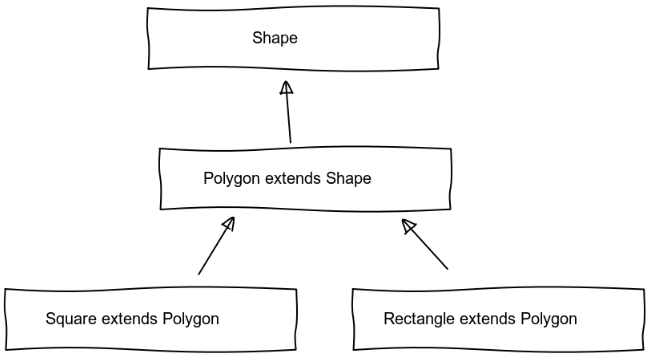
IS-A and *HAS-A* Relationship

IS-A relationship

- expressed with class inheritance or interface implementation
- 'this thing is a type of that thing'
- cat is a animal
- String is a Object
- Triangle is a Shape

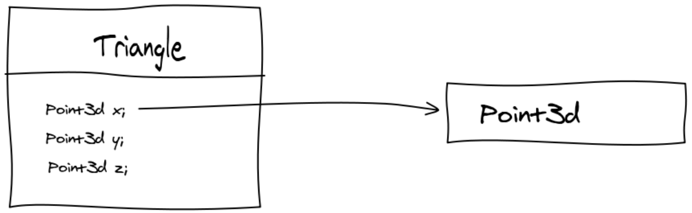
IS-A Relationship



HAS-A Relationship

- is base on usage, rather than inheritance
- instance of a class Triangle has reference to instance of a class Point

HAS-A Relationship



Polymorphism

Polymorphism is the ability of an object to take on many forms

Type of polymorphism

- *ad-hoc* - methods overloading vs overriding
- parametric
- dynamic method binding

Dynamic method binding

The ability of a program to resolve references to subclass methods at runtime. For example assume that three subclasses (Square, Rectangle,Triangle) have been created based on the Shape abstract class, each having their own calculateArea() method. Although each method reference is to an Shape (but no shape objects exist), the program is will resolve the correct method reference at runtime.

Overridden methods

Methods that are redefined within an inherited or subclass. They have the same signature and the subclass definition is used.

example

```
public abstract class Polygon {

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "polygon";
    }

}

public class Square extends Polygon{

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "square";
    }
}
```

test

```
public abstract class Polygon {

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "polygon";
    }

}

public class Square extends Polygon{

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "square";
    }
}
```

```
public class OverrideTest {

    public void overrideTest() {


        Polygon polygon = new Polygon();

        String toStringPolygon = polygon.toString();

        assertEquals( toStringPolygon, "polygon" );


    }
}
```

test - FAIL

```java
public abstract class Polygon {

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "polygon";
    }

}

public class Square extends Polygon{

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "square";
    }
}
```

```java
public class OverrideTest {

    public void overrideTest() {


        Polygon polygon = new Polygon(); //abstract class

        String toStringPolygon = polygon.toString();

        assertEquals( toStringPolygon, "polygon" );


    }
}
```

test

```java
public abstract class Polygon {

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "polygon";
    }

}

public class Square extends Polygon{

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "square";
    }
}
```

```java
public class OverrideTest {

    public void overrideTest() {


        Polygon polygon = new Square();

        Square square = new Square();

        String toStringPolygon = polygon.toString();
        String toStringSquare = square.toString();

        assertEquals( toStringPolygon, "polygon" ); // ??
        assertEquals( toStringSquare, "square" ); // ??

    }
}
```

test

```
public abstract class Polygon {

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "polygon";
    }

}

public class Square extends Polygon{

    public double calculateArea() {
        /* body */
    }

    public String toString() {
        return "square";
    }
}
```

```
public class OverrideTest {

    public void overrideTest() {


        Polygon polygon = new Square();

        Square square = new Square();

        String toStringPolygon = polygon.toString();
        String toStringSquare = square.toString();

        assertEquals( toStringPolygon, "polygon" ); // false
        assertEquals( toStringSquare, "square" ); // true

    }
}
```

example

```java
public class Square extends Polygon {

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    public double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

test

```java
public class Square extends Polygon {

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    public double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

```java
public class OverrideTest {

    public void overrideTest() {

        Square square = new Square();
        String toStringSquare  = square.toString();
        assertEquals( toStringSquare, "square" ); // ??


        Square square3d = new Square3d();
        assertEquals(square3d.toString,
                     "square3"); // ??


        square = new Square3d();
        assertEquals(square.calculateArea(), ??); // ??
        assertEquals(square.calculateVolumn(), ??); // ??

    }
}
```

test - result

```java
public class Square extends Polygon {

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    public double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

```java
public class OverrideTest {

    public void overrideTest() {

        Square square = new Square();
        String toStringSquare  = square.toString();
        assertEquals( toStringSquare, "square" ); // true


        Square square3d = new Square3d();
        assertEquals(square3d.toString,
                     "square3"); // true


        square = new Square3d();
        assertEquals(square.calculateArea(), ??); // 10.0
        assertEquals(square.calculateVolumn(), ??); // error

        assertEquals(
             (Square3d)square).calculateVolumn(), ??);

    }
}
```

example

```
public class Square extends Polygon {

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    private double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

test

```java
public class Square extends Polygon {

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    private double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

```java
public class OverrideTest {

    public void overrideTest() {

        Square square = new Square();
        assertEquals( square.calculateArea(),
                      ?? ); // ??


