ECE 462
Object-Oriented Programming using C++ and Java

Inheritance (2)

Yung-Hsiang Lu
yunglu@purdue.edu

public or private inheritance in C++

public inheritance in C++

- **interface**

```
class Person {
protected:
        string name;
public:
        string getName() { return name; }
};
class Student: public Person ...
Person * p = new Student ...     // allowed
Student * s = new Person ...     // not allowed
```

- **implementation** (attributes and methods)

```
Student * s = new Student ...
s -> getName();                  // allowed
```

calling the method in base class

```
class Person {
public:
   void print(...) { ... }                  // Person::print
};
class Student: public Person {
public:
   void print(...) {
     Person::print();                       // call print in base class
     // print additional attributes
   }
};
Person * p = new Student ...
p-> print();            // call Student::print, polymorphism
```

C/C++ - inheritance/DerivedConstructor.cc - Eclipse SDK

```
//DerivedConstructor.cc

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

class User { // BASE
    string name;
    int age;
public:
    User(string nm, int a) {
        name = nm;
        age = a;
    }
    void print() {
        cout << "Name: " << name << "   Age: " << age;
    }
};

class StudentUser : public User { // DERIVED
    string schoolEnrolled;
public:
    StudentUser(string nam, int y, string school) :
        User(nam, y) { //(A)
        schoolEnrolled = school;
    }
    void print() {
        User::print();
```

calling base's constructor

C/C++ - inheritance/DerivedConstructor.cc - Eclipse SDK

File Edit Refactor Navigate Search Project Run Window Help

DerivedConstructor.cc

```cpp
    User(string nm, int a) {
        name = nm;
        age = a;
    }
    void print() {
        cout << "Name: " << name << "   Age: " << age;
    }
};

class StudentUser : public User { // DERIVED
    string schoolEnrolled;
public:
    StudentUser(string nam, int y, string school) :
        User(nam, y) { //(A)
        schoolEnrolled = school;
    }
    void print() {
        User::print();
        cout << "   School Enrolled:  " << schoolEnrolled << endl;
    }
};

int main() {
    StudentUser student("Maura", 20, "ece");
    student.print();
    // Name: Maura   Age: 20   School Enrolled: ece
    return 0;
}
```

calling base's method

Writable | Overwrite | 5 : 21

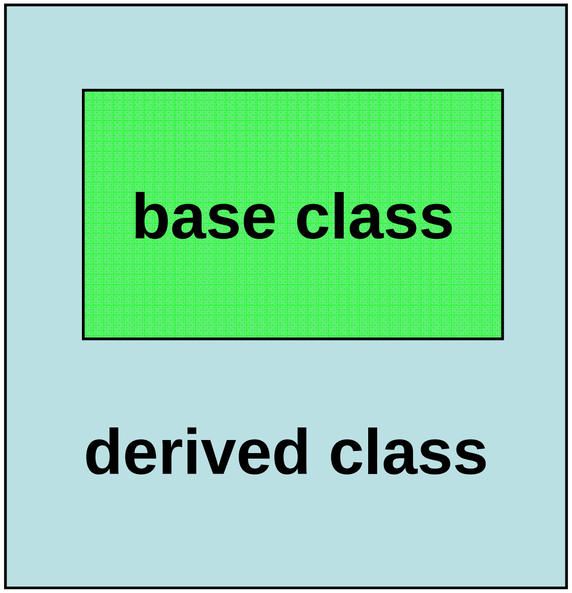
Derived Class is "Bigger"

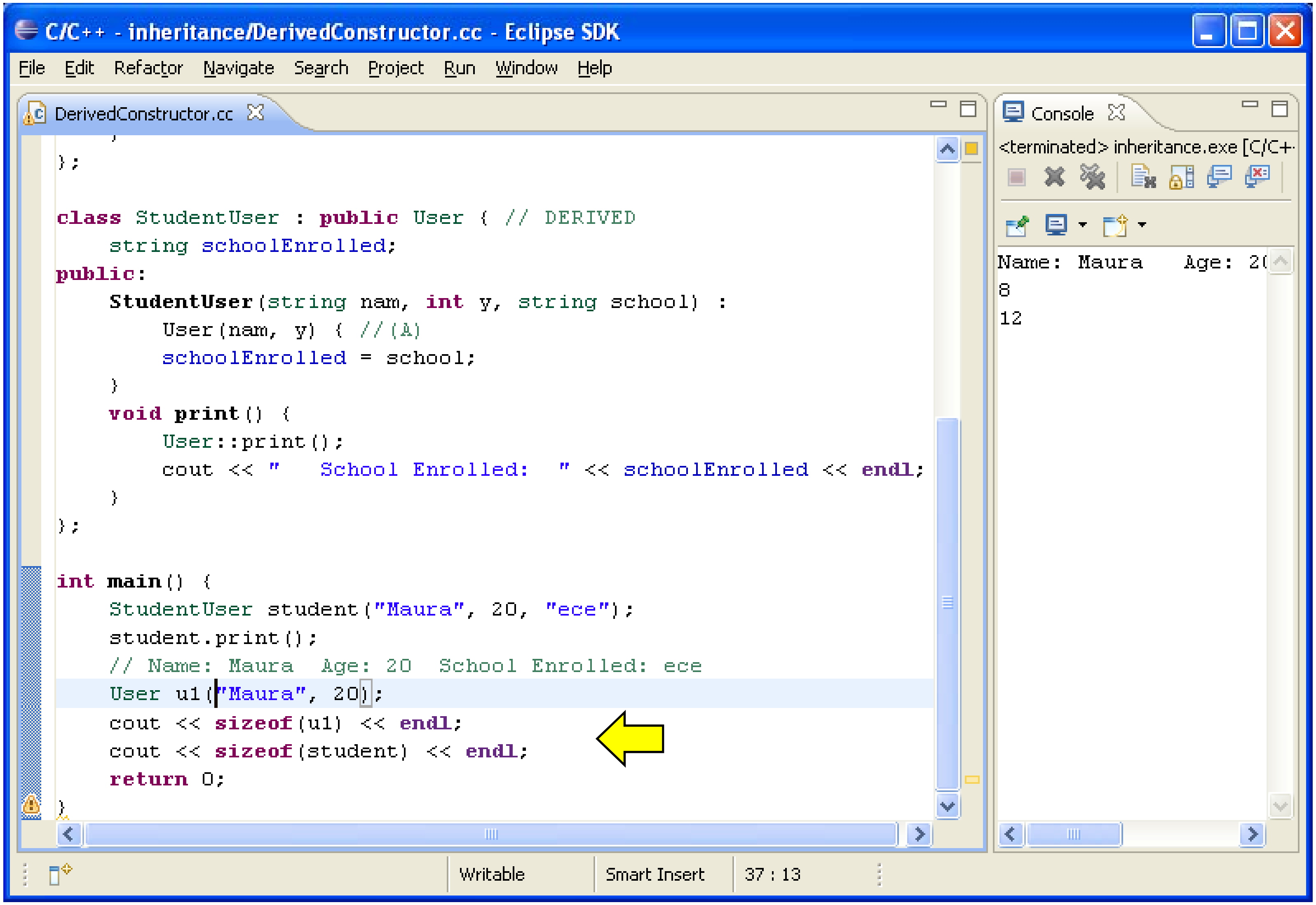
- A derived class has everything (all attributes + all public and protected methods) in the base class.
- A derived class probably has additional attributes and methods.

