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 an abstract method in it, when that method is called on an object with a compile-time type of the parent, it will call the non-abstract method in the run-time type of the object, which would be the child

- 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 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
class Shape {
    protected char name;
    public Shape(char n) {
        name = n;
    }
    public void printName() {
        System.out.println(name);
    }
}

class TwoD extends Shape {
    public TwoD(char name) {
        super(name);
    }
    public float getArea() {
        return 0.5f * base * height;
    }
}

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);
    }
}
```
• How would I print the area of the shape?
```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 }

1 class TwoD extends Shape {
2   public TwoD(char name) {
3     super(name);
4   }
5 }

1 class Triangle extends TwoD {
2   private float base;
3   private float height;
4   public Triangle(char nm, float b, float h) {
5       super(nm);
6       base = b;
7       height = h;
8   }
9   public float getArea() {
10      return 0.5f * base * height;
11   }
12 }

1 class Rectangle extends TwoD {
2   protected float width, length;
3   public Rectangle(char nm, float w, float l) {
4       super(nm);
5       width = w;
6       length = l;
7   }
8   public float getArea() {
9       return width * length;
10   }
11 }

1 class Square extends Rectangle {
2   public Square(char nm, float s) {
3       super(nm, s, s);
4   }
5 }
```
Polymorphism Example 2 (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);
    }

    • Does this print the area of the shape?
```
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);
  }
}
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

```java
1 class C0 extends C1 {
2   public int meth3() {
3     System.out.println("3");
4     return 3;
5   }
6 }
7
8 class C1 extends C2 implements I1, I2 {
9   public void meth(int num) {
10     System.out.println(num);
11   }
12   public int meth1() {
13     System.out.println("1");
14     return 1;
15   }
16   public int meth2() {
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 meth(int i);
30 }
31
32 interface I1 {
33   public int meth1();
34 }
35
36 interface I2 {
37   public int meth2();
38 }
39
40 public class Test {
41   public static void main(String [] args) {
42     C1 c = new C0();
43     c.meth(0);
44     c.meth1();
45     c.meth2();
46     c.meth3();
47     c.foo();
48   }
49 }
```
Advanced Polymorphism Example

1  class C0 extends C1 {
2    public int meth3() {
3      System.out.println("3");
4      return 3;
5    }
6  }
7
8  abstract class C1 extends C2 implements I1, I2 {
9    public void meth(int num) {
10       System.out.println(num);
11     }
12    public int meth1() {
13       System.out.println("1");
14       return 1;
15    }
16    public int meth2() {
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 meth(int i);
30  }
31
32  interface I1 {
33    public int meth1();
34  }
35  interface I2 extends I3 {
36    public int meth2();
37  }
38
39  interface I3 { // is I3 necessary?
40    public int meth3();
41  }
42
43  public class Test {
44    public static void main(String [] args) {
45       C1 c = new C0();
46       c.meth(0);
47       c.meth1();
48       c.meth2();
49       c.meth3();
50       c.foo();
51    }
52  }
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++**
    - When the function is called on an instance of the derived class (compile-time type), the redefined function in the derived class will be called and the function in the base class will NOT be called.
    - When the function is called on an instance of the base class (compile-time type), even if the run-time type is different, the function defined in the base class will be called and the function in the derived class will NOT be called.
  - **Java**
    - There is no redefining of methods in Java. Methods that are defined in a derived class the same 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
class C0 {
    public void meth3() {
        System.out.println("3");
    }
}

class C1 extends C0 {
    public void meth3() {
        System.out.println("4");
    }
}

class HelloClass {
    public static void meth(C0 c) {
        c.meth3();
    }
    public static void main(String[] args) {
        C0 c0 = new C1();
        c0.meth3();
        meth(c0);
        C1 c1 = new C1();
        c1.meth3();
    }
}
```

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

```cpp
#include <iostream>
using namespace std;
class C0 {
    public:
        void meth3() {
            cout << "3" << endl;
        }
};
class C1 : public C0 {
    public:
        void meth3() {
            cout << "4" << endl;
        }
};

void meth(C0 *c) {
    c->meth3();
}

int main() {
    C0 *c0 = new C1();
    c0->meth3();
    meth(c0);
    C1 *c1 = new C1();
    c1->meth3();
    return 1;
}
```
Outline

- Polymorphism
- Program
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.

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

c:\>