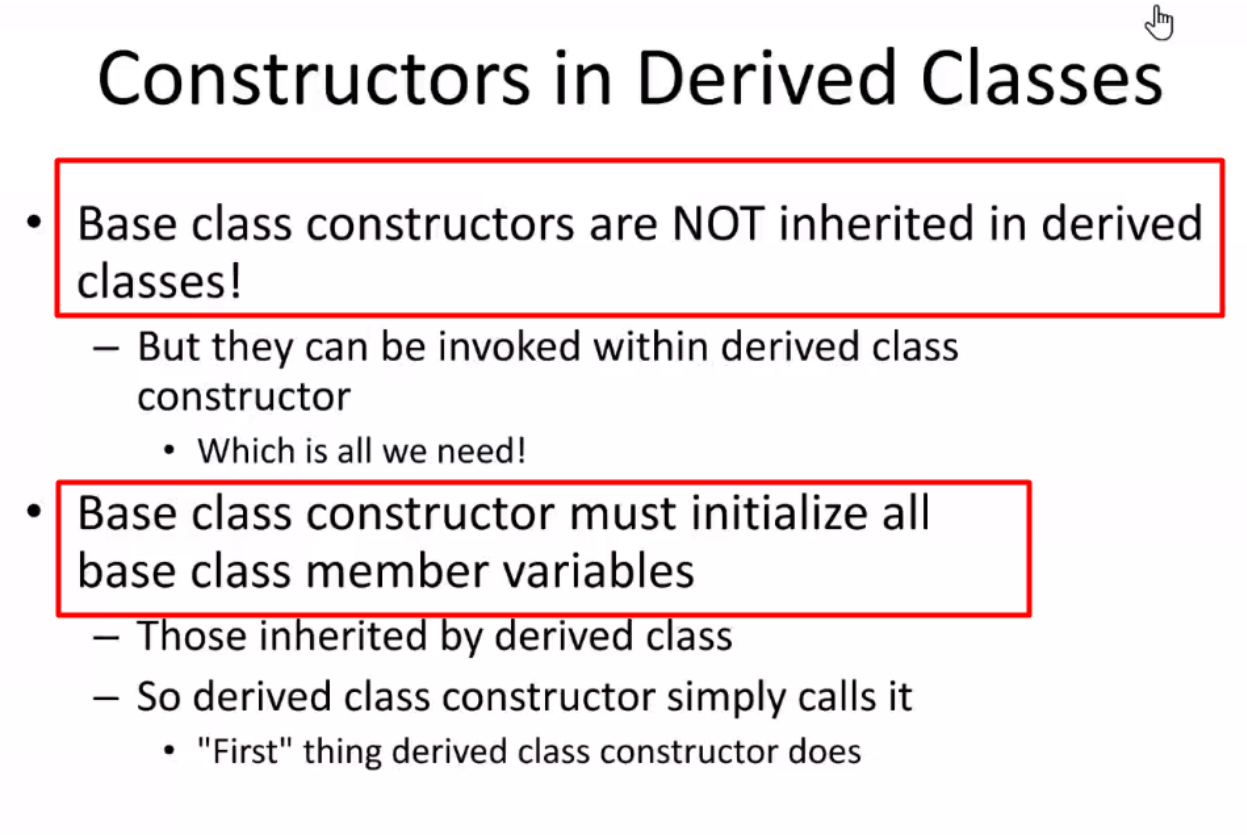
CSE241 - OOP

A virtual function is a member function that is declared in a base class and that is redefined by derived class. Virtual function are hierarchical in order of inheritance. When a derived class does not override a virtual function, the function defined within its base class is used.

A pure virtual function is one that contains no definition relative to the base class. It has no implementation in the base class. Any derived class must override this function.



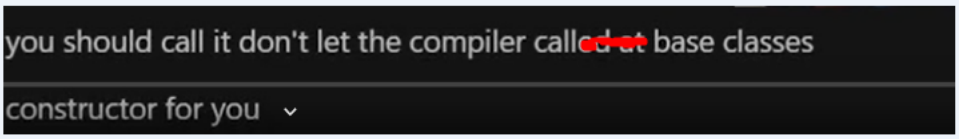
💡 Should always invoke base classs constructors

Accessing Redefined Base Function

- When redefined in derived class, base class's definition not "lost"
- Can specify it's use:

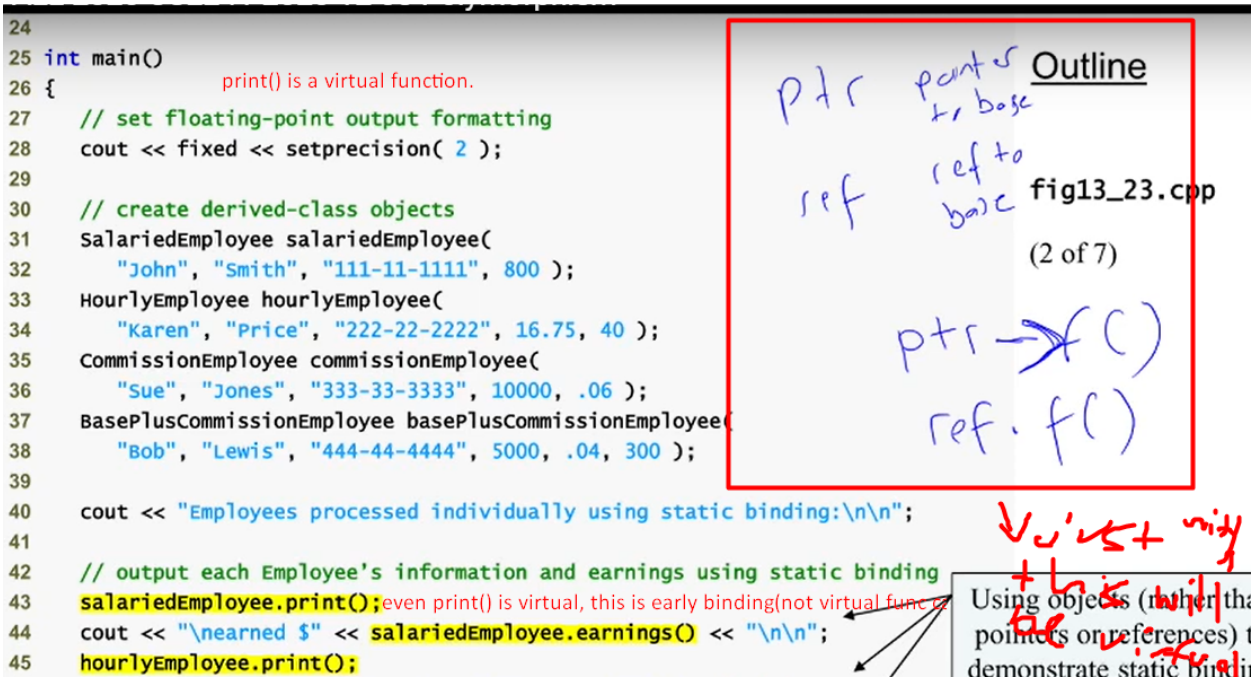
```
Employee        JaneE;
HourlyEmployee SallyH;
JaneE.printCheck();  → calls Employee's
                                printCheck function
SallyH.printCheck();  → calls HourlyEmployee
                                printCheck function
SallyH.Employee::printCheck();  → Calls Employee's
                                printCheck function!
```

