POLYMORPHISM IN DEPTH

Run Time Binding

Late Binding

Dynamic Binding

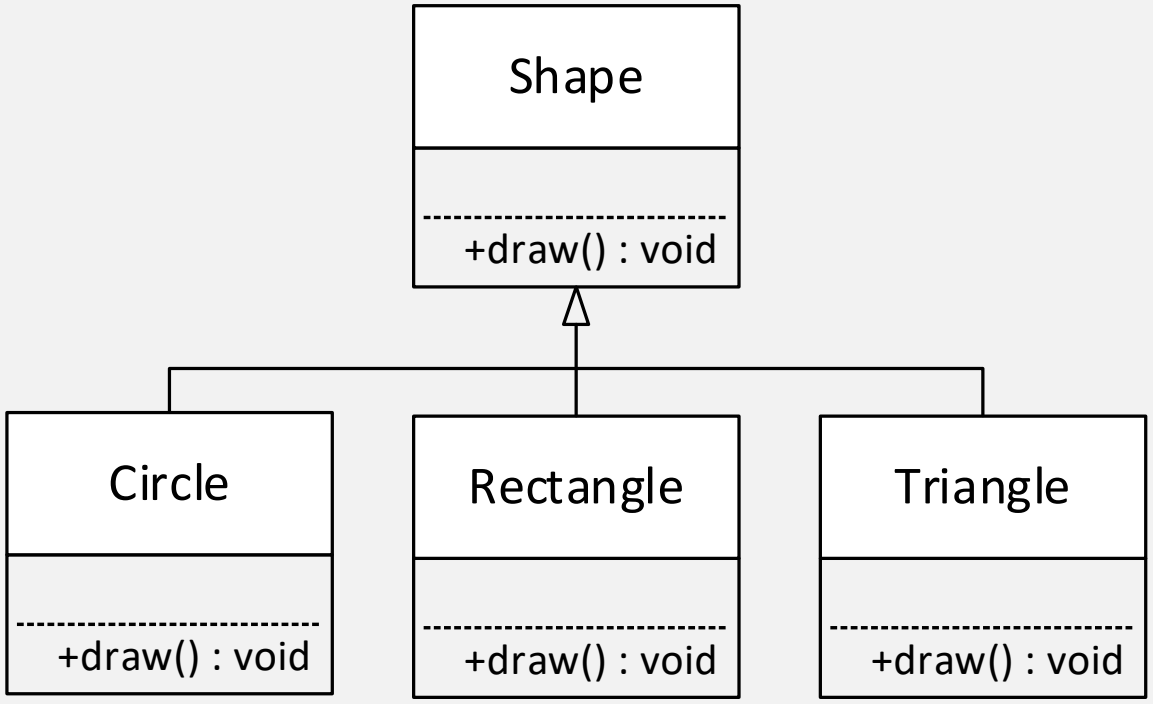
Dynamic Dispatch

Delroy A. Brinkerhoff

REQUIREMENTS

- Inheritance
- Function overriding
- Upcasting
- Pointer or reference variable
- “virtual” function

INHERITANCE AND OVERRIDDEN FUNCTIONS



“virtual” ENABLES POLYMORPHISM

```
class Shape
{
    public:
        virtual void draw();
};

class Circle : public Shape
{
    public:
        virtual void draw();
};
```

```
class Rectangle : public Shape
{
    public:
        virtual void draw();
};

class Triangle : public Shape
{
    public:
        virtual void draw();
};
```

UPCASTING WITH POINTERS

ASSIGNMENT

- `Shape* s = new Circle(...);`

FUNCTION CALL

- `void render(Shape* s) {...}`
- `render(new Circle(...));`

UPCASTING WITH POINTERS

ASSIGNMENT

- `Shape* s = new Circle(...);`

FUNCTION CALL

- `void render(Shape* s) {...}`
- `render(new Circle(...));`

```
s->draw();
```

when polymorphism is inactive (off)

UPCASTING WITH POINTERS

ASSIGNMENT

- `Shape* s = new Circle(...);`

FUNCTION CALL

- `void render(Shape* s) {...}`
- `render(new Circle(...));`

```
s->draw();
```

when polymorphism is active (on)