$\Rightarrow$ A derived class is “bigger”.

- base class: general
- derived class: specific,

  more requirements
- inheritance = specialization




C/C++ - inheritance/DerivedConstructor.cc - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
DerivedConstructor.cc
};

class StudentUser : public User { // DERIVED
string schoolEnrolled;
public:
StudentUser(string nam, int y, string school) :
User(nam, y) { //(A)
schoolEnrolled = school;
}
void print() {
User::print();
cout << "   School Enrolled: " << schoolEnrolled << endl;
}
};

int main() {
StudentUser student("Maura", 20, "ece");
student.print();
// Name: Maura Age: 20 School Enrolled: ece
User u1("Maura", 20);
cout << sizeof(u1) << endl;
cout << sizeof(student) << endl;
return 0;
}
Console
<terminated> inheritance.exe [C/C+
Name: Maura   Age: 2(
8
12
Writable
Smart Insert
37 : 13

Copy Constructor in Derived Class

C/C++ - inheritance/DerivedCopyConstruct.cc - Eclipse Platform

File Edit Refactor Navigate Search Run Project Window Help

DerivedCopyConstruct.cc

```cpp
//DerivedCopyConstruct.cc
#include <iostream>
using namespace std;

class X
{ // BASE
    int m;
public:
    //base class constructor:
    X(int mm) :
        m(mm)   {
    }
    //base class copy constructor:
    X(const X& other) :
        m(other.m)  {
    } //(A)
    void print()    {
        cout << "m of X obj: " << m << endl;
    }
};
class Y : public X
{ // DERIVED
    int n;
public:
    //derived class constructor:
    Y(int mm, int nn) :
        X(mm), n(nn)    {
    }
    //derived class copy constructor:
```

Console

<terminated> inheritance.exe [C/C++ l

```
m of X obj: 5

m of X obj: 2
n of Y obj: 3

m of X obj: 2
n of Y obj: 3
```

Writable | Smart Insert | 33 : 17

C/C++ - inheritance/DerivedCopyConstruct.cc - Eclipse Platform

File Edit Refactor Navigate Search Run Project Window Help

DerivedCopyConstruct.cc

```
public:
    //derived class constructor:
    Y(int mm, int nn) :
        X(mm), n(nn)    {
    }
    //derived class copy constructor:
    Y(const Y& other) :
        X(other), n(other.n)    {
    } //(B)
    void print()    {
        X::print();
        cout << "n of Y obj: " << n << endl;
    }
};
int main()
{
    X* xptr1 = new X( 5 );
    xptr1->print(); // m of X object: 5
    cout << endl;
    Y y1( 2, 3);
    y1.print(); // m of X subobject: 2
    // n of Y object: 3
    cout << endl;
    Y y2 = y1; // invokes copy constructor for Y
    y2.print(); // m of X subobject: 2
    // n of Y object: 3
    return 0;
}
```

Console

<terminated> inheritance.exe [C/C++ l

```
m of X obj: 5

m of X obj: 2
n of Y obj: 3

m of X obj: 2
n of Y obj: 3
```

Writable | Smart Insert | 33 : 17

Operator = and Derived Class

C/C++ - inheritance/DerivedAssignOp.cc - Eclipse Platform

File Edit Refactor Navigate Search Run Project Window Help

DerivedAssignOp.cc

```
//DerivedAssignOp.cc

#include <iostream>
using namespace std;

class X {                                   // BASE
    int m;
public:
    //constructor:
    X( int mm ) : m( mm ) {}
    //copy constructor:
    X( const X& other ) : m( other.m ) {}
    //assignment op:
    X& operator=( const X& other ) {                    //(A)
        if ( this == &other ) return *this;
        m = other.m;
        return *this;
    }
    void print() {
        cout << "m of X obj: " << m << endl;
    }
};

class Y : public X {                        // DERIVED
    int n;
public:
    //constructor:
    Y( int mm, int nn ) : X( mm ), n( nn ) {}
    //copy constructor:
```

Console

<terminated> inheritance.exe [C/C++ l

```
m of X obj: 5

m of X obj: 100
n of Y obj: 110
```

Writable | Smart Insert | 32 : 53

C/C++ - inheritance/DerivedAssignOp.cc - Eclipse Platform

File Edit Refactor Navigate Search Run Project Window Help

DerivedAssignOp.cc

```
class Y : public X {                          // DERIVED
    int n;
public:
    //constructor:
    Y( int mm, int nn ) : X( mm ), n( nn ) {}
    //copy constructor:
    Y( const Y& other ) : X( other ), n( other.n ) {}
    //assignment op:
    Y& operator=( const Y& other ) {                       //(B)
        if ( this == &other ) return *this;
        X::operator=( other );
        n = other.n;
        return *this;
    }
    void print() {
        X::print();
        cout << "n of Y obj: " << n << endl; }
};

int main()
{
    X xobj_1( 5 );                    // X's constructor
    X xobj_2 = xobj_1;                // X's copy constructor

    X xobj_3( 10 );
    xobj_3 = xobj_2;                  // X's assignment op
    xobj_3.print();                   // m of X obj: 5
    cout << endl;
```

