432 review

OOP Concepts:
Abstraction  Abstract Data Type – combine variables & methods
Encapsulation  accessibility, properties, indexers, delegates, events
Inheritance
   Polymorphism: classes and interfaces
      overriding Vs virtual methods
   Interfaces: design by contract, “mixins”
      Abstract methods Vs Interfaces
   Multiple Inheritance: name collision (search path) Vs Interfaces

OOP languages:  dynamic Vs static typing
   Smalltalk: class / method browser, inspector, blocks (anon method)
   C#: safer C++, single inheritance, properties, windows / web
   Python: class method self reference – def methodName(self, args)

OOP class libraries:
   Collections: dynamic storage with iterators, indexers
   GUI: events, containment, layout mgmt, Model / View
Type membership

Class A

Class B

Class C

Interface A

Class D

Class A

Interface A

Collections of

as ClassA
aClassB
aClassC

as InterfaceA
aClassC
aClassD

object aClassC is-a ClassA and is-a InterfaceA
Collections

Smalltalk collections are data structures (there are more than shown)
Sorted Collection

SortedCollections maintain a sort order based on their sort block. The sort block can be set with the sortBlock: class method.

```smalltalk
Object subclass: #SortableObject
    instanceVariableNames: 'sortOn '
    ...

new: value
    ^super new initialize: value

initialize: value
    sortOn := value.

getSortValue
    ^sortOn
```

```smalltalk
SC := SortedCollection new.
SC sortBlock: [ :x :y |
    x getSortValue <= y getSortValue ].
SO2 := SortableObject new: 10.
SO3 := SortableObject new: 15.
SC add: SO1.
SC add: SO2.
SC add: SO3.
```
C# Collection interfaces

IEnumerable      iterate / enumerate through collection
                using foreach statement

ICollection     provides CopyTo(), and Count, ISReadOnly,
                ISSynchronized and SyncRoot properties

IComparer        compares two objects in collection for sorting

IList           used by array-indexable collections

IDictionary      used by key / value based collections

IDictionaryEnumerator allow foreach use with Dictionaries

List<AnObject> list = new List<AnObject>();
// add anObjects to list
list.Add(new AnObject(...));
...
foreach(AnObject ao in list)
   Console.WriteLine(ao.ToString());
Sortable Object

public int CompareTo(Object rhs) {
    SortableObject so = (SortableObject) rhs;
    return SortOn.CompareTo(so.SortOn);
}
Event Driven apps

Running programs can be "interrupted" by various "events": exceptions and events.

Exception and event handling changes normal execution flow.

Exceptions usually represent an error condition or unmet system request and provide a mechanism for the application to recovery and continue. Usage generates robust applications.

Events are usually generated by the user via the GUI. There are internal events like timers(animations) that are also available.
Graphical User Interface programming uses Morphic classes.

GUI applications are event driven.

Hello432 example.

graphical container, halo
edges and shape
user interactivity—events
application morph
Class Hierarchy

Object

MarshalByRefObject

Component

Control

ScrollableControl

ContainerControl

Form

Object

remote object reference across application domains (IPC)

components

visual components, size, UI events

scroll capability

Container, focus & component control

Basic container for apps & dialogs
Property
Properties are "lite, nested classes" that provide accessor / mutators methods.

+ data object is hidden
+ data usage appears to be direct.

```
anObject.Property = aValue;
```
```
aValue = anObject.Property;
```

Indexer Property

```
class AClass {
    private int [,] storage;
    public AClass(int i, int j) {
        storage = new int[i, j]; }

    public int this [int i, int j] {
        get {return storage[i, j]; }
        set {storage[i, j] = value; } }
}
```

```java
AClass aClass;
...
aClass[10, 20] = 30;
```
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>title</td>
</tr>
<tr>
<td>ForeColor</td>
<td>foreground color</td>
</tr>
<tr>
<td>BackColor</td>
<td>background -- also BackgroundImage</td>
</tr>
<tr>
<td>Width, Height</td>
<td>size</td>
</tr>
<tr>
<td>Cursor</td>
<td>see Cursors class for static values</td>
</tr>
<tr>
<td>StartPosition</td>
<td>Center.Parent, Center.Screen, Manual, WindowsDefaultBounds,</td>
</tr>
<tr>
<td></td>
<td>WindowsDefaultLocation (default).</td>
</tr>
<tr>
<td>Anchor</td>
<td>Bottom, Left, None, Top, Right</td>
</tr>
<tr>
<td></td>
<td>(Top, Left default)</td>
</tr>
<tr>
<td>Bounds</td>
<td>location and size of control</td>
</tr>
<tr>
<td>ClientSize</td>
<td>Area of client (! TitleBar and Borders)</td>
</tr>
<tr>
<td>ClientRectangle</td>
<td>Area of client (! title, menu, borders, scrollbars)</td>
</tr>
<tr>
<td>Icon</td>
<td>image used in task bar and control box</td>
</tr>
<tr>
<td>Parent</td>
<td>The container control of the current control</td>
</tr>
<tr>
<td>Owner</td>
<td>Owned forms min/max w/ &amp; on top of Owner</td>
</tr>
<tr>
<td>Location</td>
<td>position of origin</td>
</tr>
</tbody>
</table>
Delegates, events

Delegates and events (ADTs) – declared and instantiated can’t be derived.

Delegates encapsulate a method – a type safe pointer to method

Events are delegates invokable only by their defining class and subscribed to, or unsubscribed to, by other classes with += and -=

The .Net framework event handler methods associated with events by convention have the following form:

```csharp
access void eventHandlerName
    ( object sender, EventArgs eventArg) {
        ... }
```

EG: cancelButton.Click += new EventHandler(doCancelEvent);
Use Static show methods
Can add to controls using Designer
Use properties to get values

Open dialogs in event handler method

```csharp
DialogResult dr = 
aDialog.ShowDialog();
if (dr == DialogResult.OK)
    // get property of
    // aDialog
```

Menus have similar class hierarchy
Generics, Serialization, Reflection

Generics (C++ templates) abstract (factor out, or parameterize) data types for classes and methods. Substitutes type @ compile time.

Serialization is the process of converting the state of an object into a form that can be persisted (local disk) or transported (distributed apps).

- persistence – binary serialization – disk format
- transportation – SOAP (Simple Object Access Protocol) XML based protocol / format for web use (http, smtp)

Threads abstracts thread creation and synchronization

```java
Thread aThread = new Thread (new ThreadStart (aMethod));
```

Reflection (IRellect, MemberInfo) enables run-time determination of object information

- EG: Object has method `public Type GetType();`

OOP compilation – creates implicit arguments, variables, VMT

```java
type methodName([type_i*, arg_i*]) {...}
```

converted by compiler to

```java
type methodName(ClassName this, [type_i*, arg_i*]) {...}
```
Mobile Devices
what to do next wrt OOP

Practise, practise, practise… design && implementation

Design as a verb: explore many problem solving techniques:
    simulate w/ game | objects … SimGame
    CRC index card sides: private members | public methods, associates

Design as a noun: document results of problem solving
    UML diagrams: class | object, use cases, sequence, state diagrams

Experiment w/ new languages: Ruby, Effiel "only change is constant"

CSUN elective courses using OOP – others ??

586 Object Oriented Software Development
585 Graphical User Interfaces (Java Swing / C# Windows Forms)
480/L Software System Development (depends on project)
496 ebt E Business Technology (depends on tools used ???)
565 Advanced computer graphics (1/4 C# / DirectX, JOGL)