PV178: Programming for the CLI Environment

Seminar: Week 5

Tomáš Pochop

Institute of Computer Science and **Faculty of Informatics**
Masaryk University

April 2, 2007

Example 1

Write a class *BinaryHeap* that represents a binary heap. Nodes in heap are of the same generic type T, where T implements the $IComparable < T >$ interface. The *BinaryHeap* class is serializable. The binary heap is represented by an array, allow to set the length of the array via constructor. Implement all heap operations. Create one instance of *BinaryHeap* class of some type (e.g. *int*), fill it with some values. Demonstrate serialization and deserialization.

Binary heap

Binary heap is a binary tree with two additional constraints:

1. The shape property: the tree is either a perfectly balanced binary tree (all leaves are at the same level), or, if the last level of the tree is not complete, the nodes are filled from left to right.
2. The heap property: each node is higher than or equal to each of its children according to some comparison predicate which is fixed for the entire data structure.

Binary heap operations

1. Add an item to the heap
2. Remove the root from the heap

Binary heap - adding

1. Add the element on the bottom level of the heap.
2. Compare the added element with its parent; if they are in the correct order, stop.
3. If not, swap the element with its parent and return to the previous step.

Binary heap - removing

1. Remove the root.
2. Replace it with the last element on the last level.
3. Compare the replacing element with its children; if they are in the correct order, stop.
4. If not, swap the element with the bigger child and return to the previous step.

IComparable<T> interface

Defines a generalized comparison method that a value type or class implements to create a type-specific comparison method for ordering instances. One method – `public int CompareTo(T` *otherObject*)

1. Compares the current object with another object of the same type.
2. Returns 0 if this object is equal to other.
3. Less than zero if this object is less than the other parameter.
4. Greater than zero if this object is greater than other.

Type parameters

- the generic types are defined with type parameters

```
class List <T> { T method() {}}
```

- when used, type parameters are substituted with type arguments

```
List<int> lint = new List<int>
```

- by instantiation a type is constructed substituting type arguments in all occurrences of the type parameter

```
class List<int> {int method() {}
```

Constraints on Type parameters

- can apply restrictions on types that can be used as type parameters
- `where` keyword
- five types of constraints:
  - where T : struct value type (except Nullable)
  - where T : class reference type
  - where T : new() must have a public parameterless constructor (must be specified last)
  - where T : <base class name> must be or derive from the specified class.
  - where T : <interface name> must be or implement the specified interface (multiple can be specified, can also be generic)
  - where T : U must be or derive from the argument supplied for U (naked type constraint)

Serialization

1. default serialization – objects marked with `[Serializable]` attribute
2. explicit serialization – objects implementing `ISerializable` interface
3. Class `BinaryFormatter` serializes and deserializes an object, or an entire graph of connected objects, in binary format
4. `void BinaryFormatter.Serialize(Stream` *stream*`, Object` *object*`)` - serializes an object, or graph of connected objects, to the given stream.
5. `Object BinaryFormatter.Deserialize(Stream` *stream*`)` - deserializes a stream into an object graph

Explicit serialization - ISerializable interface

Only one method:
`void ISerializable.GetObjectData(SerializationInfo` *si*`,`
`StreamingContext` *sc*`)`

1. this method is called by serialization
2. `SerializationInfo` carries data to be serialized
3. `StreamingContext` describes the source and destination of a given serialized stream, and provides an additional caller-defined context.
4. `si.AddValue(string` *name*`, Object` *value*`)` adds the specified object into the `SerializationInfo` store, where it is associated with a specified name. (Overloaded for more types.)

Explicit deserialization

The class must implement a special constructor which is used by deserialization.

1. `MySerializableClass(SerializationInfo` *si*`, StreamingContext` *sc*`){ ... }`
2. `int si.GetInt32(string` *name*`)` returns the int value associated to the name
3. `Object si.GetValue(string` *name*`, Type` *type*`)` returns the Object associated to the name, *type* is the type of the value to retrieve (the `System.Type typeof(`*sometype*`)` operator can be used to obtain the type)

Namespaces

1. System.Collections.Generic
2. System.Runtime.Serialization
3. System.Runtime.Serialization.Formatters.Binary
4. System.IO