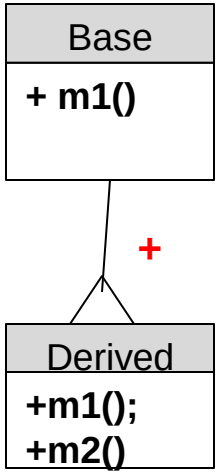
Java™ Education & Technology Services

Object Oriented programming Using C++

Clear points

Main

```
Derived obj;
Derived *ptr;
Obj.m1() ;          Derived
obj.Base:: m1();    Base
obj.m2();           Derived
ptr = & obj ;
ptr -> m1 ();                  Derived
ptr->Base:: m1();                    Base
ptr-> m2();                    Derived
```



- Object From the Derived works as pointer to Derived One;

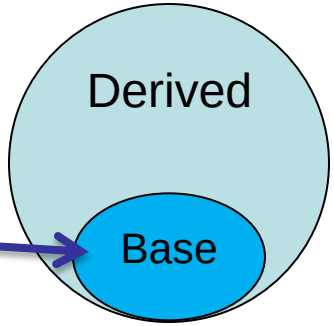
Clear points

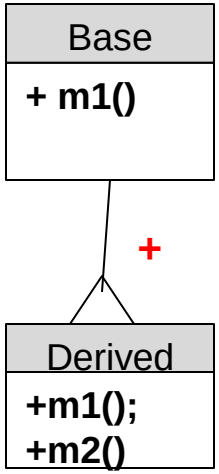
Main

```
Derived obj;
Base *ptr;
ptr = & obj ;
ptr -> m1 ();
ptr-> m2();
```

check Base at compile time and run Base version

compiler error





- Pointer From the Base to object of Derived will only access the Base part in the derived

Dynamic Binding

Dynamic Binding

- Late Binding.
- The most clear implementation of polymorphism.

- **Virtual Function:**
  - Define in the Base class for Derived class.
  - Only in the public inheritance type.
  - Virtual function in Base to run Derived on in the runtime.
  - Not needed to define the function as virtual in the last class in the tree.
  - Virtual effects only in the child.
  - Need to define as virtual at the beginning of the tree

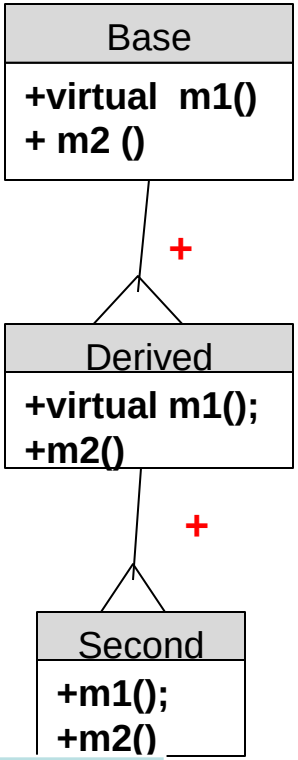
Dynamic Binding

- **Example:**

```
Base *ptr;

Base o1;

Derived o2;

Second o3;
```

- **Case 1:**

```
ptr = &o1;

ptr - > m1();

ptr - > m2();
```



Compile	Runtime
Base	Static Binding know from Compile time
Base	

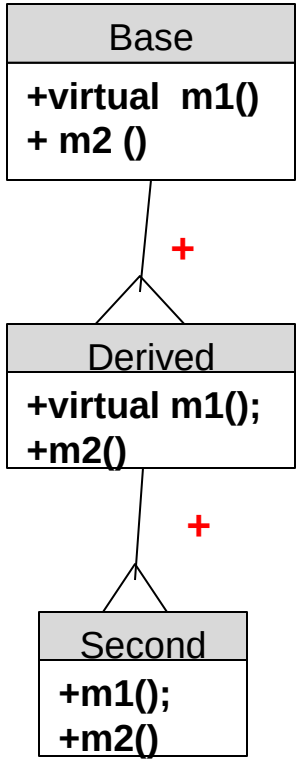
Dynamic Binding

- **Example:**

```
Base *ptr;

Base o1;

Derived o2;

Second o3;
```

- **Case 2:**

```
ptr = &o2;

ptr - > m1();

ptr - > m2();
```



Compile	Runtime
Check at Base	Derived --- Late Binding
Check at Base	Base--- Static Binding