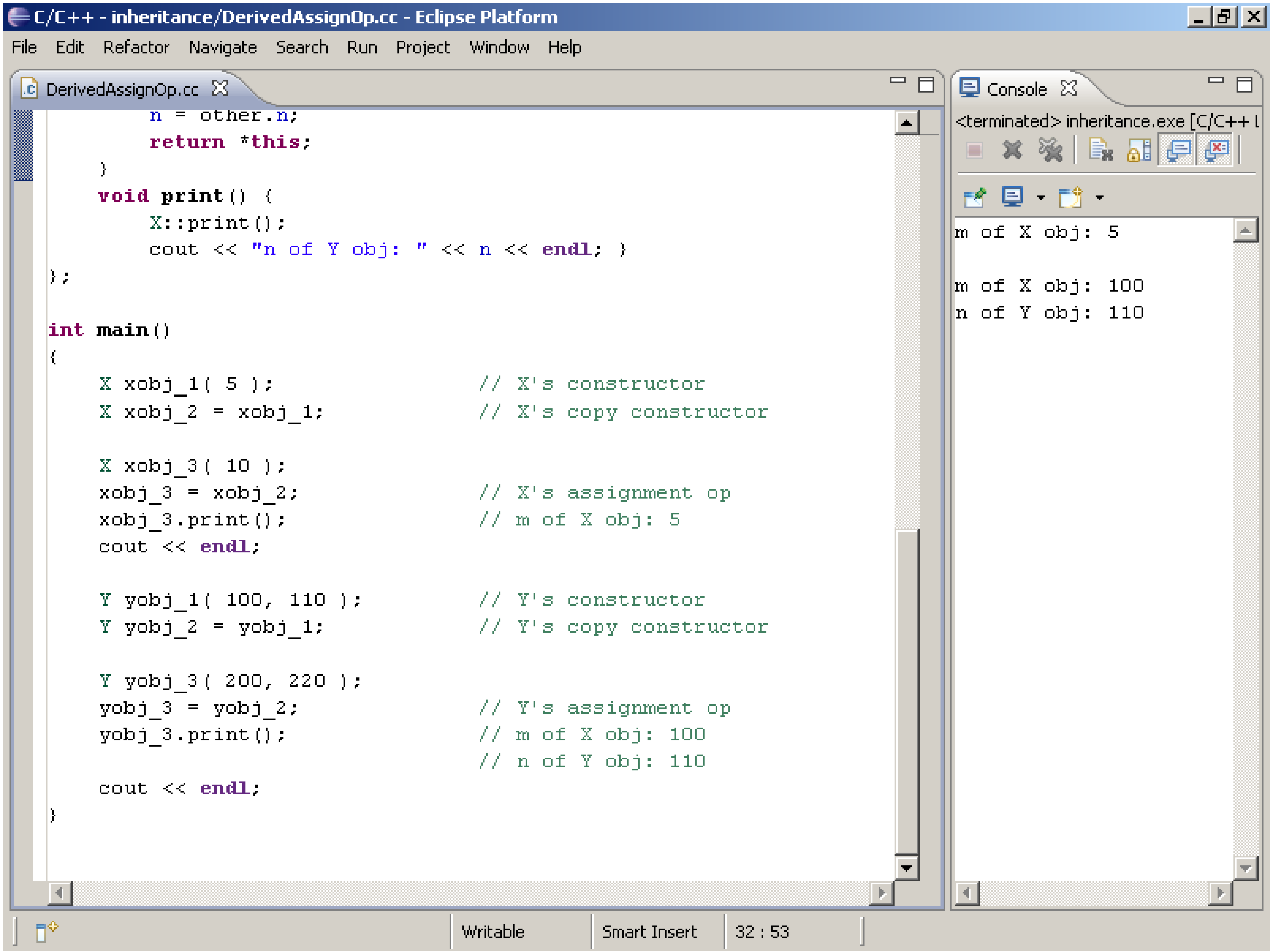
Console

<terminated> inheritance.exe [C/C++ l

```
m of X obj: 5

m of X obj: 100
n of Y obj: 110
```

Writable | Smart Insert | 32 : 53



C/C++ - inheritance/DerivedAssignOp.cc - Eclipse Platform
File Edit Refactor Navigate Search Run Project Window Help
DerivedAssignOp.cc
        n = other.n;
        return *this;
    }
    void print() {
        X::print();
        cout << "n of Y obj: " << n << endl; }
};

int main()
{
    X xobj_1(5);   // X's constructor
    X xobj_2 = xobj_1;   // X's copy constructor

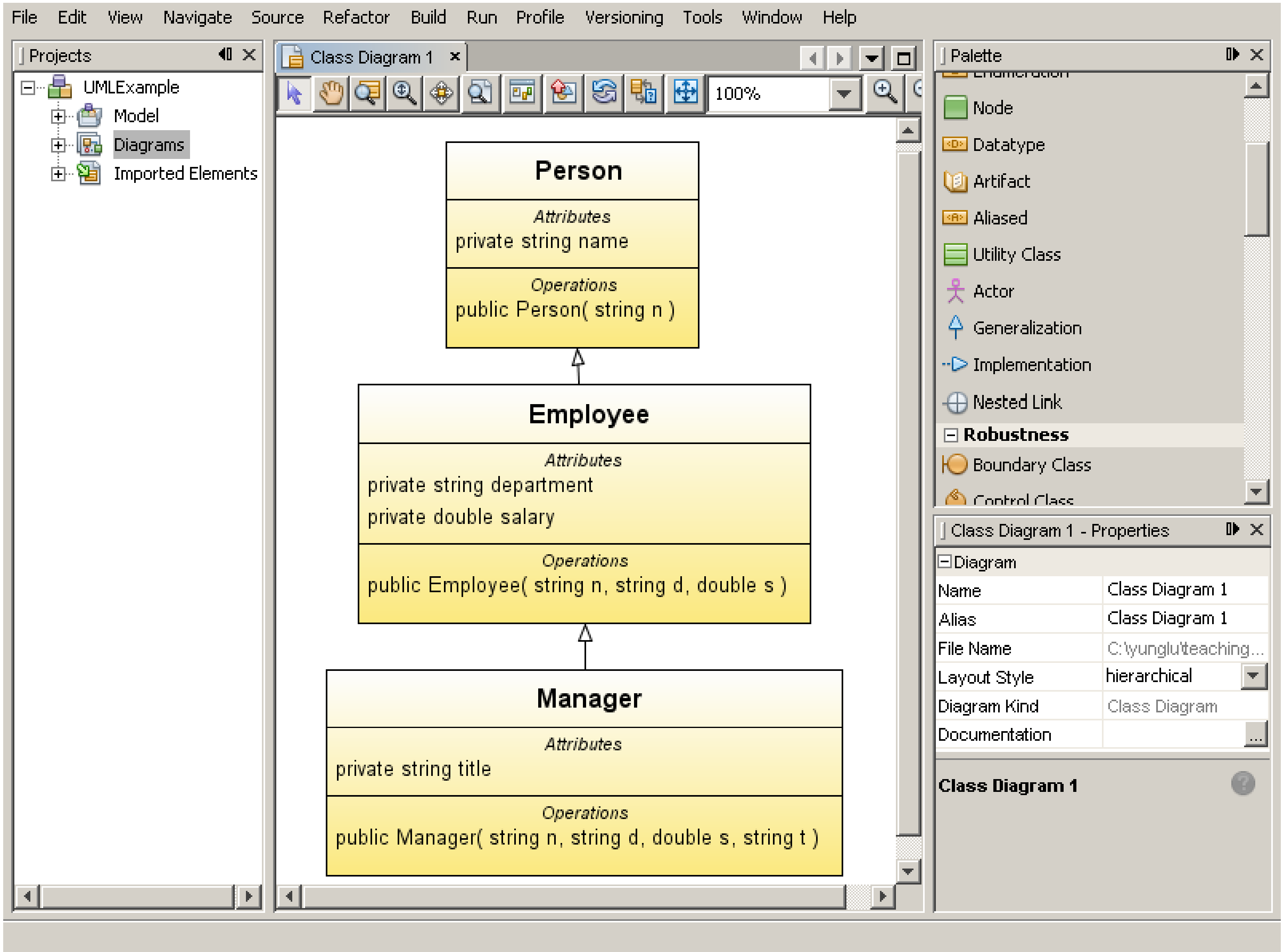
    X xobj_3(10);
    xobj_3 = xobj_2;   // X's assignment op
    xobj_3.print();   // m of X obj: 5
    cout << endl;

