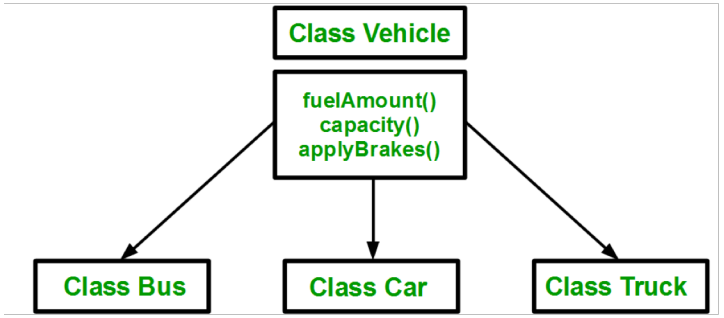
Object orientated programming

Inheritance, aggregation, composition, super, final

Inheritance

- Is a mechanism in which one object acquires all the properties and behaviors of a parent object (important part of OOP).
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and fields of the parent class.
  - You can also add new fields and methods in your current class
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.
- Why use inheritance?
  - For Method Overriding (so runtime polymorphism can be achieved).
  - For Code Reusability.



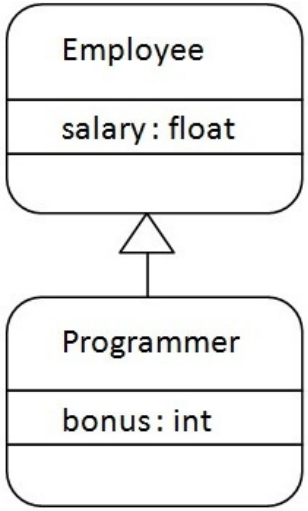
Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **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 where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** Reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Syntax of inheritance

- The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
- In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

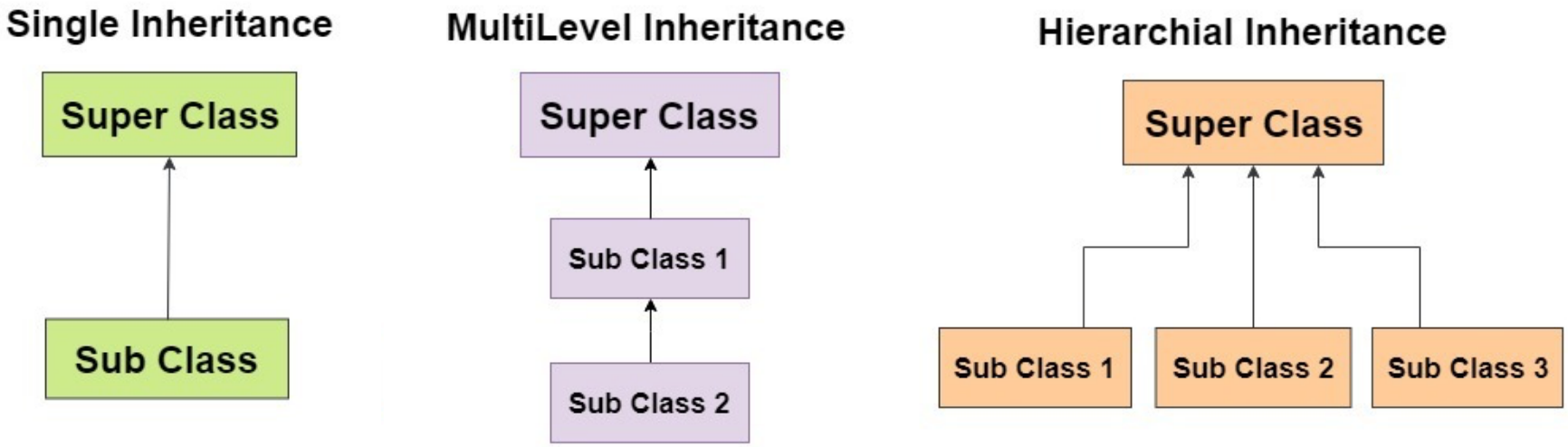
```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```



- Programmer is the subclass and Employee is the superclass. The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee (Employee.java).