- Not typical here, but useful sometimes

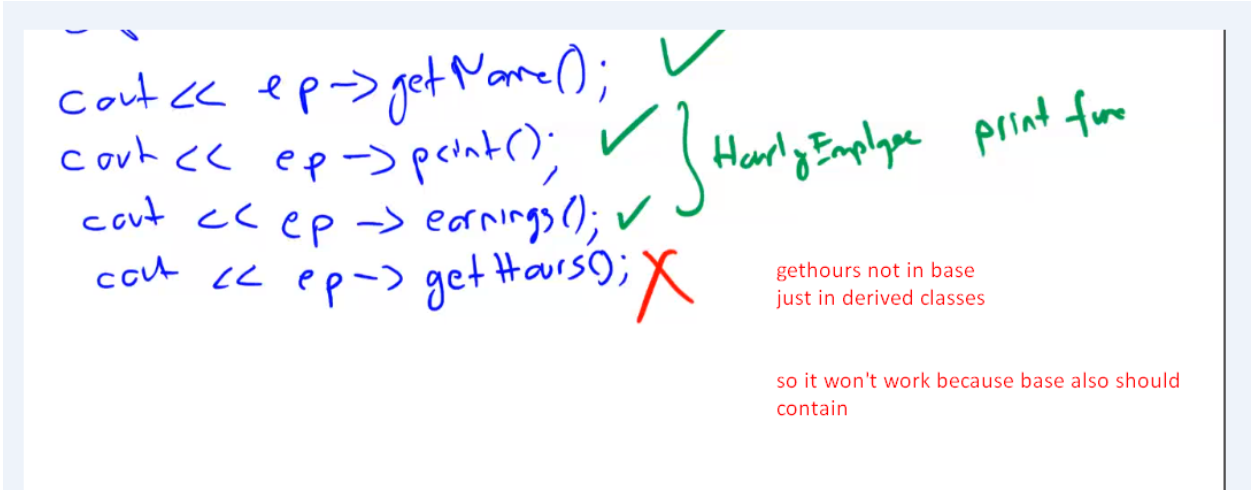


move semantics

basein assignment operatorı varsa derivedda da kesin yaz. cmpiler otomatik cagırmasın



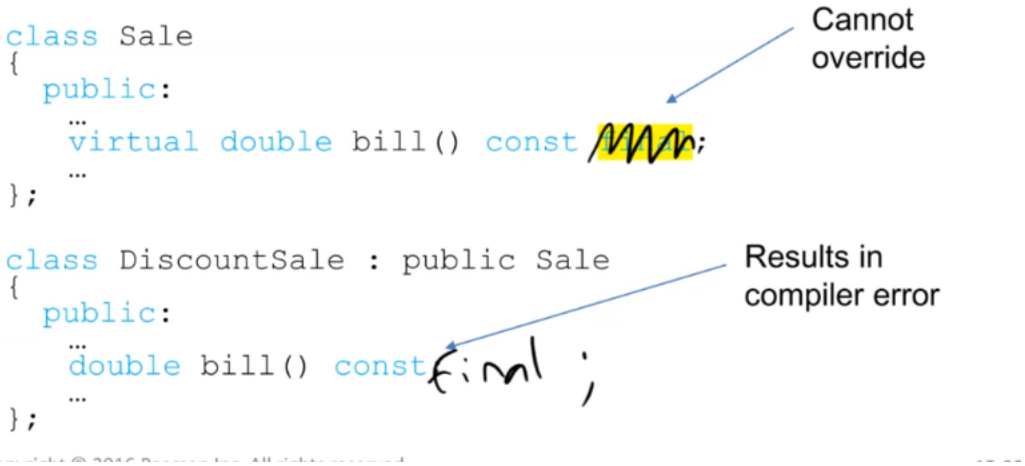
early binding virtual değil. Object type isn't revealed at the time of instantiation.



- So:
  - Virtual functions changed: ***overridden***
  - Non-virtual functions changed: ***redefined***

C++11 **final** keyword

C++11 includes the **final** keyword to prevent a function from being overridden. Useful if a function is overridden but don't want a derived classes to override it again.




```
class Sale
{
  public:
    …
    virtual double bill() const final;
    …
};

class DiscountSale : public Sale
{
  public:
    …
    double bill() const final;
    …
};
```

Virtual Destructors

- Recall: destructors needed to de-allocate dynamically allocated data
- Consider:
  Base *pBase = new Derived;
  …
  delete pBase;
  - Would call base class destructor even though pointing to Derived class object!
  - Making destructor ***virtual*** fixes this!
- Good policy for all destructors to be virtual

template prefix → template<class T>

Multiple Type Parameters

- Can have:
  template<class T1, class T2>
- Not typical
  - Usually only need one "replaceable" type
  - Cannot have "unused"
    template parameters
    - Each must be "used" in definition
    - Error otherwise!

defined

- Example: an array:
  int a[10], b[10];
  swapValues(a, b);
  - Arrays cannot be "assigned"!

catch içinde cout değil cerr kullan. ———- Exception& referans kullann.

```
catch(const Errors& e){ e.what() }
```

try'ın içindeki error throw ile eşleşirse catchin içi gerçekleşir.