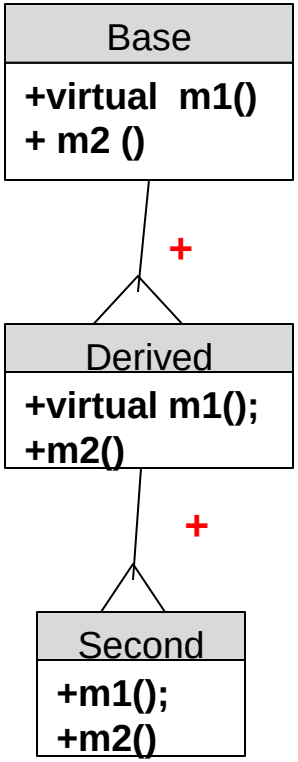
Dynamic Binding

- **Example:**

```
Base *ptr;

Base o1;

Derived o2;

Second o3;
```

- **Case 3:**

```
ptr = &o3;

ptr - > m1();

ptr - > m2();
```



Compile	Runtime
Check at Base	Second--- Late Binding
Check at Base	Base--- Static Binding

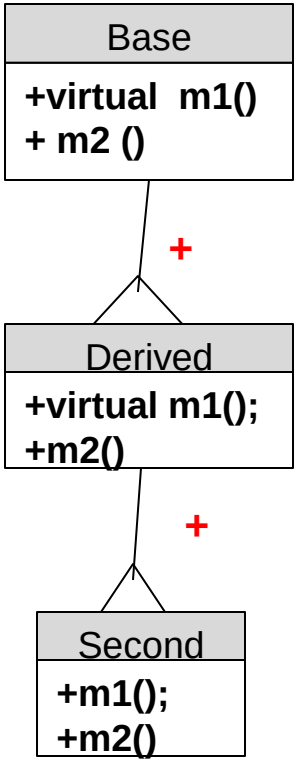
Dynamic Binding

- **Example:**

```
Derived *ptr;

Base o1;

Derived o2;

Second o3;
```

- **Case 4:**

```
ptr = &o3;

ptr - > m1();

ptr - > m2();
```



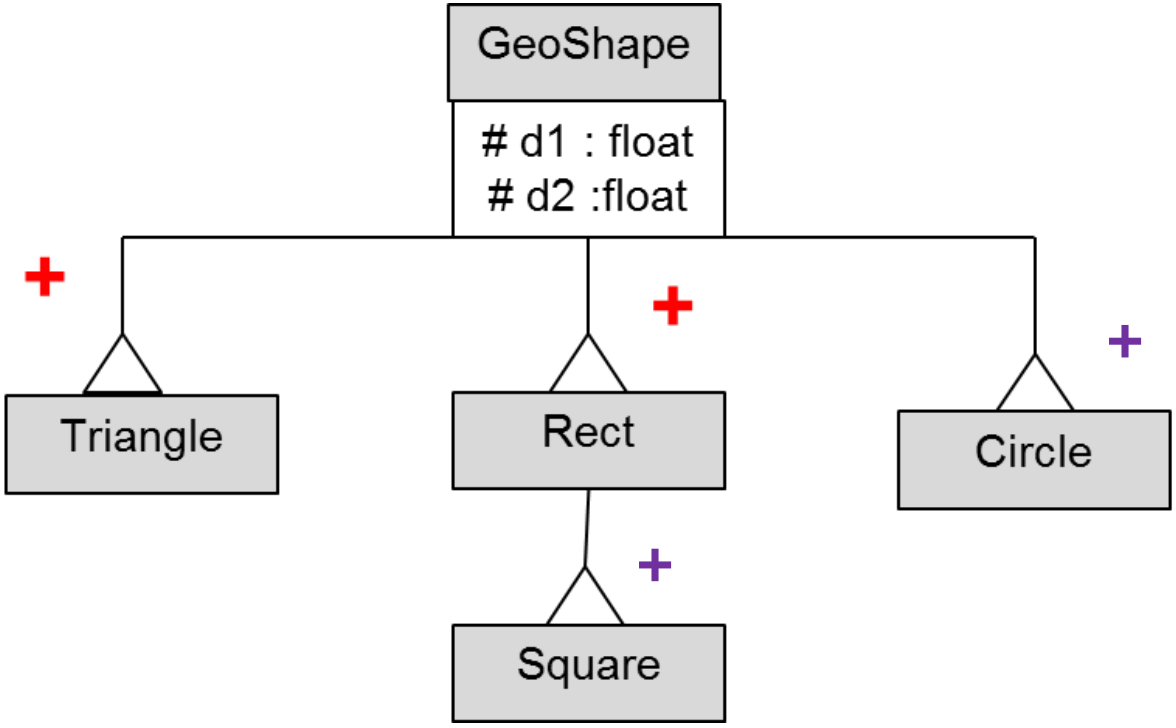
Compile	Runtime
Check at Derived	Second--- Late Binding
Check at Derived	Derived-- Static Binding

