Generics

Generics are used to allow generic data type to be a parameter to methods and classes. This can be implemented in C++ using Templates.

For example:

```
template <typename myType>
void myfunc(myType x){
        cout<<"you have passed"<<x;
}
myfunc(10);
myfunc(10.5);
myfunc("hello");

=================================================================

template <typename T>
void myfunc(T x, T y){          //x and y must be of same type
        cout<<"you have passed"<<x <<"and" <<y;
}

=================================================================

template <class T>
void myfunc(T x, T y){                  //x and y must be of same type
        cout<<"you have passed"<<x <<"and" <<y;
}

=================================================================

template <typename T1, typename T2>
void myfunc(T1 x, T2 y){        //x and y can be diff types
        cout<<"you have passed"<<x <<"and" <<y;
}

=================================================================
```

Inheritance in C++:

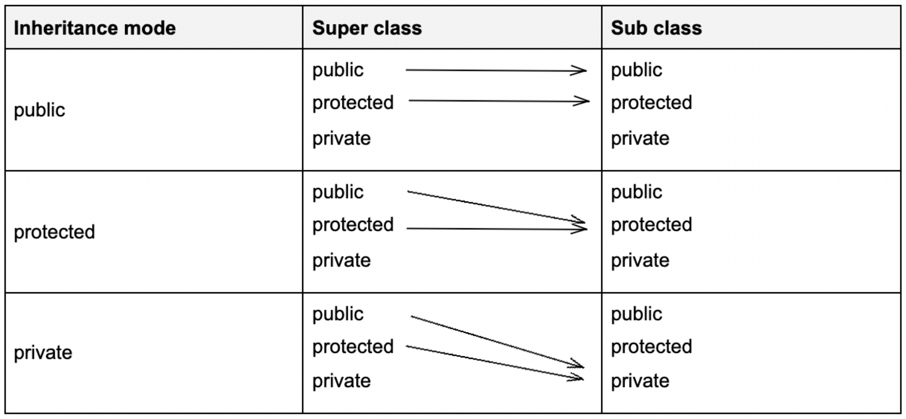
- Subclasses inherit all Non-private methods except:
    - Constructor

- Destructor
- Overloaded operators
- Friend functions of the base class.

Example:

```
class Person{
    private:
        int id;
    public:
        string name;
};
class Student : public Person{
    public:
        double gpa;
};
int main(){
    Student s1;
    cout<<s1.name<<endl;
    cout<<s1.gpa<<endl;
}
```

- It is important to pay attention to the order of execution (calls of constructors and destructors) when we have inheritance. This will be as follows:
    - 1.superClassConstructor → 2.subClassConstructor
    - 1.subClassDestructor → 2.superClassDestructor
- There are different modes of inheritance in C++: (public, protected, and private).
- The default mode is private inheritance.



Inheritance mode	Super class	Sub class
public	public protected private	public protected private
protected	public protected private	public protected private
private	public protected private	public protected private