        Square square3d = new Square3d();
        assertEquals(square3d.calculateArea(),
                     ?? ); // ??

    }
}
```

test - result

```
public class Square extends Polygon {

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    private double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

```
public class OverrideTest {

    public void overrideTest() {

        Square square = new Square();
        assertEquals( square.calculateArea(),
                      ?? ); // ??


        Square square3d = new Square3d();
        assertEquals(square3d.calculateArea(),
                     ?? ); // RTE

    }
}
```

Rules

- the argument list must exactly match that of the overridden method. If they don't match, you can end up with an overloaded method you didn't intend.
- the return type must be the same as, or a subtype of, the return type declared in the original overridden method in the superclass.
- the access level can't be more restrictive than the overridden method's.
- the access level CAN be less restrictive than that of the overridden method.
- you cannot override a method marked final.
- you cannot override a method marked static.

example

```
public class Square extends Polygon {

    protected int corners = 4;

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    protected int corners = 8;

    public double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

test

```
public class Square extends Polygon {

    protected int corners = 4;

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    protected int corners = 8;

    public double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

```
public class OverrideTest {

    public void overrideTest() {

        Square square = new Square();

        assertEquals( square.corners , ?? ); // ??


        Square square3d = new Square3d();
        assertEquals(square3d.corners, ?? ); // ??


        square = new Square3d();
        assertEquals(square.corners, ??); // ??



    }
}
```

test - result

```java
public class Square extends Polygon {

    protected int corners = 4;

    public double calculateArea() {
        return 2.0;
    }

    public String toString() {
        return "square";
    }
}

public class Square3d extends Square {

    protected int corners = 8;

    public double calculateArea() {
        return 10.0;
    }

    public double calculateVolumn() {
        return 20.0;
    }

    public double toString() {
        return "square3d";
    }
}
```

```java
public class OverrideTest {

    public void overrideTest() {

        Square square = new Square();

        assertEquals( square.corners , 4 ); // 4


        Square square3d = new Square3d();
        assertEquals(square3d.corners, 8 ); // 8


        square = new Square3d();
        assertEquals(square.corners, 4); // at compile time



    }
}
```

Overloaded method

Methods with the same name signature but either a different number of parameters or different types in the parameter list.

Rules

- overloaded methods MUST change the argument list.
- overloaded methods CAN change the return type.
- overloaded methods CAN change the access modifier.
- overloaded methods CAN declare new or broader checked exceptions.

example

```
public class Square extends Polygon {


    public Square() { }

    public Square(double a, double b) { }

    public void move(double x) { }

    public void move(double x,
                     double y) { }

    public void move(double x,
                     double y
                     double z) { }


}
```

Polymorphic method invocations apply only to instance methods. You can always refer to an object with a more general reference variable type (a su- perclass or interface), but at runtime, the ONLY things that are dynamically selected based on the actual object (rather than the reference type) are instance methods. Not static methods. Not variables. Only overridden instance meth- ods are dynamically invoked based on the real object's type.

Comparison operators

Two operators

- *==*
- *.equals()*

```java
public class TestEquality {

    Point p1 = new Point(1,1);
    Point p2 = new Point(1,1);
    Point p3 = new Point(1,2);
    Point p4 = p3;


    public void testEquality() {

    }
}
```

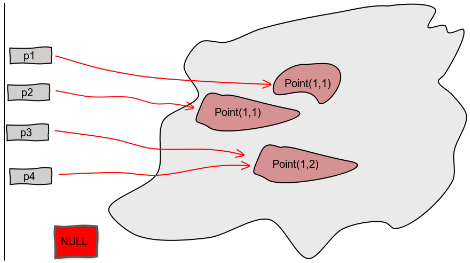
test

```
public class TestEquality {

    Point p1 = new Point(1,1);
    Point p2 = new Point(1,1);
    Point p3 = new Point(1,2);
    Point p4 = p3;


    public void testEquality() {

    }
}
```

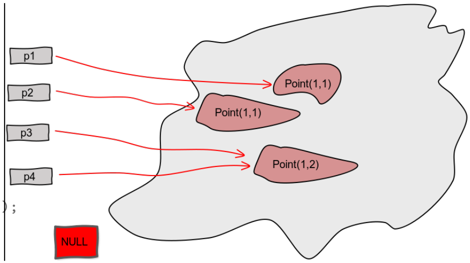


test

```
public class TestEquality {

    Point p1 = new Point(1,1);
    Point p2 = new Point(1,1);
    Point p3 = new Point(1,2);
    Point p4 = p3;


    public void testEquality() {
        assertEquals( p1 == p3, ?? );
        assertEquals( p1.equals(p4), ?? );
    }
}
```

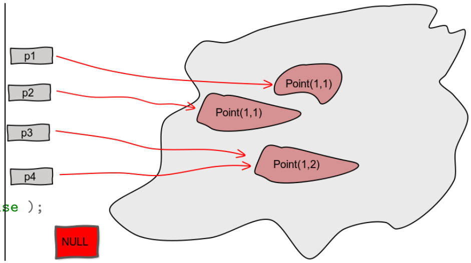


test

```
public class TestEquality {

    Point p1 = new Point(1,1);
    Point p2 = new Point(1,1);
    Point p3 = new Point(1,2);
    Point p4 = p3;


    public void testEquality() {
        assertEquals( p1 == p3, false );
        assertEquals( p1.equals(p4), false );
    }
}
```


p1
p2
p3
p4
Point(1,1)
Point(1,1)
Point(1,2)
NULL