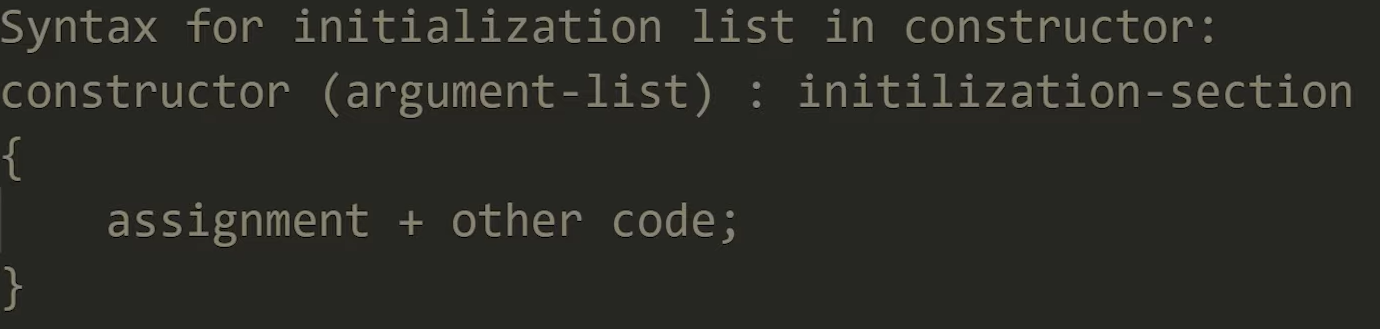
**­­­­­­­­­Object Oriented Programming –**

Refer - <https://www.youtube.com/watch?v=a199KZGMNxk&ab_channel=AnujBhaiya>

**Initializer List –**

**Refer** - <https://www.youtube.com/watch?v=-Re7K7mHtv4&ab_channel=CodeWithHarry>

****

**IMP -**

When you **initialize** fields via Member initializer list, the constructors will be called once and the object will be constructed and initialized in one operation.

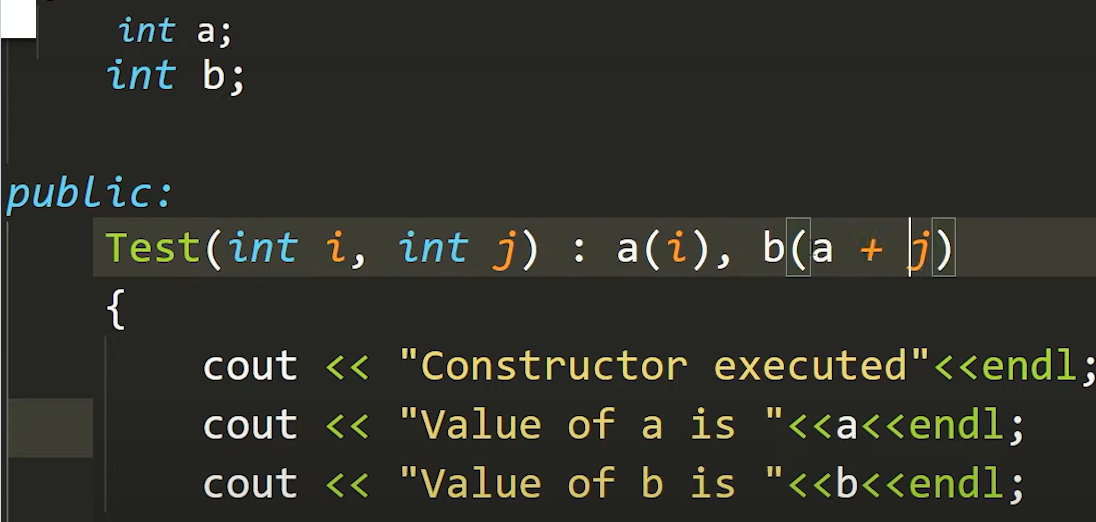
If you use **assignment** then the fields will be first initialized with default constructors and then reassigned (via assignment operator) with actual values.

Cost of Member Initialization = Object Construction

Cost of Member Assignment = Object Construction + Assignment

**Eg –**

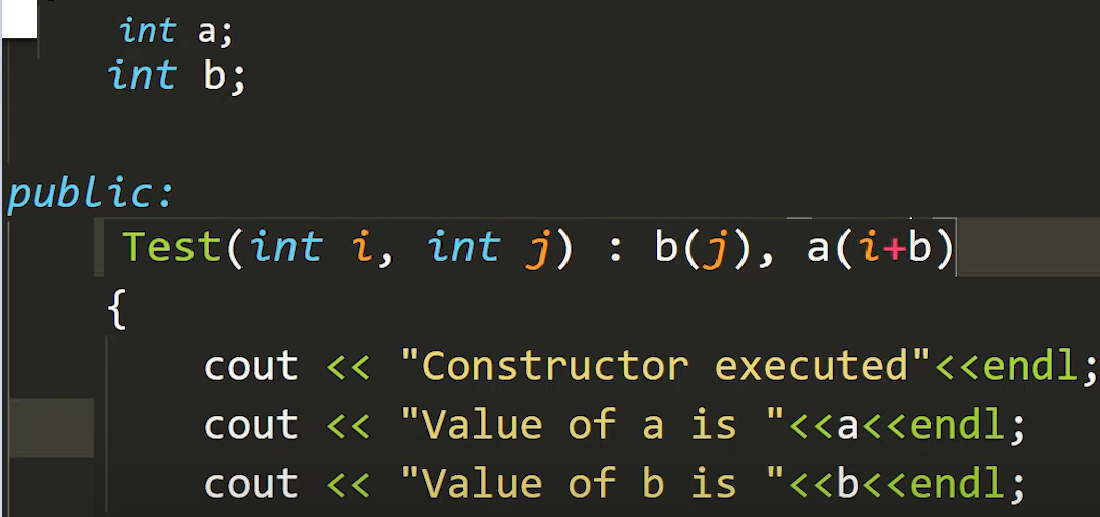
Test obj(4,6)