    Y yobj_1(100, 110);   // Y's constructor
    Y yobj_2 = yobj_1;   // Y's copy constructor

    Y yobj_3(200, 220);
    yobj_3 = yobj_2;   // Y's assignment op
    yobj_3.print();   // m of X obj: 100
                                    // n of Y obj: 110
    cout << endl;
}
Console
<terminated> inheritance.exe [C/C++ l
m of X obj: 5
m of X obj: 100
n of Y obj: 110
Writable
Smart Insert
32 : 53

Operator Overloading and Derived Class

```
ostream& operator<<(ostream& os, const Person& p)  {
   os << p.name;
return os;
}
```



File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Projects
UMLExample
Model
Diagrams
Imported Elements
Class Diagram 1
100%
Person
Attributes
private string name
Operations
public Person(string n)
Employee
Attributes
private string department
private double salary
Operations
public Employee(string n, string d, double s)
Manager
Attributes
private string title
Operations
public Manager(string n, string d, double s, string t)
Palette
Node
Datatype
Artifact
Aliased
Utility Class
Actor
Generalization
Implementation
Nested Link
Robustness
Boundary Class
Class Diagram 1 - Properties
Diagram
Name
Class Diagram 1
Alias
Class Diagram 1
File Name
C:\yunglu\teaching...
Layout Style
hierarchical
Diagram Kind
Class Diagram
Documentation
Class Diagram 1

C/C++ - inheritance/DerivedOverloadOp.cc - Eclipse Platform

DerivedOverloadOp.cc

```cpp
//DerivedOverloadOp.cc

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

///////////////////////////  class Person  /////////////////////
class Person {
    string name;
public:
    Person(string nom) :
        name(nom)   {
    }
    Person(const Person& p) :
        name(p.name)    {
    }
    Person& operator=(const Person& p)   {
        if ( this != &p)
            name = p.name;
        return *this;
    }
    virtual ~Person()   {
    }
    friend ostream& operator<<(ostream& os, const Person& p);
};

//overload << for base class Person:
ostream& operator<<(ostream& os, const Person& p)   { //(D)
    os << p.name;
```

Console

<terminated> inheritance.exe [C/C++ L

```
Zahpod assembly 100 di
Trillion sales 200 vice
```

Writable | Smart Insert | 82 : 18

C/C++ - inheritance/DerivedOverloadOp.cc - Eclipse Platform

File Edit Refactor Navigate Search Run Project Window Help

DerivedOverloadOp.cc

```
ostream& operator<<(ostream& os, const Person& p)  { //(D)
    os << p.name;
    return os;
}

//////////////////////////  class Employee  ////////////////////
class Employee : public Person  {
    string department;
    double salary;
public:
    Employee(string name, string dept, double s) :
        Person(name), department(dept), salary(s)   {
    }
    Employee(const Employee& e)  :
        Person(e), department(e.department), salary(e.salary)   {
    }
    Employee& operator=(const Employee& e)  {
        if ( this != &e)        {
            Person::operator=( e );
            department = e.department;
            salary = e.salary;
        }
        return *this;
    }
    ~Employee()
    {}
    friend ostream& operator<<( ostream& os, const Employee& p );
};
```

Console

<terminated> inheritance.exe [C/C++ L

```
Zahpod assembly 100 di
Trillion sales 200 vic
```

Writable | Smart Insert | 82 : 18

C/C++ - inheritance/DerivedOverloadOp.cc - Eclipse Platform

```cpp
//overload << for derived class Employee:
ostream& operator<<(ostream& os, const Employee& e)  { //(E)
    const Person* ptr = &e; //upcast
    os << *ptr;
    os << " " << e.department << " " << e.salary;
    return os;
}

///////////////////////////  class Manager  ////////////////////
class Manager : public Employee  {
    string title;
public:
    Manager(string name, string dept, double salary, string atitle)
        Employee(name, dept, salary), title(atitle)    {
    }
    Manager(const Manager& m)  :
        Employee(m), title(m.title)     {
    }
    Manager& operator=(const Manager& m)     {
        if ( this != &m)          {
            Employee::operator=( m );
            title = m.title;
        }
        return *this;
    }
    ~Manager()   {
    }
    friend ostream& operator<<( ostream& os, const Manager& m );
};
```

upcasting

Console

```
<terminated> inheritance.exe [C/C++ L
d assembly 100 di
Trillion sales 200 vic
```

Writable | Smart Insert | 82 : 18

C/C++ - inheritance/DerivedOverloadOp.cc - Eclipse Platform

File Edit Refactor Navigate Search Run Project Window Help

DerivedOverloadOp.cc

```cpp
            }
            return *this;
        }
        ~Manager()  {
        }
        friend ostream& operator<<( ostream& os, const Manager& m );
};

//overload << for derived class Manager:
ostream& operator<<(ostream& os, const Manager& m)
{ //(F)
    const Employee* ptr = &m; //upcast
    os << *ptr;
    os << " " << m.title;
    return os;
}

//////////////////////////////// main ///////////////////////////
int main()
{
    Manager m1("Zahpod", "assembly", 100, "director");
    Manager m2(m1); // invokes copy construct
    cout << m2 << endl; // Zaphod assembly 100 director
    Manager m3("Trillion", "sales", 200, "vice_pres");
    m2 = m3; // invokes assignment oper
    cout << m2 << endl; // Trillion sales 200 vice_pres
    return 0;
}
```

Console

<terminated> inheritance.exe [C/C++ L

```
Zahpod assembly 100 di
Trillion sales 200 vic
```

Writable | Smart Insert | 82 : 18

Operator and Polymorphism?

```
Person * p1 = new Manager;
p1 -> print();
cout << (* p1) << endl;
```

C/C++ - inheritance/OverloadPolymorphism.cc - Eclipse SDK

OverloadPolymorphism.cc

```cpp
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <string>
using namespace std;

class Person
{
public:
    virtual void print()
    {
        cout << "Person::print" << endl;
    }
    friend ostream& operator<<(ostream& os, const Person& p);
};
ostream& operator<<(ostream& os, const Person& p)
{
    cout << "operator<<(ostream& os, const Person& p)" << endl;
    return os;
}

class Employee : public Person
{
public:
    void print()
    {
        cout << "Employee::print" << endl;
    }
```

