Object-Oriented Programming (OOP)
Lecture No. 30

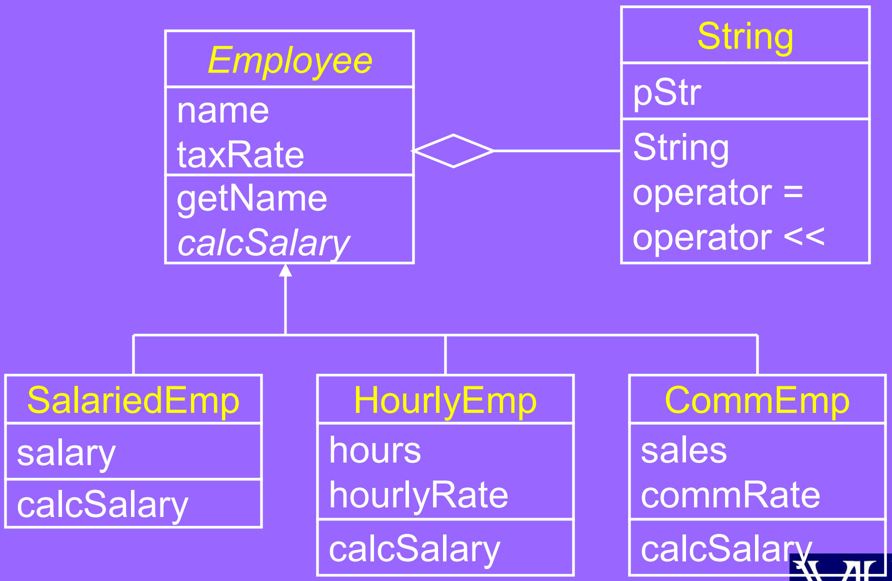
Polymorphism – Case Study

A Simple Payroll Application

Problem Statement

- Develop a simple payroll application. There are three kinds of employees in the system: salaried employee, hourly employee, and commissioned employee. The system takes as input an array containing employee objects, calculates salary polymorphically, and generates report.

OO Model



Class *Employee*

```
class Employee {
  private:
     String name;
     double taxRate;
  public:
     Employee( String&, double );
     String getName();
     virtual double calcSalary() = 0;
}
```

… Class *Employee*

```
Employee::Employee( String& n,
         double tr ): name(n){
 taxRate = tr;
}

String Employee::getName() {
  return name;
}
```

Class SalariedEmp

```
class SalariedEmp : public Employee
{
private:
     double salary;
public:
  SalariedEmp(String&,double,double);
  virtual double calcSalary();
}
```

… Class SalariedEmp

```
SalariedEmp::SalariedEmp(String& n,
               double tr, double sal)
               : Employee( n, tr ) {
  salary = sal;
}

double SalariedEmp::calcSalary() {
  double tax = salary * taxRate;
  return salary - tax;
}
```

Class HourlyEmp

```
class HourlyEmp : public Employee {
private:
  int hours;
  double hourlyRate;
public:
  HourlyEmp(string&,double,int,double);
  virtual double calcSalary();
}
```

… Class HourlyEmp

```
HourlyEmp ::HourlyEmp( String& n,
     double tr, int h, double hr )
                : Employee( n, tr ) {
  hours = h;
  hourlyRate = hr;
}
```

… Class HourlyEmp

```
double HourlyEmp::calcSalary()
{
  double grossPay, tax;

  grossPay = hours * hourlyRate;
  tax = grossPay * taxRate;

  return grossPay – tax;
}
```

Class CommEmp

```
class CommEmp : public Employee
{
private:
  double sales;
  double commRate;
public:
  CommEmp( String&, double, double,
                              double );
  virtual double calcSalary();
}
```

… Class CommEmp

```
CommEmp::CommEmp( String& n,
     double tr, double s, double cr )
     : Employee( n, tr ) {
  sales = s;
  commRate = cr;
}
```

… Class CommEmp

```
double CommEmp::calcSalary()
{
  double grossPay = sales * commRate;
  double tax = grossPay * taxRate;

  return grossPay – tax;
}
```

A Sample Payroll

```
int main() {
  Employee* emp[10];
  emp[0] = new SalariedEmp( "Aamir",
                    0.05, 15000 );
  emp[1] = new HourlyEmp( "Faakhir",
                    0.06, 160, 50 );
  emp[2] = new CommEmp( "Fuaad",
               0.04, 150000, 10 );
  …
  generatePayroll( emp, 10 );
  return 0;
}
```

…A Sample Payroll

```
void generatePayroll(Employee* emp[],
                               int size) {

  cout << "Name\tNet Salary\n\n";

  for (int i = 0; i < size; i++) {
     cout << emp[i]->getName() << '\t'
           << emp[i]->calcSalary()
           << '\n';
  }
}
```