Errors.h

```
throw( Errors("There is no last move. Thrown an exception.\n") );
```

catch(...) kullanırsan en altta olsun.

iterator will work in STL.

```
//Program to demonstrate STL iterators.
#include <iostream>
#include <vector>
using namespace std;
int main( )
{
    vector<int> container;

    for (int i = 1; i <= 4; i++)
        container.push_back(i);

    cout << "Here is what is in the container:\n";
    vector<int>::iterator p;
    for (p = container.begin( ); p != container.end( ); p++)
        cout << *p << " ";
    cout << endl;

    cout << "Setting entries to 0:\n";
    for (p = container.begin( ); p != container.end( ); p++)
         *p = 0;
    cout << "Container now contains:\n";

    for (p = container.begin( ); p != container.end( ); p++)
        cout << *p << " ";
    cout << endl;

    return 0;
}
/*
Here is what is in the container:
1 2 3 4
Setting entries to 0:
Container now contains:
0 0 0 0
*/
```

List = no random access / no index operator

begin, end, rbegin, rend →>>>>>>> returns operator

STACK

```
//Program to demonstrate use of the stack template class from the STL.
#include <iostream>
#include <stack>
using std::cin;
using std::cout;
using std::endl;
using std::stack;

int main( )
{
    stack<char> s;

    cout << "Enter a line of text:\n";
    char next;
    cin.get(next);
    while (next != '\n')
    {
        s.push(next);
        cin.get(next);
    }

    cout << "Written backward that is:\n";
    while ( ! s.empty( ) )
    {
        cout << s.top( );
        s.pop( );
    }
    cout << endl;

    return 0;
}
//SONUC: LOBLOG NAMYELUS
```

```
template <class T, class R = vector<R>>
class stack{
    private:
        R data_;
    public:
        stack(){};
        void push(const T& e) { data_.push_back(e);}
        T pop() {return data_.pop_back();}
        bool empty() const {return data_.size()==0;}
};

stack<int> s1;
stack<int, list<int>> s2;
```

More set Template Class

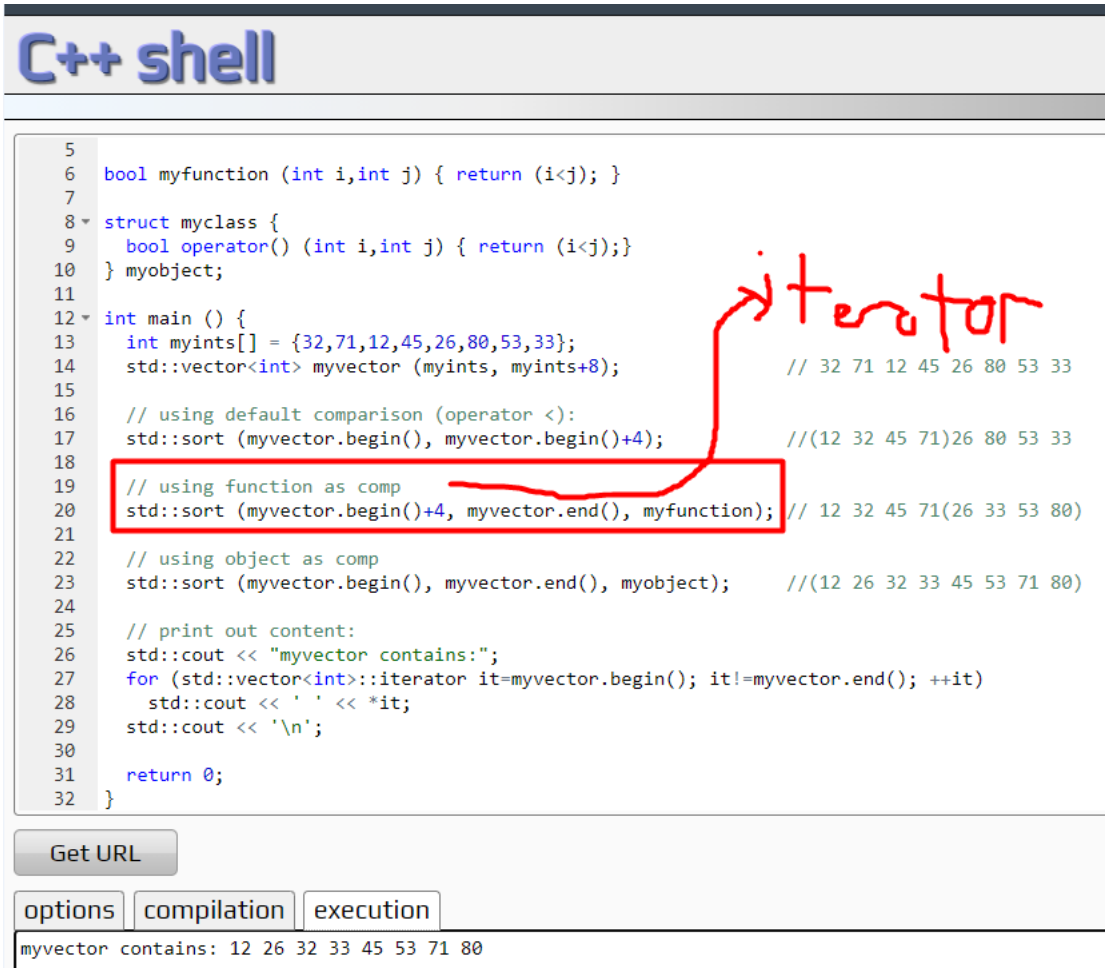
- Designed to be efficient
  - Stores values in sorted order
  - Can specify order:
    set<T, *Ordering*> s;
    - Ordering is well-behaved ordering relation that returns bool
    - None specified: use < relational operator

