Polymorphism

CSCI 201
Principles of Software Development

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu
Outline

• Polymorphism
• Program
Polymorphism

- Based on the inheritance hierarchy, an object with a compile-time type of a parent can take on the form of a child at runtime
  - Then, if the parent has a method `declared` in it,
    (note that this does not say `defined` or `implemented`)
    when that method is called on an object with a compile-time type of the parent, it will call the method in the run-time type of the object, which would be the child (this is Java-specific)
- You are only able to call methods based on the compile-time type of an object
  - If a method only exists in the child and you have an instance of a parent, you won’t be able to call the child method since not every child of that parent is required to implement that method
    - You can downcast the parent object to a child, but this is potentially dangerous since you may not know the runtime type of the object
Polymorphism Example 1

```java
1 class Shape {
2    protected char name;
3    public Shape(char n) {
4        name = n;
5    }
6    public void printName() {
7        System.out.println(name);
8    }
9 }

10 class TwoD extends Shape {
11    public TwoD(char name) {
12        super(name);
13    }
14 }

15 class Square extends Rectangle {
16    public Square(char nm, float s) {
17        super(nm, s, s);
18    }
19 }

20 class Triangle extends TwoD {
21    private float base;
22    private float height;
23    public Triangle(char nm, float b, float h) {
24        super(nm);
25        base = b;
26        height = h;
27    }
28    public float getArea() {
29        return 0.5f * base * height;
30    }
31 }

32 class Rectangle extends TwoD {
33    protected float width, length;
34    public Rectangle(char nm, float w, float l) {
35        super(nm);
36        width = w;
37        length = l;
38    }
39    public float getArea() {
40        return width * length;
41    }
42 }
```
Polymorphism Example 1 (cont.)

public class ShapeTest {
    public static void printShape(Shape s) {
        System.out.print("First letter of shape is ");
        s.printName();
        System.out.print("Shape area: "+ s.getArea());
    }

    public static void main(String[] args) {
        Shape sh;
        if (args[0].equals("triangle")) {
            sh = new Triangle('t', 5.0f, 4.0f);
        }
        else if (args[0].equals("rectangle")) {
            sh = new Rectangle('r', 3.0f, 2.0f);
        }
        else {
            sh = new Square('s', 4.0f);
        }
        printShape(sh);
    }

    • How would I print the area of the shape?
Polymorphism Example 2

```java
1 class Shape {
2     protected char name;
3     public Shape(char n) {
4         name = n;
5     }
6     public void printName() {
7         System.out.println(name);
8     }
9     public float getArea() {
10         return 0; // how to get area?
11     }
12 }

13 class TwoD extends Shape {
14     public TwoD(char name) {
15         super(name);
16     }
17 }

18 class Square extends Rectangle {
19     public Square(char nm, float s) {
20         super(nm, s, s);
21     }
22 }

23 class Triangle extends TwoD {
24     private float base;
25     private float height;
26     public Triangle(char nm, float b, float h) {
27         super(nm);
28         base = b;
29         height = h;
30     }
31     public float getArea() {
32         return 0.5f * base * height;
33     }
34 }

35 class Rectangle extends TwoD {
36     protected float width, length;
37     public Rectangle(char nm, float w, float l) {
38         super(nm);
39         width = w;
40         length = l;
41     }
42     public float getArea() {
43         return width * length;
44     }
45 }
```
```java
public class ShapeTest {
    public static void printShape(Shape s) {
        System.out.print("First letter of shape is ");
        s.printName();
        System.out.print("Shape area: "+s.getArea());
    }

    public static void main(String [] args) {
        Shape sh;
        if (args[0].equals("triangle")) {
            sh = new Triangle('t', 5.0f, 4.0f);
        }
        else if (args[0].equals("rectangle")) {
            sh = new Rectangle('r', 3.0f, 2.0f);
        }
        else {
            sh = new Square('s', 4.0f);
        }
        printShape(sh);
    }
}
```