Writable | Smart Insert | 5 : 21

C/C++ - inheritance/OverloadPolymorphism.cc - Eclipse SDK

File Edit Refactor Navigate Search Run Project Window Help

OverloadPolymorphism.cc

```
    }
    friend ostream& operator<<(ostream& os, const Employee& p);
};
ostream& operator<<(ostream& os, const Employee& e)
{
    cout << "operator<<(ostream& os, const Employee& e)" << endl;
    const Person* ptr = &e; //upcast
    os << *ptr;
    return os;
}
class Manager : public Employee
{
public:
    void print()
    {
        cout << "Manager::print" << endl;
    }
    friend ostream& operator<<(ostream& os, const Manager& m);
};

ostream& operator<<(ostream& os, const Manager& m)
{
    cout << "operator<<(ostream& os, const Manager& m)" << endl;
    const Employee* ptr = &m;
    os << *ptr;
    return os;
}
int main()
```

Writable | Smart Insert | 5 : 21

C/C++ - inheritance/OverloadPolymorphism.cc - Eclipse SDK

File Edit Refactor Navigate Search Run Project Window Help

OverloadPolymorphism.cc

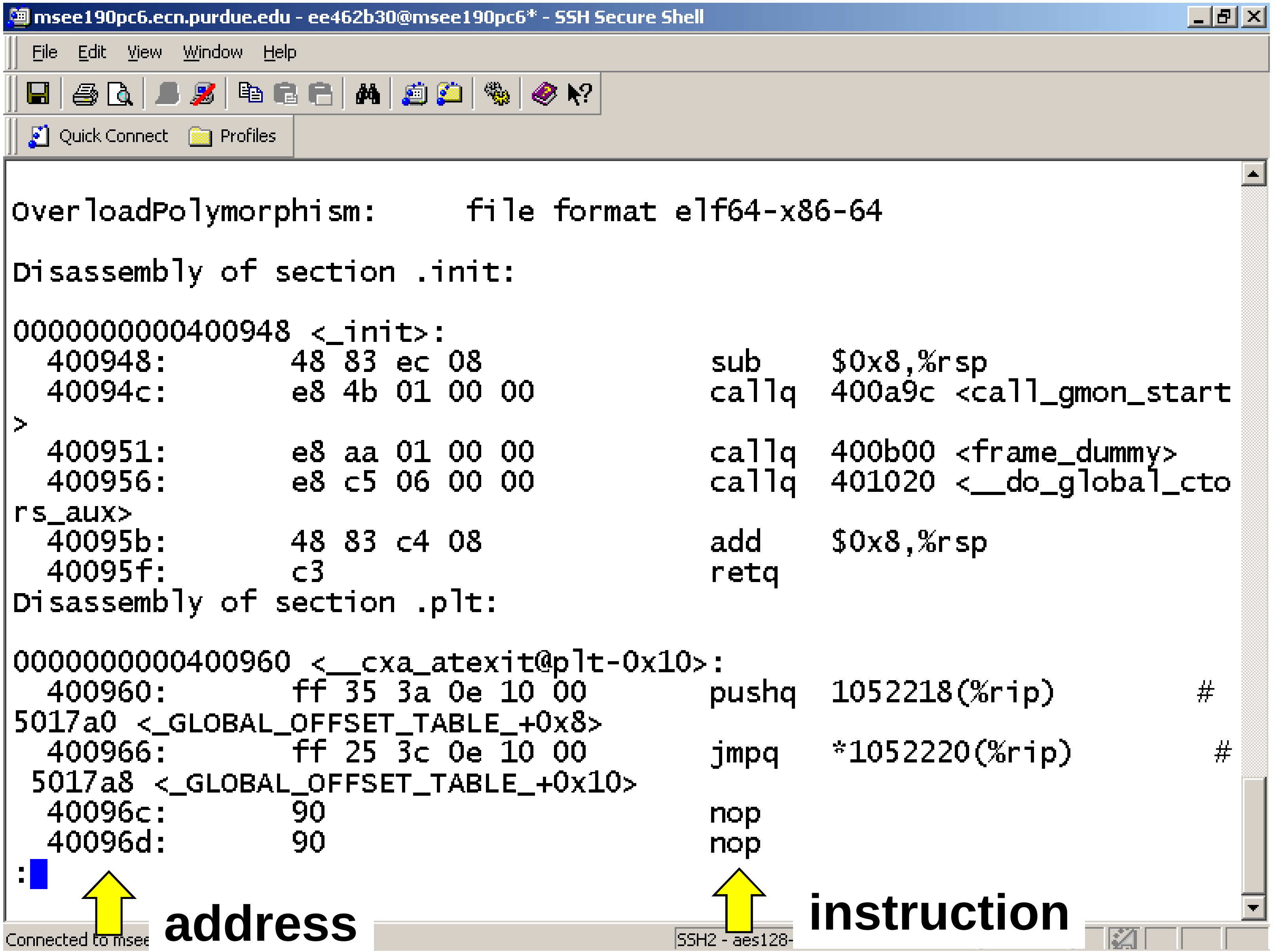
```
ostream& operator<<(ostream& os, const Manager& m)
{
    cout << "operator<<(ostream& os, const Manager& m)" << endl;
    const Employee* ptr = &m;
    os << *ptr;
    return os;
}
int main()
{
    Person * p1;
    srand(time(NULL));
    int val = rand() % 2;
    if (val == 0)
    {
        p1 = new Manager;
    }
    else
    {
        p1 = new Person;
    }
    cout << "val = " << val << endl;
    p1 -> print();
    cout << (*p1) << endl;
    delete p1;
    return 0;
}
```

Writable | Smart Insert | 5 : 21

ECE 462
Object-Oriented Programming using C++ and Java

Virtual Function (2)

Yung-Hsiang Lu
yunglu@purdue.edu


msee190pc6.ecn.purdue.edu - ee462b30@msee190pc6* - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
OverloadPolymorphism: file format elf64-x86-64
Disassembly of section .init:
0000000000400948 <_init>:
400948: 48 83 ec 08 sub $0x8,%rsp
40094c: e8 4b 01 00 00 callq 400a9c <call_gmon_start
>
400951: e8 aa 01 00 00 callq 400b00 <frame_dummy>
400956: e8 c5 06 00 00 callq 401020 <__do_global_cto
rs_aux>
40095b: 48 83 c4 08 add $0x8,%rsp
40095f: c3 retq
Disassembly of section .plt:
0000000000400960 <__cxa_atexit@plt-0x10>:
400960: ff 35 3a 0e 10 00 pushq 1052218(%rip) #
5017a0 <_GLOBAL_OFFSET_TABLE_+0x8>
400966: ff 25 3c 0e 10 00 jmpq *1052220(%rip) #
5017a8 <_GLOBAL_OFFSET_TABLE_+0x10>
40096c: 90 nop
40096d: 90 nop
:
Connected to msee
SSH2 - aes128-
address
instruction

