# Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

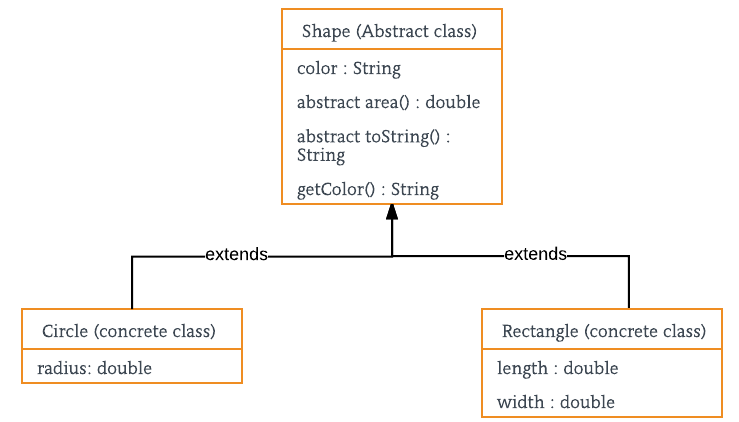
In java, abstraction is achieved by [interfaces](https://www.geeksforgeeks.org/interfaces-in-java/) and [abstract classes](https://www.geeksforgeeks.org/abstract-classes-in-java/). We can achieve 100% abstraction using interfaces.

1. An abstract class is a class that is declared with [abstract keyword.](https://www.geeksforgeeks.org/abstract-keyword-in-java/)
2. An abstract method is a method that is declared without an implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods
4. A method defined abstract must always be redefined in the subclass,thus making [overriding](http://contribute.geeksforgeeks.org/overriding-in-java/) compulsory OR either make subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with abstract keyword.
6. There can be no object of an abstract class.That is, an abstract class can not be directly instantiated with the [new operator](https://www.geeksforgeeks.org/new-operator-java/).
7. An abstract class can have parametrized constructors and default constructor is always present in an abstract class.

**When to use abstract classes and abstract methods with an example**

There are situations in which we will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. That is, sometimes we will want to create a superclass that only defines a generalization form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.

Consider a classic “shape” example, perhaps used in a computer-aided design system or game simulation. The base type is “shape” and each shape has a color, size and so on. From this, specific types of shapes are derived(inherited)-circle, square, triangle and so on – each of which may have additional characteristics and behaviors. For example, certain shapes can be flipped. Some behaviors may be different, such as when you want to calculate the area of a shape. The type hierarchy embodies both the similarities and differences between the shapes.



**Advantages of Abstraction**

1. It reduces the complexity of viewing the things.
2. Avoids code duplication and increases reusability.
3. Helps to increase security of an application or program as only important details are provided to the user.

# Interface

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

* Interfaces specify what a class must do and not how. It is the blueprint of the class.
* An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
* If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.

To declare an interface, use **interface** keyword. It is used to provide total abstraction. That means all the methods in an interface are declared with an empty body and are public and all fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement interface use **implements** keyword.

All methods are abstract and public.

Interface does not contain constructor.

**Why do we use interface ?**

* It is used to achieve total abstraction.
* Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
* It is also used to achieve loose coupling.
* Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?

The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

**New features added in interfaces in JDK 8**

* Prior to JDK 8, interface could not define implementation. We can now add default implementation for interface methods. This default implementation has special use and does not affect the intention behind interfaces. Suppose we need to add a new function in an existing interface. Obviously the old code will not work as the classes have not implemented those new functions. So with the help of default implementation, we will give a default body for the newly added functions. Then the old codes will still work.
* Another feature that was added in JDK 8 is that we can now define static methods in interfaces which can be called independently without an object. Note: these methods are not inherited.

|  |
| --- |
| // An example to show that interfaces can  // have methods from JDK 1.8 onwards  interface In1  {      final int a = 10;      static void display()      {          System.out.println("hello");      }  }    // A class that implements the interface.  class TestClass implements In1  {      // Driver Code      public static void main (String[] args)      {          In1.display();      }  } |

**Important points about interface or summary of article:**

* We can’t create instance(interface can’t be instantiated) of interface but we can make reference of it that refers to the Object of its implementing class.
* A class can implement more than one interface.
* An interface can extends another interface or interfaces (more than one interface) .
* A class that implements interface must implements all the methods in interface.
* All the methods are public and abstract. And all the fields are public, static, and final.
* It is used to achieve multiple inheritance.
* It is used to achieve loose coupling.