 Output –> a=4, b = 10

**Eg2 –**

Test obj(4,6)

­­­­

 output –> a = garbage value, b = ­6

**Important points to Note while using Member Initializer Lists:**

Class Member variables are always initialized in the order in which they are declared in the class.

They are not initialized in the order in which they are specified in the Member Initializer List.

In short, Member initialization list does not determine the order of initialization.

When do you HAVE TO use Member Initializer list?

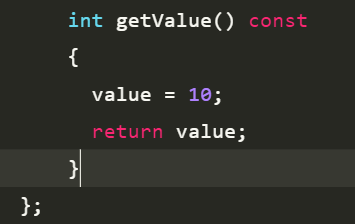
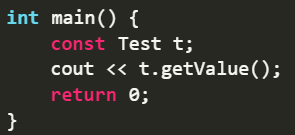
You will **have(rather forced) to** use a Member Initializer list if:

* Your class has a reference member
* Your class has a non static const member or
* Your class member doesn't have a default constructor or
* For initialization of base class members or
* When constructor’s parameter name is same as data member(this is not really a MUST)

**Const object , member functions, variable and function arguments –**

* **const object** - can invoke only const member functions
* **non-const object** – can invoke const and non-const member function

**Const with method - Const with objects -**

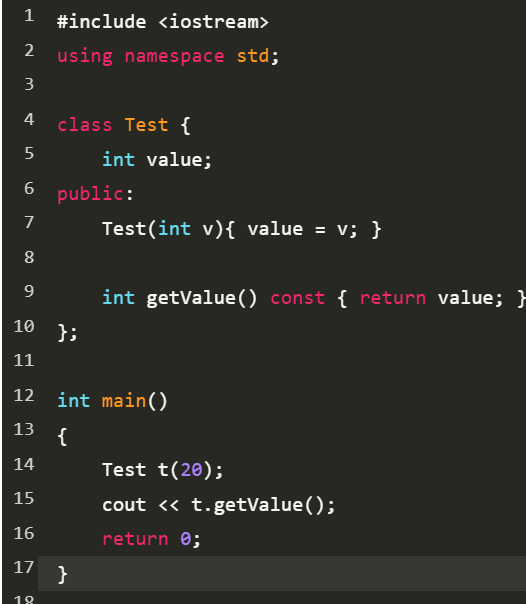
**Explanation -**

When we don’t want an object to be modified, then either we can make it a const object, so that only const functions can be called using it and a const function cannot change the value of an object.

It is recommended the practice to make as many functions const as possible so that accidental changes to objects are avoided.

**Eg –**

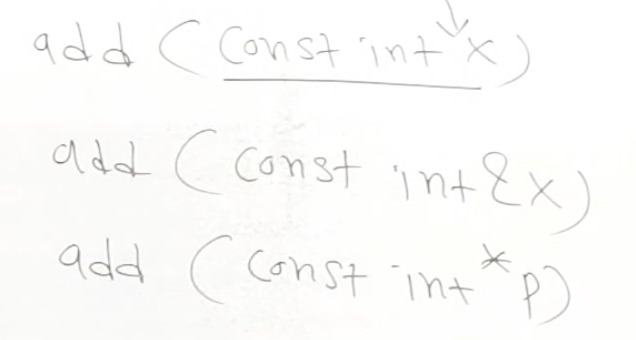
**No error - Error -**

**Const arugments –**

The qualifier const tells the compiler that the value of that argument cannot be modified, and any such attempt will generate a compile time error. We can indirectly modify the value of the const argument by using pointer. Compile time error occurs only when we pass arguments by reference or pointer.

Eg –



### **Const variable –**

it prevents the variable to modify its data items value.

Rules –

* The const variable cannot be left un-initialized at the time of the assignment.
* It cannot be assigned value anywhere in the program.
* Explicit value needed to be provided to the constant variable at the time of declaration of the constant variable.

Eg – const int a = 5;

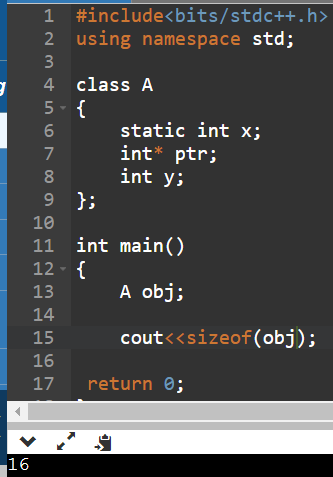
### **Is it always necessary to create objects from class?**

No. An object is necessary to be created if the base class has non-static methods. But if the class has static methods, then objects don’t need to be created. You can call the class method directly in this case, using the class name.