Dynamic Binding

- **V-Table:**
  - Build at the compile tine for overriding functions in it
  - Put static Binding in it at the compile time.
  - Put dynamic Binding at the run time.
  - Load the table at the run time.
  - the pointer always refers to the last implemented function for running object in the table
  - V-Table for m1() :
    - » Check the pointer class
    - » Check there if the function is defined as virtual.
    - » Go to object class type.
    - » Run its object implementation.

Dynamic Binding

- **Example:**
  - Change all inheritance type to public
  - Add virtual function to GeoShape class




```
class GeoShape
{
    :
    virtual float calculateArea()
    {
        return 0;
    }

};
```

Dynamic Binding

- **Example:**
  - In Main Method:

GeoShape *p;

Circle c(10);

Rect r(30,40);

Triangle t(100,150);

Square s(60);

```
class GeoShape
{
    :
    virtual float calculateArea()
    {
        return 0;
    }

};
```

p = & c; p->calculateArea();	p = & r; p>calculateArea();	p = &t; p->calculateArea();	p = & s; p->calculateArea();

Dynamic Binding

- **Example:**
  - Add standalone function to sum areas of any three Geoshapes

```
float sumAreas(GeoShape *p1, GeoShape *p2, GeoShape *p3)
{
    return p1->calculateArea() + p2->calculateArea() + p3->calculateArea() ;
}
```

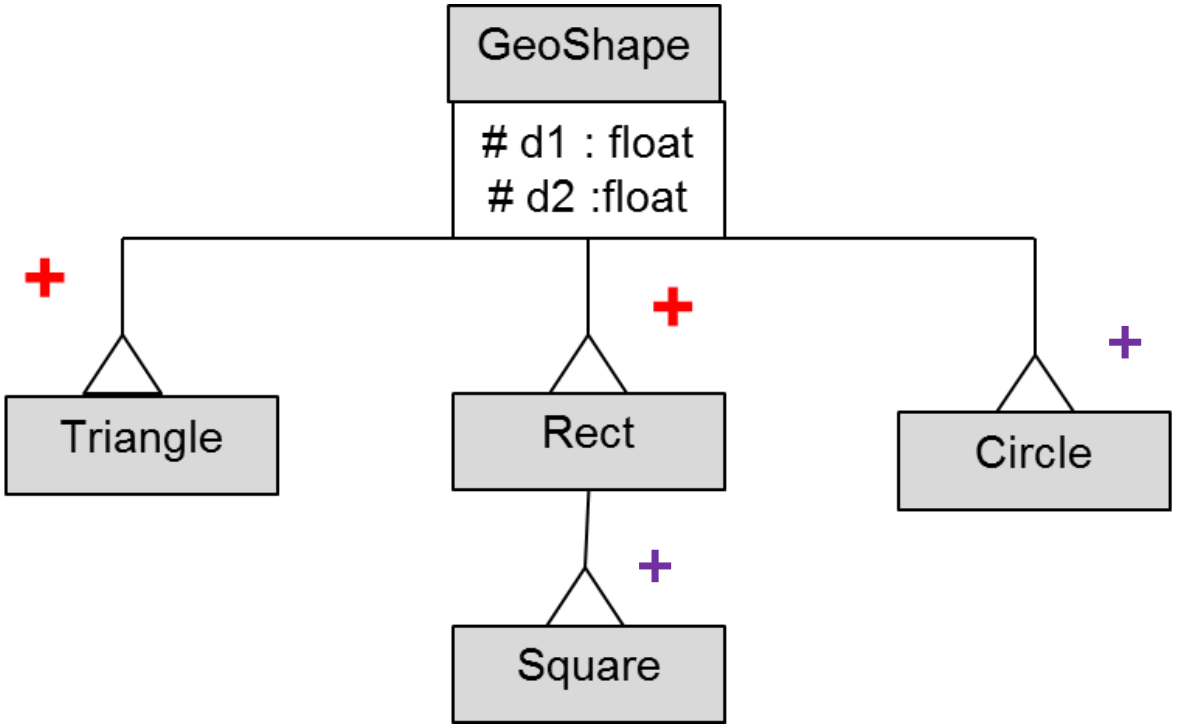
```
int main()
{
     Triangle myT(20, 10);
     Rect myR(2, 5);
     Circle myc(5);
     cout<<sumAreas(&myT, &myR, &myS)<<endl ;
}
```

  - It is applicable for any Geoshape as calculateArea () in Geoshape is virtual for Late Binding.