- Does this print the area of the shape?
Polymorphism Example 3

```java
abstract class Shape {
    protected char name;
    public Shape(char n) {
        name = n;
    }
    public void printName() {
        System.out.println(name);
    }
    public abstract float getArea();
}

abstract class TwoD extends Shape {
    public TwoD(char name) {
        super(name);
    }
}

class Triangle extends TwoD {
    private float base;
    private float height;
    public Triangle(char nm, float b, float h) {
        super(nm);
        base = b;
        height = h;
    }
    public float getArea() {
        return 0.5f * base * height;
    }
}

class Rectangle extends TwoD {
    protected float width, length;
    public Rectangle(char nm, float w, float l) {
        super(nm);
        width = w;
        length = l;
    }
    public float getArea() {
        return width * length;
    }
}

class Square extends Rectangle {
    public Square(char nm, float s) {
        super(nm, s, s);
    }
}
```

 USC CSCI 201L 8/18

• Polymorphism

```rust```
Polymorphism Example 3 (cont.)

```java
public class ShapeTest {
    public static void printShape(Shape s) {
        System.out.print("First letter of shape is ");
        s.printName();
        System.out.print("Shape area: "+s.getArea());
    }

    public static void main(String[] args) {
        Shape sh;
        if (args[0].equals("triangle")) {
            sh = new Triangle('t', 5.0f, 4.0f);
        } else if (args[0].equals("rectangle")) {
            sh = new Rectangle('r', 3.0f, 2.0f);
        } else {
            sh = new Square('s', 4.0f);
        }
        printShape(sh);
    }
}
```
Advanced Polymorphism Example

1 class C0 extends C1 {
2   public int meth4() {
3     System.out.println("3");
4     return 3;
5   }
6 }
7
8 class C1 extends C2 implements I1, I2 {
9   public void meth1(int num) {
10      System.out.println(num);
11   }
12   public int meth2() {
13      System.out.println("1");
14      return 1;
15   }
16   public int meth3() {
17      System.out.println("2");
18      return 2;
19   }
20 }
21
22 abstract class C2 extends C3 {
23   public void foo() {
24      System.out.println("foo");
25   }
26 }
27
28 abstract class C3 {
29   public abstract void meth1(int i);
30 }
31
32 interface I1 {
33   public int meth2();
34 }
35
36 interface I2 {
37   public int meth3();
38 }
39
40 public class Test {
41   public static void main(String [] args) {
42      C1 c = new C0();
43      c.meth1(0);
44      c.meth2();
45      c.meth3();
46      c.meth4();
47      c.foo();
48   }
49 }
Advanced Polymorphism Fix #1

1 class C0 extends C1 {
2   public int meth4() {
3     System.out.println("3");
4     return 3;
5   }
6 }
7
8 class C1 extends C2 implements I1, I2 {
9   public void meth1(int num) {
10      System.out.println(num);
11   }
12   public int meth2() {
13      System.out.println("1");
14      return 1;
15   }
16   public int meth3() {
17      System.out.println("2");
18      return 2;
19   }
20 }
21
22 abstract class C2 extends C3 {
23   public void foo() {
24      System.out.println("foo");
25   }
26 }
27
28 abstract class C3 {
29   public abstract void meth1(int i);
30 }
31
32 interface I1 {
33   public int meth2();
34 }
35
36 interface I2 {
37   public int meth3();
38 }
39
40 public class Test {
41   public static void main(String [] args) {
42      C1 c = new C0();
43      c.meth1(0);
44      c.meth2();
45      c.meth3();
46      // c.meth4();
47      c.foo();
48   }
49 }
50
class C0 extends C1 {
    public int meth4() {
        System.out.println("3");
        return 3;
    }
}

class C1 extends C2 implements I1, I2 {
    public void meth1(int num) {
        System.out.println(num);
    }
    public int meth2() {
        System.out.println("1");
        return 1;
    }
    public int meth3() {
        System.out.println("2");
        return 2;
    }
}

abstract class C2 extends C3 {
    public void foo() {
        System.out.println("foo");
    }
}

abstract class C3 {
    public abstract void meth1(int i);
}

interface I1 {
    public int meth2();
}

interface I2 {
    public int meth3();
}

public class Test {
    public static void main(String[] args) {
        C0 c = new C0();
        c.meth1(0);
        c.meth2();
        c.meth3();
        c.meth4();
        c.foo();
    }
}
Advanced Polymorphism Fix #3

1  class C0 extends C1 {
2    public int meth4() {
3      System.out.println("3");
4      return 3;
5    }
6  }
7
8  abstract class C1 extends C2 implements I1, I2 {
9    public void meth1(int num) {
10       System.out.println(num);
11    }
12    public int meth2() {
13       System.out.println("1");
14       return 1;
15    }
16    public int meth3() {
17       System.out.println("2");
18       return 2;
19    }
20    public abstract int meth4();
21 }
22
23 abstract class C2 extends C3 {
24    public void foo() {
25       System.out.println("foo");
26    }
27 }
28
29 abstract class C3 {
30    public abstract void meth1(int i);
31 }
32
33 interface I1 {
34    public int meth2();
35 }
36
37 interface I2 {
38    public int meth3();
39 }
40
41 public class Test {
42    public static void main(String [] args) {
43       C1 c = new C0();
44       c.meth1(0);
45       c.meth2();
46       c.meth3();
47       c.meth4();
48       c.foo();
49    }
50 }
class C0 extends C1 {
    public int meth4() {
        System.out.println("3");
        return 3;
    }
}

abstract class C1 extends C2 implements I1, I2 {
    public void meth1(int num) {
        System.out.println(num);
    }
    public int meth2() {
        System.out.println("1");
        return 1;
    }
    public int meth3() {
        System.out.println("2");
        return 2;
    }
}

abstract class C2 extends C3 {
    public void foo() {
        System.out.println("foo");
    }
}

abstract class C3 {
    public abstract void meth1(int i);
}

interface I1 {
    public int meth2();
}

interface I2 extends I3 {
    public int meth3();
}

interface I3 {
    public int meth4();
}

public class Test {
    public static void main(String [] args) {
        C1 c = new C0();
        c.meth1(0);
        c.meth2();
        c.meth3();
        c.meth4();
        c.foo();
    }
}
Redefining/Overriding

- A derived class in C++ is able to redefine a method in a base class by giving the method the same name and parameters.

- Java and C++ behave differently
  
  > C++
  
  - If the function in the parent is `virtual`, the function in the child overrides it. Then the function executed is based on the runtime type of the variable.
  - If the function in the parent is not `virtual`, the function in the child redefines it. Then the function executed is based on the compile-time type of the variable.

  > Java
  
  - There is no redefining of methods in Java. Methods that are defined in a derived class with the same signature as they are defined in a base class are always overridden.
  - When the method is called on an instance of either the derived class or the base class, the method in the runtime instance of the object is called.
  - Java doesn’t have a sense of redefining methods - it always overrides, regardless of the use of the word `abstract`
Redefining Methods/Functions Example

NOTE: Java does not have redefined methods. These methods have all been overridden.