**Static variables and methods –**

When we declare a variable/method as static, it becomes the property of a class and not the object. They can be called using the class name. So when finding the size of an object of a class, we consider the size of the static variable as 0.

**Eg –**



Size of a pointer is always = 8bytes(64 bit compiler), size of int = 4bytes (64 bit compiler)

Total size of object = 16 bytes (due to padding concept)

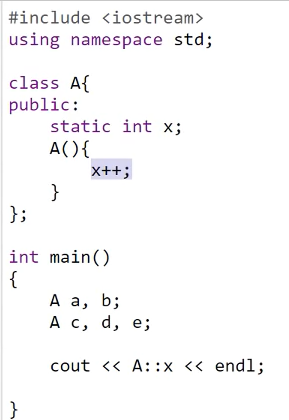
There is only one copy of a static variable or method(of another or same class) for the whole class. But we can change its value any number of times.f

Static methods can be public or private.

Static methods only have access to other static variables and static methods.

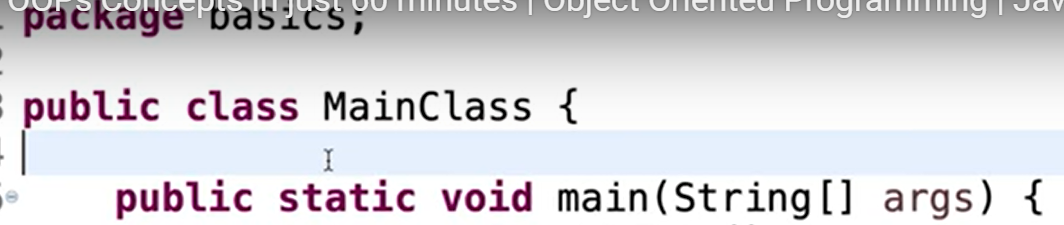
non-static methods have access to all variables and methods (static or non-static) in the class.

Eg –

, **output** = 5

**Eg 2 -** Just like the main method can be called without creating the object of mainClass

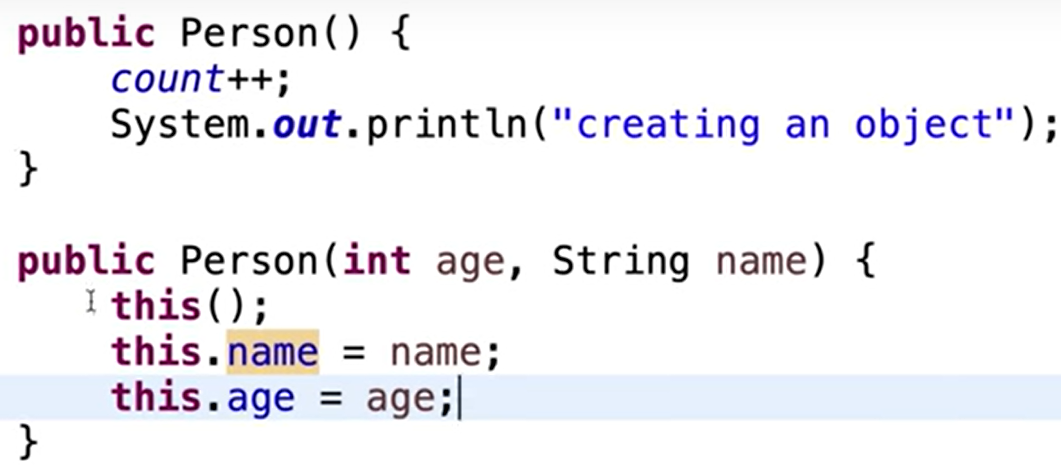
Refer -



**This keyword vs super keyword – (JAVA)**

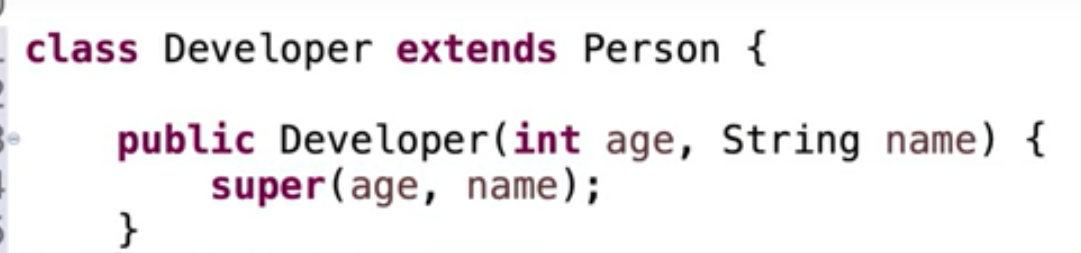
* 1. We can invoke the constructor of the current class.
  2. We use this to denote the global variables of that class.