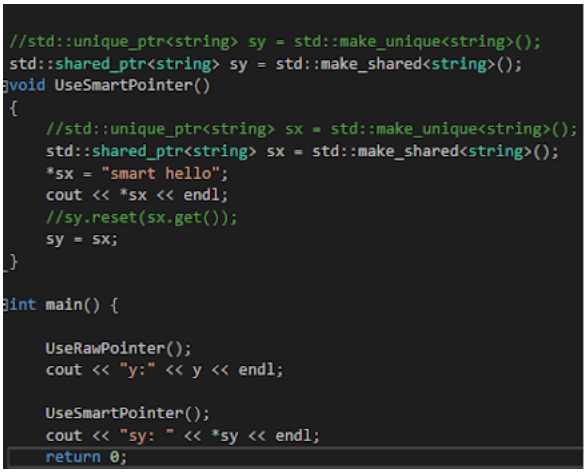
Map

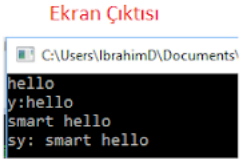
```
23
24      cout << "Entry for Mercury - " << planets["Mercury"]
25          << endl << endl;
26
27      if (planets.find("Mercury") != planets.end())
28          cout << "Mercury is in the map." << endl;
29      if (planets.find("Ceres") == planets.end())
30          cout << "Ceres is not in the map." << endl << endl;
31
32      // Iterator outputs planets in order sorted by key
33      cout << "Iterating through all planets: " << endl;
34      map<string, string>::const_iterator iter;
35      for (iter = planets.begin(); iter != planets.end(); iter++)
36      {
37         cout << iter->first << " - " << iter->second << endl;
38      }
39
40      return 0;
41  }
42
```

```
Entry for Mercury - Hot planet

Mercury is in the map.
Ceres is not in the map.

Iterating through all planets:
Earth - Home
Jupiter - Largest planet in our solar system
Mars - The Red Planet
Mercury - Hot planet
Neptune - 1500 mile per hour winds
Pluto - Dwarf planet
Saturn - Has rings
Uranus - Tilts on its side
Venus - Atmosphere of sulfuric acid
```

first ve second keylere denk geliyor.

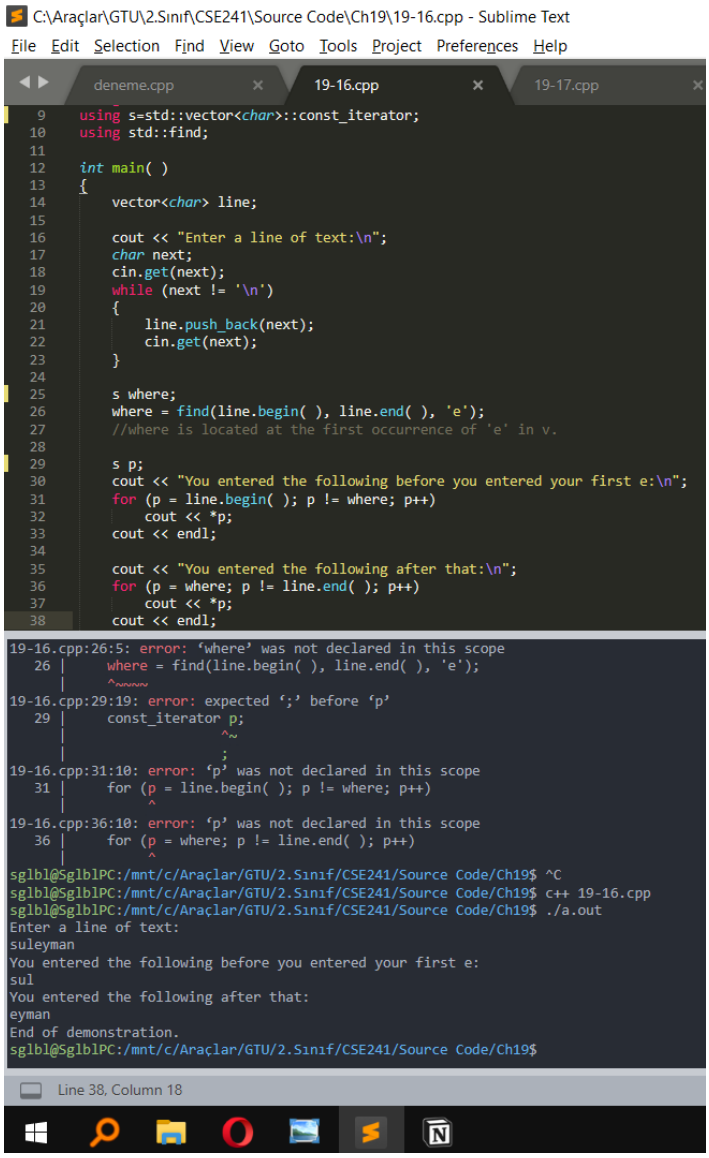






Ekran çıktısına baktığımızda raw pointer ile aynı davrandı. **sx** pointer ı kapsama alanı dışında kaldığı için otomatik olarak silindi, Fakat **sy** hala verinin saklandığı yeri gösteriyor. Buradaki avantaj akıllı pointer **sx** i silmek ile uğraşmadık, boş yere de yer kaplamamış oldu.

shared_ptr kullanarak ilgili kaynağı kaç farklı pointer ın işaret ettiğini de öğrenebiliriz (reference counting). Örneğin aşağıdaki örnekte 2 pointer aynı değeri işaret ediyor, ve bu pointer lardan birisini silince **use_count** değeri bir azalıyor.


