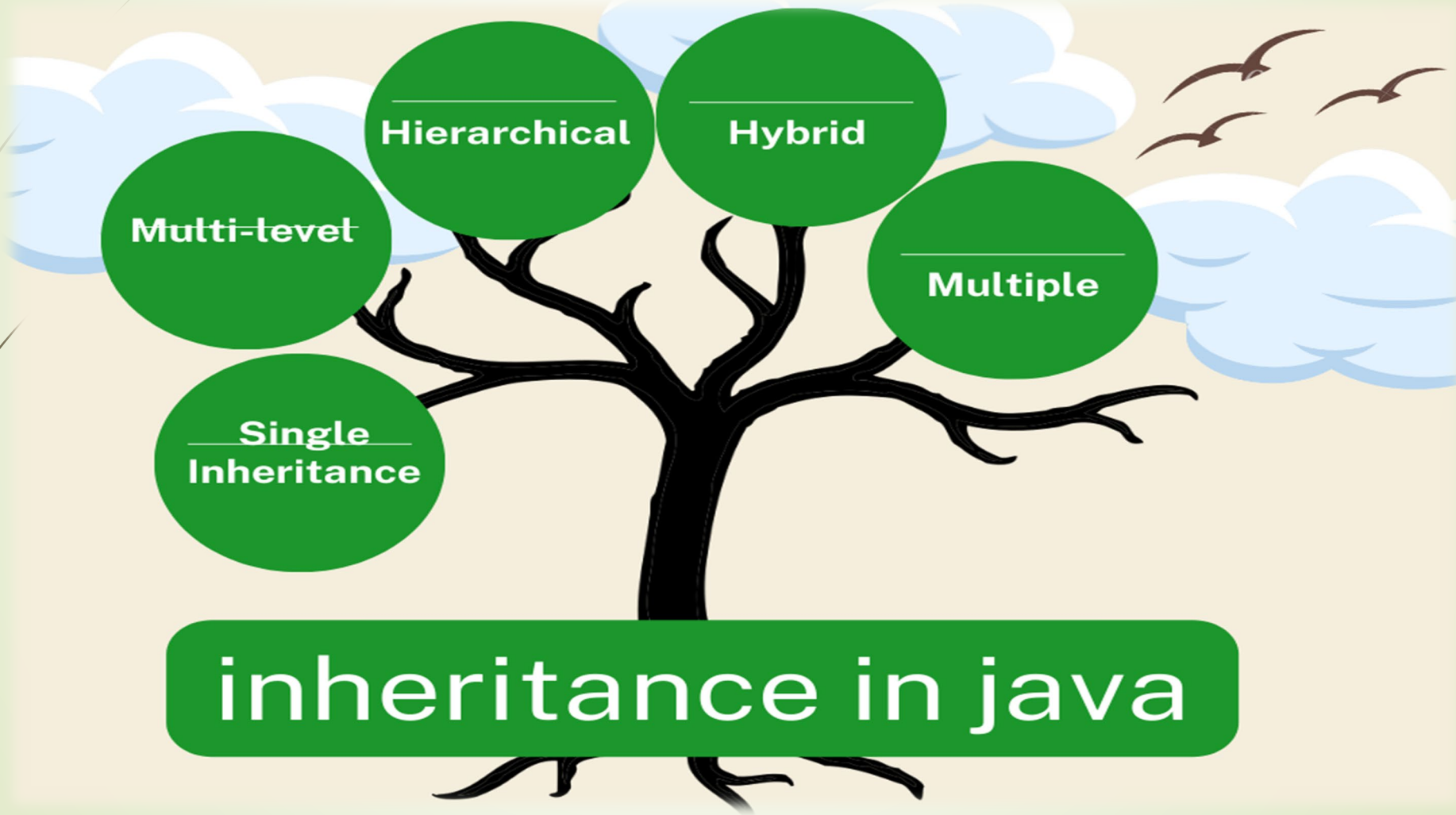
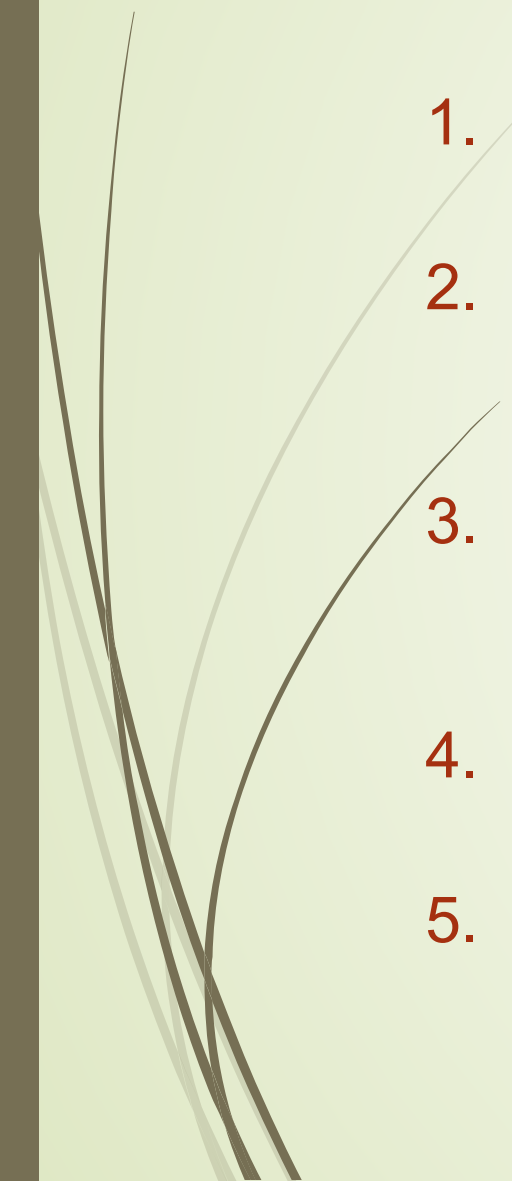


Inheritance in Java





Inheritance in Java

1. The technique of creating a new class by using an existing class functionality is called inheritance in Java.
 2. In other words, inheritance is a process by which a child class acquires all the properties and behaviors of the parent class.
 3. The existing class is called parent class (a more general class) and the new class is called child class (a more specialized class).
 4. The child class inherits data and behavior from the parent class.
 5. Inheritance represents the **IS-A relationship**, also known as a parent-child relationship.
- 