**Eg -**

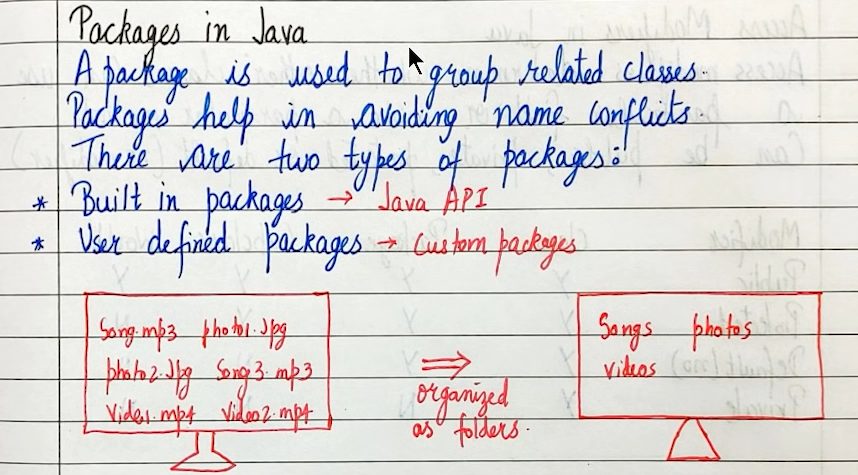
****

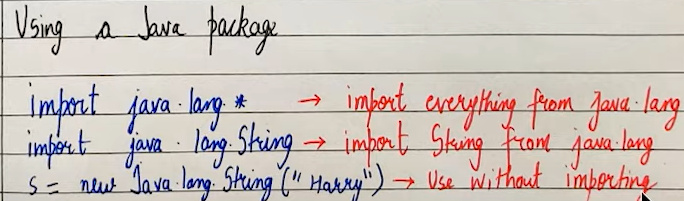
**Super keyword –**

It is used to call the constructor of it’s parent.

