CS 241 — Introduction to Problem Solving and Programming

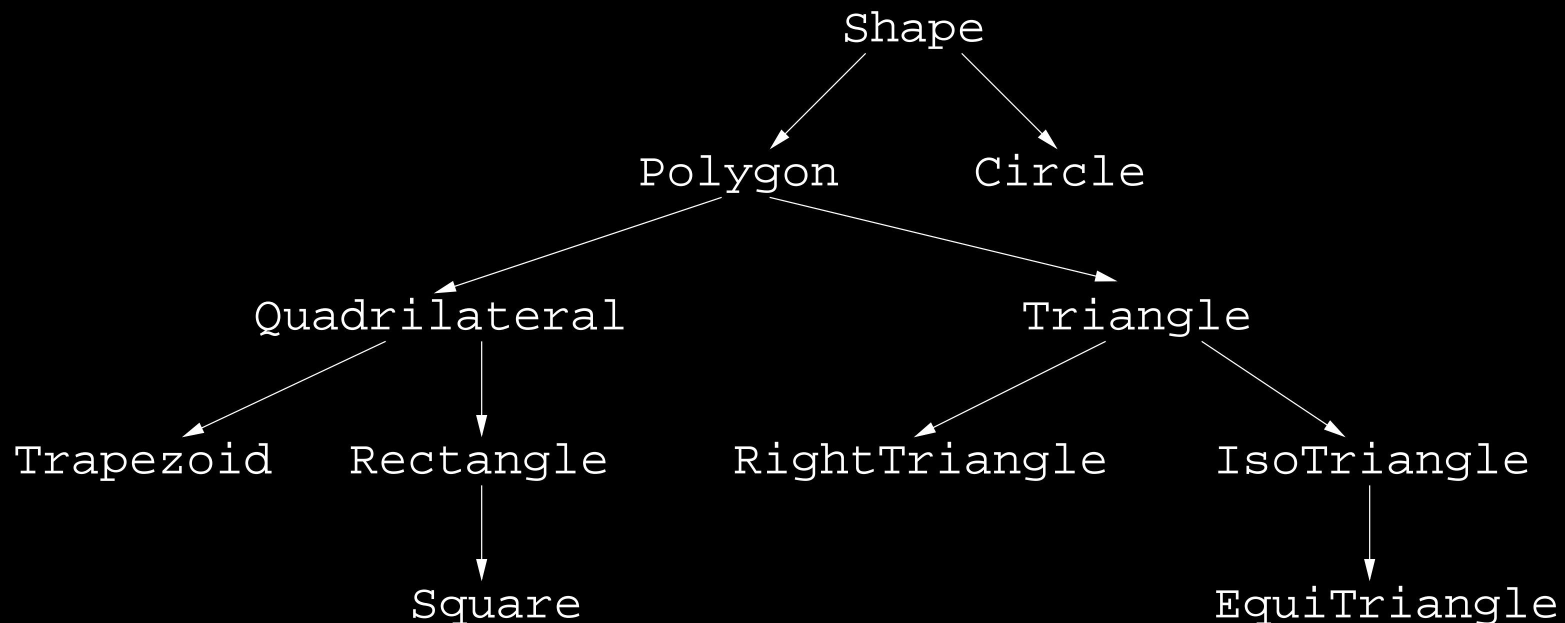
Object-Oriented Programming

The `Object` class; dynamic binding

April 6, 2005

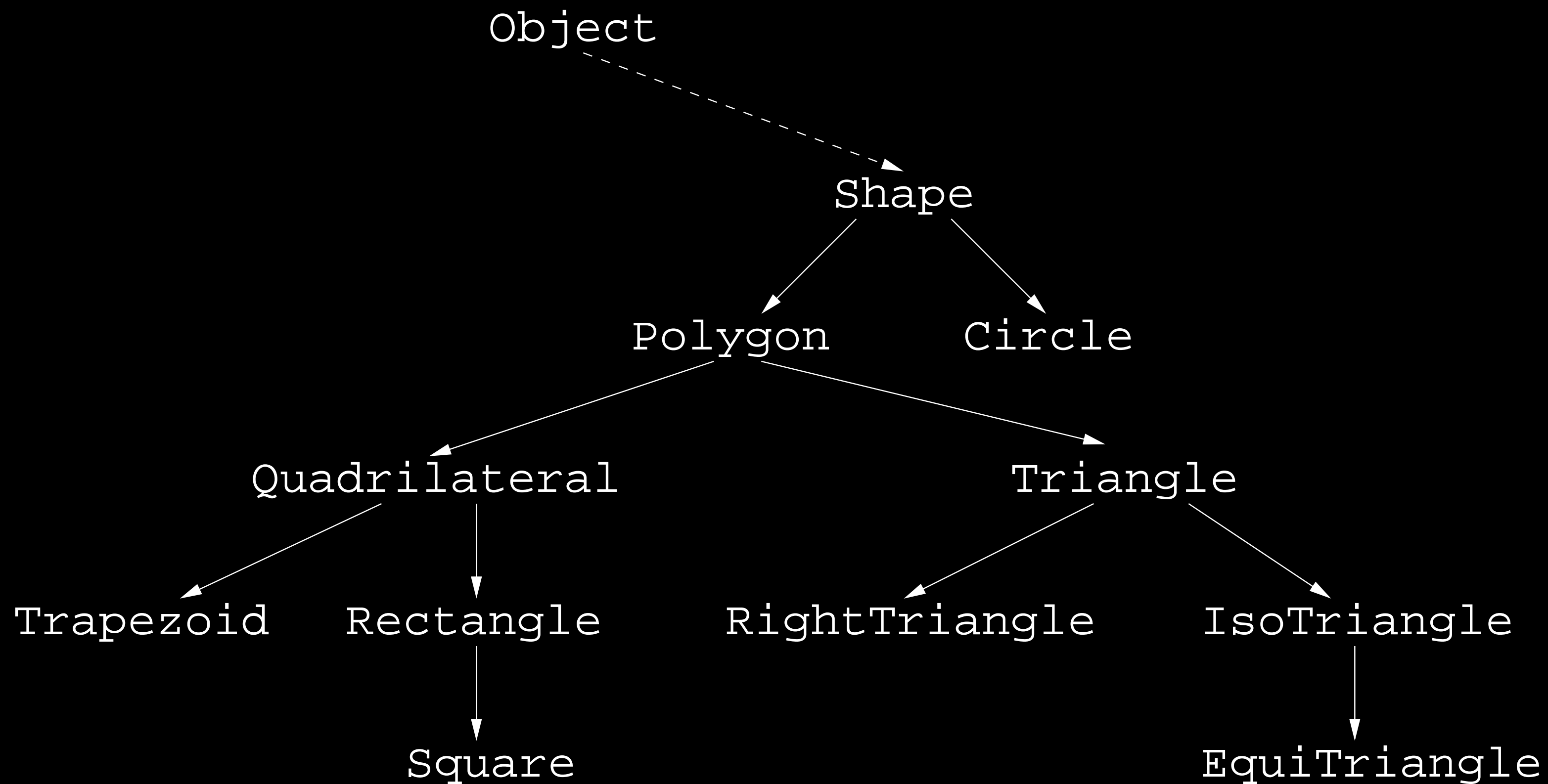
Class hierarchy

We've talked about parents and children, ancestors and predecessors, and class hierarchies. . .



Object

At the top of every hierarchy is a pre-defined class called `Object`.



Object

If you do not declare a class to extend another, it automatically extends `Object`

Some methods of class `Object`. . .

```
public class Object {

     public boolean equals(Object obj) { ... }

     public String toString() { ... }
}
```

Object

`equals(Object obj)` — recall in class `String`; intended to compare contents of objects. Notice the parameter type is `Object`

Contract: `equals` should be reflexive, symmetric, and transitive.

Default: works like ==

Object

```
public class Point {
    private double xCoord;
    private double yCoord;
    public boolean equals(Object obj) {
          if (obj instanceof Point)
              return ((Point) obj).xCoord == xCoord
                  && ((Point) obj).yCoord == yCoord;
          else return false;
    }
}
```

Object

`toString()` — create a `String` representation.

Default: the string representation is the name of the class plus an internal unique identifier.

The set assignments called for `asString()` because that was before we know about `public`. `toString()` methods are very useful for debugging.

`toString()` in class `String` just returns the object.

Object

```
public class Point {
    private double xCoord;
    private double yCoord;

    public String toString() {
        return "(" + xCoord + "," + yCoord + ")";
    }
}
```

Polymorphism

What is polymorphism?

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

How does it work?

```
public class A {
    public int m(A a) { ... }
    public int m(B b) { ... }
}

public class B extends A{
    public int m(A a) { ... }
    public int m(B b) { ... }
}
```

```
A a1 = new A();