Dynamic Binding

- **Pure virtual Function:**

```
class GeoShape
{
    :
    virtual float calculateArea() = 0;

};
```



Dynamic Binding

- **Pure virtual Function:**

```
class GeoShape
{
    :
    virtual float calculateArea() = 0;

};
```

  - Now Geoshape is abstract class as it contain at least one pure virtual function.
  - can not make object from Geoshape.
  - can make a pointer from Geoshape to one of its Childs objects
  - any class can inherit abstract class.
  - when inheritance from abstract the child must implement all the pure virtual functions of his base or it will be converted from concrete class to abstract one.
  - pure virtual function should be implemented in the leaf of the tree at least one.

Template Classes

Template Classes

- I you need to create objects of int_stack and double_stack

```
// Stack carries int element
class StackI
{
  private:
    int top ;
    int size;
    int *st;
    static int counter ;
  public:
    StackI();
    StackI(int n);
    ~StackI();
    static int getCounter();
    StackI(StackI &) ;
    void push(int);
    int pop();
    StackI& operator= (StackI&);
    friend void viewContent(StackI) ;
};
int StackI::counter = 0 ;
```

```
// Stack carries double element
class StackD
{
  private:
    int top ;
    int size;
    double *st;
    static int counter ;
  public:
    StackD();
    StackD(int n);
    ~StackD();
    static int getCounter();
    StackD(StackD &) ;
    void push(double);
    double pop();
    StackD& operator= (StackD&);
    friend void viewContent(StackD) ;
};
int StackD::counter = 0 ;
```

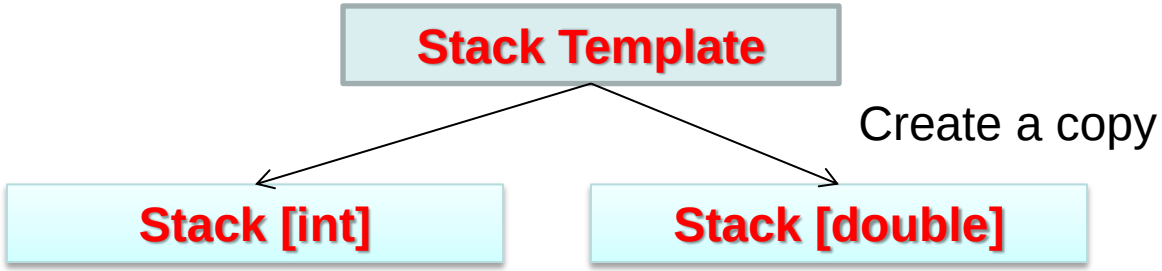
- There are two classes which are different only in some variable data type

Template Classes

```
int main()
{
    clrscr();
    StackI s1(5);
    cout << "\nNumber of Integer Stacks is: " << StackI::getCounter();
    s1.push(10);
    s1.push(3);
    s1.push(2);

    cout << "\n1st integer: " << s1.pop();
    cout << "\n2nd integer: " << s1.pop();

    StackD s2;
    cout << "\nNumber of Character Stacks is: " << StackD::getCounter();


    getch() ;
return 0;
}
```

