
# Polymorphism

## Polymorphism

**Polymorphism** basically means **many forms**.

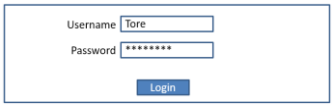
It's the ability to give code more than one behaviour or "form".

**Inheritance based polymorphism** allows us to provide different behaviours for both classes and methods.

We never actually used the **GuiControl** classes to any great extent. How would we use them?

# Polymorphism (2)

First, we would probably like to add our controls to our GUI.




If we were to create such a class (called **Gui**), it would probably contain methods like:

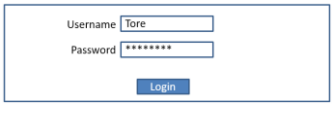
```
public class Gui {
    public void addTextBox(TextBox textBox) {
        // Code to add a text box
    }

    public void addButton(Button button) {
        // Code to add a button
    }

    // More methods will follow (one for each type).
}
```

One add-method for each type of **GuiControl**.

# Polymorphism (3)

The usage of this class would be:

```
// New GUI to add controls to
Gui myGui = new Gui();

// Create controls
Button okButton = new Button();
TextBox firstName = new TextBox();
TextBox lastName = new TextBox();

// Add controls
myGui.addButton(okButton);
myGui.addTextBox(firstName);
myGui.addTextBox(lastName);
```




The following approach will work for a while, but it will be a nightmare to maintain.

## Polymorphism (4)

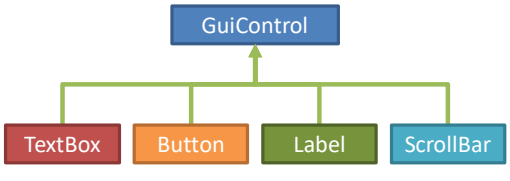
If we're writing code in this manner, then we're missing out on a big feature of OOP.

Right now, we don't really cash in on the **is a** relationship of inheritance. Remember that a **TextBox** **is a** **GuiControl** and a **Button** **is a** **GuiControl** and so on.

The GUI class should neither care nor know what kind of GUI control we are adding, just that it **is a** **GuiControl**.

## Polymorphism (5)

The solution is to first look at the class-hierarchy:




Then we just add one Add method:

We can take any **GuiControl** as an argument.

```
public class Gui {
    public void addControl(GuiControl control) {
        // Code to add a control to the GUI
    }

    // Methods pertaining to other GUI-related
    // operations will follow.
}
```

## Polymorphism (6)

The updated usage would now be:

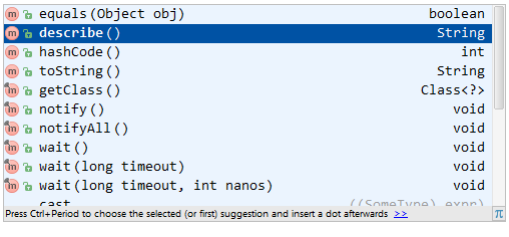
```
// New GUI to add controls to
Gui myGui = new Gui();

// Create controls
Button okButton = new Button();
TextBox firstName = new TextBox();
TextBox lastName = new TextBox();

// Add controls
myGui.addControl(okButton);
myGui.addControl(firstName);
myGui.addControl(lastName);
```

One method is enough, as long as what we pass in is actually **GuiControl** objects.

## Polymorphism (7)

In the Add method we'll only have access to the **superclass members**, since we are taking a **GuiControl** as input, not a specific control.

```
public class Gui {
    public void addControl(GuiControl control) {
        control.
    }
}
```



## Polymorphism (8)

Depending on where we put an object, we will see different members:

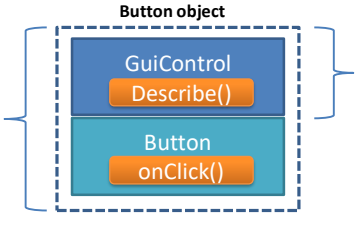
```
Button okButton1 = new Button();

okButton1.describe();
okButton1.onClick();

GuiControl okButton2 = okButton1;

okButton2.describe();
okButton2.onClick();
```

Not available as a GuiControl

## Polymorphism (9)

Depending of where we put an object, we will get access to and see different members:

```
Button b1= new Button();

b1.describe();
b1.onClick();
```




```
GuiControl b2= new Button();

b2.describe();
```

Keep in mind that we never loose any data when placing the Button in a **GuiControl**!, the object is still the same and never changed.

# Exercise 18

Lets do exercise 18