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

Lecture 3

Yung-Hsiang Lu

yunglu@purdue.edu

Mini Review

concepts explained so far:

- object: identity, states (attributes), behavior (methods)
- class: interface and implementation
- inheritance: code reuse
- polymorphism: different derived classes have different behavior
- encapsulation: hide information inside objects

Execute C++ and Java Programs

- C++
  - Linux: g++
  - Windows: g++ in cygwin
- Java
  - Linux: javac to compile and java to execute or Netbeans to compile and execute
  - Windows: javac to compile and java to execute or Netbeans to compile and execute

```
//AddArray.java
public class AddArray {                                    //(A)
    public static void main( String[] args )               //(B)
    {
        int[] data = { 0, 1, 2, 3, 4, 5, 9, 8, 7, 6 };     //(C)
        System.out.println( "The sum is: "                 //(D)
                            + addArray(data) );
    }
    public static int addArray( int[] a ) {                //(E)
        int sum = 0;
        for ( int i=0; i < a.length; i++ )   sum += a[i];
        return sum;
    }
}
```

similar to (int argc, char * argv[]) in C

similar to printf in C

```
//Polymorph.java
class User {
    private String name;
    private int age;
    public User( String str, int yy ) { name = str; age = yy; }
    public void print() {
        System.out.print( "name: " + name + "   age: " + age );
    }
}                                                   // ; not needed for Java
class StudentUser extends User {                    // derived class
    private String schoolEnrolled;
    public StudentUser( String nam, int y, String sch ) {
        super(nam, y);                  // User::print(); in C++
        schoolEnrolled = sch;
    }
      public void print() {
        super.print();
        System.out.print( "   School: " + schoolEnrolled );
    }
}
```

```
class Test {
    public static void main( String[] args )
    {
        User[] users = new User[3];
        users[0] = new User( "Buster Dandy", 34 );
        users[1] = new StudentUser("Missy Showoff",25,"Math");
        users[2] = new User( "Mister Meister", 28 );
        for (int i=0; i<3; i++) {
            users[i].print(); System.out.println();
        }
    }
}
```

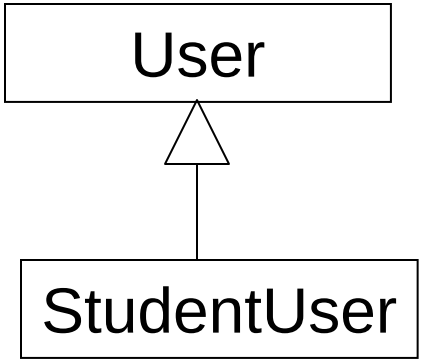
Encapsulation

- users[2].name ⇒ compile-time error
- users[2].age ⇒ compile-time error
- Information is available only if an object allows the visibility; visibility is specified in the class interface.

- Inheritance:
  - A derived class has all the public and protected properties (= attributes + methods) of the base class. Private properties are not inherited.
  - A pointer for a base class can point to an object of a derived class.
  - A pointer for a derived class can **not** point to an object of a base class.
  - A StudentUser is a User, incorrect the other way
- Polymorphism:
  - A pointer's behavior changes based on the object being pointed to.
  - determined at run-time

Polymorphism

- users[1] = new StudentUser("Missy Showoff",25,"Math");
  - behavior as a StudentUser
  - users[1].print will include the school
- users[1] = new User("Missy Showoff",25);
  - behavior as a User
  - users[1].print will not include the school



Self Test

What concept is used? Choose your answers from

1. encapsulation
2. inheritance
3. polymorphism

Questions:

- a StudentUser object can call super.print();
- a StudentUser object can be assigned to (RHS) to a User object
- in StudentUser's print, accessing name is an error
- users[0].print() and users[1].print() behave differently

Self Test

What concept is used? Choose your answers from

1. encapsulation
2. inheritance
3. polymorphism

Questions:

- a StudentUser object can call super.print(); $\Rightarrow$ 2
- a StudentUser object can be assigned to (RHS) to a User object $\Rightarrow$ 2
- in StudentUser's print, accessing name is an error $\Rightarrow$ 1
- users[0].print() and users[1].print() behave differently $\Rightarrow$ 3

```
class User {
    private String name;
    private int age;
    public User( String str, int yy ) { name = str; age = yy; }
    public void setAge(int newage) {
            // an example of information consistency
            if (age > newage)
                    { System.out.println("You can't become younger."); }
            else
            { age = newage; }
    }
    public void print() {
       System.out.print( "name: " + name + "   age: " + age );
    }
}
```

In general, it is a bad programming habit to provide "setXXX" methods, worse to make them public.

```
// ---------------------- User1.cc
#include <iostream>
#include <string>
using namespace std;
class User {
    string name;  // default visibility is private
    int age;
public:
    User( string str, int yy ) { name = str;  age = yy; }        // constructor
     void print() { cout << "name: " << name << " age: "  << age << endl; }
}; // ; not needed for Java
int main()
{
    User u( "Zaphod", 119 );        // create an object
    u.print();
    return 0;
}
```