**New features added in interfaces in JDK 9**  
From Java 9 onwards, interfaces can contain following also

1. Static methods
2. Private methods
3. Private Static methods

# Concrete class

A concrete class is a class that has an implementation for all of its methods that were inherited from abstract or implemented via interfaces. It also does not define any abstract methods of its own. This means that an instance of the class can be created/allocated with the **new** keyword without having to implement any methods first. Therefore it can be inferred that any class that is not an abstract class or interface is a concrete class.

# Inheritence

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.  
**Important terminology:**

* **Super Class:**The class whose features are inherited is known as super class(or a base class or a parent class).
* **Sub Class:** The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
* **Reusability:**Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

The keyword used for inheritance is **extends**.  
Syntax :

class derived-class extends base-class

{

//methods and fields

}

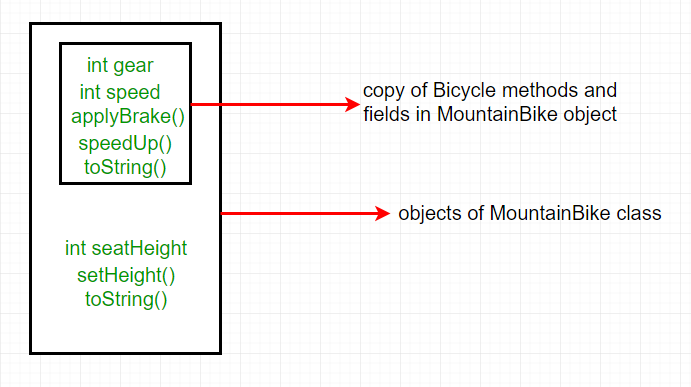
|  |
| --- |
| //Java program to illustrate the  // concept of inheritance    // base class  class Bicycle  {      // the Bicycle class has two fields      public int gear;      public int speed;        // the Bicycle class has one constructor      public Bicycle(int gear, int speed)      {          this.gear = gear;          this.speed = speed;      }        // the Bicycle class has three methods      public void applyBrake(int decrement)      {          speed -= decrement;      }        public void speedUp(int increment)      {          speed += increment;      }        // toString() method to print info of Bicycle      public String toString()      {          return("No of gears are "+gear                  +"\n"                  + "speed of bicycle is "+speed);      }  }    // derived class  class MountainBike extends Bicycle  {        // the MountainBike subclass adds one more field      public int seatHeight;        // the MountainBike subclass has one constructor      public MountainBike(int gear,int speed,                          int startHeight)      {          // invoking base-class(Bicycle) constructor          super(gear, speed);          seatHeight = startHeight;      }        // the MountainBike subclass adds one more method      public void setHeight(int newValue)      {          seatHeight = newValue;      }        // overriding toString() method      // of Bicycle to print more info      @Override      public String toString()      {          return (super.toString()+                  "\nseat height is "+seatHeight);      }    }    // driver class  public class Test  {      public static void main(String args[])      {            MountainBike mb = new MountainBike(3, 100, 25);          System.out.println(mb.toString());        }  } |

**Output:**

No of gears are 3

speed of bicycle is 100

seat height is 25

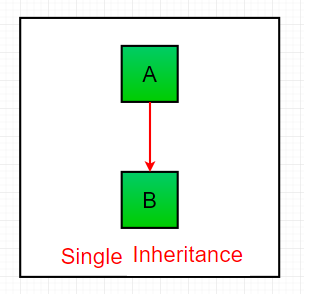
In above program, when an object of MountainBike class is created, a copy of the all methods and fields of the superclass acquire memory in this object. That is why, by using the object of the subclass we can also access the members of a superclass.  
Please note that during inheritance only object of subclass is created, not the superclass. For more, refer [Java Object Creation of Inherited Class](https://www.geeksforgeeks.org/gfact-52-java-object-creation-of-inherited-classes/).  
**Illustrative image of the program:**  
[](https://media.geeksforgeeks.org/wp-content/uploads/inheritence1.png)

In practice, inheritance and [polymorphism](https://www.geeksforgeeks.org/overriding-in-java/) are used together in java to achieve fast performance and readability of code.

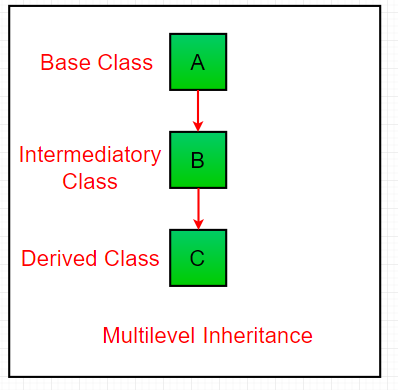
**Types of Inheritance in Java**

Below are the different types of inheritance which is supported by Java.