Inheritance Example:

```
public class Vehicle {  
    .....  
    .....  
}  
  
    IS-A Relationship  
    ↙      ↘  
public class Car extends Vehicle {  
    .....  
    .....  
}  
  
    IS-A Relationship  
    ↙      ↘  
public class Alto extends Car {  
    .....  
    .....  
}
```

Fig a: Is-A relationship between classes

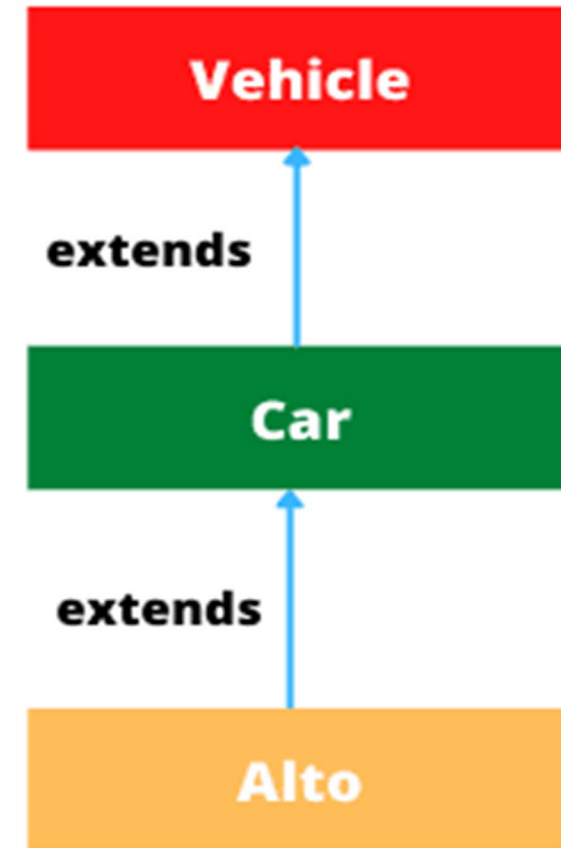


Fig b: Inheritance hierarchy for Vehicle, Car, and Alto

How is Inheritance Implemented/Achieved in Java?

Inheritance in Java can be implemented or achieved by using two keywords:

1. **extends**: extends is a keyword that is used for developing the inheritance between two classes and two interfaces. Note that a class always extends another class. An interface always extends another interface and can extend more than one interface.
2. **Implements**: implements keyword is used for developing the inheritance between a class and interface. A class always implements the interface.

Points to remember

1. Is-A relationship in Java represents Inheritance.
2. It is implemented in Java through keywords **extends** (for class inheritance) and **implements** (for interface implementation).
3. All the classes extends **java.lang.object** by default.
4. This means Object is the root class of all the classes in Java.
5. Therefore, by default, every class is the subclass of **java.lang.object**.

```
class A {  
    // statements  
}
```

Java compiler will treat the above code like this:

```
class A extends Object {  
    // statements  
}
```

Why do we use Inheritance in Java?

1. We can reuse the code from the base class.
2. Using inheritance, we can increase features of class or method by overriding.
3. Inheritance is used to use the existing features of class.
4. It is used to achieve runtime polymorphism i.e **method** overriding.
5. Using inheritance, we can organize the information in a hierarchal form.

Constructor in Java Inheritance

1. When the constructor of a subclass is called during the object creation, by default, it calls the default constructor of superclass.
2. The superclass constructor can be called using keyword “super” from the subclass constructor.
3. It must be the first statement in a constructor.
4. We can call other constructors of same class using this keyword, but you cannot call a subclass constructor from the superclass constructor.