C:\Araçlar\GTU\2.Sınıf\CSE241\Source Code\Ch19\19-16.cpp - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
deneme.cpp
19-16.cpp
19-17.cpp
using s=std::vector<char>::const_iterator;
using std::find;
int main( )
{
vector<char> line;
cout << "Enter a line of text:\n";
char next;
cin.get(next);
while (next != '\n')
{
line.push_back(next);
cin.get(next);
}
s where;
where = find(line.begin( ), line.end( ), 'e');
//where is located at the first occurrence of 'e' in v.
s p;
cout << "You entered the following before you entered your first e:\n";
for (p = line.begin( ); p != where; p++)
cout << *p;
cout << endl;
cout << "You entered the following after that:\n";
for (p = where; p != line.end( ); p++)
cout << *p;
cout << endl;
19-16.cpp:26:5: error: 'where' was not declared in this scope
26 | where = find(line.begin( ), line.end( ), 'e');
| ^~~~~
19-16.cpp:29:19: error: expected ';' before 'p'
29 | const_iterator p;
| ^~
| ;
19-16.cpp:31:10: error: 'p' was not declared in this scope
31 | for (p = line.begin( ); p != where; p++)
| ^
19-16.cpp:36:10: error: 'p' was not declared in this scope
36 | for (p = where; p != line.end( ); p++)
| ^
sglbl@SglblPC:/mnt/c/Araçlar/GTU/2.Sınıf/CSE241/Source Code/Ch19$ ^C
sglbl@SglblPC:/mnt/c/Araçlar/GTU/2.Sınıf/CSE241/Source Code/Ch19$ c++ 19-16.cpp
sglbl@SglblPC:/mnt/c/Araçlar/GTU/2.Sınıf/CSE241/Source Code/Ch19$ ./a.out
Enter a line of text:
suleyman
You entered the following before you entered your first e:
sul
You entered the following after that:
eyman
End of demonstration.
sglbl@SglblPC:/mnt/c/Araçlar/GTU/2.Sınıf/CSE241/Source Code/Ch19$
Line 38, Column 18

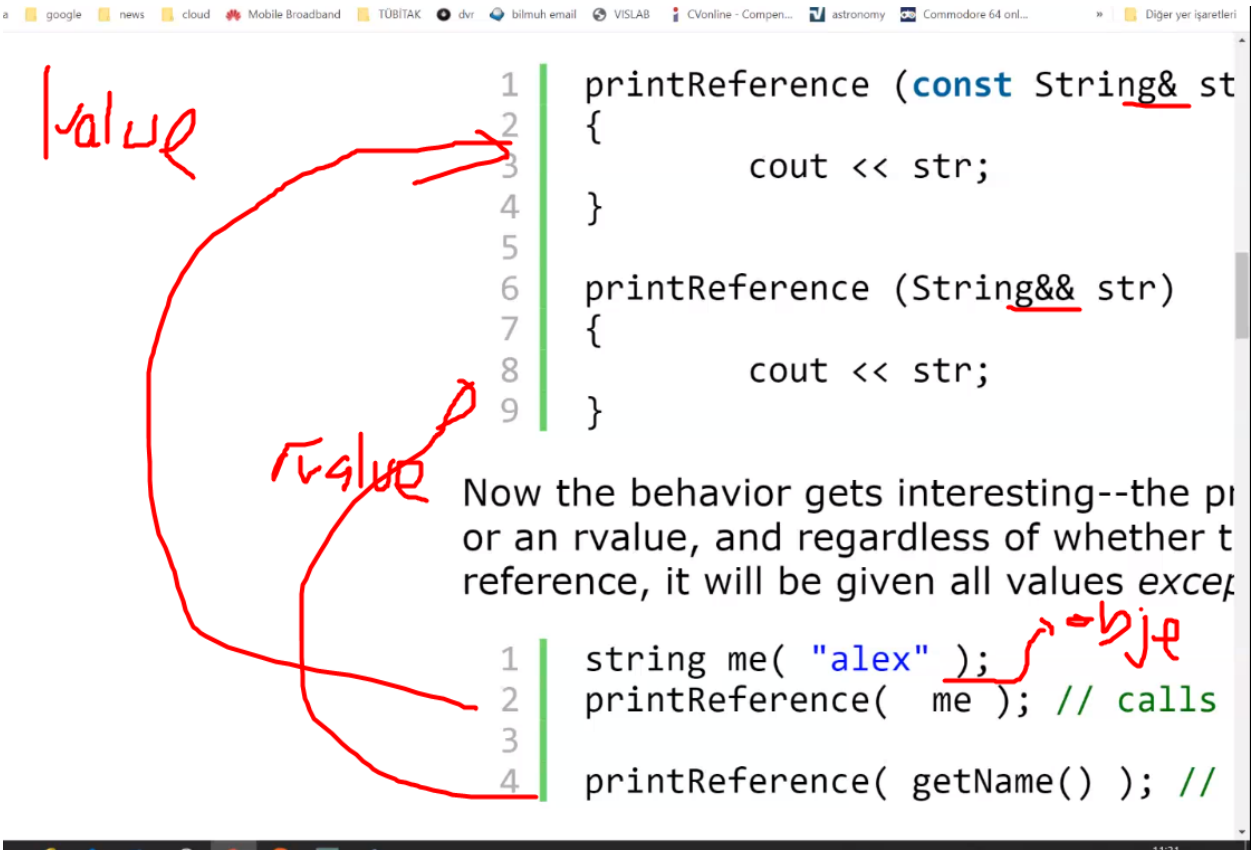
It's a "pointer to member" - the following code illustrates its use:

```
#include <iostream>
using namespace std;

class Car
{
    public:
    int speed;
};

int main()
{
    int Car::*pSpeed = &Car::speed;

    Car c1;
    c1.speed = 1;       // direct access
    cout << "speed is " << c1.speed << endl;
    c1.*pSpeed = 2;     // access via pointer to member
    cout << "speed is " << c1.speed << endl;
    return 0;
}
```


Right value reference

- For overloading, we need a new type
  - Reference type for performance reasons
  - Overload resolution should prefer this new type on rvalue objects

```
void f(X& arg_)       // lvalue reference parameter
void f(X&& arg_)      // rvalue reference parameter
void f(const X& arg_) // const lvalue reference parameter

X x;
X g();

f(x);    // lvalue argument --> f(X&)
f(g());  // rvalue argument --> f(X&&)
```

