Object-oriented features of C# PL

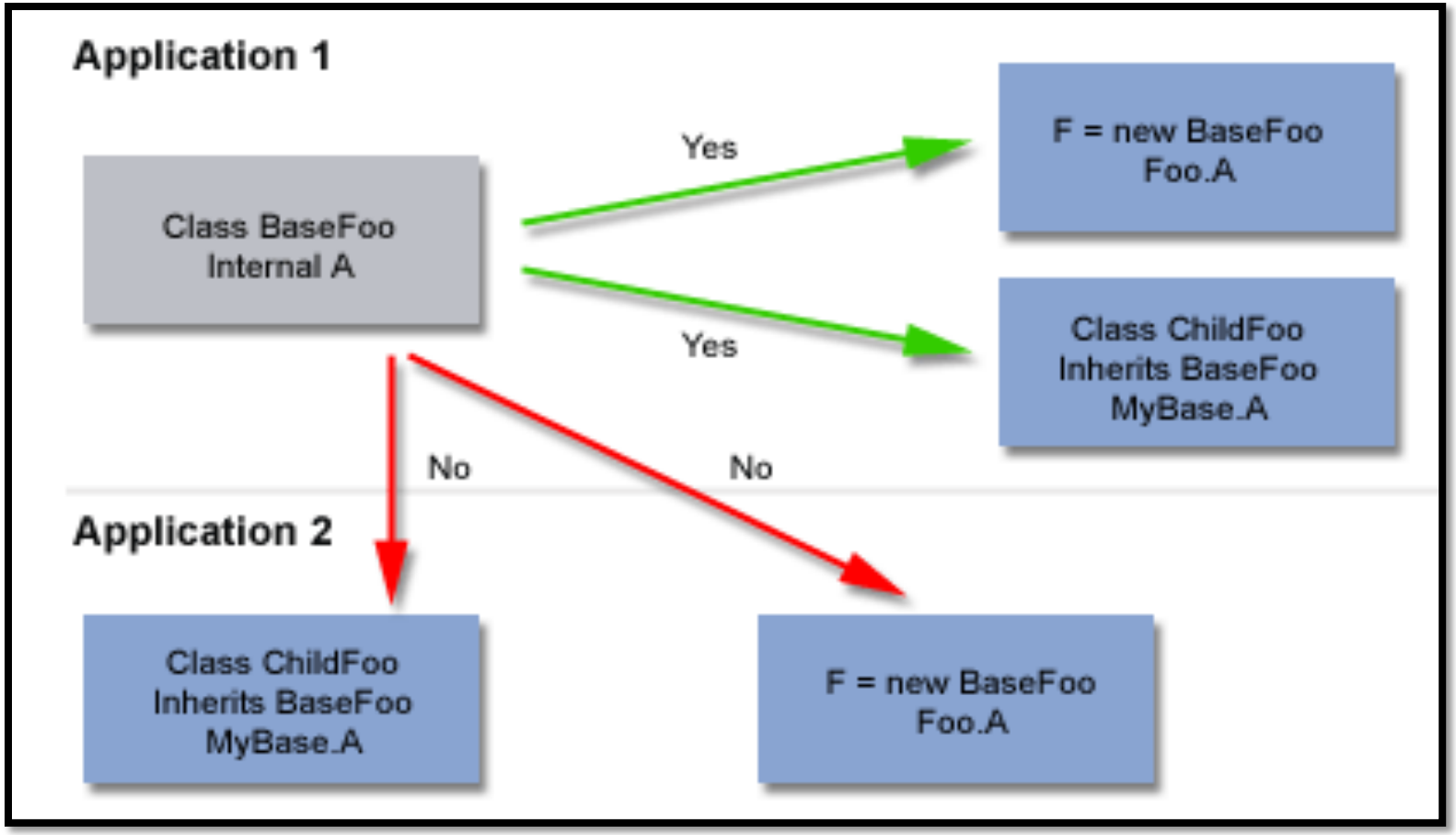
Pakita Shamoi, Spring ~~2018~~

Intro

- Key Concepts of Object Orientation:
  - Abstraction. *Through what ?*
  - Encapsulation. *Through what?*
  - Polymorphism
  - Inheritance.

- C# provides full support for object-oriented programming concepts
  - Let's concentrate on some peculiar ones

Access Modifiers



Access Modifiers

- Besides familiar access modifiers you all know, C# provides **internal** access modifier
- **internal** is public to the entire application but private to any outside applications.

- In java default modifier for fields is … ?
  - In c# it is private

- Difference in meaning of **protected modifier**

Properties

- In c#, besides fields and methods, we also have properties, having **get** and **set** procedures, which provide more control on how values are set or returned.

```
class SampleClass
{
    private int _sample;
    public int Sample
    {
        // Return the value stored in a field.
        get { return _sample; }
        // Store the value in the field.
        set { _sample = value; }
    }
}
```

Java vs. C#

protected Modifier

Java

```
public class A
{
    protected int x;
    static void F(A a, B b) {
        a.x = 1;         // Ok
        b.x = 1;         // Ok
    }
}
public class B extends A
{
    static void F(A a, B b) {
        a.x = 1;         // Ok
        b.x = 1;         // Ok
    }
}
```

C#

```
public class A
{
    protected int x;
    static void F(A a, B b) {
        a.x = 1;         // Ok
        b.x = 1;         // Ok
    }
}
public class B: A
{
    static void F(A a, B b) {
        a.x = 1;    // Error, must access
                    // through instance of B
        b.x = 1;    // Ok
    }
}
```

Inheritance

```
class DerivedClass:BaseClass{}
```

- **base** -> access the members of the base class.
- **this** -> refer to the current object for which a method is called.
- If you override the method of a superclass, you MUST use **override :**
  - public **override** void GetInfo(){…}
- For a class that can not be extended use **sealed** keyword

- Always provide no-arg constructor for your classes

this , base keywords

```
class foo
    {
  public foo(string s) { }
  public foo(string s1, string s2) : this(s1) {}
  }
```

```
public B(int value, int value2): base(value){…}
```

```
class Employee: Person
{
public string id = "ABC567EFG";
public override void GetInfo()
{
// Calling the base class GetInfo method:
base.GetInfo();
Console.WriteLine("Employee ID: {0}", id);
}
```

Virtual, abstract

C# Modifier	Definition
virtual (C# Reference)	Allows a class member to be overridden in a derived class.
override (C# Reference)	Overrides a virtual (overridable) member defined in the base class.
abstract (C# Reference)	Requires that a class member to be overridden in the derived class.

```
public virtual string Name { get; set; }
```

base class

```
public override string Name
{
    get
    {
        return name;
    }
    set
    {
        if (value != String.Empty)
        {
            name = value;
        }
        else
        {
            name = "Unknown";
        }
    }
}
```

derived class

Anonymous types

- Provide a convenient way to encapsulate a set of read-only properties into a single object without having to explicitly define a type first.
- The class has no usable name and contains the properties you specify in declaring the object.

```
var v = new { Amount = 108, Message = "Hello" };
Console.WriteLine(v.Amount + v.Message);
```