Inheritance

Inheritance is one of the useful feature of OOPs. It allows a class to use the properties and methods of another class. The **purpose of inheritance in java**, is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be inherited from the another class.
A class can only inherit the fields and methods of another class, if it extends the class. For example in the following snippet, class A extends class B. Now class A can access the fields and methods of class B.

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

Let's learn the concept of parent and child class in Inheritance:

Child Class:

The class that extends the features of another class is known as child class, sub class or derived class. In the above code, class A is the child class.

Parent Class:

The class that shares the fields and methods with the child class is known as parent class, super class or Base class. In the above code, Class B is the parent class.

Advantages of Inheritance

•Inheritance **removes redundancy** from the code. A class can reuse the fields and methods of parent class. No need to rewrite the same redundant code again.
•Inheritance allows us to reuse of code, it **improves reusability** in your java application.
•**Reduces code size:** By removing redundant code, it reduces the number of lines of the code.
•**Improves logical structure:** Improves the logical structure of the code that allows programmer to visualize the relationship between different classes.

Types of inheritance in Java

There are four types of inheritance in Java:

- Single
- Multilevel
- Hierarchical
- Hybrid

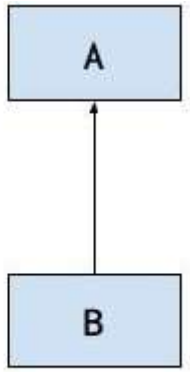
Single Inheritance

In Single inheritance, a **single child class** inherits the properties and methods of a **single parent class**. In the following diagram: class B is a child class and class A is a parent class.

```
class A {

}
class B extends A {

}
```



Multilevel Inheritance

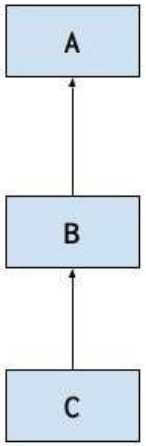
In multilevel inheritance a parent class becomes the child class of another class. In the following diagram: class B is a parent class of C, however it is also a child class of A. In this type of inheritance, there is a concept of intermediate class, here class B is an intermediate class.

```
class A {

}
class B extends A {

}
class C extends B {

}
```



Hierarchical Inheritance

In hierarchical inheritance, more than one class extends the same class. As shown in the following diagram, classes B, C & D extends the same class A.

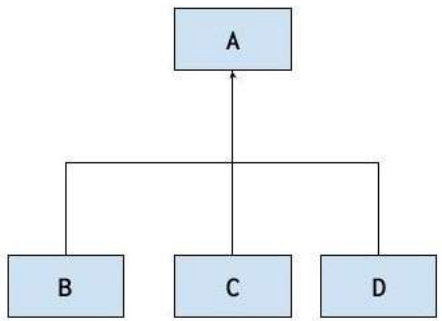
```
class A {

}
class B extends A {

}
class C extends A {

}
class D extends A {

}
```



Hybrid Inheritance

Combination of more than one types of inheritance in a single program. For example class B & C extends A and another class D extends class C then this is a hybrid inheritance example because it is a combination of single and hierarchical inheritance.

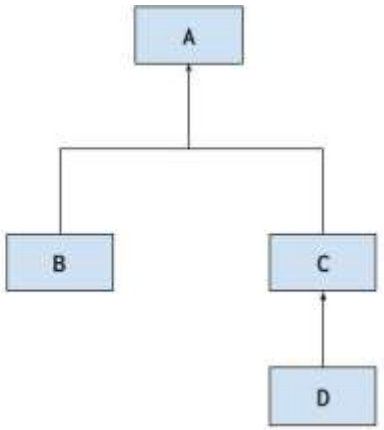
```
class A {

}
class B extends A {

}
class C extends A {

}
class D extends C {

}
```