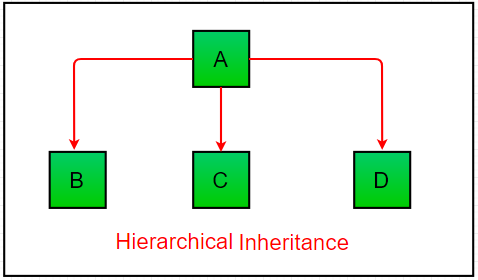
* 1. **Single Inheritance :**In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance1.png)

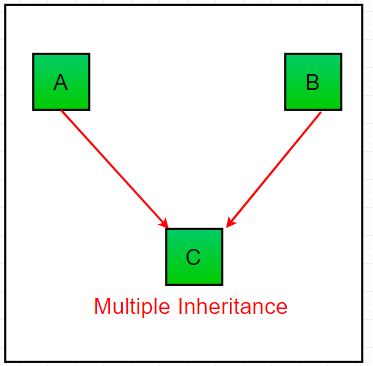
**Multilevel Inheritance :**In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the[grandparent’s members](https://www.geeksforgeeks.org/g-fact-91/).

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance3.png)

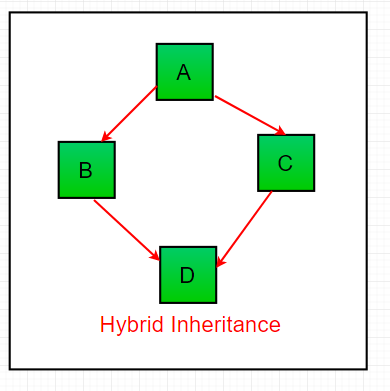
1. **Hierarchical Inheritance :**In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance4.png)

[**Multiple Inheritance**](https://www.geeksforgeeks.org/java-and-multiple-inheritance/)**(Through Interfaces) :**In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. In java, we can achieve multiple inheritance only through [Interfaces](http://quiz.geeksforgeeks.org/interfaces-in-java/). In image below, Class C is derived from interface A and B.

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance2-1.png)

**Hybrid Inheritance(Through Interfaces) :**It is a mix of two or more of the above types of inheritance. Since java doesn’t support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through [Interfaces](http://quiz.geeksforgeeks.org/interfaces-in-java/).

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance-1.png)

**Important facts about inheritance in Java**

* **Default superclass**: Except [Object](https://www.geeksforgeeks.org/object-class-in-java/) class, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of [Object](https://www.geeksforgeeks.org/object-class-in-java/) class.
* **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only **one** superclass. This is because Java does not support [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. Although with interfaces, multiple inheritance is supported by java.
* **Inheriting Constructors:**A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
* **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods(like getters and setters) for accessing its private fields, these can also be used by the subclass.
* When child class constructor called by default super class constructor is called.

**What all can be done in a Subclass?**

In sub-classes we can inherit members as is, replace them, hide them, or supplement them with new members:

* The inherited fields can be used directly, just like any other fields.
* We can declare new fields in the subclass that are not in the superclass.
* The inherited methods can be used directly as they are.
* We can write a new *instance* method in the subclass that has the same signature as the one in the superclass, thus [overriding](https://www.geeksforgeeks.org/overriding-in-java/) it (as in example above, *toString()* method is overridden).
* We can write a new *static* method in the subclass that has the same signature as the one in the superclass, thus [hiding](https://www.geeksforgeeks.org/g-fact-63/) it.
* We can declare new methods in the subclass that are not in the superclass.
* We can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword [super](http://quiz.geeksforgeeks.org/super-keyword/).

# **super and this keywords in Java**

super keyword is used to access methods of the **parent class** while this is used to access methods of the **current class**

[**this keyword**](https://www.geeksforgeeks.org/this-reference-in-java/)

1. **this** is a reserved keyword in java i.e, we can’t use it as an identifier.
2. **this** is used to refer **current-class’s instance as well as static members.**

[**super**](https://www.geeksforgeeks.org/super-keyword/)

1. **super** is a reserved keyword in java i.e, we can’t use it as an identifier.
2. **super** is used to refer **super-class’s instance as well as static members**.
3. We can use this as well as super **anywhere except static area**.

Because this and super refers to the object instance. There is no object instance in a call of a static method.