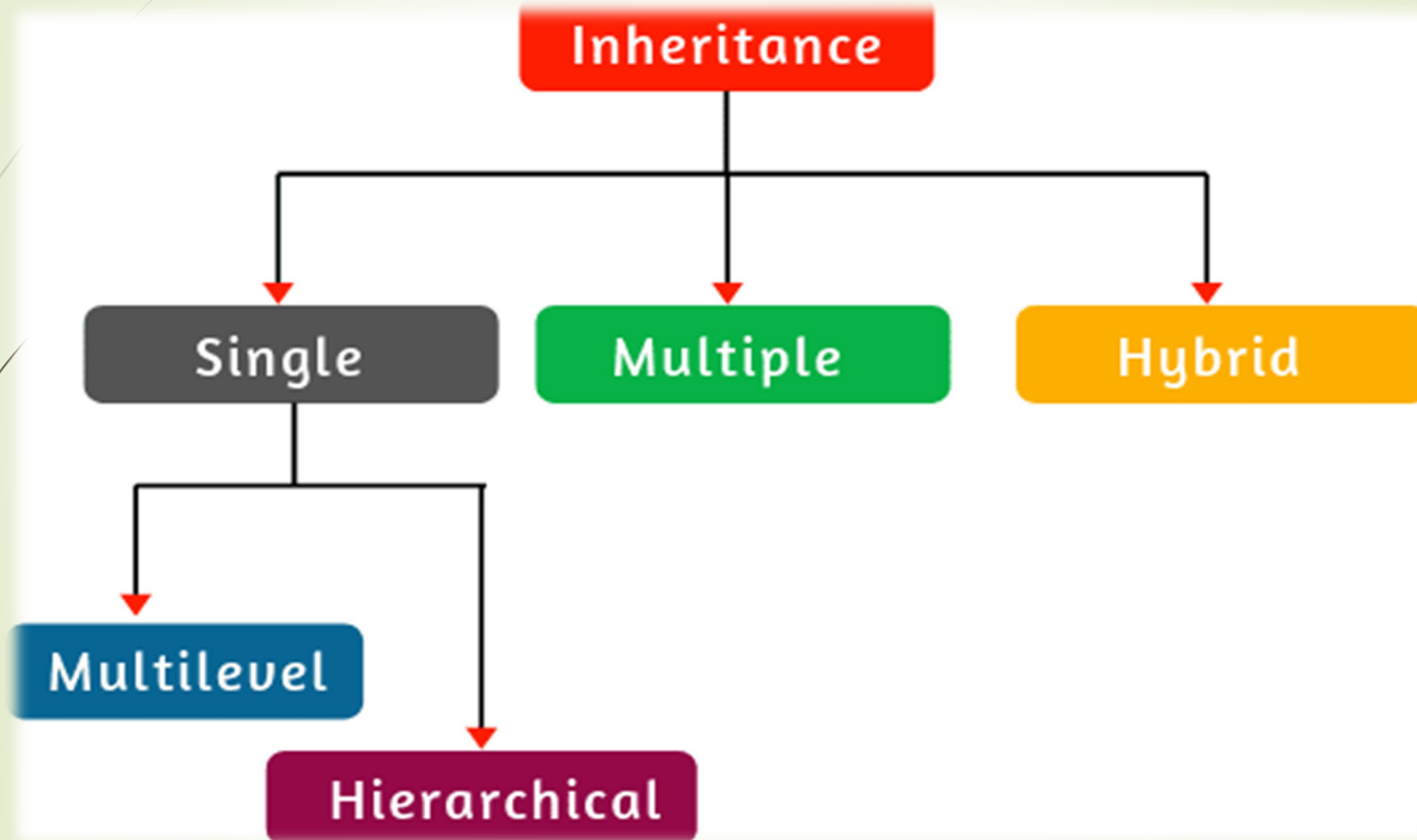
Rules of Inheritance in Java

1. We cannot assign a superclass to the subclass.
2. We cannot extend the final class.
3. A class cannot extend itself.
4. One class can extend only a single class.
5. We cannot extend a class having a private constructor, but if we have private constructor as well as public constructor, then we can extend superclass to subclass. In this case, the only public constructor will work.

Rules of Inheritance in Java (Conti...)

6. If we assign subclass reference to superclass reference, it is called dynamic method dispatch in java.
7. Constructor, Static initialization block (SIB), and Instance initialization block (IIB) of the superclass cannot be inherited to its subclass, but they are executed while creating an object of the subclass.
8. A static method of superclass is inherited to the subclass as a static member and non-static method is inherited as a non-static member only.

Types of Inheritance in Java





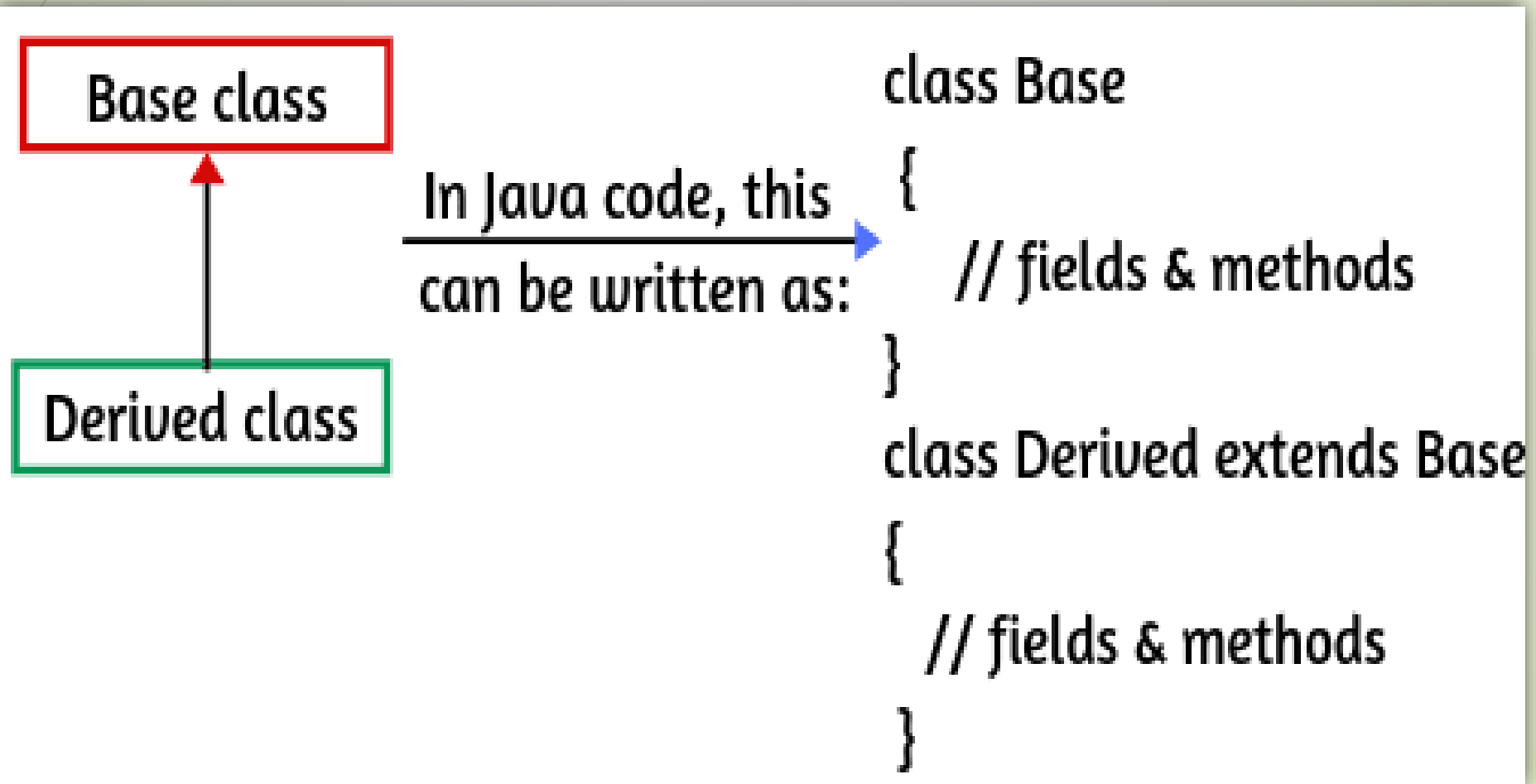
Types of Inheritance in Java

On the basis of class, there are three types of inheritance in Java.
They are as follows:

- 1. Simple/Single level Inheritance**
 - 2. Multiple Inheritance**
 - 3. Hybrid Inheritance**
- 

Single level Inheritance in Java

When a class is extended by only one class, it is called **single-level inheritance** in Java or simply **single inheritance**.