```java
1  class C0  {
2    public void meth3() {
3      System.out.println("3");
4    }
5  }
6
7  class C1 extends C0  {
8    public void meth3() {
9      System.out.println("4");
10   }
11 }
12
13 public class HelloClass {
14   public static void meth(C0 c) {
15     c.meth3();
16   }
17   public static void main(String[] args) {
18     C0 c0 = new C1();
19     c0.meth3();
20     meth(c0);
21     C1 c1 = new C1();
22     c1.meth3();
23     meth(c1);
24   }
25 }
26 }
```

NOTE: C++ functions are redefined if they are not defined as virtual in the parent.

```cpp
1  #include <iostream>
2  using namespace std;
3  class C0  {
4    public:
5      void meth3() {
6        cout << "3" << endl;
7      }
8  }
9  class C1 : public C0  {
10     public:
11      void meth3() {
12        cout << "4" << endl;
13      }
14   }
15 void meth(C0 *c) {
16    c->meth3();
17 }
18 void main() {
19    C0 *c0 = new C1();
20    c0->meth3();
21    meth(c0);
22    C1 *c1 = new C1();
23    c1->meth3();
24    meth(c1);
25 }
26 }
```
Outline

• Polymorphism
• Program
Write the shape program provided in these slides, but make the Shape class an interface instead. Make the `name` variable `protected` so it can be set in the derived classes. Think about the advantages to using an interface instead of a class.

```bash
java csci201.ShapeTest triangle
First letter of shape is t
Shape area: 10.0
```