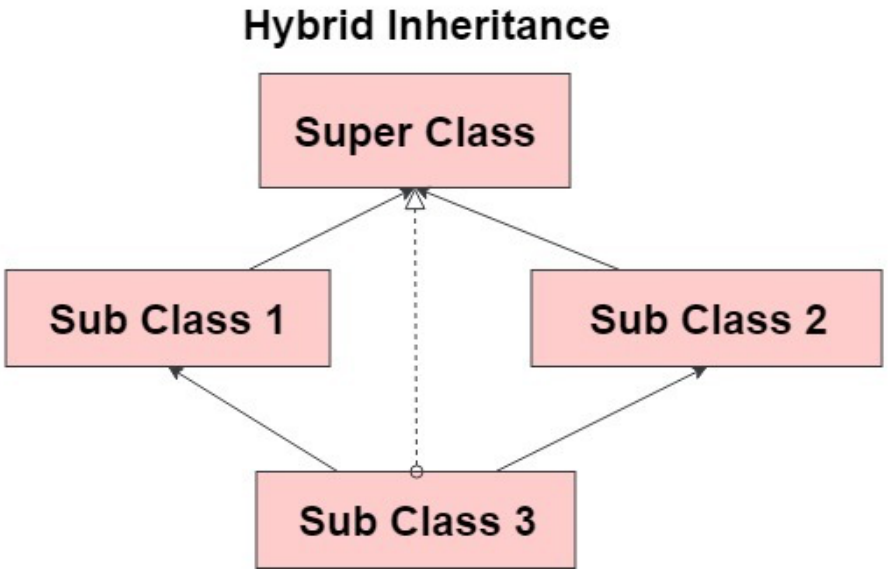
Types of inheritance (i)

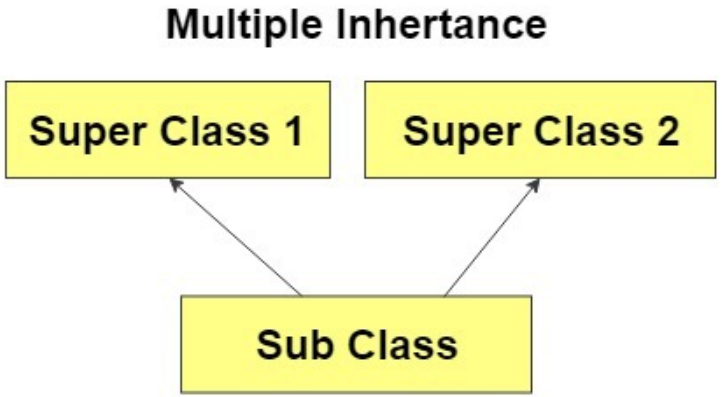
- On the basis of class, there can be three types of inheritance in java: **single, multilevel and hierarchical**.



Types of inheritance (ii)

- When one class inherits multiple classes, it is known as multiple inheritance.
- In java programming, multiple and hybrid inheritance is supported through interface only.





Inhertiance examples

- Single inheritance example (TestInheritance.java)
- Multilevel inheritance example (TestMultilevelInheritance.java)
- Hierarchical inheritance example (TestHierarchicalInheritance.java)

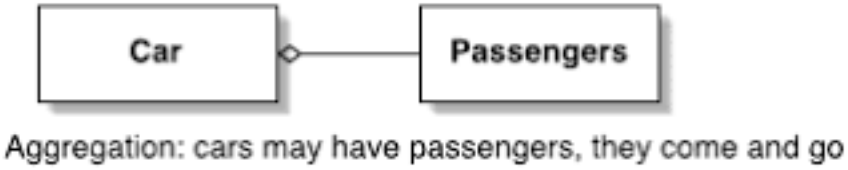
Why multiple inheritance is not supported in java

- Main reason: reduce the complexity and simplify the language.
- **Why?**
- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
- Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have the same method or different, there will be a compile time error (CompilTimeError.java).

Aggregation

- Aggregation represents HAS-A relationship.
  - When a class has an entity reference
  - For code reusability
- Example: An Employee object contains information such as id, name, emailId etc. It contains one more object named address, which contains its own information such as city, state, country, zipcode etc.

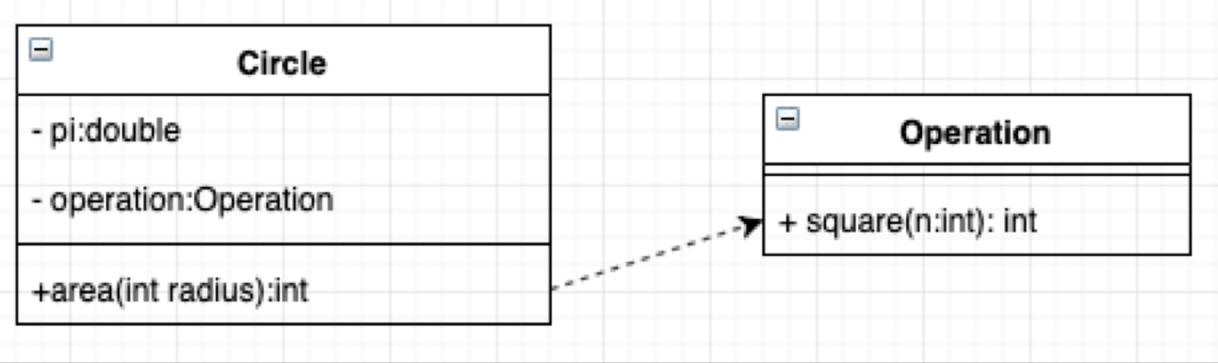
```
class Employee{
int id;
String name;
Address address;//Address is a class
...