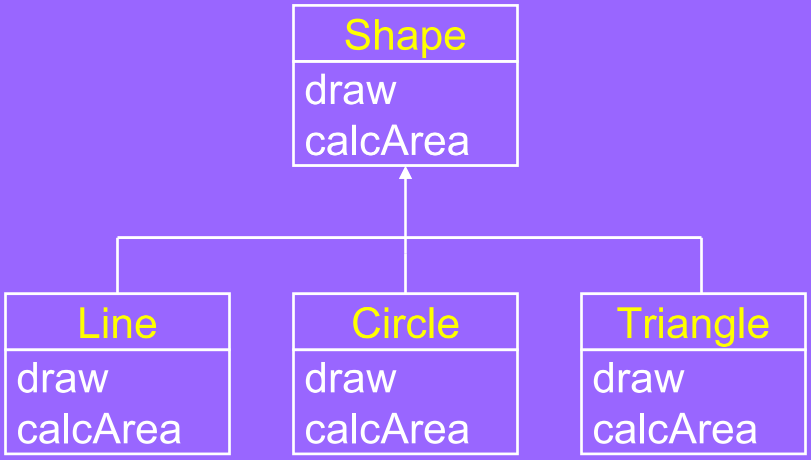
Sample Output

```
Name          Net Salary

Aamir         14250
Fakhir        7520
Fuaad         14400
…
```