msee190pc6.ecn.purdue.edu - ee462b30@msee190pc6* - SSH Secure Shell

```
0000000000400970 <__cxa_atexit@plt>:
  400970:       ff 25 3a 0e 10 00       jmpq   *1052218(%rip)        #
 5017b0 <_GLOBAL_OFFSET_TABLE_+0x18>
  400976:       68 00 00 00 00          pushq  $0x0
  40097b:       e9 e0 ff ff ff          jmpq   400960 <_init+0x18>

0000000000400980 <__libc_start_main@plt>:
  400980:       ff 25 32 0e 10 00       jmpq   *1052210(%rip)        #
 5017b8 <_GLOBAL_OFFSET_TABLE_+0x20>
  400986:       68 01 00 00 00          pushq  $0x1
  40098b:       e9 d0 ff ff ff          jmpq   400960 <_init+0x18>

0000000000400990 <_ZNSt8ios_base4InitC1Ev@plt>:
  400990:       ff 25 2a 0e 10 00       jmpq   *1052202(%rip)        #
 5017c0 <_GLOBAL_OFFSET_TABLE_+0x28>
  400996:       68 02 00 00 00          pushq  $0x2
  40099b:       e9 c0 ff ff ff          jmpq   400960 <_init+0x18>

00000000004009a0 <_ZNSolsEi@plt>:
  4009a0:       ff 25 22 0e 10 00       jmpq   *1052194(%rip)        #
 5017c8 <_GLOBAL_OFFSET_TABLE_+0x30>
  4009a6:       68 03 00 00 00          pushq  $0x3
  4009ab:       e9 b0 ff ff ff          jmpq   400960 <_init+0x18>
:
```

Connected to msee190pc6.ecn.purdue.edu   SSH2 - aes128-cbc - hmac-md5 - none   70x24

How Do Function Call Work?

address	instruction
0X40096a0	
0X40096a4	
0X40096a8	
0X40096ac	
0X40096b0	
0X40096b4	
0X40096b8	**call**
0X40096bc	

Function call: change the address of the program counter.

1. push the current address at the call stack
2. **change program counter**
3. execute the instruction in the called function
4. pop the call stack and restore the program counter

virtual **in C++**

- allow polymorphism: a derived class to change the behavior (“override”, “new implementation”). a virtual method is virtual for **all derived** classes.
- If a derived class can use the same method, do not override the method.
- A virtual method must have the same prototype (i.e. return type and argument types).
- virtual $\Rightarrow$ derived class may (not have to) override
- not virtual $\Rightarrow$ should not override, compiler will allow, but don’t ask for trouble
- why virtual in C++? improve performance ... but ... cause too much confusion

C/C++ - inheritance/virtualfunction.cc - Eclipse Platform

virtualfunction.cc

```
#include <iostream>
#include <stdlib.h>
#include <string>
using namespace std;

class Person
{
public:
    virtual void fv1()
    {
        cout << "Person::fv1" << endl;
    }
    void fv2()
    {
        cout << "Person::fv2" << endl;
    }
};

class Employee : public Person
{
public:
    void fv1()
    {
        cout << "Employee::fv1" << endl;
    }
    virtual void fv2()
    {
        cout << "Employee::fv2" << endl;
    }
```

Writable | Smart Insert | 5 : 1

C/C++ - inheritance/virtualfunction.cc - Eclipse Platform

File Edit Refactor Navigate Search Run Project Window Help

virtualfunction.cc

```cpp
class Manager : public Employee
{
public:
    void fv1()
    {
        cout << "Manager::fv1" << endl;
    }
    void fv2()
    {
        cout << "Manager::fv2" << endl;
    }
};

int main()
{
    Person * p1;
    p1 = new Manager;
    p1 -> fv1();
    p1 -> fv2();
    delete p1;
    Employee * e1;
    e1 = new Manager;
    e1 -> fv1();
    e1 -> fv2();
    delete e1;
    return 0;
}
```

`p2' was not declared in this scope | Writable | Smart Insert | 56 : 14

C/C++ - inheritance/virtualfunction.cc - Eclipse Platform

File Edit Refactor Navigate Search Run Project Window Help

virtualfunction.cc

```cpp
        cout << "Manager::fv2" << endl;
    }
};

int main()
{
    Person * p1;
    p1 = new Manager;
    p1 -> fv1();
    p1 -> fv2();
    delete p1;
    Employee * e1;
    e1 = new Manager;
    e1 -> fv1();
    e1 -> fv2();
    delete e1;
    return 0;
}
```

Console

<terminated> inheritance.exe [C/C++ Local Application] C:\yunglu\workspace\inheritance\Debug\inheritance.exe (6/5/08 10:53 AM)

```
Manager::fv1
Person::fv2
Manager::fv1
Manager::fv2
```

Writable | Smart Insert | 44 : 1

How Does virtual Work?

```
class Base {
   virtual void xxx() { /* xxx1 */ }
   void yyy() { /* yyy1 */ }
};
class Derived: public Base {
   void xxx() { /* xxx2 */ }
};
```

Base * bobj $\Rightarrow$ NVF $\leftarrow$ yyy1

VF $\leftarrow$ unknown

bobj = new Base(...)

VF $\leftarrow$ xxx1;

bobj = new Derived(...);

VF $\leftarrow$ xxx2;

object's storage
attributes
non-virtual functions: locations of functions (**NVF**)
virtual functions: **pointers** to be assigned at run time (**VF**)

In general, all C++ methods should be virtual, unless you have strong reasons to make them not virtual.

All Java methods are virtual.

Calling Virtual Functions inside Overloaded Functions

C/C++ - inheritance/OverloadVirtual.cc - Eclipse SDK

OverloadVirtual.cc

```
#include <iostream>
using namespace std;
class Person {
public:
    virtual void print() {
        cout << "Person::print" << endl;
    }
};
class Employee : public Person {
public:
    void print() {
        cout << "Employee::print" << endl;
    }
};
void f1(Person * p) {
    cout << "f1(Person * p)" << endl;
    p-> print();
}
void f1(Employee * p) {
    cout << "f1(Employee * p)" << endl;
    p-> print();
}
int main() {
    Person * p1;
    p1 = new Employee;
    Employee * e1;
    e1 = new Employee;
    f1(p1);
```

Writable | Smart Insert | 4 : 8

C/C++ - inheritance/OverloadVirtual.cc - Eclipse SDK

File Edit Refactor Navigate Search Run Project Window Help

OverloadVirtual.cc

```
void f1(Person * p) {
    cout << "f1(Person * p)" << endl;
    p-> print();
}
void f1(Employee * p) {
    cout << "f1(Employee * p)" << endl;
    p-> print();
}
int main() {
    Person * p1;
    p1 = new Employee;
    Employee * e1;
    e1 = new Employee;
    f1(p1);
    f1(e1);
    delete p1;
    delete e1;
    return 0;
}
```