****

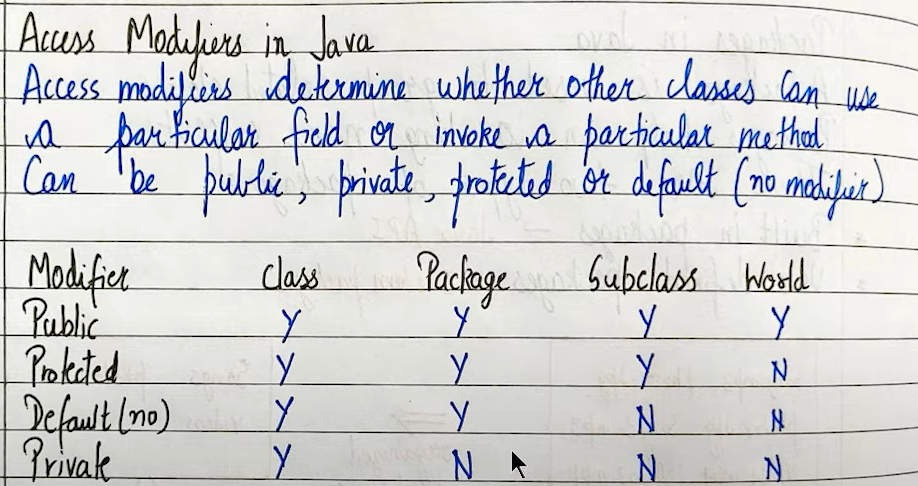
**Packages in JAVA –**

****

****

**It is just like libraries. Even we can create a lot of classes and put it in the same package.**

**Access Modifiers in Java -**

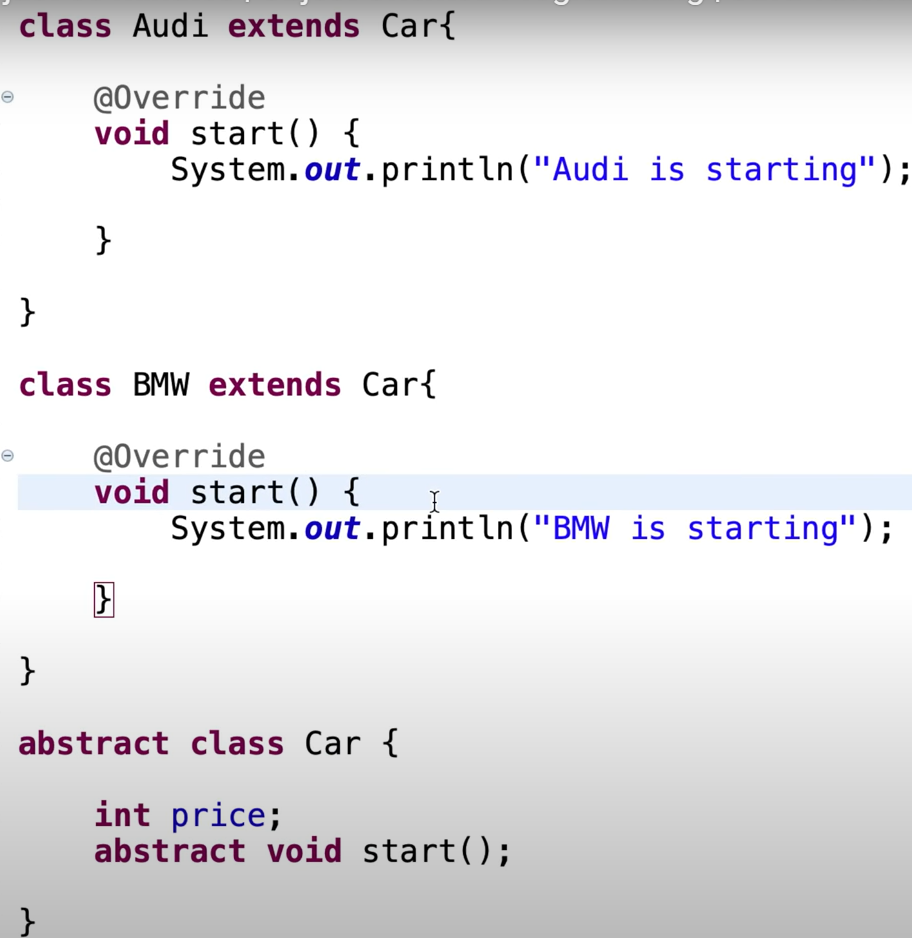
****

### **Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

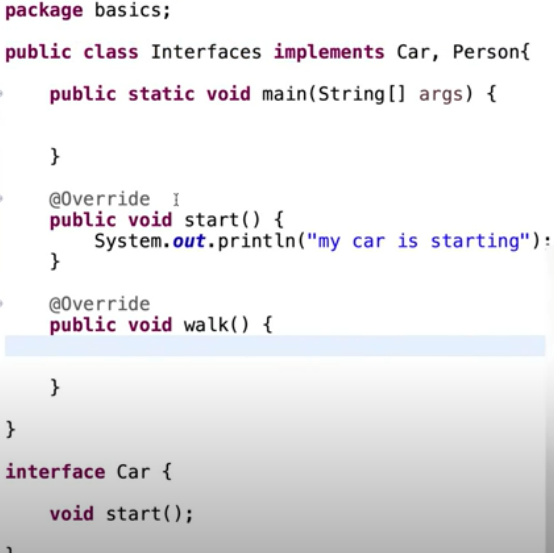
**Abstraction in Java –**

****

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated. this signifies that this class is just an abstract concept and cannot be really implemented. Similarly when we write abstract behind a method, this means we will define this method in the subclasses by overriding it.

But still we can define the method here.

**Interface –**

****

We define a class as interface when we put the ‘interface’ keyword behind it.

All the method in an interface will always be public and abstract. Here cannot define a method at all.

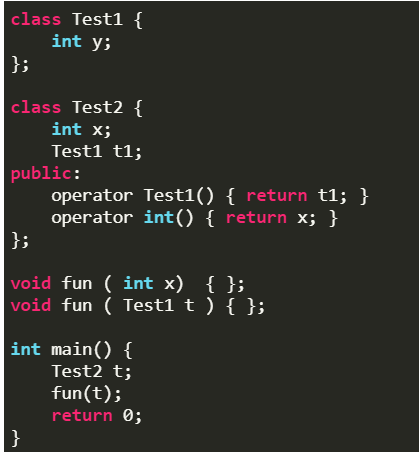
**To inherit from a class we use** – extend

**To inherit from an interface we use** – implements

There is no multiple inheritance in java, but we can use interface just like implemented in the above image.

**Conversion Operator – (doubt)**

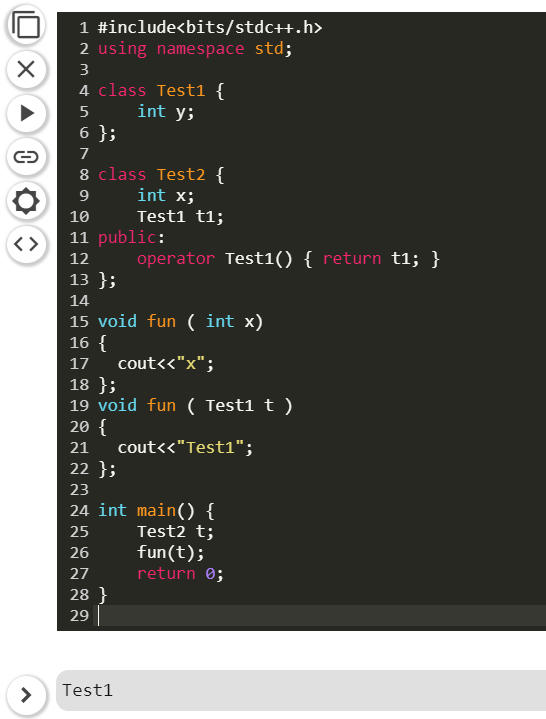
Sometimes it is required to convert one concrete type to another concrete type or primitive type implicitly. To make this conversion we can use conversion operator. This is created like operator overloading function in class.

****

Test2 objects can automatically be converted to both int and Test1 data types.

Therefore, the function call fun(t) is ambiguous as there are two functions void fun(int ) and void fun(Test1 ), compiler has no way to decide which function to call.

**This is correct –**

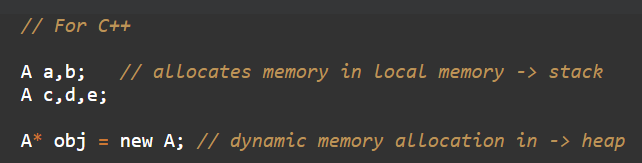
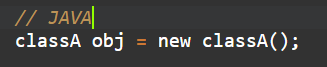
****

**Some of the major Object-Oriented Programming languages include -** Java, C++, Javascript, Python, PHP

**Definition -**

Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.  
  