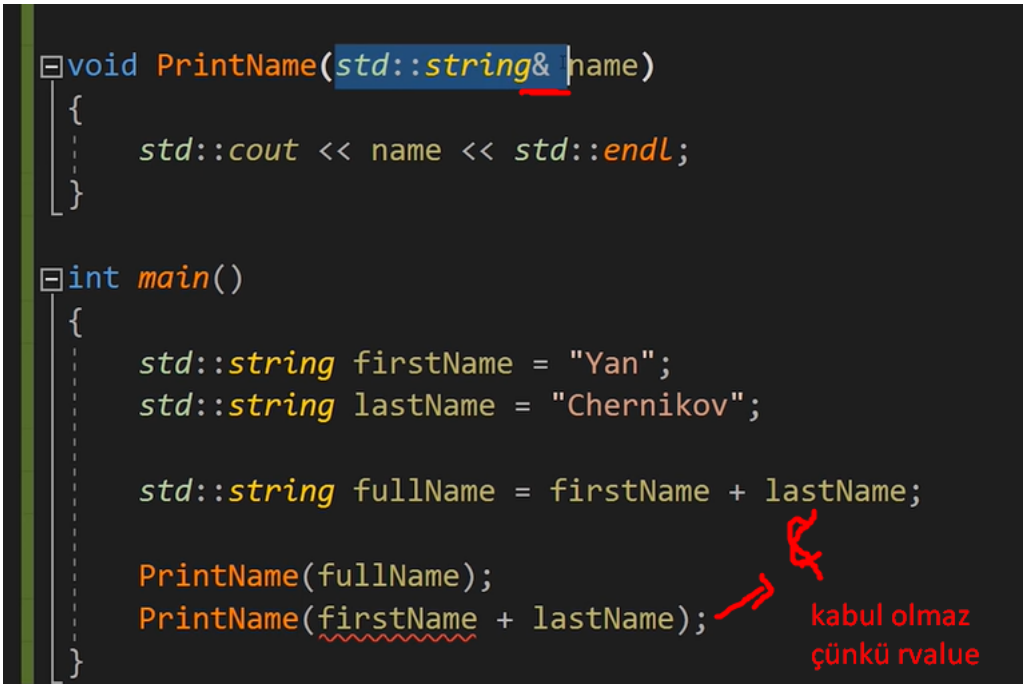


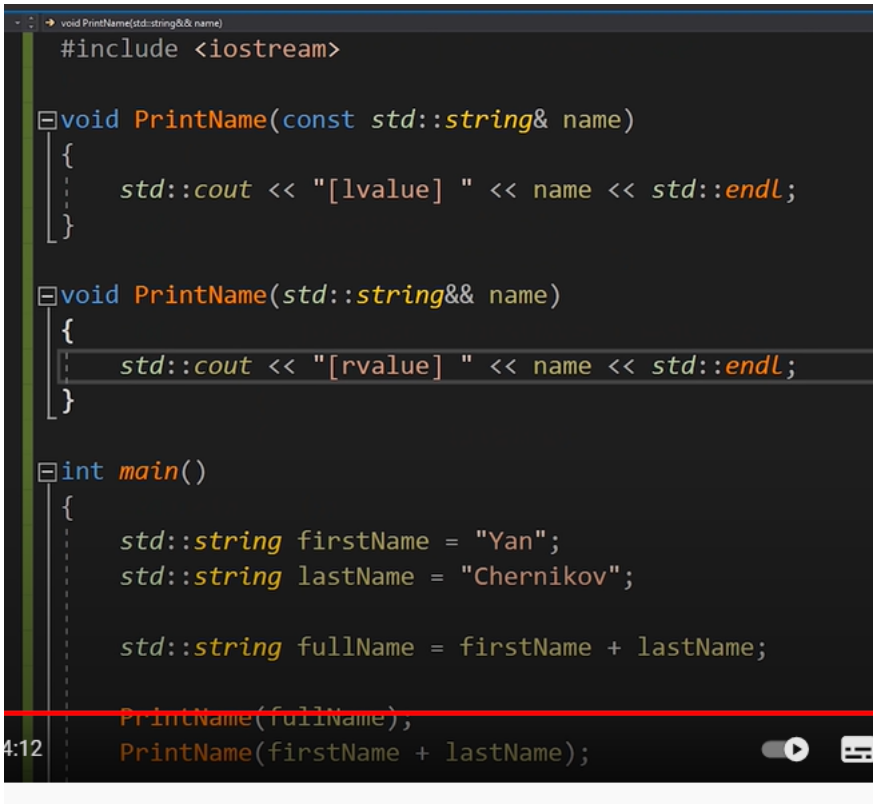

```
ArrayWrapper & operator= ( ArrayWrapper&& other)
{
    delete [] _p_vals;
    _size = other._size;
    _p_vals = other._p_vals;

    other._p_vals = nullptr;
    other._size = 0;

    return *this;
}
```

