Polymorphism

- Polymorphism: "Poly" meaning "many" and "morph" meaning "form" – many forms
- When the implementation for the same member signature varies between two or more related classes, you have a key object-oriented principle of *polymorphism*.
- Polymorphism relies are two important principles:
  - All derived class objects can be stored and partially accessed with a base class reference
  - Overridden members of derived classes have different implementations – a base class reference will still access the appropriate overridden derived class member

The "is a" Relationship

- All derived classes have an '**is a**' relationship to its base class…
  - For base class *Person*, derived class *Student* – every student **is a** person but not every person is a student (they could be an employee)
  - For base class *Boat*, derived class *PowerBoat* – every powerboat **is a** boat but not every boat is a powerboat (they could be a sailboat)
  - For base class *BankAccount*, derived class *SavingsAccount* – every savings account **is a** bank account but not every bank account is a savings account (they could be a chequing account)
- All classes have **Object** as their base class – so every class **is a** Object!

- This '*is a*' relationship allows derived class object to be stored with a base class reference:

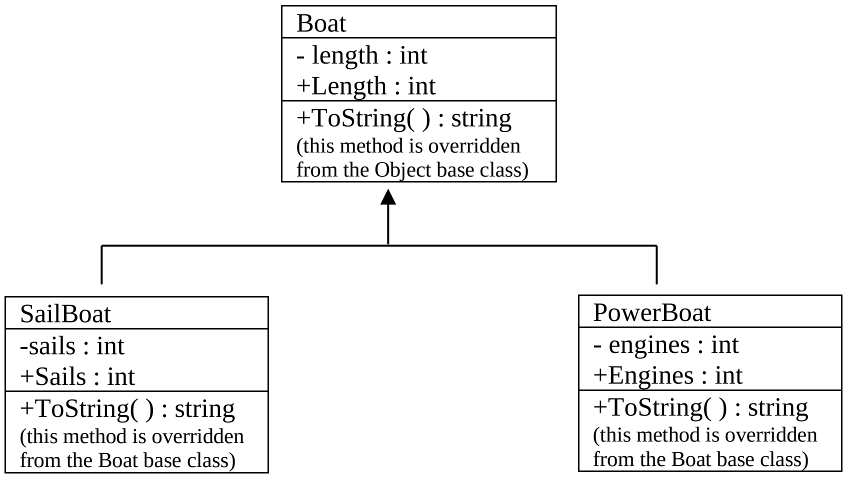
  Examples: Derived class objects stored with a base class reference…

```
Boat boat1 = new PowerBoat();
Person person1 = new Student();
BankAccount account1 = new SavingsAccount();
Object object1 = new PowerBoat();
Object object2 = new Student();
Object object3 = new SavingsAccount();
```

- The limitation of using a base class reference to access a derived class object is that the reference can only access those members in the base class.
- All derived class members cannot be accessed directly *except* members that have been overridden from the base class because the base class *had* its own version of the overridden member.

Polymorphism Example

Consider the following simplified inheritance example:



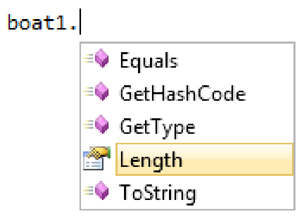
Code:

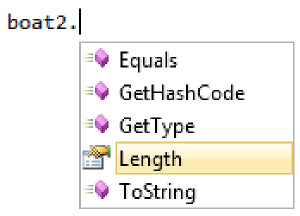
```
public class Boat : Object
{
    // private field - public property
    private int length = 5;
    public int Length
    {
        get { return length; }
        set { length = (value >= 5) ? value : 5; }
    }
    // override the ToString() method from the Object class
    public override string ToString()
    {
        return "Length: " + length;
    }
}
public class PowerBoat : Boat
{
    // private field - public property
    private int engines = 1;
    public int Engines
    {
        get { return engines; }
        set { engines = (value >= 1) ? value : 1; }
    }
    // override the ToString() method from the Object class
    public override string ToString()
    {
        return base.ToString() + "\nEngines : " + engines;
    }
}
public class SailBoat : Boat
{
    // private field - public property
    private int sails = 1;
    public int Sails
    {
        get { return sails; }
        set { sails = (value >= 1) ? value : 1; }
    }
    // override the ToString() method from the Object class
    public override string ToString()
    {
        return base.ToString() + "\nSails : " + sails;
    }
}
```

- Since both a `PowerBoat` and `SailBoat` **is a** `Boat`, we can declare `Boat` references and assign them either `PowerBoat` or `SailBoat` objects…

```
    Boat boat1 = new PowerBoat();
    Boat boat2 = new SailBoat();
```

- Using the `Boat` references we can access only those members that belong to the `Boat` class…





```
    string display = "Boat 1...\nLength: " + boat1.Length + "\nBoat 2...\nLength: " + boat2.Length;
    txtTextbox.Text = display;
```

Output:

```
Boat 1…
Length: 5
Boat 2…
Length: 5
```

- However…using a `Boat` reference on an overridden member produces unique results…

```
string display = "Boat 1...\n" + boat1.ToString() + "\nBoat 2...\n" + boat2.ToString();
txtTextbox.Text = display;
```

Output:

```
Boat 1…
Length: 5
Engines: 1
Boat 2…
Length: 5
Sails: 1
```

Boat 1 ToString()

Boat 2 ToString()

This is Polymorphism…

- Using a base class reference produced two different results!
- For *boat1*, the *ToString()* version included engine info.
- For *boat2*, the *ToString()* version included sails info.
- The Boat class has a *ToString()* but it is replaced with a newer version when either a SailBoat or PowerBoat is created.