**Class -** It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.  
  
**Object -** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

**Object declaration –**

**Advantages -**   
OOPs is very helpful in solving very complex level of problems.  
Highly complex programs can be created, handled, and maintained easily using object-oriented programming.  
  
**4 pillars of OOP –**

* Encapsulation
* Abstraction
* Inheritance
* Polymorphism

**Encapsulation** - is defined as wrapping up of data under a single unit.  
Eg - In a company, there are different sections like the accounts, finance, sales section. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales. In thiscase, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section.  
  
Encapsulation also leads to 'data abstraction' and 'hiding'.   
Eg - data of sales section is hidden from people of financial section  
  
**Abstraction** - displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.  
  
Eg - Just like a driver doesn't know how brakes are working but still applies brake.  
In c++, just by writing math.h header file, we can use pow function without knowing how it works.

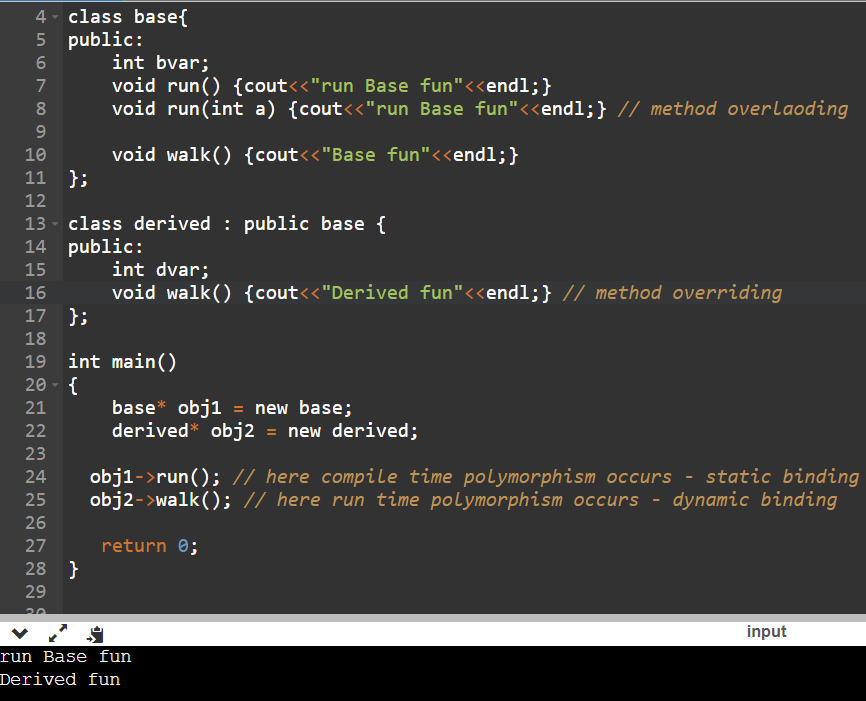
**Polymorphism** - means having many forms.  
Just like a man can be a father, brother, son at the same time.  
  
**It is of 2 types -**   
**1. Compile Time polymorphism 2. Run-time polymorphism**

* Function Overloading
* Operator Overloading

**Static binding** **-**   
It is the binding which can be resolved at compile time by compiler that is , here compiler knows for which particular method call which method is to be executed.  
It is also known as **early binding**, because this binding takes place before the program is executed.  
Eg - Method Overloading, etc.

**Dynamic Binding -**   
Compiler is not able to resolve the binding at compile time, so it is known as **late binding**.  
Here binding takes place at the time of program execution. Here compiler doesn’t know if it will encounter virtual function or not. **Eg -** Method Overriding

**Combined Eg -**



**Explanation –**

Here, during **method overloading**, the compiler knows which method is going to called.