MOVE OPERATOR

FOR RVALUE

NOT CONST BECAUSE WE ARE CHANGING



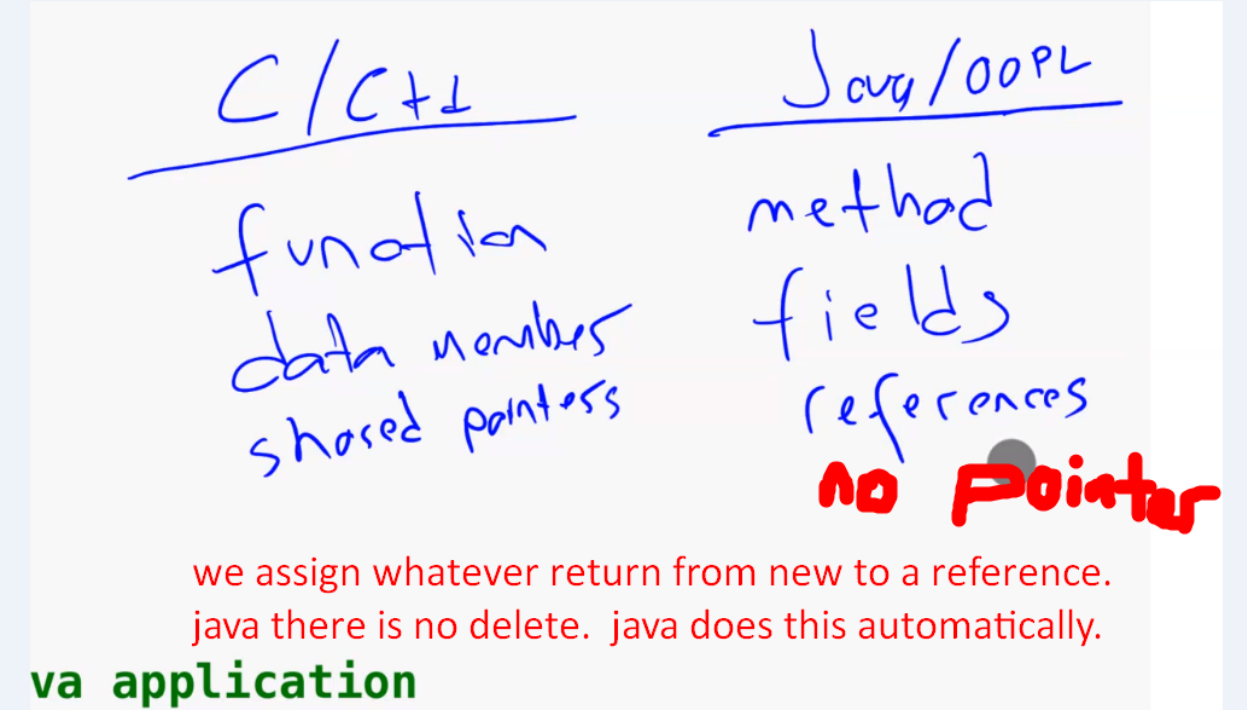


&& sadece rvalue için çalışacak artık.

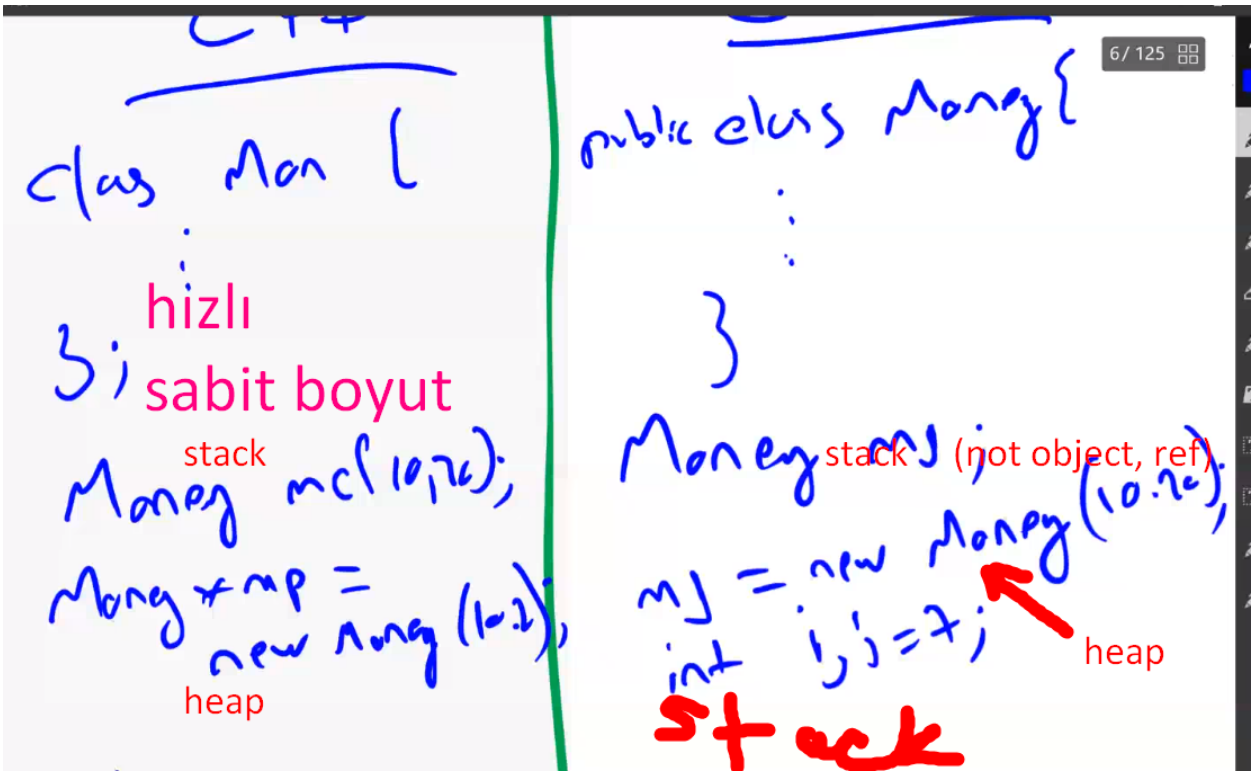
https://stackoverflow.com/questions/274626/what-is-object-slicing 2.sine bakk.

dynamic_cast<> kullan

```
class rule_of_five
{
    char* cstring; // raw pointer used as a handle to a dynamically-allocated memory block
 public:
    rule_of_five(const char* s = "")
    : cstring(nullptr)
    {
        if (s) {
            std::size_t n = std::strlen(s) + 1;
            cstring = new char[n];      // allocate
            std::memcpy(cstring, s, n); // populate
        }
    }

    ~rule_of_five()
    {
        delete[] cstring;  // deallocate
    }

    rule_of_five(const rule_of_five& other) // copy constructor
    : rule_of_five(other.cstring)
    {}

    rule_of_five(rule_of_five&& other) noexcept // move constructor
    : cstring(std::exchange(other.cstring, nullptr))
    {}

    rule_of_five& operator=(const rule_of_five& other) // copy assignment
    {
         return *this = rule_of_five(other);
    }

    rule_of_five& operator=(rule_of_five&& other) noexcept // move assignment
    {
        std::swap(cstring, other.cstring);
        return *this;
    }
```



```
public class Ders3{
  static int kareAl(int sayi){
    return sayi*sayi;
  }

  public static void main(String[] args){
    // Arrayler
    // int dizim[5][4];    // yanlis kullanim
    int[] dizi = new int[6];
    int[] dizi2= {1,2,3,4,5};

    // 2D Arrayler
    int[][] ikiBoyut = new int[5][6];
    for(int i=0; i<ikiBoyut.length; i++)
      for(int j=0; j<ikiBoyut[0].length; j++) //2.indexin size'i icin ikiBoyut[0].length yap.
        ikiBoyut[i][j] = i;

  } //END OF MAIN FUNCTION

}
```



Early Binding - Kod çalışmadan önce her şey belirlenmiştir. Bu nedenle baya baya hızlıdır. Overloading methodlar derleme anında kararlaştırıldığından bu yöntemi kullanır.

