Object Oriented C#

The OO features of C# are very similar to Java. We'll point out the exceptions along the way.

Classes & Members

1. A class is defined in C# the same as in Java. The access modifier can be either *public* or *internal*. Internal is similar to package level accessibility in Java. A basic class is declared just like Java:

```
public class Dog {
        ...
}
```

2. The basic members of a class are the same as Java: fields (instance variables), constants, methods, and constructors. In addition, classes can have these additional members: properties, events, finalizers, indexers, operators, and nested types.

3. The accessibility levels for members of a class are: *public, protected, internal, protected internal* (default)*, private*.

   - *protected* – Slightly different meaning in C#. It means the member is only available in subclasses.
   - *internal* – Similar to default access (unspecified) in Java.
   - *protected internal* – Similar to *protected* in Java. C# also defines other accessibility modifiers which we will not

4. The convention in .NET is to use an initial capital letter for class names and all public members of the class, followed by camel-case notation.

Instance Variables & Properties

5. *Properties* are succinct way to define a getter, and a setter for an instance variable. The example below shows how we declare a *name* instance variable in Java and in C# using a property declaration.

Java	C#

`string name;`
`public string getName() {`
`  return name;`
`}`
`public void setName(string name) {`
`  if (name.Length < 1)`
`  this.name = "unknown";`
`  else`
`  this.name = name;`
`}`
	`string name;`
	`public string Name {`
	`  get { return name; }`
	`  set {`
	`  if (value.Length < 1)`
	`  name = "unknown";`
	`  else`
	`  name = value;`
	`  }`
	`}`

Notes:

- *private* is a valid access modifier for class members; however, the default is private so there is not need to declare a member private.

- *value* is a keyword in C# and is used to denote the value being assigned to a property.

- This simplifies access:

```
Dog dog = new Dog();
dog.Name = “Snoopy”;
string name = dog.Name;
```

- If no special processing is needed in the getter and setter, then no instance variable is needed. For example:

```
public int Age { get; set; }
```

- If a property is read-only, the omit the setter:

```
public int Id { get; }
```

- A getter and/or setter can be declared with less accessibility than the property:

```
string name;
public string Name {
	get { return name; }
	protected set {
		...
	}
}
```

Constructors

6. A basic constructor is programmed the same as in Java. The use of *this* to call a constructor in the same class is the same, though the way it is called is slightly different. For example:

```
public class Dog {

	public string Name { get; }
	public int Age { get; set; }

	public Dog(string name) : this(name, 0) {

	}

	public Dog(string name, int age) {
		Name = name;
		Age = age;
	}

	public override string ToString() {
		return "Dog name: " + Name;
	}


}
```

Objects

7. Creating objects is the same as in Java. For example:

```
Dog dog1 = new Dog("Juno");
Dog dog2 = new Dog("Chaps", 6);
```

Methods

8. The basic methods that we write in C# are identical to the way we write them in Java. C# does introduce three modifiers for parameter, *in*, *out* and *ref*, which we will not consider.

9. To override a method, you must use the *override* keyword in the signature. For example:

```
public override string ToString() {
	return "Dog name: " + name;
}
```

C# Collection Classes

10. Comparison between Java & C#

Java	C#
ArrayList<T>	List<T>
LinkedList<T>	LinkedList<T>
HashSet<T>	HashSet<T>
TreeSet<T>	SortedSet<T>
HashMap<K,V>	Dictionary<K,V>
TreeMap<K,V>	SortedDictionary<K,V>
	SortedList<K,V> (similar)

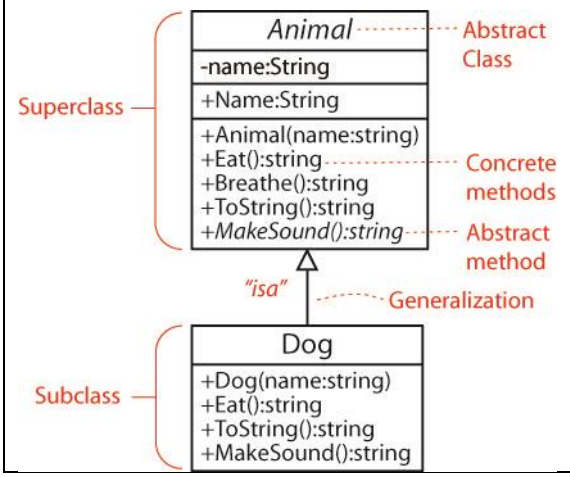
11. Resources:

List<T>	
http://www.csharp-examples.net/list/	Simple example of all List methods
https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.7.2	.NET Api
Dictionary<K,V>	
https://www.tutorialsteacher.com/csharp/csharp-dictionary	
SortedSet<T>	
https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.sortedset-1?view=netframework-4.7.2	.NET Api
https://www.codeproject.com/Articles/86794/SortedSet-Collection-Class-in-NET-4-0	Simple examples