}
```



Aggregation: cars may have passengers, they come and go

- Employee has an entity reference address, so relationship is Employee HAS-A address.

Aggregation example



- com.pgr103.aggregation

When to use aggregation?

- Code reuse most efficient by aggregation when there is no is-a relationship.
- If the relationship **is-a** is maintained throughout the lifetime of the objects involved then inheritance is the way to go: otherwise, aggregation is the best choice.
- Meaningful example of aggregation
  - Employee has an object of Address. The address object contains its own information such as city, state, country etc. Relationship: Employee HAS-A address (com.pgr103.com.pgr103.aggregation.example2).

Composition

Composition: every car has an engine.

Aggregation: cars may have passengers, they come and go

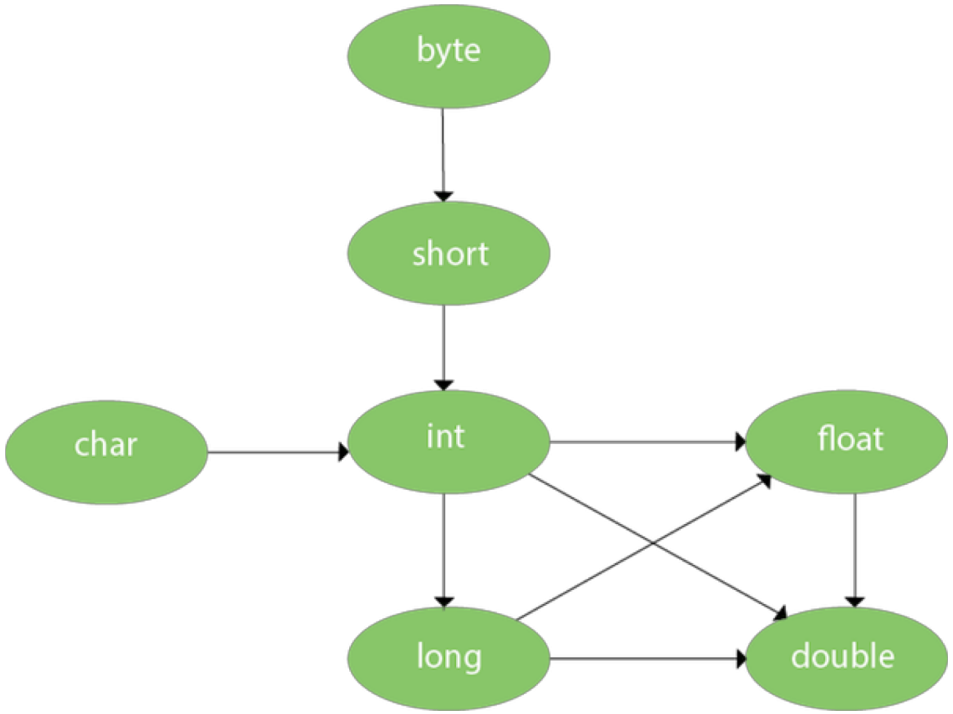
- Composition is a special case of aggregation.
  - Restricted aggregation
- When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.
- Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.
- Composition is the design technique to implement has-a relationship in classes. We can use java inheritance or Object composition for code reuse.
- Java composition is achieved by using instance variables that refers to other objects (com.pgr103.composition example).

Method overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
  - Increase readability of our program
- Two different ways:
  - By changing number of arguments (com.pgr103.overloading.adder.java)
  - By changing the data type (com.pgr103.overloading.adder2.java)
  - Method Overloading is not possible by only changing the return type of the method (**because of ambiguity**, com.pgr103.overloading.adder3.java).
- You could also overload the main method! You can have any number of main methods in a class by method overloading. But JVM calls the main() method which only receives a string array as arguments. Others are ignored.

Method overloading and type promotion

- One type is promoted to another implicitly if no matching datatype is found (OverlaodingCalculation1, 2 and 3 examples).
  - Byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on.

