Polymorphism

Object-Oriented Programming

Topics

- Polymorphism
- Subtyping
- Overriding
- Overloading
- Extension Methods

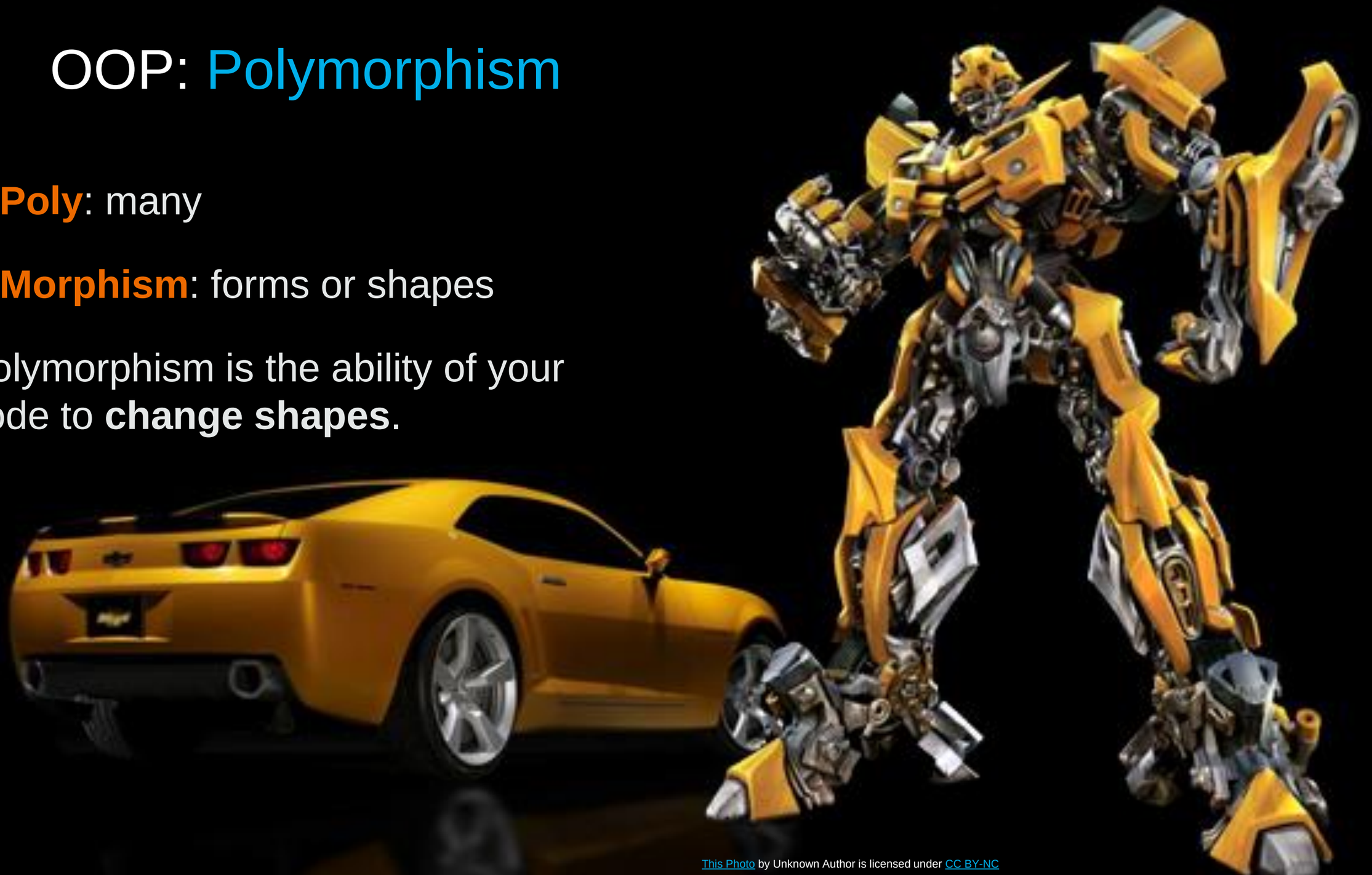
Polymorphism

Object-Oriented Programming

OOP: Polymorphism

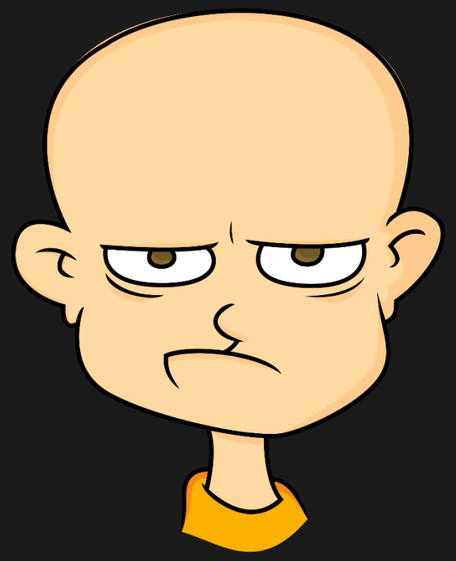
- **Poly**: many
- **Morphism**: forms or shapes

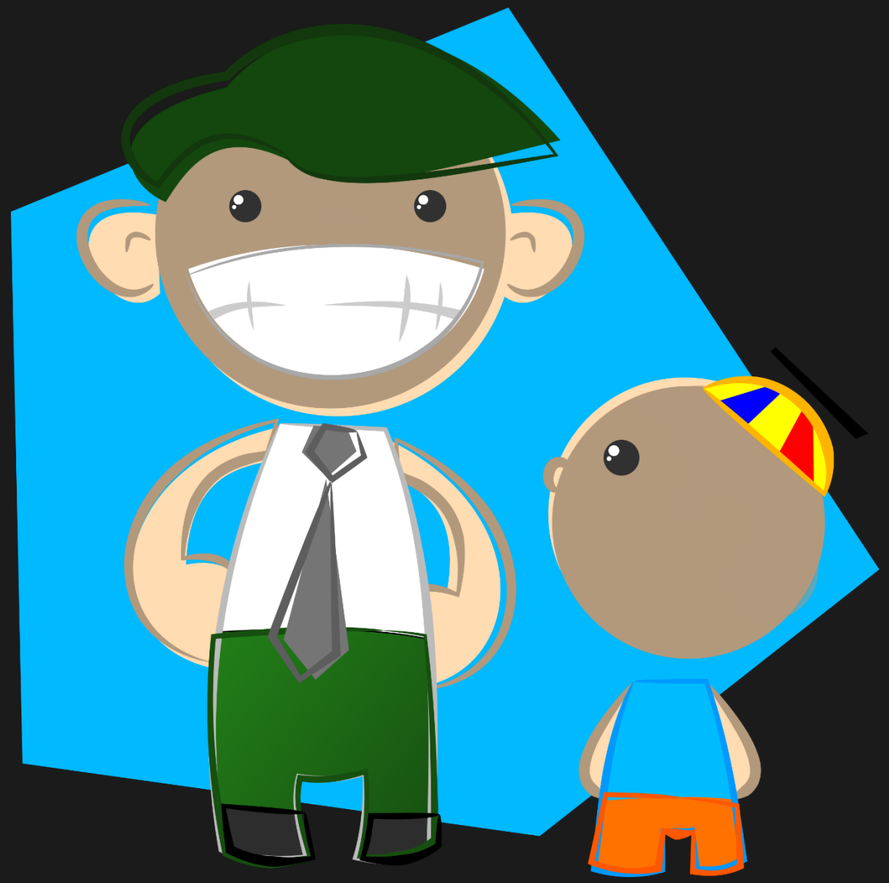
Polymorphism is the ability of your code to **change shapes**.



OOP: Polymorphism

BLAH!
BLAH!
BLAH!





MAN

DAD

TEACHER

OOP: Polymorphism

- Ways to do Polymorphism in your code:
  - **Subtyping** (inheritance)
  - **Overriding** (changing the behavior)
  - **Overloading** (different implementations)

Subtyping

- When you upcast or downcast a variable to a different type, you are in essence changing what the variable can do.

```
Warship enterprise = new Warship("Enterprise", 9.9F);
enterprise.CloakShip(true);
Spaceship federationShip = enterprise;
federationShip.CloakShip(false);
```

Can’t see this method anymore

Overriding

Overriding

- Overriding lets you change the behavior of a base method
  - You can extend the method (add to the behavior)
  - Or you can fully override the method (add all new behavior)

How-To: Overriding

- To override, you need to do 2 things:
  1. Mark the base method as **virtual**
  2. Add the method to the derived class marked with **override**.

