

Announcements

Inheritance: a simple first example

-
-

```
class sphere {  
public:  
    sphere();  
    sphere(double r);  
    double getVolume();  
    void setRadius(double r);  
    void display();  
  
private:  
    double theRadius;  
  
};
```

```
class ball:public sphere {  
public:  
    ball();  
    ball(double r string n);  
    string getName();  
    void setName(string n);  
    void display();  
  
private:  
    string name;  
};
```

inheritance rules:

-
-
-

Subclass substitution (via examples):

```
void printVolume(sphere t){  
    cout << t.getVolume() << endl;}  
  
int main() {  
    sphere s(8.0);  
    ball b(3.2, "pompon");  
  
    double a = b.getVolume();  
  
    printVolume(s);  
    printVolume(b);  
}
```

```
Base b;  
Derived d;  
  
b=d;  
  
d=b;
```

```
Base * b;  
Derived * d;  
  
b=d;  
  
d=b;
```

something to consider:

```
class sphere {  
public:  
    sphere();  
    sphere(double r);  
    ...  
};
```

```
class ball:public sphere {  
public:  
    ball();  
    ball(double r string n);  
    ...  
};
```

```
void sphere::display() {  
    cout << "sphere" << endl;  
}
```

```
void ball::display() {  
    cout << "ball" << endl;  
}
```

void display();

void display();

```
private:  
    double theRadius;  
};
```

```
private:  
    string name;  
};
```

ex1

```
sphere s;  
ball b;  
s.display();  
b.display();
```

ex2

```
sphere * sptr;  
sptr = &s;  
sptr->display();
```

ex3

```
sphere * sptr;  
sptr = &b;  
sptr->display();
```

“virtual” functions:

```
class sphere {  
public:  
    sphere();  
    sphere(double r);  
    ...  
};
```

```
class ball:public sphere {  
public:  
    ball();  
    ball(double r string n);  
    string getName();  
};
```

```
void sphere::display() {  
    cout << "sphere" << endl;  
}
```

```
void ball::display() {  
    cout << "ball" << endl;  
}
```

void display();

void display();

```
private:  
    double theRadius;  
};
```

```
private:  
    string name;  
};
```

ex4

```
if (a==0)  
    sptr = &s;  
else sptr = &b;  
sptr->display();
```

virtual functions – the rules:

A virtual method is one a _____ can override.

A class's virtual methods _____ be implemented. If not, then the class is an “abstract base class” and no objects of that type can be declared.

A derived class is not *required* to override an existing implementation of an _____ virtual method.

Constructors _____ be virtual

Destructors can and _____ virtual

Virtual method return type _____ be overwritten.

Constructors for derived class:

```
ball::ball():sphere()  
{  
    name = "not known";  
}
```

```
ball b;
```

```
ball::ball(double r, string n):  
sphere(r)  
{  
    name = n;  
}
```

```
ball b(0.5,"grape");
```

“virtual” destructors:

```
class Base{
public:
    Base(){cout<<"Ctor: B"<<endl;}
    ~Base(){cout<<"Dtor: B"<<endl;}
};

class Derived: public Base{
public:
    Derived(){cout<<"Ctor: D"<<endl;}
    ~Derived(){cout<<"Dtor: D"<<endl;}
};
```

```
void main(){
    Base * V = new Derived();
    delete V;
}
```

Abstract Base Classes:

```
class flower {  
public:  
    flower();  
    virtual void drawBlossom() = 0;  
    virtual void drawStem() = 0;  
    virtual void drawFoliage() = 0;  
    ...  
};
```

```
void daisy::drawBlossom() {  
    // whatever  
}  
void daisy::drawStem() {  
    // whatever  
}  
void daisy::drawFoliage() {  
    // whatever  
}
```

```
class daisy:public flower {  
public:  
    virtual void drawBlossom();  
    virtual void drawStem();  
    virtual void drawFoliage();  
    ...  
private:  
    int blossom; // number of petals  
    int stem; // length of stem  
    int foliage // leaves per inch  
};
```

```
flower f;  
daisy d;  
flower * fptr;
```