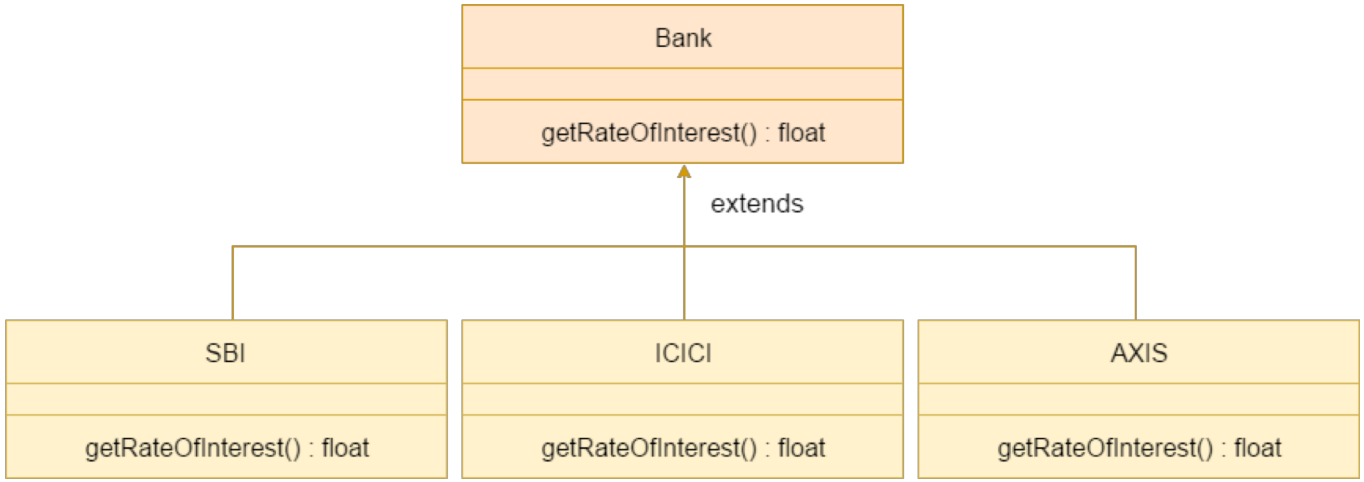


Method overriding

- If a subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
  - The subclass provides the specific implementation of the method that has been declared by one of its parent classes
- Usage of 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
  - 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).

Method overriding examples

- No method overriding (com.pgr103.overriding.bike)
- With method overriding (com.pgr103.overriding.bike)
- A real world example of method overriding
  - Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to the banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest (com.pgr103.overriding.BankRateExample).



Can we override static methods?

- No, a static method cannot be overridden.
- The reason is that the static method is bound with the class whereas the instance method is bound with an object. Static belongs to the class area, and an instance belongs to the object area.
- The main method can also not be overridden because it is a static method.

Differences between method overloading and overriding

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Super keyword

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever an instance of subclass is created, an instance of the parent class is created implicitly which is referred by the super reference variable.
- Usage of super
  - super can be used to refer to an immediate parent class instance variable.
  - super can be used to invoke an immediate parent class method.
  - super() can be used to invoke an immediate parent class constructor.

Super examples

- 1) super is used to refer to an immediate parent class instance variable (com.pgr103.superexamples.TestSuper1).
- 2) super can be used to invoke parent class method (com.pgr103.superexamples.TestSuper2).
- 3) super is used to invoke parent class constructor (com.pgr103.superexamples.TestSuper3).
- Note: super() is added in each class constructor automatically by the compiler if there is no super() or this().
- super example and a real use case
  - Employee class inherits Person class so all the properties of Person will be inherited to Employee by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor (com.pgr103.superexamples.SuperEmployee).

Final keyword

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context.
- Any variable declared as final cannot be changed (It will be constant).
- Final can be:
  - 1) variable – cannot be changed (com.pgr103.finalexamples.Bike)
  - 2) method – cannot be overridden (com.pgr103.finalexamples.Bike2)
  - 3) Class – cannot be extened (com.pgr103.finalexamples.Bike3)
- A final method is inherited but cannot be overridden! (com.pgr103.finalexamples.Bike4)
- A final variable that is not initialized at the time of declaration is known as blank final variable.
  - If you want to create a variable that is initialized at the time of creating an object and once initialized may not be changed, it is useful. For example speed limit of your bike. It can be initialized in the constructor (com.pgr103.finalexamples.Bike5)

Where are we now?

- Constructors
- Static key word
- This key word
- Inheritance
- Aggregation

- Method overloading and overriding in context with OOP
- Polymorphism
- Abstract classes
- Interfaces
- … exception handling…