2) We can use this as well as super **any number of times in a program**.

# Polymorphism

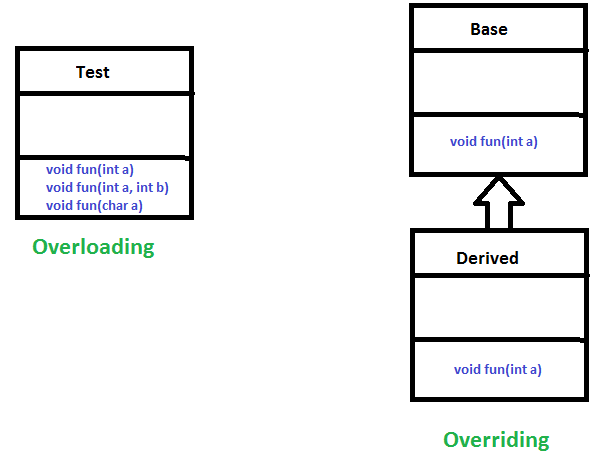
The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

**Real life example of polymorphism:** A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.

Polymorphism is considered as one of the important features of Object Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

**In Java polymorphism is mainly divided into two types:**

* Compile time Polymorphism
* Runtime Polymorphism

[](https://www.geeksforgeeks.org/overloading-in-java/)

**Compile time polymorphism**: It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

**Method Overloading**: When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

**Operator Overloading**: Java also provide option to overload operators. For example, we can make the operator (‘+’) for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands. So a single operator ‘+’ when placed between integer operands, adds them and when placed between string operands, concatenates them.

In java, Only “+” operator can be overloaded:

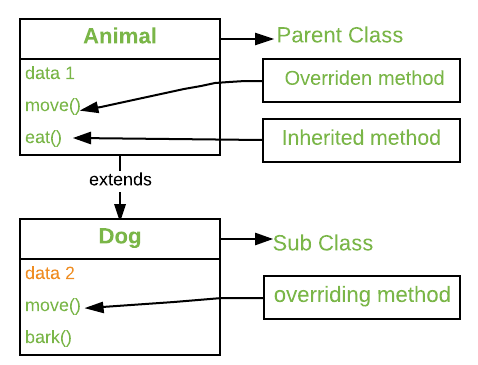
* To add integers
* To concatenate strings

[**Runtime polymorphism**](https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/): It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

* [**Method overriding**](https://www.geeksforgeeks.org/overriding-in-java/), on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

# Overriding

In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.



Method overriding is one of the way by which java achieve [Run Time Polymorphism](https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/).The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

**Rules for method overriding:**

1. **Overriding and Access-Modifiers :**The [access modifier](https://www.geeksforgeeks.org/access-modifiers-java/) for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the super-class can be made public, but not private, in the subclass. Doing so, will generate compile-time error.
2. **Final methods can not be overridden :**If we don’t want a method to be overridden, we declare it as [final](https://www.geeksforgeeks.org/final-keyword-java/).
3. **Static methods can not be overridden(Method Overriding vs Method Hiding) :**When you define a static method with same signature as a static method in base class, it is known as [method hiding](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/).
4. The following table summarizes what happens when you define a method with the same signature as a method in a super-class.

|  |  |  |
| --- | --- | --- |
|  | **SUPERCLASS INSTANCE METHOD** | **SUPERCLASS STATIC METHOD** |
| **SUBCLASS INSTANCE METHOD** | Overrides | Generates a compile-time error |
| **SUBCLASS STATIC METHOD** | Generates a compile-time error | Hides |

1. **Private methods can not be overridden :**[Private methods](https://www.geeksforgeeks.org/can-override-private-methods-java/)cannot be overridden as they are bonded during compile time. Therefore we can’t even override private methods in a subclass.(See [this](https://www.geeksforgeeks.org/can-override-private-methods-java/) for details).

1. **The overriding method must have same return type (or subtype) :**From Java 5.0 onwards it is possible to have different return type for a overriding method in child class, but child’s return type should be sub-type of parent’s return type. This phenomena is known as [**covariant return type**](https://www.geeksforgeeks.org/covariant-return-types-java/).