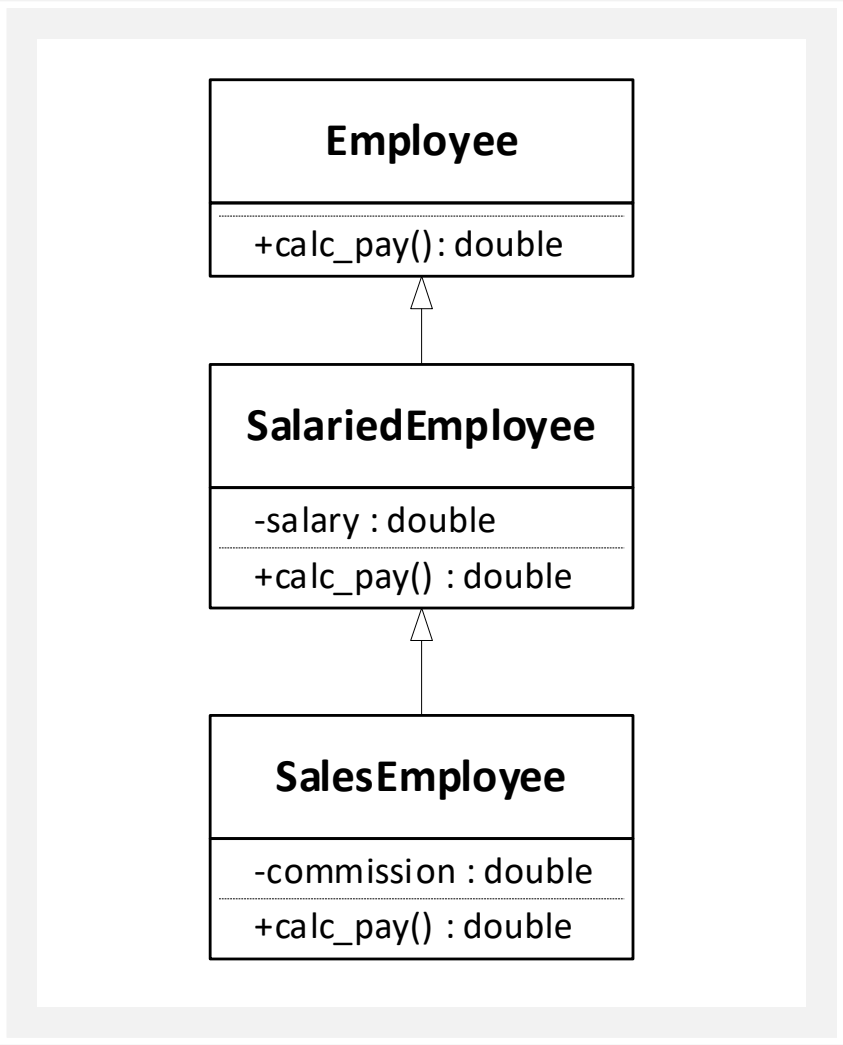
USING POLYMORPHISM

```
void render(Shape* s)
{
        . . .
    s->draw();
        . . .
}
```

- In object speak, calling a function is often referred to as sending a message.
- Send object s the “draw” message
- Polymorphism:
  - An object responds appropriately to a message for the kind of object that it is
  - Function binding takes place at the time of the function call – binds to the function belonging to the object

ACCESSING MEMBER VARIABLES WITHOUT DOWNCASTING

- SalariedEmployee has a private member named salary
  - pay = salary / 24
- SalesEmployee
  - Inherits salary
  - Has a private member named commission
  - pay = salary / 24 + commission
  - But a SalesEmployee object cannot access salary!



USING PUBLIC FUNCTIONS TO ACCESS PRIVATE DATA

SalariedEmployee

```
double calc_pay()
{
    return salary / 24;
}
```

SalesEmployee

```
double calc_pay()
{
    return SalariedEmployee::calc_pay()
       + commission;
}
```

POLYMORPHISM REDUCES THE NEED TO DOWNCAST

SalariedEmployee

```
virtual double calc_pay()
{
    return salary / 24;
}
```

SalesEmployee

```
virtual double calc_pay()
{
    return SalariedEmployee::calc_pay()
       + commission;
}
```

```
Employee* e = new SalesEmployee(...);
double pay = e->calc_pay();
```

QUIZ: GROUP 1

```
class Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
                    void funcC() {...}
};

class Child : public Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
};
```

```
Parent*  P1 = new  Parent;


P1->funcA();  // (a)_______

P1->funcB();  // (b)_______

P1->funcC();  // (c)_______
```

QUIZ: GROUP 1

```
class Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
                    void funcC() {...}
};

class Child : public Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
};
```

```
Parent*  P1 = new  Parent;


P1->funcA();  // (a) Parent

P1->funcB();  // (b) Parent

P1->funcC();  // (c) Parent
```

QUIZ: GROUP 2

```
class Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
                    void funcC() {...}
};

class Child : public Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
};
```

```
Parent*  P2 = new  Child;


P2->funcA();  // (d)_______

P2->funcB();  // (e)_______

P2->funcC();  // (f)_______
```

QUIZ: GROUP 2

```
class Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
                    void funcC() {...}
};

class Child : public Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
};
```

```
Parent*  P2 = new  Child;


P2->funcA();  // (d) Parent

P2->funcB();  // (e) Child

P2->funcC();  // (f) Parent
```

QUIZ: GROUP 3

```
class Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
                    void funcC() {...}
};

class Child : public Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
};
```

```
Child*  C2 = new  Child;


C2->funcA();  // (g)_______

C2->funcB();  // (h)_______

C2->funcC();  // (i)_______
```

QUIZ: GROUP 3

```
class Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
                    void funcC() {...}
};

class Child : public Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
};
```

```
Child*  C2 = new  Child;


C2->funcA();  // (g) Child

C2->funcB();  // (h) Child

C2->funcC();  // (i) Parent
```

QUIZ: GROUP 4

```
class Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
                    void funcC() {...}
};

class Child : public Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
};
```

```
Child  temp;

Parent P3 = temp;


P3.funcA();   // (j)_______


P3.funcB();   // (k)_______


P3.funcC();   // (l)_______
```

QUIZ: GROUP 4

```
class Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
                    void funcC() {...}
};

class Child : public Parent
{
    public:
                    void funcA() {...}
       virtual      void funcB() {...}
};
```

```
Child  temp;

Parent P3 = temp;


P3.funcA();   // (j) Parent


P3.funcB();   // (k) Parent


P3.funcC();   // (l) Parent
```