Console

<terminated> inheritance.exe [C/C++ Local Application] C:\yunglu\eclipse\workspace\inheritance\Debug\inheritance.exe (6/5/08 9:10 PM)

```
f1(Person * p)
Employee::print
f1(Employee * p)
Employee::print
```

Writable | Smart Insert | 21 : 17

ECE 462
Object-Oriented Programming using C++ and Java

Private and Protected Inheritance

Yung-Hsiang Lu
yunglu@purdue.edu

class Derived: public Base {	class Derived: protected Base {	class Derived: private Base {
interface + implementation	implementation	implementation
Base * bptr; bptr = new Derived ...	**not allowed**	**not allowed**
public member → public	public member → protected	public member → private
protected member → protected	protected member → protected	protected member → private
private member → inaccessible	same	same
override virtual function	same	same

Private Inheritance as Composition

[24] Inheritance -- private and protected inheritance, C++ FAQ Lite - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

http://www.parashift.com/c++-faq-lite/private-inheritance.html

[24] Inheritance — `private` and `protected` inheritance

(Part of [*C++ FAQ Lite*](), [Copyright © 1991-2006](), [Marshall Cline](), [`cline@parashift.com`]())

FAQs in section [24]:

[24.1] How do you express "`private` inheritance"?

When you use `: private` instead of `: public`. E.g.,

Done

[24] Inheritance -- private and protected inheritance, C++ FAQ Lite - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

http://www.parashift.com/c++-faq-lite/private-inheritance.html

[24.2] How are "`private` inheritance" and "composition" similar?

`private` inheritance is a syntactic variant of composition (AKA aggregation and/or has-a).

E.g., the "`Car` has-a `Engine`" relationship can be expressed using *simple composition*:

```
class Engine {
public:
  Engine(int numCylinders);
  void start();                  // Starts this Engine
};

class Car {
public:
  Car() : e_(8) { }              // Initializes this Car with 8 cylinders
  void start() { e_.start(); }   // Start this Car by starting its Engine
private:
  Engine e_;                     // Car has-a Engine
};
```

The "`Car` has-a `Engine`" relationship can also be expressed using *`private` inheritance*:

```
class Car : private Engine {    // Car has-a Engine
public:
  Car() : Engine(8) { }          // Initializes this Car with 8 cylinders
```

Done

[24] Inheritance -- private and protected inheritance, C++ FAQ Lite - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

http://www.parashift.com/c++-faq-lite/private-inheritance.html

```
class Car : private Engine {    // Car has-a Engine
public:
  Car() : Engine(8) { }         // Initializes this Car with 8 cylinders
  using Engine::start;          // Start this Car by starting its Engine
};
```

There are several similarities between these two variants:

- In both cases there is exactly one `Engine` member object contained in every `Car` object
- In neither case can users (outsiders) convert a `Car*` to an `Engine*`
- In both cases the `Car` class has a `start()` method that calls the `start()` method on the contained `Engine` object.

There are also several distinctions:

- The simple-composition variant is needed if you want to contain several `Engine`s per `Car`
- The `private`-inheritance variant can introduce unnecessary multiple inheritance
- The `private`-inheritance variant allows members of `Car` to convert a `Car*` to an `Engine*`
- The `private`-inheritance variant allows access to the `protected` members of the base class
- The `private`-inheritance variant allows `Car` to override `Engine`'s `virtual` functions
- The `private`-inheritance variant makes it *slightly* simpler (20 characters compared to 28 characters) to give `Car` a `start()` method that simply calls through to the `Engine`'s `start()` method

Done

Protected Inheritance as Role

C/C++ - privateinheritance/ImplementationInherit.cc - Eclipse SDK

File Edit Refactor Navigate Search Run Project Window Help

ImplementationInherit.cc

```cpp
//ImplementationInherit.cc

#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Employee {
    string firstName, lastName;
    int age, yearsInService;
public:
    Employee(string fnam, string lnam) :
        firstName(fnam), lastName(lnam) {
    }
    virtual void print() const {
        cout << firstName << " " << lastName << endl;
    }
    void sayEmployeeHello() {
        cout << "hello from Employee class" << endl;
    }
};

class ExecutiveRole {
public:
    void sayExecutiveHello() {
        cout << "Hello from Executive ranks" << endl;
    }
};
```

Writable | Smart Insert | 83 : 2 | C/C++ Indexer: (0%)

ImplementationInherit.cc

```cpp
//class Manager
//        : public Employee, private ExecutiveRole { // WILL NOT COMPILE
class Manager : public Employee, protected ExecutiveRole { // WORKS FINE
    short level;
public:
    Manager(string fnam, string lnam, short lvl) :
        Employee(fnam, lnam), level(lvl) {
        cout<< "In Manager constructor: ";
        sayEmployeeHello();
        sayExecutiveHello(); //(A)
    }
    void print() const {
        Employee::print();
        cout << "level: " << level << endl;
    }
};

class Director : public Manager {
    short grade;
public:
    Director(string fnam, string lnam, short lvl, short gd) :
        Manager(fnam, lnam, lvl), grade(gd) {
        cout << "In Director constructor: ";
        sayEmployeeHello();
        sayExecutiveHello(); //(B)
    }
    void print() const {
        Manager::print();
```

C/C++ - privateinheritance/ImplementationInherit.cc - Eclipse SDK

```cpp
    void print() const {
        Manager::print();
        cout << "grade: " << grade << endl << endl;
    }
};

int main() {
    vector<Employee*> empList;

    Employee* e1 = new Employee( "joe", "schmoe" );
    Employee* e2 = (Employee*) new Manager( "ms", "importante", 2 );
    Employee* e3 = (Employee*) new Director("mister", "bigshot", 3, 4); //(C)

    empList.push_back(e1);
    empList.push_back(e2);
    empList.push_back(e3);

    vector<Employee*>::iterator p = empList.begin();
    while (p < empList.end() )
        (*p++)->print();

    Manager* m = new Manager( "jane", "doe", 2 );
    m->sayEmployeeHello();

    Director* d = new Director( "john", "doe", 3, 4 );
    d->sayEmployeeHello();
    return 0;
}
```

Writable | Smart Insert | 83 : 2 | C/C++ Indexer

C/C++ - privateinheritance/ImplementationInherit.cc - Eclipse SDK

File Edit Refactor Navigate Search Run Project Window Help

ImplementationInherit.cc

```
public:
    Director(string fnam, string lnam, short lvl, short gd) :
        Manager(fnam, lnam, lvl), grade(gd) {
```