Abstract Classes

12. An example of an abstract class and a subclass.

Design



Class

```
public class Dog : Animal {
    public Dog(string name) : base(name) {
    }

    public override string MakeSound() {
        return Name + " barks";
    }

    public override string Eat() {
        return base.Eat() + " meat";
    }

    public override string ToString() {
        return base.ToString() +
                    " and I am a Dog";
    }
}
```

Abstract Class

```
public abstract class Animal {
    private string name;

    public string Name {
        get { return name; }
        set {
            if (value.Length < 1)
                this.name = "unknown";
            else
                this.name = value;
        }
    }

    public Animal(string name) {
        Name = name;
    }

    public abstract string MakeSound();

    public virtual string Eat() {
        return Name + " eats";
    }

    public string Breathe() {
        return Name + " breathes";
    }

    public override string ToString() {
        return "My name is " + Name;
    }
}
```

Test Code

```
Dog d = new Dog("Max");
Animal a = new Dog("GiGi");
string msg = d + "\n" + d.MakeSound()
+"\n";
msg += a + "\n" + a.MakeSound() + "\n";
txtMsg.Text = msg;
```

13. Notes on abstract classes:

    a. As with Java, an abstract class is declared *abstract* and abstract methods are also declared *abstract*.

```
public abstract class Animal {
    ...
    public abstract string MakeSound();
```

b. Unlike Java, a method (in an abstract class or concrete class) cannot be overridden unless it is declared *virtual*.

```
public virtual string Eat() {
    return Name + " eats";
}
```

c. Though not shown above, an abstract class can declare abstract properties.

d. The *base* keyword is used like *super* in Java. It can be use to call a superclass constructor or to call a super class method that is overridden.

```
public Dog(string name) : base(name) {
}


public override string Eat() {
   return base.Eat() + " meat";
}
```

e. As with Java, C# allows only single inheritance. The *colon* (:) indicates that one class extends (in the Java sense) another class.

```
public class Dog : Animal
```

f. The implementation of abstract methods must be declared *override.* Similarly, a method that overrides a virtual method must be declared *override*

```
public override string MakeSound() {
    return Name + " barks";
}
```

g. As with Java, the reference type for an object can be the class itself or any superclass.

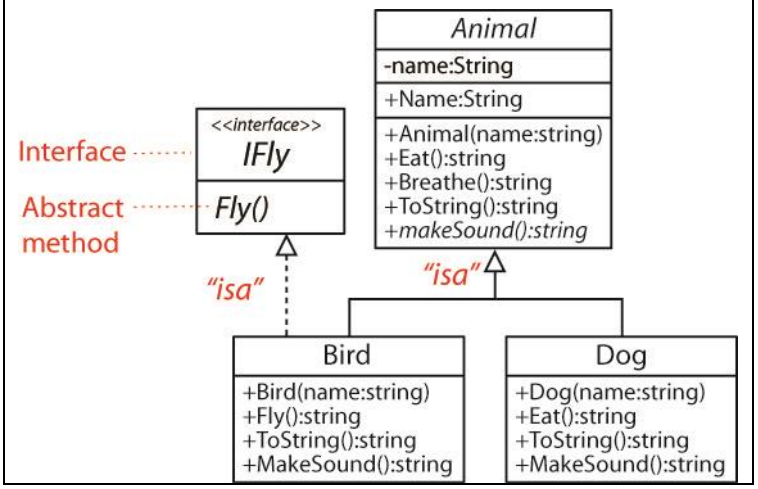
```
Dog d = new Dog("Max");
Animal a = new Dog("GiGi");
```

a. A C# class can be *sealed* which means it cannot be extended (in the Java sense, *e.g.* the same thing as *final*). A method is final (in the Java sense) if is NOT declared *override*. For example, the *Breathe* method in the abstract *Animal* class cannot be extended.

Interfaces

14. An example of an interface and implementing class.

Design



Test Code

```
Bird b = new Bird("Tweety");
msg = b + "\n" + b.MakeSound() +
                 "\n" + b.Fly();
txtMsg.Text += msg;
```

Interface

```
public interface IFly {
       string Fly();
}
```

Implementing Class

```
public class Bird : Animal, IFly {
    public Bird(string name) : base(name) {
    }

    // Implement abstract method
    public override string MakeSound() {
        return Name + " chirps";
    }
    // Implement interface method
    public string Fly() {
        return Name + " flys";
    }

    public override string ToString() {
        return base.ToString() +
                     " and I am a Bird";
    }
}
```

15. Notes on interfaces.

    a. An interface in C# is identical to one in Java; however, C# allows properties to be specified in an interface (not shown in the example).

    b. Like Java, C# allows a class to extend another class and implement any number of interfaces. In this case, we specify the class like this:

```
public class SubClass : SuperClass, Interface1, Interface2, …
```

    Or, if there is no superclass:

```
public class MyClass : Interface1, Interface2, …
```

    c. Implemented methods in the implementing class can NOT be declared *override*.

```
public string Fly() {
    return Name + " flys";
}
```

    Remember, the implementation of an abstract method defined in an abstract class must be declared *override*.

http://www.csharp-examples.net/	Great site with examples of some API classes: string formatting, files & folders, xml, reflection, linq, threading