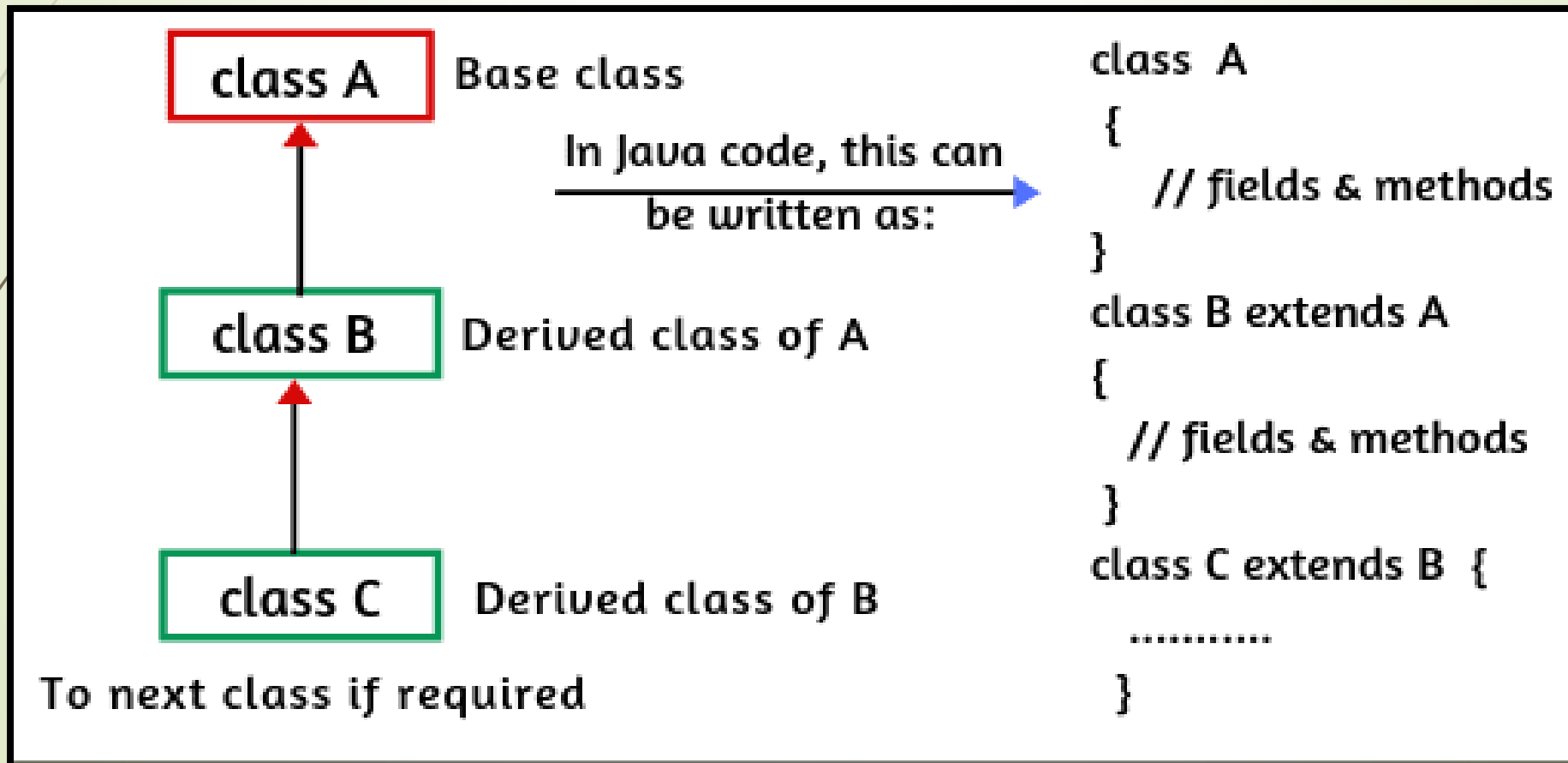
Single level Inheritance in Java (Conti...)

Single inheritance is further classified into types that are as follows:

1. Multilevel Inheritance
2. Hierarchical Inheritance

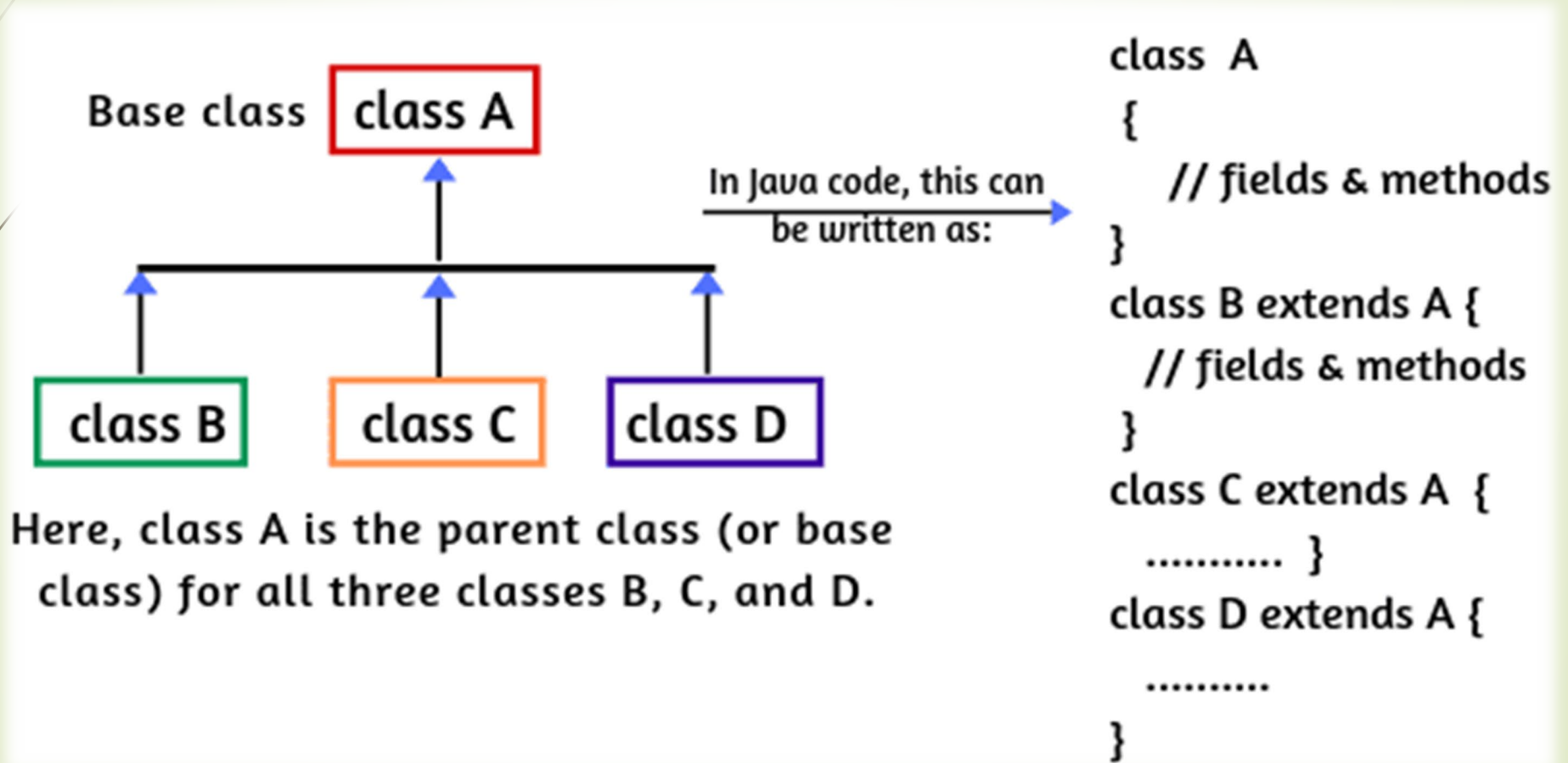
Multilevel Inheritance in Java

A class that is extended by a class and that class is extended by another class forming chain inheritance is called **multilevel inheritance** in Java.