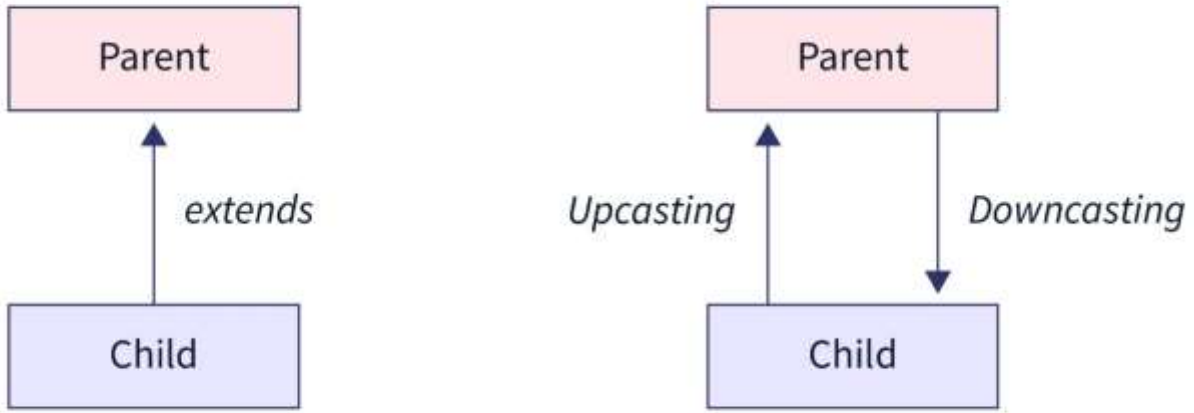
Multiple Inheritance

It refers to the concept of one class extending more than one classes, which means a child class has more than one parent classes. For example class C extends both classes A and B. **Java doesn't support multiple inheritance**

What is Upcasting and Downcasting in Java?

Upcasting (or widening) is the typecasting from a subclass to a superclass (Child to Parent class). In this way, we can access the properties (methods and variables) of the superclass, i.e., the Parent as well. Downcasting (or narrowing) is typecasting from a superclass to a subclass so that we can access the methods of the subclass, i.e., the **child** class.

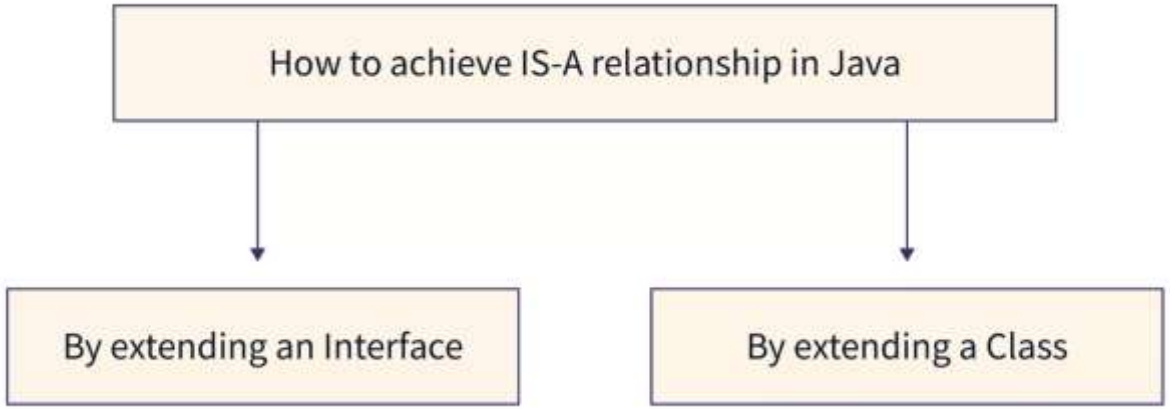
Upcasting and downcasting in Java are the two forms of typecasting where conversion of data type takes place. Typecasting is one of the important concepts in Java where the conversion of data types can be used for various purposes. Here, we're going to see how the conversion of an object takes place (because we have usually seen the typecasting in the case of datatypes only).



