What is Polymorphism?

• The definition of **polymorphism** is:

  The ability to treat objects of different classes in a uniform way.

• What does this mean?

• It is best explained with an example.
Consider the following example:

- Let’s say we are implementing a drawing program that allows the user to draw shapes to a canvas. We have the following inheritance hierarchy:
Each class has a **draw** method

- Let’s say each class in the previous slide has a method called **draw** that draws its shape to a canvas:

```java
public void draw()
{
    // Draws itself
}
```

- The **Circle**’s **draw** method draws a circle. What does the **Shape**’s **draw** method draw? Let’s implement **Shape** so that by default it simply draws a square.
Example

Shape shape = new Shape();
shape.draw();

Produces:

Circle circle = new Circle();
circle.draw();

Produces:
Okay, so far so good

- So far we have two classes: **Shape** and **Circle**
- Circle override’s its superclass’s **draw**
- That is, each class’s **draw** method draws a different shape
Let’s Implement a method called DrawShape

• Now let’s say we define a method called DrawShape that accepts a single Shape object and draws it:

```java
public static void drawShape(Shape shape)
{
    shape.draw();
}
```
What actual shape will drawn in the following example?

• Let’s say in our main method, we implement the following code:

```java
Circle circle = new Circle();
drawShape(circle);
```

• What actual shape will be drawn to the canvas?
• Answer: a circle!

This is what is actually drawn to the canvas
What Happened?

- Even though inside the `drawShape` method we call `draw` on a reference variable of type `Shape`, Java knows that the object being referenced by this variable is actually an instance of Circe:

```java
shape.draw();
```
How does Java determine which method to call?

• Why did Java call `Circle.draw` instead of `Shape.draw` even though the reference variable “shape” was type `Shape`?

• In Java, method calls are always determined by the type of the actual object, not on the type of the variable containing the object reference.

• This is called dynamic method lookup.

• Dynamic method lookup allows us to treat objects of different classes in a uniform way. This ability is called Polymorphism.