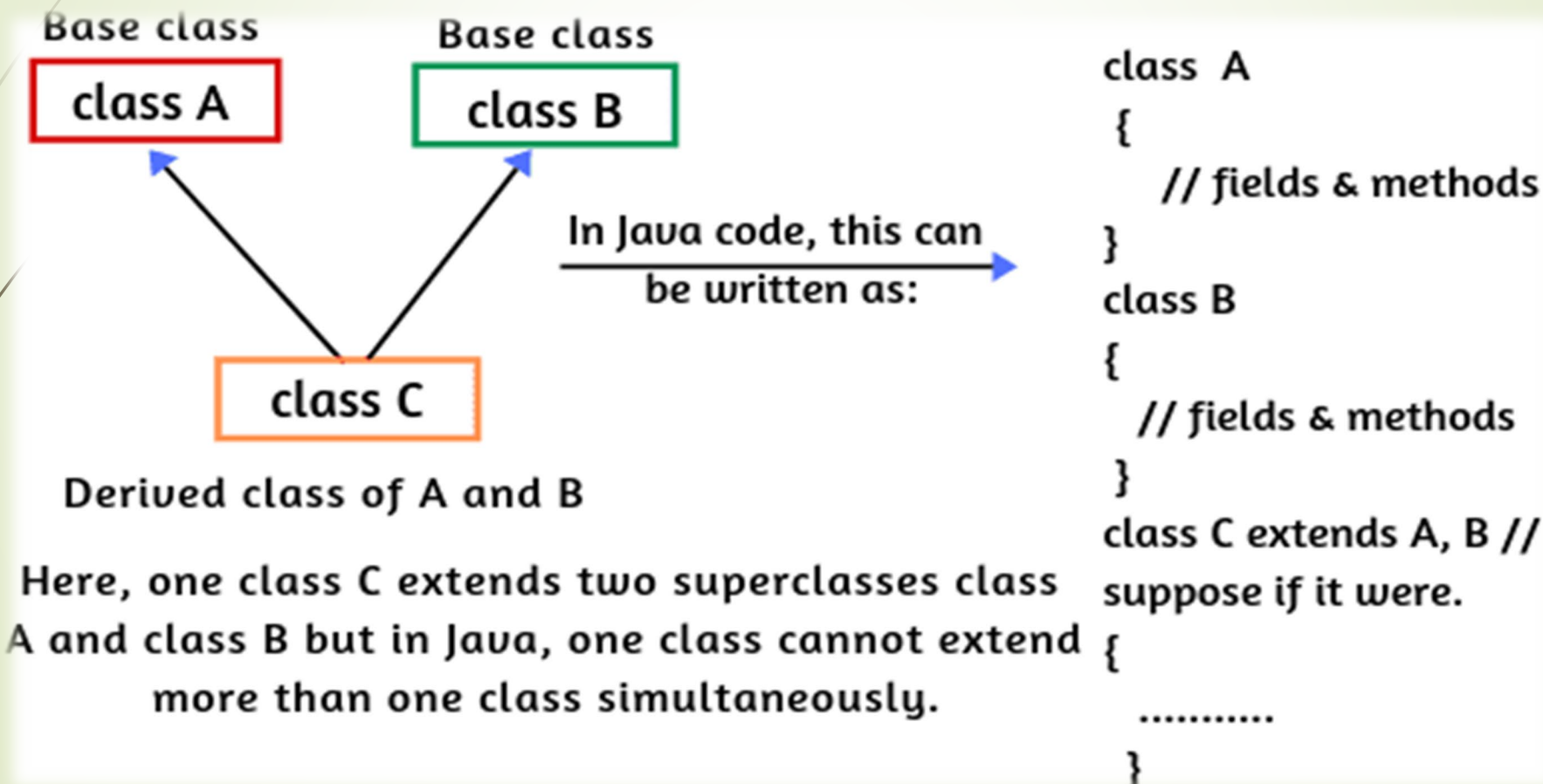
Hierarchical Inheritance in Java

A class that is inherited by many subclasses is known as hierarchical inheritance in Java.



Multiple Inheritance in Java

Deriving subclasses from more than one superclass is known as **multiple inheritance** in Java.



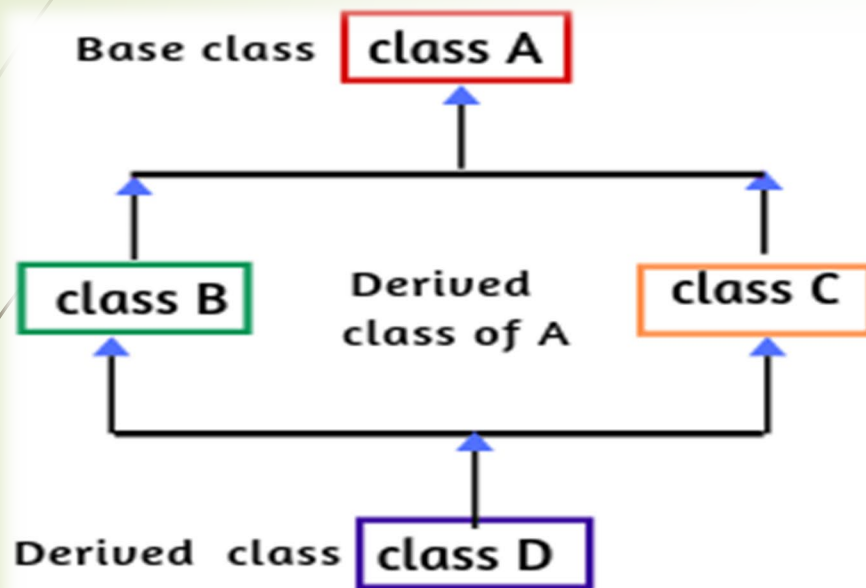
Points to remember in Multiple Inheritance

1. Java does not support multiple inheritance through class.
2. The ambiguity created by the multiple inheritance is called **diamond problem** in Java.
3. Practically, it is very rare and difficult to use in a software project because it creates **ambiguity**, **complexity**, and **confusion** when a class inherits methods from two **superclasses** with the same method signature.
4. Multiple inheritance in Java can be achieved by using interfaces.