A a2 = new B();
B b = (B) a2;

a1.m(a1);
a1.m(a2);
a1.m(b);
a2.m(a1);
a2.m(a2);
a2.m(b);
b.m(a1);
b.m(a2);
b.m(b);
```

Polymorphism

Steps that are taken:

1. At compile-time, the compiler considers the static types of the parameters to determine the method signature.
2. At compiler time, the compiler checks the static type of the receiver to determine if it has a compatible method (and adjusts the signature to the signature that is compatible).
3. At run time, the virtual machine discovers the dynamic type of the receiver.
4. At run time, the virtual machine finds the method with the compile-time signature in the most recent ancestor of the dynamic type.
5. At run time, the virtual machine executes the method.

Dynamic binding

Finding the method to execute for a given call is called binding the method to the call.

Since the binding of methods in Java happen at run time, this is called dynamic binding

Dynamic binding

```
public class A { ... }
public class B extends A { ... }
public class C {
    int m1(A a) { ... }
    int m2(A a) { ... }
    int m2(B b) { ... }
}
```

```
C c = new C();
B b = new B();

c.m1(b);
c.m2(b);
```

Dynamic binding

```
public class A { ... }
public class B extends A { ... }
public class C {
    int m1(A a) { ... }
    int m2(A a) { ... }
    int m2(B b) { ... }
}
```

```
C c = new C();
B b = new B();

c.m1(b);
c.m2(b);
```

No exact match, but this is compatible.

Dynamic binding

```
public class A { ... }
public class B extends A { ... }
public class C {
    int m1(A a) { ... }
    int m2(A a) { ... }
    int m2(B b) { ... }
}
```

```
C c = new C();
B b = new B();

c.m1(b);
c.m2(b);
```

More than one is compatible, but exact match is better.

Dynamic binding

Ambiguity is possible.

```
public class B extends A {
    int m(A a, B b) {
      return 5;
    }
    int m(B b, A a) {
      return 6;
    }
    public static void main(String[] args) {
        B b1, b2, b3;
        b1 = new B();
        b2 = new B();
        b3 = new B();

        System.out.println(b3.m(b1, b2));
    }
}
```