24.0
Weight of shipment2 is 0.76
Shipping cost: $1.28
```

A Second Use for super

- The second form of **super** acts somewhat like **this**, except that it always refers to the superclass of the subclass in which it is used. This usage has the following general form:

  super.*member*

- Here, *member* can be either a method or an instance variable.
- This second form of **super** is most applicable to situations in which member names of a subclass hide members by the same name in the superclass.

```
// Using super to overcome name hiding.
class A {
int i;
}
// Create a subclass by extending class A.
class B extends A {
int i; // this i hides the i in A
B(int a, int b) {
super.i = a; // i in A
i = b; // i in B
}
void show() {
System.out.println("i in superclass: " +
super.i);
System.out.println("i in subclass: " + i);
}
}
class UseSuper {
public static void main(String args[]) {
B subOb = new B(1, 2);
subOb.show();
}
}
```

```
Output
i in superclass: 1
i in subclass: 2
```

Namah Shivaya!