Hybrid Inheritance in Java

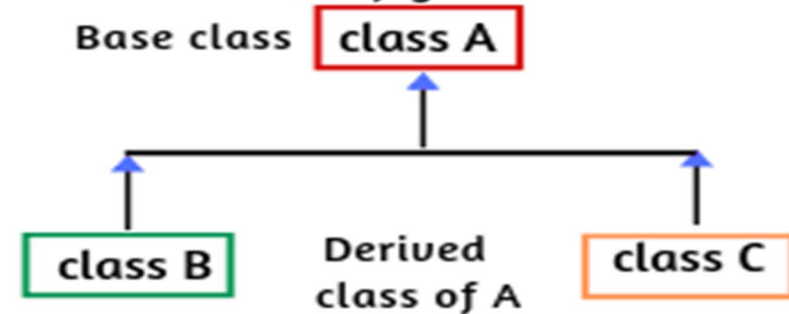
A hybrid inheritance in Java is a combination of single and multiple inheritance.

It can be achieved in the same way as multiple inheritance using interfaces in Java.

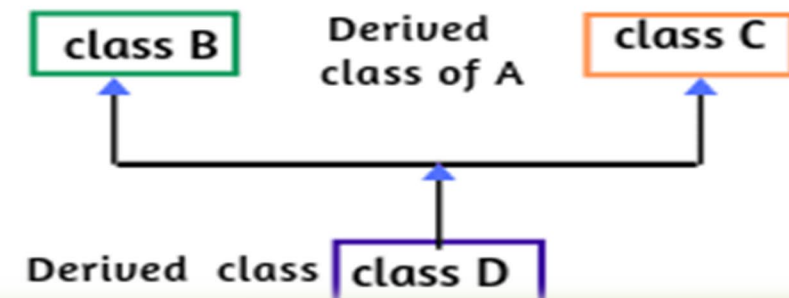


Combining together them as shown in fig , we have a hybrid inheritance.

For the first half of the fig, we have hierarchical inheritance as shown by breaking the fig.



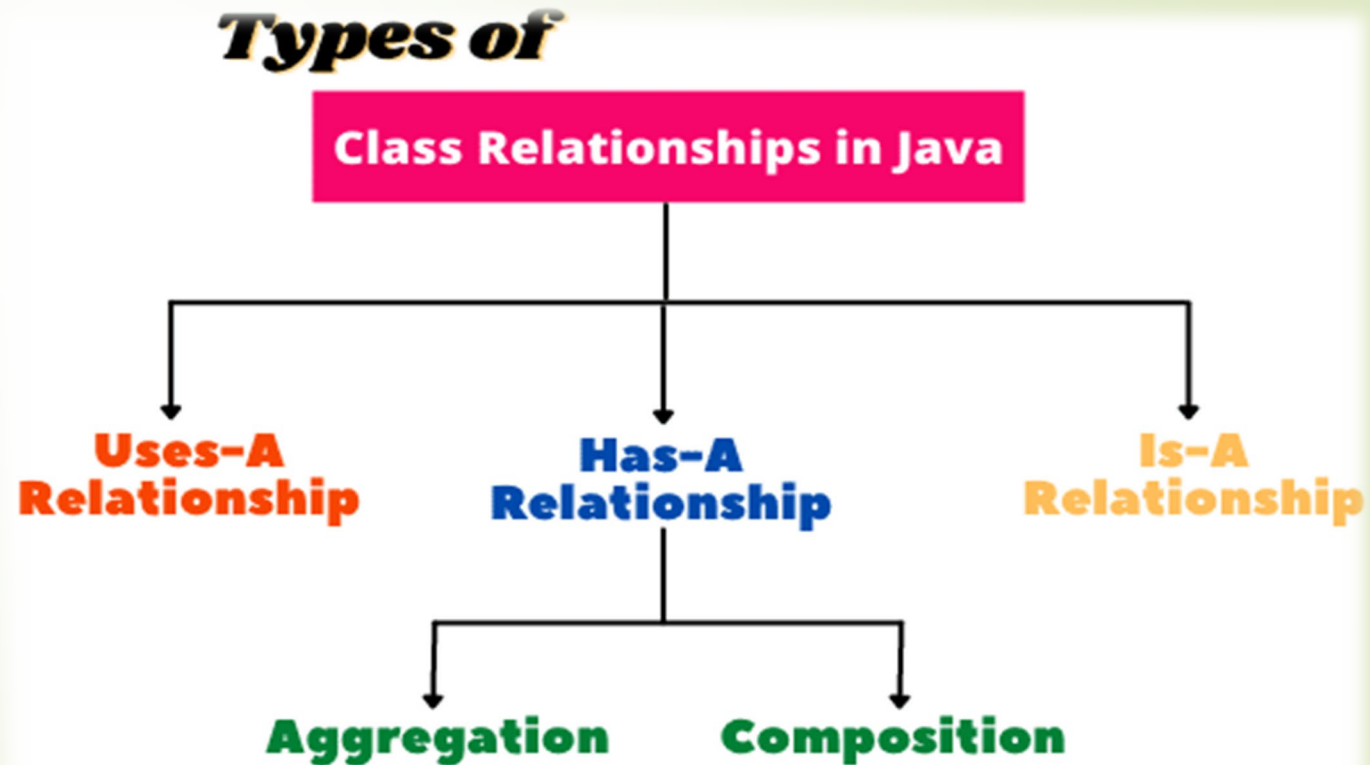
In second half, we have multiple inheritance as shown in the fig.