Casting and the `is` Keyword

- The power of using base class references is that they can be used as the data type for an array or collection…
  - An array of Person references can hold any derived class objects (Students and Employees)
  - A List<> of Boat references can hold any derived class objects (PowerBoat and SailBoat)

Example: An Array of Boats

```
Boat[] boats = new Boat[4];
boats[0] = new PowerBoat();   // first two elements are powerboat objects
boats[1] = new PowerBoat();
boats[2] = new SailBoat();   // last two elements are sailboat objects
boats[3] = new SailBoat();
```

Example: A List<> of Boats

```
List<Boat> boats = new List<Boat>();
boats.Add(new PowerBoat());   // first two elements are powerboat objects
boats.Add(new PowerBoat());
boats.Add(new SailBoat());   // last two elements are sailboat objects
boats.Add(new SailBoat());
```

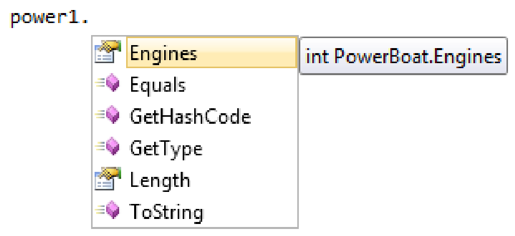
- As mentioned previously, using a base class reference only allows access to base class members except those that are overridden. If you want access to the derived class members, you must use a *type cast.*

Example: Casting Base Class References as Derived Class References

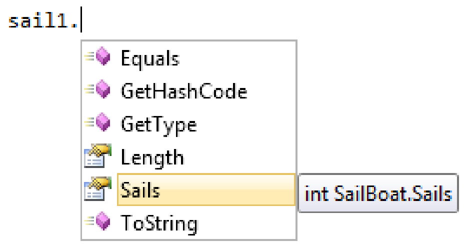
```
Boat boat1 = new PowerBoat();
Boat boat2 = new SailBoat();

PowerBoat power1 = (PowerBoat)boat1;
SailBoat sail1 = (SailBoat)boat2;
```

  - With a derived class reference, all members of the derived class are now accessible…

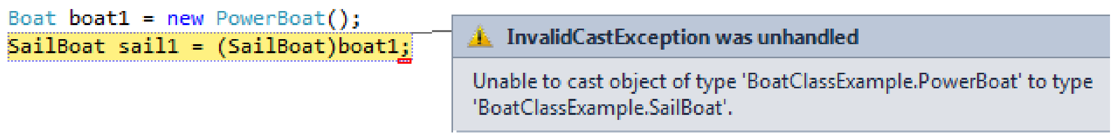


```
power1.
```



```
sail1.
```

- Care must be take when casting – if casted to the wrong derived class type, an exception occurs:



```
Boat boat1 = new PowerBoat();
SailBoat sail1 = (SailBoat)boat1;
```

InvalidCastException was unhandled

Unable to cast object of type 'BoatClassExample.PowerBoat' to type 'BoatClassExample.SailBoat'.

- The ***is*** keyword can be used to test if a particular base class reference is actually a specific derived class…

Example:

```
Boat boat1 = new PowerBoat();
if (boat1 is SailBoat)
{
    SailBoat sail1 = (SailBoat)boat1;
    …
}
else if (boat1 is PowerBoat)
{
    PowerBoat power1 = (PowerBoat)boat1;
    …
}
```

Casting Example

- Starting with the Boat classes from the Polymorphism Example…

```
// declare an array of base class Boats
Boat[] boats = new Boat[4];

// assign derived classes of base class Boat to each element in the array
boats[0] = new PowerBoat();
boats[1] = new PowerBoat();
boats[2] = new SailBoat();
boats[3] = new SailBoat();

// loop through the array
string display = "Boat Array:\n";
for (int index = 0; index < boats.Length; index++)
{
    // determine if current element is a SailBoat object
    if (boats[index] is SailBoat)
    {
        // cast current element to a SailBoat and access .Sails
        SailBoat sail1 = (SailBoat)boats[index];
        display += "\nBoat #" + (index + 1) + " is a SailBoat" +
                   "\nNumber Of Sails: " + sail1.Sails + "\n";
    }
    // determine if current element is a PowerBoat object
    else if (boats[index] is PowerBoat)
    {
        // cast current element to a PowerBoat and access .Engines
        PowerBoat power1 = (PowerBoat)boats[index];
        display += "\nBoat #" + (index + 1) + " is a PowerBoat" +
                   "\nNumber Of Engines: " + power1.Engines + "\n";
    }
}
txtTextbox.Text = display;
```

Casting Example Output:

```
Boat Array:

Boat #1 is a PowerBoat
Number Of Engines: 1

Boat #2 is a PowerBoat
Number Of Engines: 1

Boat #3 is a SailBoat
Number Of Sails: 1

Boat #4 is a SailBoat
Number Of Sails: 1
```