Classes and Objects

March 6, 2002

CS 131: Programming Languages

Classes

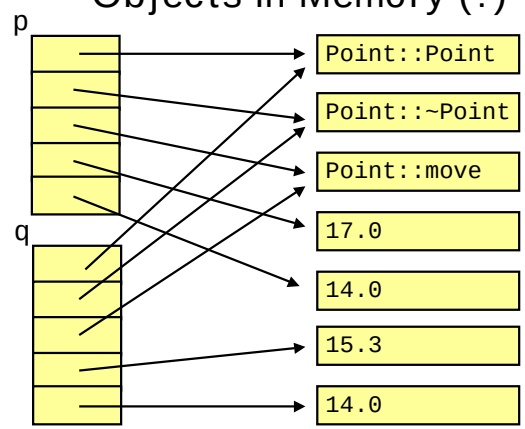
- Classes in C++ and Java provide
  - Ability to create objects
  - Repository for related code (`static`)
  - Access control (`public`/`private`)
  - Code reuse via inheritance
  - Abstract types
  - Subtyping

C++ Object Representation

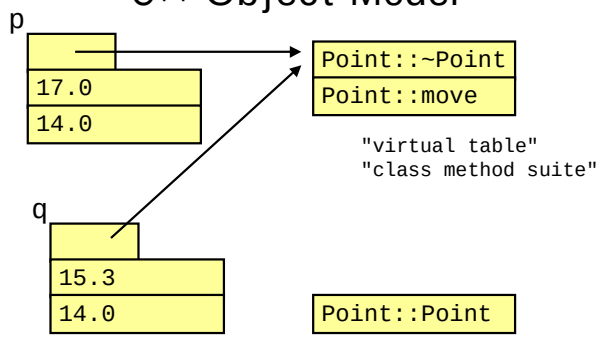
```
class Point {
  public:
     Point (double,double);
     virtual ~Point;
     virtual void move(double,double);
     double x;
     double y;
};

p = new Point(17.0, 14.0);
q = new Point(15.3, 14.0);
```

Objects in Memory (?)



C++ Object Model

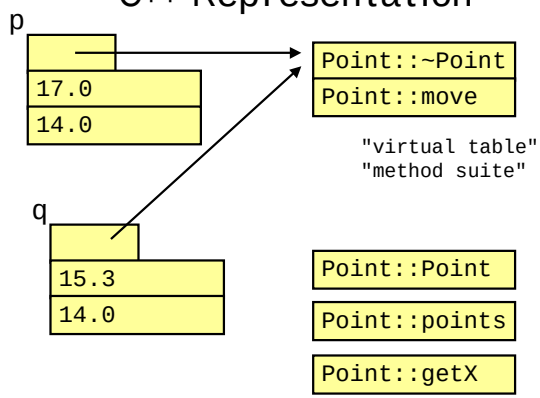


Other Class Components

```
class Point2 {
  public:
     Point (double,double);
     virtual ~Point;
     virtual void move(double,double);
     double x,y;
     static int points;
     int getX();
};

Point2* p = new Point2(17.0, 14.0);
Point2* q = new Point2(15.3, 14.0);
```

C++ Representation



Comments on C++

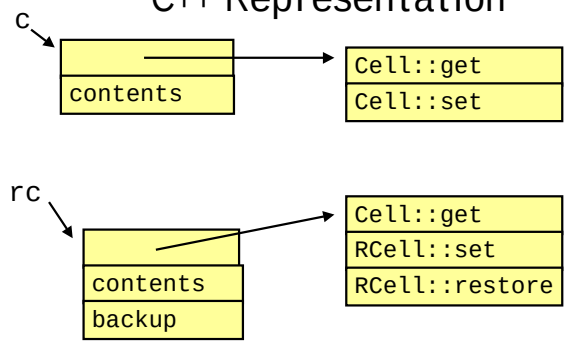
- Object layout doesn't care about `public / private / protected`
  - These are solely a matter of scoping and typechecking
- Interface of a class uniquely determines the class layout.
  - e.g., byte offsets of fields or methods
  - This is why you have to list all the private components of the object.
  - Contributes to "fragile base class" problem.

Objects and Subclasses

```
class Cell {
   int contents;
   int get()       { return this.contents; }
   void set(int x) { this.contents = x; }
};

class RCell : public Cell {
   int backup = 0;
   void restore    { this.contents = backup }
   void set(int x) { backup = this.contents;
                     this.contents = x; }
};
```

C++ Representation



Sample Code

```
Cell *c   = new Cell;
RCell *rc = new RCell;
Cell *c2  = new Cell;

c->set(0);
rc->set(1);
c2->set(2);
c->set(rc->contents);
```

Overloading

```
class Cell {
 int contents;
 int get()       { return this.contents; }
 void set(int x) { this.contents = x; }
};

class RCell : public Cell {
 int backup = 0;
 void restore    { this.contents = backup }
 void set()      { backup = this.contents;
                   this.contents = 0; }
};
```

C++ Representation

Sample Code

```
Cell *c   = new Cell;
RCell *rc = new RCell;
Cell *c2  = new Cell;

c->set(0);
rc->set(1);
c2->set(2);
c2->set();
rc->set();
```

Combined

```
class Cell {
 int contents;
 int get()       { return contents; }
 void set(int x) { contents = x; }
 void set()      { contents = 0; }
};

class RCell : public Cell {
 int backup = 0;
 void restore    { contents = backup }
 void set(int x) { backup = contents;
                   contents = x; }
};
```

C++ Representation

Sample Code

```
Cell *c   = new Cell;
RCell *rc = new RCell;
Cell *c2  = new Cell;

c->set(0);
rc->set(1);
c2->set(2);
c->set(rc->contents);
```

Object-*Based* Languages

- Languages with objects don't necessarily require classes
  - Such languages are often called object-*based*
  - Example: ECMAScript (a.k.a. Javascript, ActionScript) `http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM`
- Where do objects come from, without classes?
  - Where do tuples/records come from in SML?
  - Many object-based languages are organized around "prototypes"
    - Create new objects by copying&modifying or by referring to other objects

ECMAScript Examples

```
var ob = {a : 3,
          b : 4}

ob.c = 5

var sum = ob.a
        + ob.b
        + ob.c
```

```
function cell_get() {
   return this.contents;
}
function cell_set(n) {
   this.contents = n;
}

var mycell = new Object()
mycell.contents = 0
mycell.get = cell_get
mycell.set = cell_set
```

Constructor Functions

```
function Cell {
   this.contents = 0;
   this.get = cell_get;
   this.set = cell_set;
};

var mycell = new Cell();
```

Delegation

- Subclassing without classes
- Each object internally references another object, called the prototype
  - If we fail to find a field or method in an object, try looking in the object's parent, the parent's parent, etc.
  - Efficiency: many objects can share the same parent, so they don't have to each have a copy of the parent's fields and methods.
  - Adding methods to the parent causes new code to show up in all the child objects too. (Even built-in objects like strings!)

Delegation Example

```
var str1 = new String("a")
var str2 = new String("b")

function double() {
   return (this.toString() +
           this.toString)
}
String.prototype.double = double

alert(str1.double + str2.double)
```