1. **Invoking overridden method from sub-class :**We can call parent class method in overriding method using [super keyword](http://quiz.geeksforgeeks.org/super-keyword/).
2. **Overriding and constructor :**We can not override constructor as parent and child class can never have constructor with same name(Constructor name must always be same as Class name).
3. **Overriding and Exception-Handling :**Below are two rules to note when overriding methods related to exception-handling.
   1. **Rule#1 :** If the super-class overridden method does not throw an exception, subclass overriding method can only throws the [unchecked exception](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/), throwing checked exception will lead to compile-time error.
   2. **Rule#2 :** If the super-class overridden method does throws an exception, subclass overriding method can only throw same, subclass exception. Throwing parent exception in [Exception hierarchy](https://www.geeksforgeeks.org/exceptions-in-java/) will lead to compile time error.Also there is no issue if subclass overridden method is not throwing any exception.
4. **Overriding and abstract method:**Abstract methods in an interface or abstract class are meant to be overridden in derived concrete classes otherwise a compile-time error will be thrown.

1. **Overriding and synchronized/strictfp method :**The presence of synchronized/strictfp modifier with method have no effect on the rules of overriding, i.e. it’s possible that a synchronized/strictfp method can override a non synchronized/strictfp one and vice-versa.

# Overloading

Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both. Overloading is related to compile-time (or static) polymorphism.

**What is the advantage?**  
We don’t have to create and remember different names for functions doing the same thing. For example, in our code, if overloading was not supported by Java, we would have to create method names like sum1, sum2, … or sum2Int, sum3Int, … etc.

**Overloaded methods** in **java** may **have** different **return types** given that the argument **is** also different. Check out the sample code. The compiler **does** not consider **return type** when differentiating **methods**, so you cannot declare two **methods** with the **same** signature even if they **have** a different **return type**

**Can we overload methods on return type?**  
We **cannot** overload by return type

**Can we overload static methods?**  
The answer is ‘**Yes**’. We can have two ore more static methods with same name, but differences in input parameters.

**Can we overload methods that differ only by static keyword?**  
We **cannot** overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same).

**Can we overload main() in Java?**  
Like other static methods, we **can** [overload main() in Java](https://www.geeksforgeeks.org/gfact-48-overloading-main-in-java/)

# Encapsulation

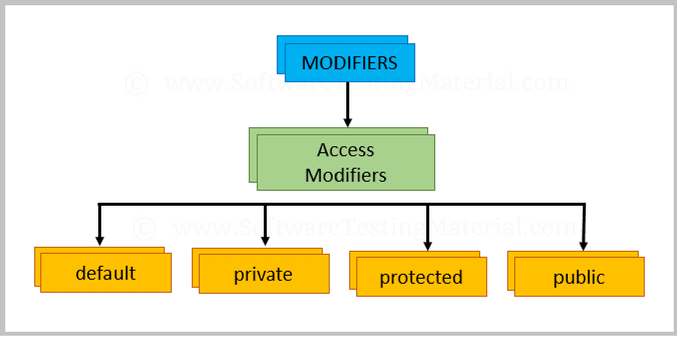
Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates.Other way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

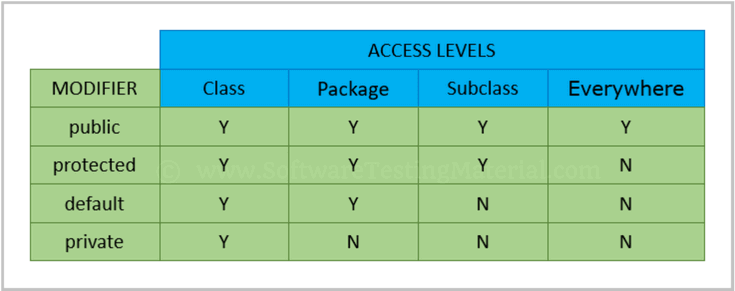
* Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
* As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of class as private and the class is exposed to the end user or the world without providing any details behind implementation using the abstraction concept, so it is also known as **combination of data-hiding and abstraction.**.
* Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

**Advantages of Encapsulation**:

* **Data Hiding:** The user will have no idea about the inner implementation of the class. It will not be visible to the user that how the class is storing values in the variables. He only knows that we are passing the values to a setter method and variables are getting initialized with that value.
* **Increased Flexibility:** We can make the variables of the class as read-only or write-only depending on our requirement. If we wish to make the variables as read-only then we have to omit the setter methods like setName(), setAge() etc. from the above program or if we wish to make the variables as write-only then we have to omit the get methods like getName(), getAge() etc. from the above program
* **Reusability:** Encapsulation also improves the re-usability and easy to change with new requirements.
* **Testing code is easy:** Encapsulated code is easy to test for unit testing.

# Access modifiers





Is a relation(4) , has a , composition

aggregarion