Late Binding -

- Her şey çalışma anında belirlenir. Bundan dolayı yavaştır.
- Overriding methodlar çalışma anında belirlendiği için bu yöntemi kullanır.
- Rahat ve esnektir. Tiplere bağlı kalmadan çalışabilirsiniz.

Java'da object için default value = null

In java string is immutable- > unmodifiable or unchangeable after it created.

In java i = j; not a deep copy. Shallow copy. It is like reference

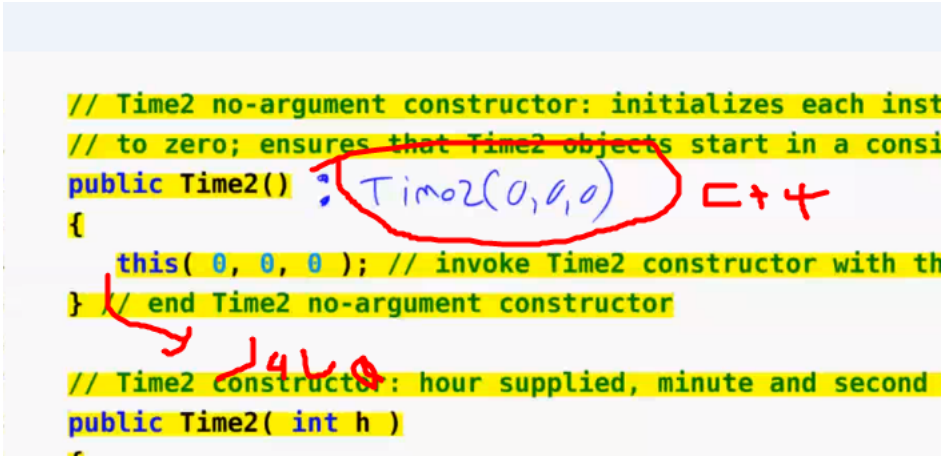
code

```
 import java.util.Scanner; //SCANNER İÇİN
```

```
for(GradeBook gbr: pa){  //EACH FOR
    gbr.printmessage();
}
```

Tüm objeler new kullanmak lazım(String, array hariç)

```
  int myint = Integer.parseInt(mystring); //String to int
```



finalize() is called by System.gc();

```
  str = String.format("total = %d " , total);   //C++'daki sprintf gibi.
```

private final static -> create this variable only once. private final -> create this variable for every object. First one saves memory, go for it. (CONST GİBİ)

String karşılaştırma → s1.equals(s2)

```
// using clone()
import java.util.ArrayList;

// An object reference of this class is
// contained by Test2
class Test
{
    int x, y;
}

// Contains a reference of Test and implements
// clone with shallow copy.
class Test2 implements Cloneable
{
    int a;
    int b;
    Test c = new Test();
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
```

```
    }
}

// Driver class
public class Main
{
    public static void main(String args[]) throws
                        CloneNotSupportedException
    {
       Test2 t1 = new Test2();
       t1.a = 10;
       t1.b = 20;
       t1.c.x = 30;
       t1.c.y = 40;

       Test2 t2 = (Test2)t1.clone();

       // Creating a copy of object t1 and passing
       //  it to t2
       t2.a = 100;

       // Change in primitive type of t2 will not
       // be reflected in t1 field
       t2.c.x = 300;

       // Change in object type field will be
       // reflected in both t2 and t1(shallow copy)
       System.out.println(t1.a + " " + t1.b + " " +
                          t1.c.x + " " + t1.c.y);
       System.out.println(t2.a + " " + t2.b + " " +
                          t2.c.x + " " + t2.c.y);
    }
}
```

CLONE protected. Client çağıracaksa public biçimde override edilmeli

downcast = super classı subclassa eşitliyorsun.

Dynamic binding = late binding

getClass.getName() ile objen hakkında bilgi ogrenebilirsin.

In java all oll object methods are virtual by default.

```
public abstract double earnings();  //Abstract class içinde
//override edilecek metodu boyle yazabilirsin
```

FINAL METHODS

-Final methods are resolved at compile time . (Static binding)

-Cannot be extended by a subclass.

If class in not abstract, it has to override all methods.

```
Scanner scanner = new Scanner(System.in);
        System.out.println("SULEYMAN GOLBOL'S OWN COLLECTION");
        while(loopFlag == true){
            try{System.out.println("5) Try ArrayList Test for Integers");
                System.out.println("7) Test Errors.\n8) Exit.");
                System.out.print("Please enter from menu: ");
                choose = scanner.nextInt(); //Getting choose from user
                scanner.nextLine(); //If wrong input, don't give error.
            }
            catch(InputMismatchException e){
                System.out.println("Input is " + e.getMessage() + "\nTry Again!");
                scanner.nextLine();
                continue;
            }
```

Static methods cannot be overridden in Java.

Eğer bi fonksiyonun override edilmesini istemiyosak sonunda final yaz

big three inherit edilmez

Derived classta override ediceksen, base'de virtual oldugunu yazman lazım
Side note: You definitely want to pass an std::vector by reference, not by value.
Side note 2: Classes with virtual functions should have a virtual destructor.
Just ensure that you don't call a pure virtual function from constructor or destructor.
Don't call virtual methods in constructor or destructor unless you understand the dynamics involved.
Abstract class'ta virtual yazdıgın her şeyin sonuna =0; koydugundan emin ol.
Virtual olan her şeyi derived class'ta implement ettiginden emin ol(yoksa derived class da abstract olur)
Compile ederken tüm .cpp dosyalarının derlendiginden emin ol
Base class'ta desturctor'ın varsa ve implement edilmemişse implement ettiginden emin ol