Console

<terminated> privateinheritance Debug [C/C++ Local Application] C:\yunglu\eclipse\workspace\privateinheritance\Debug\privateinheritance.exe (6/15/08 4:13 PM

```
In Manager constructor: hello from Employee class
Hello from Executive ranks
In Manager constructor: hello from Employee class
Hello from Executive ranks
In Director constructor: hello from Employee class
Hello from Executive ranks
joe schmoe
ms importante
level: 2
mister bigshot
level: 3
grade: 4

In Manager constructor: hello from Employee class
Hello from Executive ranks
hello from Employee class
In Manager constructor: hello from Employee class
Hello from Executive ranks
In Director constructor: hello from Employee class
Hello from Executive ranks
hello from Employee class
```

C/C++ Indexer

ECE 462
Object-Oriented Programming using C++ and Java

Java Final and Finally

Yung-Hsiang Lu
yunglu@purdue.edu

Java "final"

create a constant: (15.18)

```
interface A { public static final double PI = 3.14159; }
class X {
   double x = A.PI
}
```

Overview Package **Class** Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 6

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.lang

Class Math

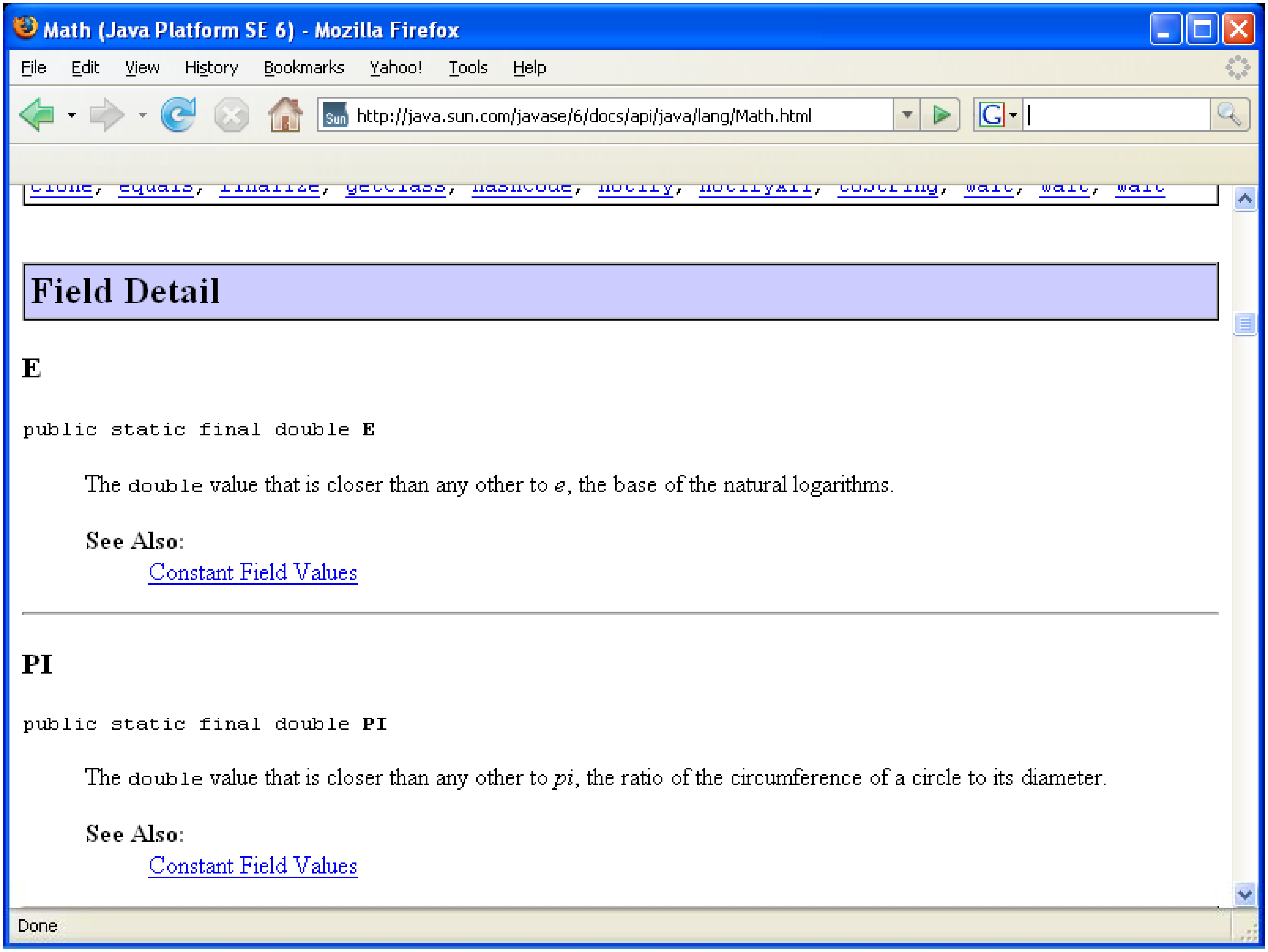
`java.lang.Object`
`└java.lang.Math`

```
public final class Math
extends Object
```

The class `Math` contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

Unlike some of the numeric methods of class `StrictMath`, all implementations of the equivalent functions of class `Math` are not defined to return the bit-for-bit same results. This relaxation permits better-performing implementations where strict reproducibility is not required.

By default many of the `Math` methods simply call the equivalent method in `StrictMath` for their implementation. Code generators are encouraged to use platform-specific native libraries or microprocessor instructions, where available, to provide



Math (Java Platform SE 6) - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

http://java.sun.com/javase/6/docs/api/java/lang/Math.html

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

E

```
public static final double E
```

The `double` value that is closer than any other to *e*, the base of the natural logarithms.

See Also:
Constant Field Values

PI

```
public static final double PI
```

The `double` value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

See Also:
Constant Field Values

Done

constant parameter: (9.8)

```
void foo (final int x) {
    x = 100;     // error
}
```

block inheritance (class): (3.6)

```
final class X {
}
class Y extends X {  // error
}
```

block overriding (method): (3.6)

```
class X {
   final public void foo () { }
}

class Y extends X {
   public void foo () { } // error
}
```

Java "finally"

exception handling: (10.5)

```
try { ...
} catch (exception_type1 ide1) { ...
} catch (exception_type2 ide2) { ...
} finally { ...
   // always execute, regardless of exceptions
}
```

Java “finalize”

Called before an object’s memory is reclaimed by the garbage collector: (11.9)

```
//GC.java
class X {
   int id;
   static int nextId = 1;
   public X() { id = nextId++; }
   protected void finalize() throws Throwable {
      if ( id%1000 == 0 )
          { System.out.println("Finalization of X object, id = " + id); }
      super.finalize();
   }
}
```

```
class Test {
    public static void main( String[] args ) {
        X[] xarray = new X[ 10000 ];
        for (int i = 0; i < 10000; i++ )
            xarray[i] = new X();
        xarray = null;
        System.gc();
    }
}
```