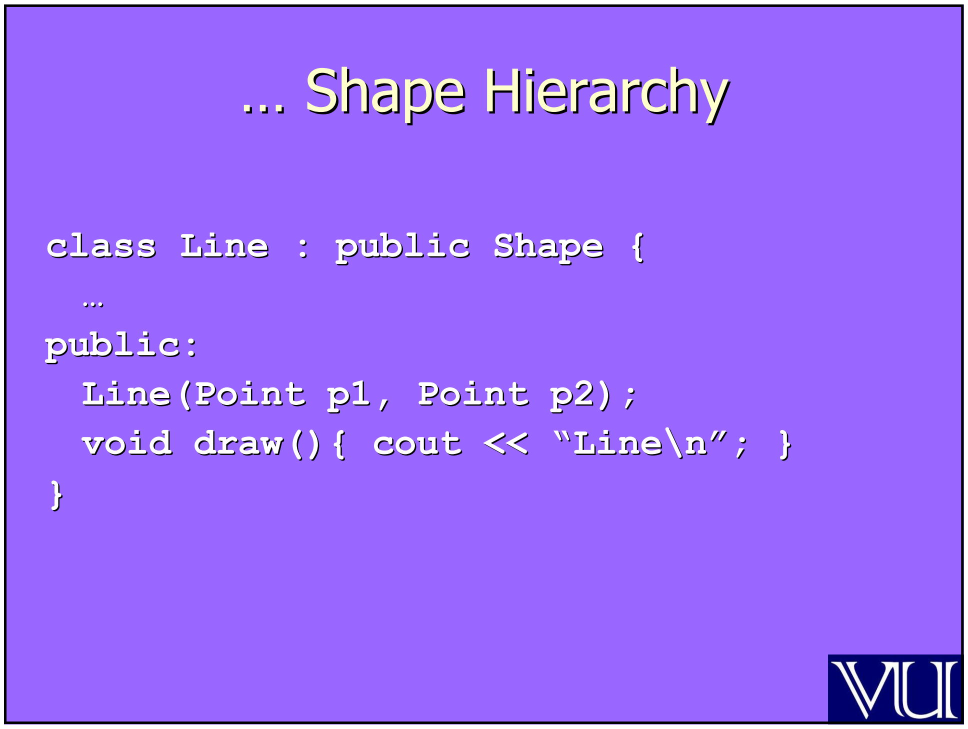
Never Treat Arrays Polymorphically

Shape Hierarchy Revisited



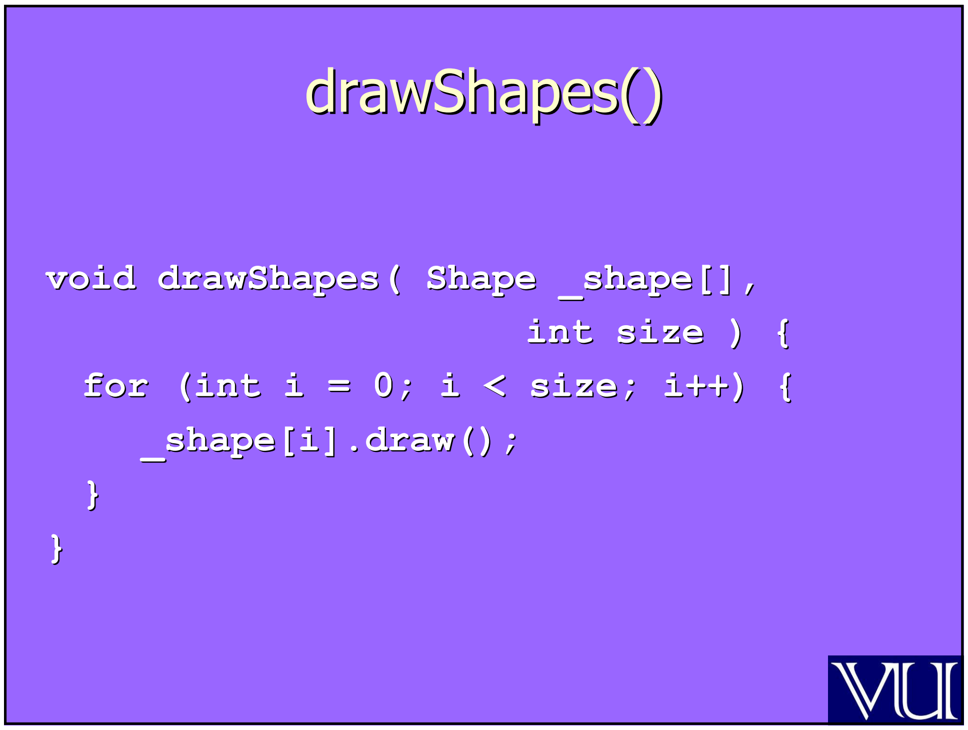
Shape Hierarchy

```
class Shape {
  …
public:
  Shape();
  virtual void draw(){
     cout << “Shape\n”;
  }
  virtual int calcArea() { return 0; }
};
```



… Shape Hierarchy

```
class Line : public Shape {
  …
public:
  Line(Point p1, Point p2);
  void draw(){ cout << "Line\n"; }
}
```



drawShapes()

```
void drawShapes( Shape _shape[],
                          int size ) {
  for (int i = 0; i < size; i++) {
     _shape[i].draw();
  }
}
```

Polymorphism & Arrays

```
int main() {
  Shape _shape[ 10 ];
  _shape[ 0 ] = Shape();
  _shape[ 1 ] = Shape();
  …
  drawShapes( _shape, 10 );
  return 0;
}
```

Sample Output

```
Shape
Shape
Shape
…
```

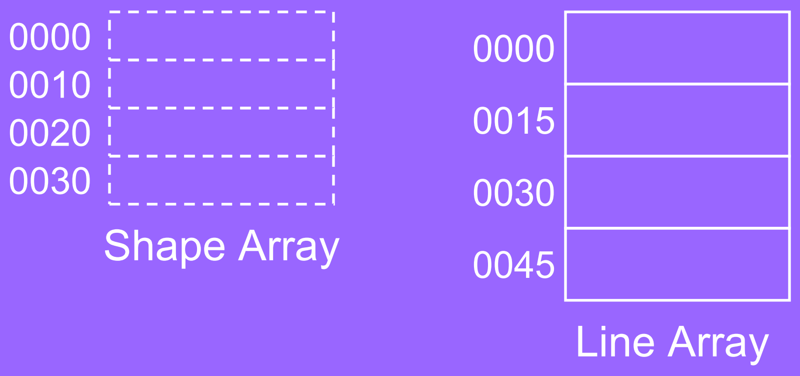
…Polymorphism & Arrays

```
int main() {
  Point p1(10, 10), p2(20, 20), …
  Line _line[ 10 ];
  _line[ 0 ] = Line( p1, p2 );
  _line[ 1 ] = Line( p3, p4 );
  …
  drawShapes( _line, 10 );
  return 0;
}
```

Sample Output

```
Shape
// Run-time error
```

Because



```
_shape[ i ].draw();
*(_shape + (i * sizeof(Shape))).draw();
```

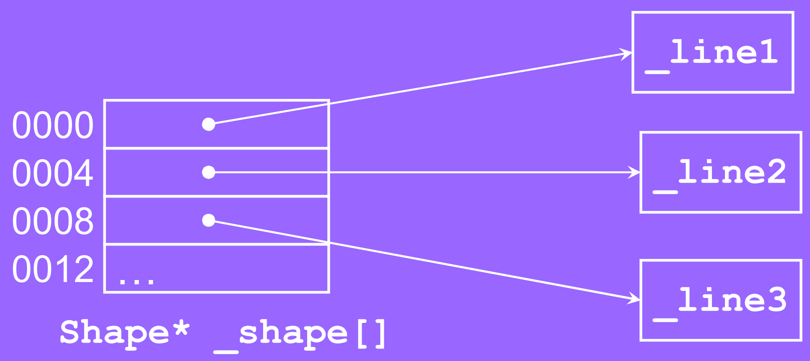
Original drawShapes()

```
void drawShapes(Shape* _shape[],
                          int size) {
  for (int i = 0; i < size; i++) {
     _shape[i]->draw();
  }
}
```

Sample Output

```
Line
Line
Line
…
```

Because



```
_shape[i]->draw();
(_shape + (i * sizeof(Shape*)))->draw();
```