comment

constructor: a method with the same name as the class

```
// ---------------------- User2.cc
#include <iostream>
#include <string>
using namespace std;
class User {
    string name;
    int age;
public:
    User( string str, int yy );
    void print();
};
User::User( string str, int yy ) {  // this does not have to in the same file
    name = str;  age = yy;
}
void User::print() {
    cout << "name: " << name  << "  age: "  << age << endl;
}
int main()
{
    User u( "Zaphod", 119 );
    u.print();
    return 0;
}
```

```
//  ---------------------- Polymorph.cc
#include <iostream>
#include <string>
using namespace std;
class User {
    string name;
    int age;
public:
    User(string nm, int a) {name=nm; age=a;}
    virtual void print() {   // virtual explained later
        cout << "Name: " << name << "  Age: " << age;
    }
};
```

```
class StudentUser : public User {
    string schoolEnrolled;                    // new attribute
public:
    StudentUser(string nam, int y, string school) : User(nam, y){
        schoolEnrolled = school;
    }
    void print() {
        User::print();        // print a user's attribute
        cout << "  School Enrolled:  " << schoolEnrolled;
    }
};
```

```
int main()
{
    User* users[3];
    users[0] = new User( "Buster Dandy", 34 );
    users[1] = new StudentUser("Missy Showoff", 25, "Math");
            // a StudentUser is a User
    users[2] = new User( "Mister Meister", 28 );
    for (int i=0; i<3; i++) {
        users[i]->print();
        cout << endl;
    }
    // this program has a memory leak; ignore it for now
    return 0;
}
```

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

Lecture 4

Yung-Hsiang Lu

yunglu@purdue.edu

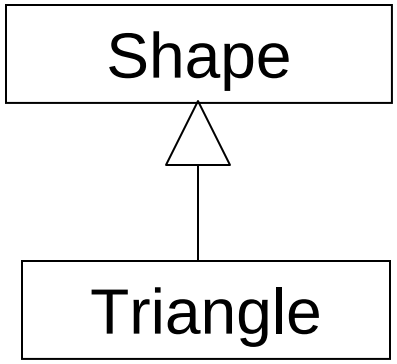
Self Test

- Java:

```
class Shape {    ...
}
class Triangle _____ Shape {   ...
}
```

- C++

```
class Shape {    ...
};
class Triangle _____ Shape {   ...
};
```



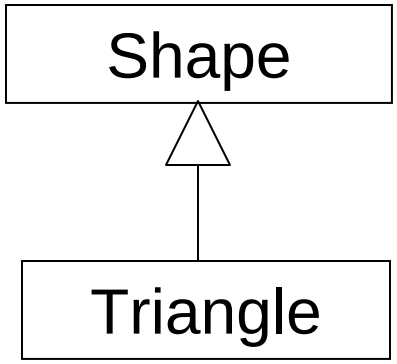
Self Test

- Java:

```
class Shape {    ...
}
class Triangle extends Shape { ...
}
```

- C++

```
class Shape {    ...
};
class Triangle: public Shape {   ...
};
```



Self Test

How to Create Objects

C++

```
class Human {
private:
    int h_height, h_weight;
public:
     Human(int height, int weight);
};
```

Java

```
class Human {
   private int h_height;
   private in h_weight;
   public Human(int height,
            int weight);
};
```

Self Test

How to Create Objects

C++

```
class Human {
private:
    int h_height, h_weight;
    // cm, kg
public:
     Human(int height, int weight);
};
```

```
Human hobj1(172, 69);
Human * hobj2 = new Human
    (181, 90);
```

Java

```
class Human {
   private int h_height;
   private in h_weight;
   public Human(int height,
          int weight);
}
```

*** in C++, no * in Java**

```
Human hobj = new Human(169,
    61);
```

another way in C++

```
Human * hobj3;
......
hobj3 = new Human(166, 58);
```

```
//  ---------------------- Polymorph.cc
#include <iostream>
#include <string>
using namespace std;
class User {
    string name;
    int age;
public:
    User(string nm, int a) {name=nm; age=a;}
    virtual void print() {        // all methods are virtual in Java, not in C++
        cout << "Name: " << name << "  Age: " << age;
    }
};
// In C++, polymorphism must be specified by adding virtual in front
// of a function.
```

```
class StudentUser : public User {
    string schoolEnrolled;                  // new attribute
public:
    StudentUser(string nam, int y, string school) : User(nam, y){
        schoolEnrolled = school;
    }
    void print() {
        User::print();        // print a user's attribute
        cout << "  School Enrolled:  " << schoolEnrolled;
    }
};
```

```
int main()
{
    User* users[3];
    users[0] = new User( "Buster Dandy", 34 );
    users[1] = new StudentUser("Missy Showoff", 25, "Math");
            // a StudentUser is a User
    users[2] = new User( "Mister Meister", 28 );
    for (int i=0; i<3; i++) {
        users[i]->print();
        cout << endl;
    }
    // this program has a memory leak; ignore it for now
    return 0;
}
```

User user[3] ⇒ error

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

Virtual ⇒ Polymorphism

```
class Base {
public:
    virtual void xxx() { /* xxx1 */ }
    void yyy() { /* yyy 1 */    };
class Derived: public Base {
public:
    void xxx() { /* xxx2 */ }
    virtual void yyy() { /* yyy2 */ }   };
class Derived2: public Derived {
public:
    void xxx() { /* xxx3 */ }
    void yyy() { /* yyy3 */ }    };
```

```
Base * bobj = new Base(...);
bobj -> xxx(); /* xxx1 */
delete bobj; // release memory
bobj = new Derived(...);
bobj -> xxx(); /* xxx2 */
Derived * dobj = new Base (...)
⇒ Error
Derived * dobj = new Derived(...);
dobj -> xxx(); /* xxx2 */
bobj = dobj;
bobj -> yyy() /* yyy1 */
dobj = new Derived2(...);
dobj -> yyy() /* yyy 3 */
```

How Does virtual Work?

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

Base * bobj ⇒ NVF ← yyy1

VF ← unknown

bobj = new Base(...)

VF ← xxx1;

bobj = new Derived(...);

VF ← 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.

Demonstration of virtual functions in C++

Lab 2: Java Draw Program Menu and Graphics

Assignment: To Reproduce the Same Program Shown in the Video

Today's lecture is 56 minutes.

Next lecture will be shorter.