Class Relationships in Java

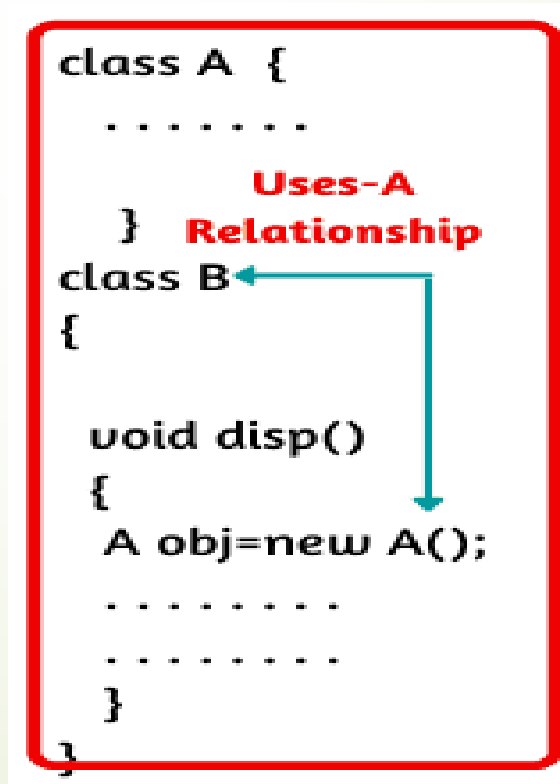
There are three most common relationships among classes in Java that are as follows:

1. Dependence (“Uses-A”)
2. Association (“Has-A”)
3. Inheritance (“Is-A”)



Uses-A Relationship in Java

- When we create an object of a class inside a method of another class, this relationship is called dependence relationship in Java, or simply Uses-A relationship.
- In other words, when a method of a class uses an object of another class, it is called dependency in java.

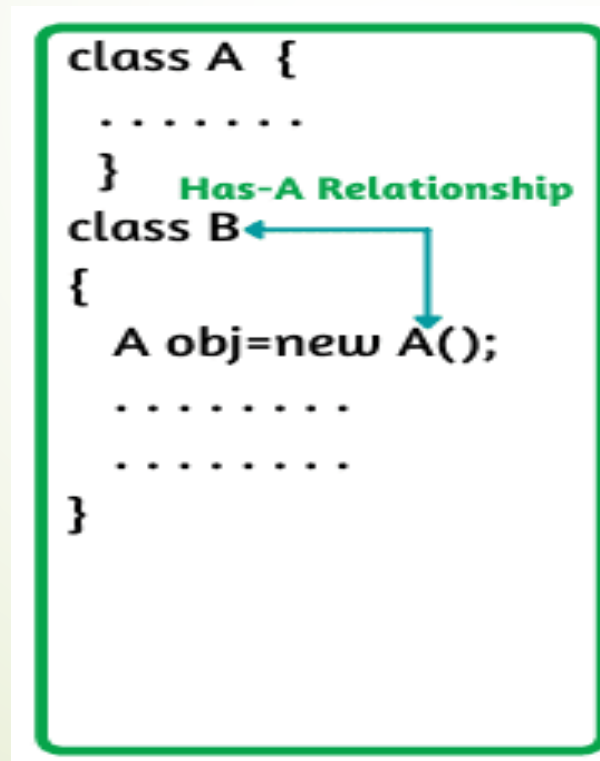


What is coupling and why does coupling matter?

- If several classes of an application program depend on each other, then we say that the coupling between classes is high.
- It is a good programming practice to minimize the dependency between classes (i.e. **coupling**) because too many dependencies make an application program difficult to manage.
- On the other hand, if there are few dependencies between classes, then we say that the coupling between classes is low.
- If a class changes its behavior in the next release of the application program, all the classes that depend on it may also be affected. In this situation, we will need to update all the coupled classes.

Association (“Has-A”) Relationship in Java

- Association is another fundamental relationship between classes that is informally known as “**Has-A**” relationship.
- When an object of one class is created as data member inside another class, it is called association relationship in java or simply **Has-A** relationship.



Types of Has-A Relationship in Java

There are two types of Has-A relationship that are as follows:

1. Aggregation
2. Composition



Aggregation:

- Aggregation is one of the core concepts of object-oriented programming.
- It focuses on establishing a Has-A relationship between two classes.
- In other words, two aggregated objects have their own life cycle, but one of the objects has an owner of Has-A relationship and child object cannot belong to another parent object.

- **For example:**

A library has students. If the library is destroyed, students will exist without library.

A Organization has employees. If the organization gets closed then employees exists without organization.

Composition

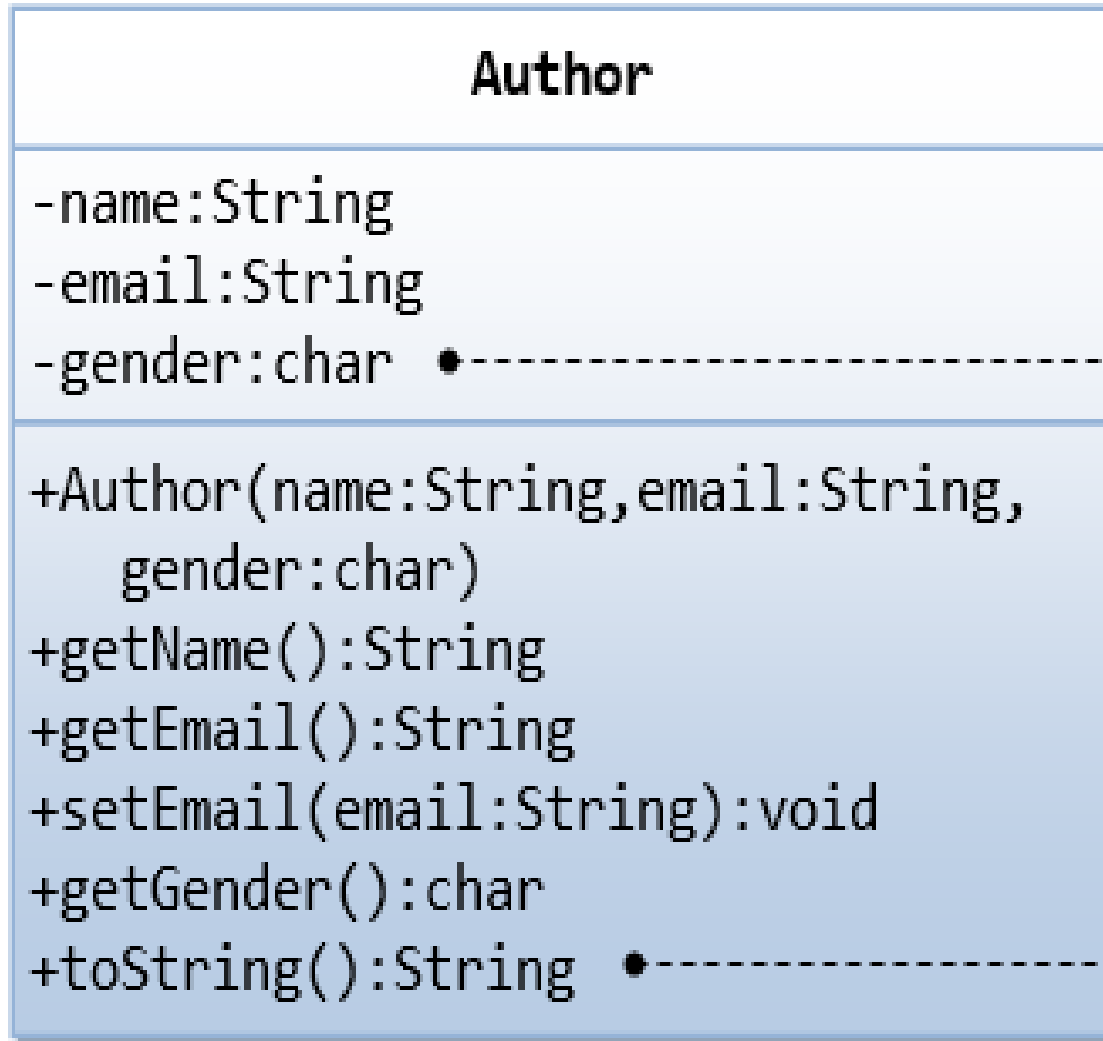
- Composition is another one of the core concepts of object-oriented programming.
- It focuses on establishing a strong Has-A relationship between the two classes.
- Two composited objects cannot have their own life cycle.
- That is, a composite object cannot exist on its own.
- If one composite object is destroyed, all its parts are also be deleted.
- For example:

A house can have multiple rooms. A room cannot exist independently, and any room cannot belong to two different houses. If the house is destroyed, all its rooms will be automatically destroyed.

Difference between Association, Aggregation, and Composition

Association	Aggregation	Composition
1. Association established the relation between two classes that is independent of each another.	1. Aggregation defines a special form of unidirectional association between two classes.	1. Composition represents a special and more restrictive form of aggregation where an object cannot exist on its own.
2. In association, there is no owner relationship.	2. In aggregation, one of the objects is the owner of the Has-A relationship.	2. In composition, both the entities are associated with each other and cannot exist on their own.
3. Association defines the relationships as one-to-one, one-to-many, many-to-one, and many-to-many.	3. It defines only a unidirectional relationship.	3. It represents an exclusive whole-part relationship.

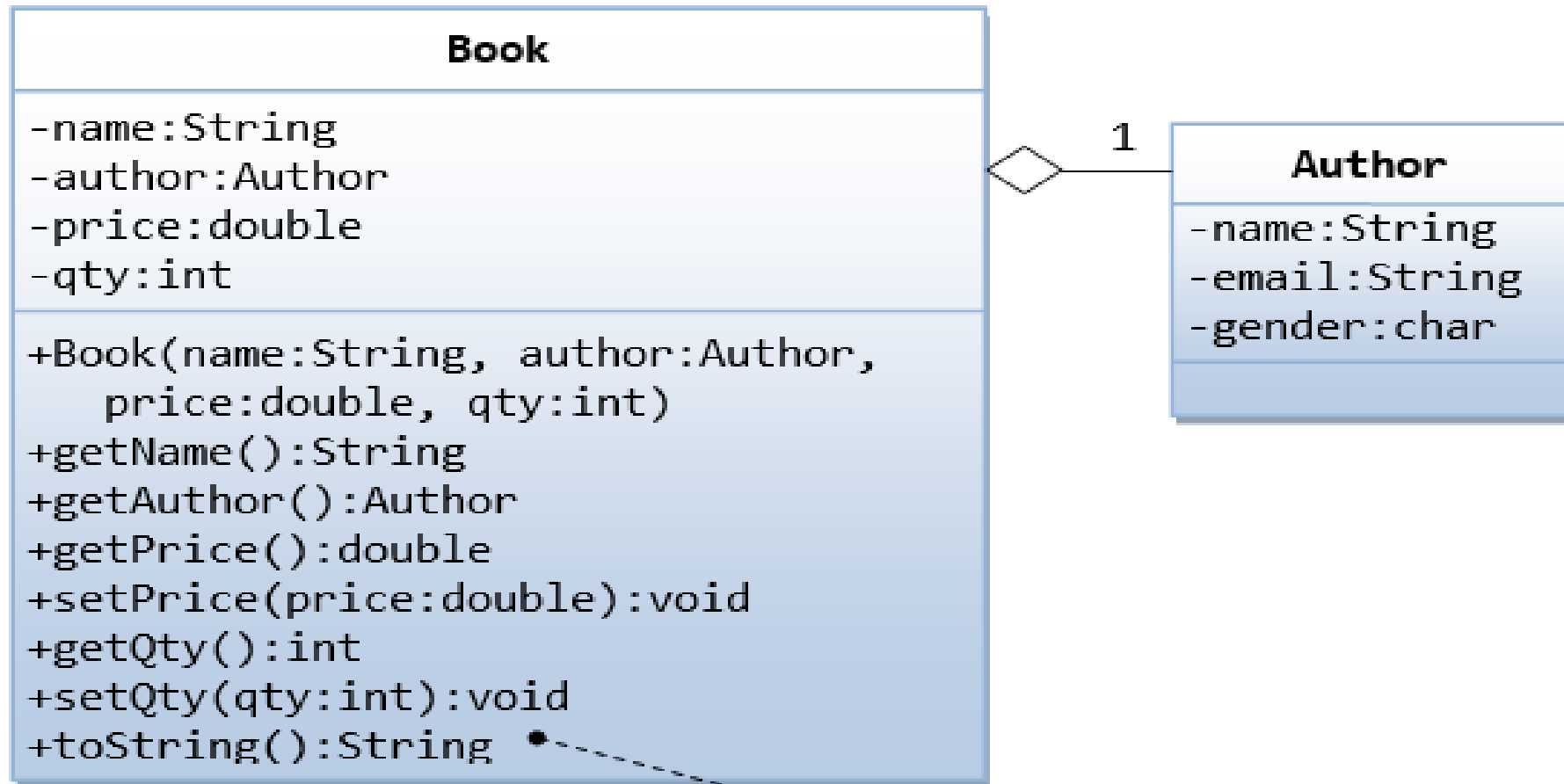
To do



'm' or 'f'

"name (gender) at email"

To do



"'book-name' by author-name (gender) at email"

To do

