An import statement tells the compiler the path of a class or the entire package. It is unlike “#include” in C++, which includes the entire code in the program. Import statement tells the compiler that we want to use a class (or classes) that is defined under a package.

Type of Inheritance

When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance. The type of inheritance is specified by the access-specifier as explained above.

We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used. While using different type of inheritance, following rules are applied −

* **Public Inheritance** − When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class. A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.
* **Protected Inheritance** − When deriving from a **protected** base class, **public** and **protected** members of the base class become **protected** members of the derived class.
* **Private Inheritance** − When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class.

Unlike Java, **instances of classes ("objects") can be allocated on the stack** in C++.

Ambiguity with multiple inheritance

When using multiple inheritance, if a class C inherits from classes B and A, and both B and A, implements a method having the same name, if we try to call the method for an object from the class C, an error occurs saying that the method is ambiguous. The solution is to use the scope operator “::” to call the method for example:

C c=C();

//c.ambiguousMethod() 🡪gives an error

c.A::ambiguouisMethod 🡪 works fine

c.B::ambiguouisMethod 🡪 works fine

inheritance and constructors

when constructing an object from the child class, if we do not call a base class constructor explicitly in the initializer list, the default constructor (without arguments) gets called.

Unlike Java, **C++ arrays can be allocated on the stack**. Java arrays are a special type of object, hence they can only be dynamically allocated via "new" and therefore allocated on the heap