How-To: Overriding

1. Mark the base method as **virtual**
2. Add the method to the derived class marked with **override**.

```
public virtual void Warp(float warpFactor)
{
    Console.WriteLine($"Warp {warpFactor}. Engage!");
}
```

How-To: Overriding

1. Mark the base method as **virtual**
2. Add the method to the derived class marked with **override**.

```
public override void Warp(float warpFactor)
{
    CloakShip(true);
    base.Warp(warpFactor);
}
```

Override Challenge

1. Add a Display method to FantasyWeapon
2. In Display, print the weapon info: rarity, level, max damage, cost
3. In BowWeapon, override Display to also print the arrow count and capacity
4. Modify the Inventory class method PrintInventory to call Display.

```
public virtual void Warp(float warpFactor)
{
    Console.WriteLine($"Warp {warpFactor}. Engage!");
}
```

```
public override void Warp(float warpFactor)
{
    CloakShip(true);
    base.Warp(warpFactor);
}
```

LINKS

Classes

VIDEOS

Overloading

Overloading

- Allows us to do the same kind of operation but with different inputs
- **The signatures of the methods must be different**.
  - Example: most of the methods of the Math class are overloaded.

How-To: Overloading

- **The signatures of the methods must be different**. How?

How-To: Overloading

- **The signatures of the methods must be different**. How?
  1. The number of parameters are different

```
public int    Add(int n1, int n2) { return n1 + n2; }
0 references
public int    Add(int n1, int n2, int n3) { return n1 + n2 + n3; }
0 references
public double Add(double n1, double n2) { return n1 + n2; }
0 references
public string Concat(string prefix, int n) { return prefix + n; }
0 references
public string Concat(int n, string postfix) { return n + postfix; }
```

How-To: Overloading

- **The signatures of the methods must be different**. How?
  1. The number of parameters are different
  2. The types of the parameters are different

```
public int    Add(int n1, int n2) { return n1 + n2; }
0 references
public int    Add(int n1, int n2, int n3) { return n1 + n2 + n3; }
0 references
public double Add(double n1, double n2) { return n1 + n2; }
0 references
public string Concat(string prefix, int n) { return prefix + n; }
0 references
public string Concat(int n, string postfix) { return n + postfix; }
```

How-To: Overloading

- **The signatures of the methods must be different**. How?
  1. The number of parameters are different
  2. The types of the parameters are different
  3. The order of the parameters are different

```
public int    Add(int n1, int n2) { return n1 + n2; }
0 references
public int    Add(int n1, int n2, int n3) { return n1 + n2 + n3; }
0 references
public double Add(double n1, double n2) { return n1 + n2; }
0 references
public string Concat(string prefix, int n) { return prefix + n; }
0 references
public string Concat(int n, string postfix) { return n + postfix; }
```

Overload Challenge

1. Overload the DoDamage method of FantasyWeapon
2. The overloaded DoDamage should take another int parameter called enchantment.
3. Add the enchantment value to the max damage before calculating the damage.
4. In Main, call the overloaded method and print the damage returned.

LINKS

Classes

VIDEOS

Extension Methods

Extension Methods

- Extension methods allow you to add functionality to classes *without* modifying the class or deriving from the class.
- Microsoft adds quite a few extension methods to its own types.

Extension Methods

1. Create a static class
2. Add a static method to the class.
3. Add a parameter to the method that declares the type of the this param

```
public static class Extensions
{
    0 references
    public static int Value(this WeaponRarity rarity)
    {
        return (int)rarity;
    }
}
```

Extension Challenge

1. Create a static Extension class.
2. Add a static Bows extension method to the Inventory class.
   - It should create a new list of just the bows in the inventory parameter.
   - Return the list of bows.

```
public static class Extensions
{
    0 references
    public static int Value(this WeaponRarity rarity)
    {
        return (int)rarity;
    }
}
```

LINKS

Classes

VIDEOS