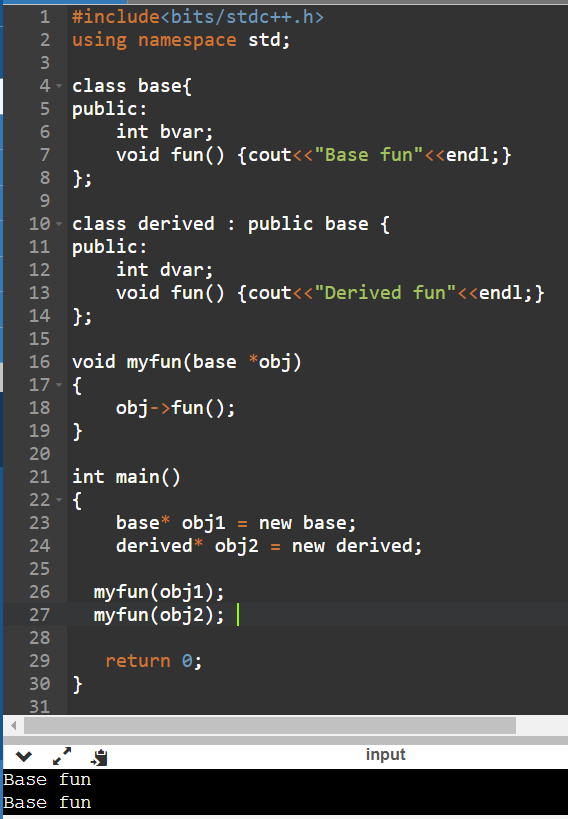
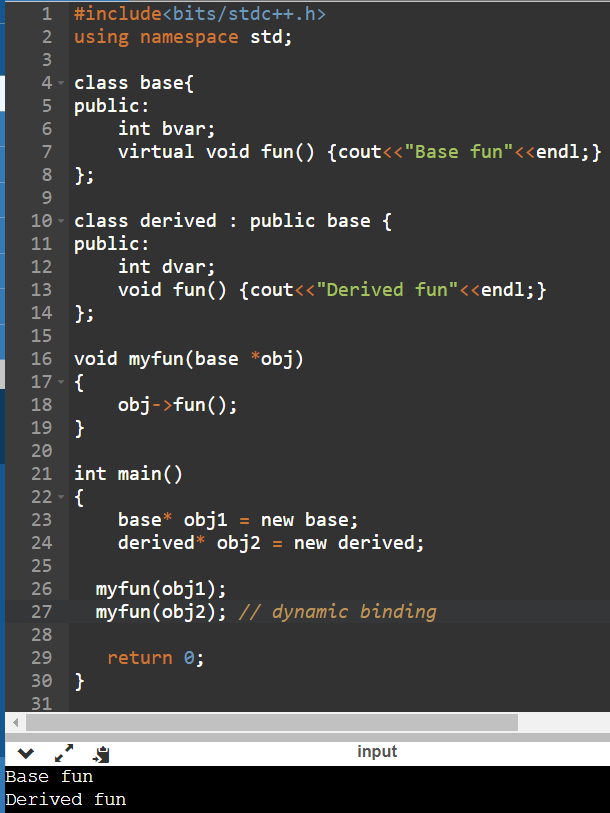
In, **method overriding**, the compiler doesn’t know which ‘walk‘ function is going to be called.

**Virtual function -**   
Refer - <https://www.youtube.com/watch?v=m9p_shyDhY0&ab_channel=CppNuts>

refer - <https://www.youtube.com/watch?v=JU8DbwBvOWE&ab_channel=SimpleSnippets>  
  
It is a function which is declared within base class and is re-defined(overridden) by derived class.   
It ensures that the correct function is called for an object, regardless of the type of pointer used for function call.

When we call a **overridden function** using the object of derived class, then if that function is defined as virtual in the base class, then the derived class’s function will execute. Here dynamic binding is occuring.

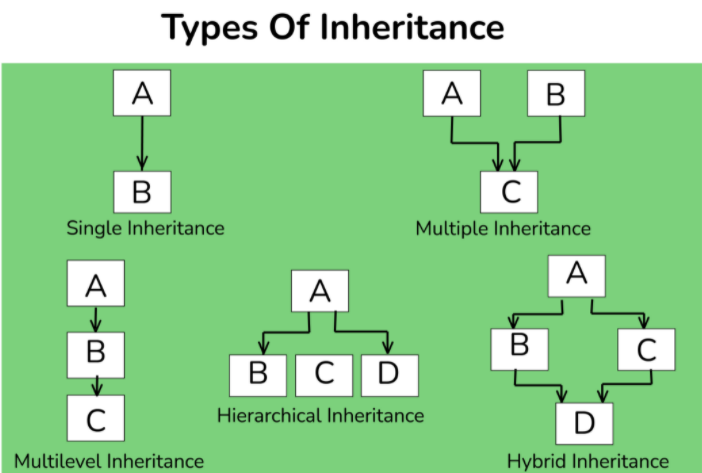
**Static binding - Dynamic binding -**

**** 

**Pure Virtual Function -**   
A pure virtual function in c++ is a virtual function for which we dont have implementation, we only declare it. It is declared by assigning 0 in declaration.  
Eg -   
We have a class shapes and it has a method to find area, but since every shape has a different formula to find area, so we make it pure virtual function so that we can declare in child clases.  
  
**Abstract Class -**   
It is a class that has atleast one pure virtual function. We cannot create an object of an abstract class.

**Inheritance – (parent = Base = Superclass | Child = sub = derived class)**

**5 types –** Single , Multiple, Multi-level, Hierarchical, Hybrid

****

“inheritance” means “receiving some quality or behavior from a parent to an offspring.  
The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.  
  
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.

**Limitations –**

* Inheritance needs more time to process, as it needs to navigate through multiple classes for its implementation.
* The base class and the child class, are very tightly coupled together. So if one needs to make some changes, they might need to do nested changes in both classes.