What is IS-A Relationship in Java?

The reason for using the IS-A relationship in java are as follows:

- Reducing redundancy
- Code reusability
- IS-A relationship in java can be achieved by using the keyword ‘extends’ in the code.
- It is used for avoiding any kind of redundancy in the code and to reuse the class and methods in the program.
- The IS-A relationship is unidirectional which means that mango is a fruit but not all fruits are mango.
- IS-A relationship is also known to be tightly coupled which means that by changing one entity, other entity will also be changed.

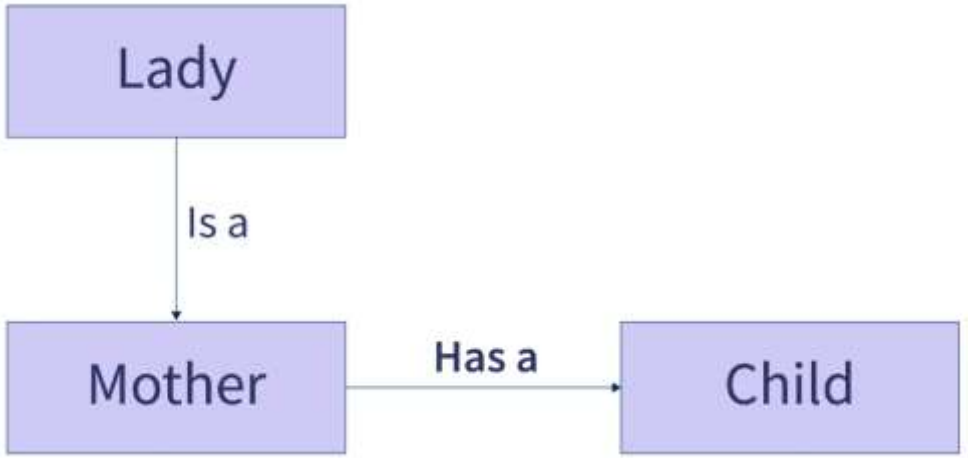


What is Has-A-Relation in Java?

Suppose we declare two different classes in Java. Between those two classes, there can be two types of relationships. This relationship can be either an Is-A relationship or a Has-A relationship. Is-A relationship is achieved by inheritance, and Has-A-relationship can be achieved using composition. Composition is a term where we declare the object of one class as a data member inside another class. In simple words, a relationship where an object has a reference to another instance of the same class or has a reference to an object of another class is called composition or has-a-relation in Java.

To create has-a-relation we need a minimum of two classes, where the instance variable of one class is declared in another class.

Let us see an example in layman's terms that will help you in understanding the above definition.



What is Has-A-Relation in Java?

- **Types of Has-A-Relationship**
- There are two types of has-a relationships. They are as follows:
- Aggregation
- Composition
- Both aggregation and composition are subsets of association. In Java, an association is a connection or relation established between two distinct classes via their objects. The following diagram shows the relationship between association, aggregation, and composition.

Aggregation

- Aggregation is a fundamental concept in object-oriented programming. It focuses on the development of a Has-A relationship between two classes. In other words, two aggregated object have their own life cycle, but one of them has a Has-A relationship owner, and a child object cannot belong to another parent object. Also, we can say that aggregation is a one-way relationship between classes. The aggregation has weak bonding between classes.
- For example, the library has students. It means that students can still study if there is no library, but if the student does not exist, then the library will be of no use.

Composition

- Composition is a more limited version of aggregation. Composition is defined as when one class that includes another class is so dependent on it that it cannot function without the class that is included. For instance, a car cannot exist without its engine. In the same way, the engine won't work for other cars (which means the engine will be useless if a car does not exist). So engine and car are two classes that have a composition relationship.
- As a result, the term composition refers to the items that something is made of, and changing the composition of things causes them to change. The composition has strong bonding between classes.