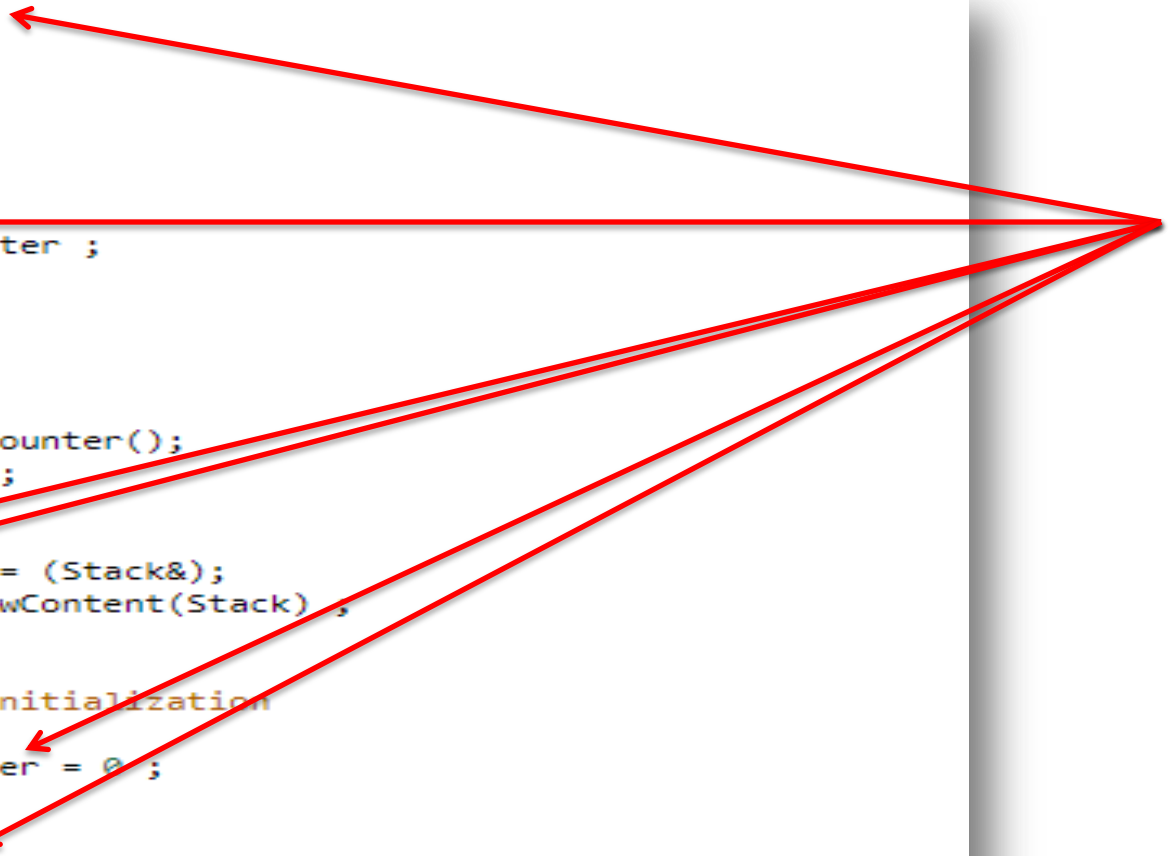
Template Classes

- **Template Classes:**
  - » Allows one to implement a generic template that has a type parameter T.
  - » T can be replaced with actual types at compiler time
  - » One place for changing the implementation.
  - » No meaning for the template until it used.
  - » The number of output classes is the number of template used.



Template Classes

```
template <class T>
class Stack
{
  private:
    int top ;
    int size;
    T *ptr;
    static int counter ;

  public:
    Stack();
    Stack(int n);
    ~Stack();
    static int getCounter();
    Stack(Stack &) ;
    void push(T);
    T pop();
    Stack& operator= (Stack&);
    friend void viewContent(Stack) ;
};

//static variable initialization
template <class T>
int Stack<T>::counter = 0 ;

template <class T>
Stack<T>::Stack()
{
        top = 0 ;
        size = 10;
        ptr = new T[size];
        counter++ ;
}
```



Template Classes

```
int main()
{
    Stack<int> s1(5);
```

Compiler creates a new stack class with int data type as T and then create object from it

```
    cout << "\nNumber of Integer Stacks is: " << Stack<int>::getCounter();
    s1.push(10);
    s1.push(3);
    s1.push(2);

    cout << "\n1st integer: " << s1.pop();
    cout << "\n2nd integer: " << s1.pop();

    Stack<char> s2;
```

Compiler creates a new class with char data type as T and then create object from it

```
    cout << "\nNumber of Character Stacks is: " << Stack<char>::getCounter();
    s2.push('q');
    s2.push('r');
    s2.push('s');
    viewContent(s2) ;
    cout << "\n1st character: " << s2.pop();
    cout << "\n2nd character: " << s2.pop();


return 0;
}
```

Lab Exercise

Lab Exercise

- **1st Assignment :**
  - Continue Geoshape Example
    - » make calculateArea in Geoshape as a pure virtual function.
    - » make a standalone function of

      sumOfAreas(int num_of_shapes, Geoshape * arr);
  - Try Template class
    - » Can make a template<T,Z>