**Constructor –**

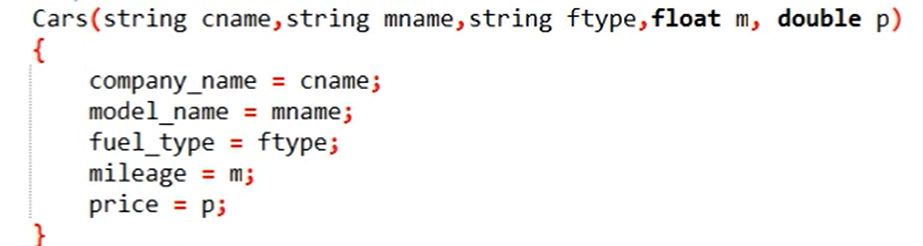
[**https://www.youtube.com/watch?v=hAA8FBq2bA4&ab\_channel=SimpleSnippets**](https://www.youtube.com/watch?v=hAA8FBq2bA4&ab_channel=SimpleSnippets)

Default, Parameterised, Copy constructor

* **Default Constructor -**

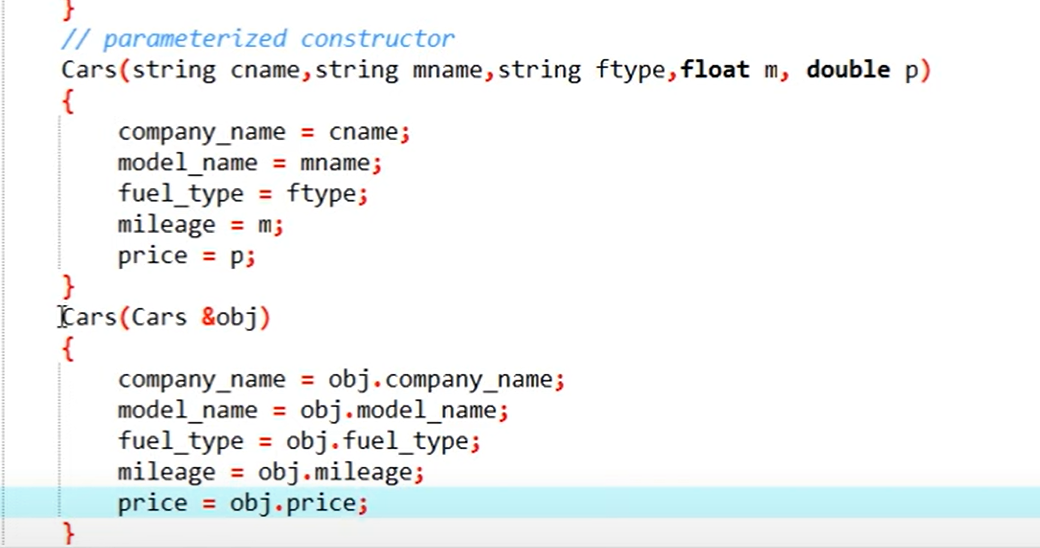
Default constructor is always called even when we don’t write it ourselves, but some random values will be assigned to the member variables. So, to assign the initial values we declare it on our own.

* **Parameterized constructor –**

****

Since we explicity define intial values in a default constructor, the values are going to be same for each object. So to have different initial values for each object, we can use parameterised constructors.

* **Copy constructor –**

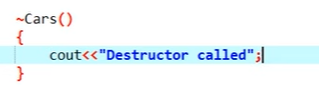
****

**Ways to define a copy constructor -**

Point p2 = p1; **or**  Point p2(p1);

A copy constructor is a member function which initializes an object using another object of the same class.If we haven’t declared a copy constructor in our class, then a default copy constructor is called when initializing an object using another object, but when we have dynamically allocated memory or pointers in our class, then we have to explicitly write it.

**Destructor –**

****

It is a member function which destructs or deletes an object. It is called when the function or the program ends. Destructor is always called without even us defining it, but when we have dynamically allocated memory or pointers in our class, then we have to explicitly write it. This must be done to avoid memory leak.

**Structure vs Class –**

Though they appear to be similar, they have differences that make them apart. For example, the structure is saved in the stack memory, whereas the class is saved in the heap memory. Also, Data Abstraction cannot be achieved with the help of structure, but with class, Abstraction is majorly used.

**Final in C++ vs Java -**

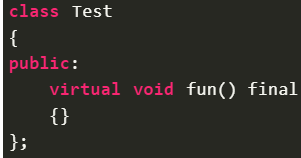
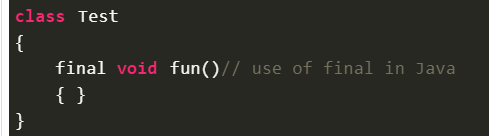
**Common use -**

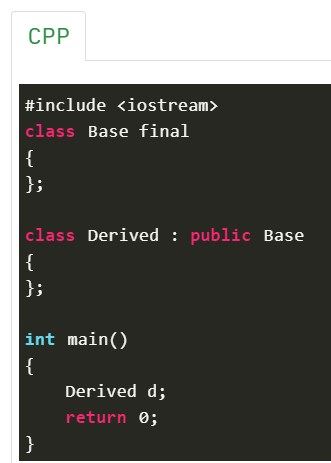
We can use final for a function to make sure that it cannot be overridden.

We can also use final to make sure that a class/struct cannot be inherited.

**In Java** - final can also be used with variables to make sure that a value can only be assigned once. We cannot change it’s value after assigning once. In java it is a keyword.

**In C++ - In Java -**



Concept of padding –

Refer - <https://www.youtube.com/watch?v=aROgtACPjjg&ab_channel=NesoAcademy>

diamond problem

29 ques  
<https://www.interviewbit.com/oops-interview-questions/#object-oriented-programming>

MCQ –

<https://www.sanfoundry.com/object-oriented-programming-questions-answers-entrance-exams/>

<https://www.geeksforgeeks.org/c-plus-plus-gq/virtual-functions-gq/>

operator overloading  
**private, protected, public**