A case study of delegates and generics in C#

Hans-Wolfgang Loidl
<hwloidl@macs.hw.ac.uk>

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 1 2017/18

Motivation

- As a case study, we want to implement a sequential *quick-sort* algorithm.
- The idea of quick-sort is:
  - Pick a *pivot* element in the list
  - *Partition* the list into elements *greater than* and elements *less than or equal* to the pivot element.
  - *Recursively* sort the two partitions.
- See this handout on quick-sort with examples[1]
- We will develop three versions:
  1. The first version operates on arrays of integers, with a fixed comparison operation.
  2. The second version uses *delegates*, so that we can *re-define the comparison operation.*
  3. The third version additionally uses *generics*, so that we can *sort values other than integers.*

[1] From Goodrich & Tamassia, "Data Structures & Algorithms" in Java

Components of in-place quicksort

We need the following functions

- A top-level function that performs sorting of an array:

```
1  private void QuickSort(int[] array, int from,
       int to, int level)
```

- A function that *partitions* a segment of the array:

```
1  private int Partition(int[] array, int from, int
       to, int pivot)
```

- A swap function that exchanges two array elements (in-place):

```
1  private static void Swap(int[] array, int i, int
       j)
```

This initial implementation fixes the base type (`int`) and the comparison fct (`>`).

Structure of in-place quicksort

```
1  private void QuickSort(int[] array, int from, int to
       , int level) {
2  if (to - from < 1) {
3     return;      // a 1 elem list is sorted, per
         definitionem
4   } else {
5     int pivot = from + (to - from) / 2;
6     pivot = Partition(array, from, to, pivot);
7     // recursive call on lower segment
8     QuickSort(array, from, pivot - 1, level+1);
9     // assert: IsSorted(array, from, pivot)
10    // recursive call on upper segment
11    QuickSort(array, pivot + 1, to, level+1);
12    // assert: IsSorted(array, pivot+1, to)
13  }
14 }
```

Partitioning (idea)

- The interface of the partitioning function is:

```
private int Partition(int[] array, int from, int
    to, int pivot)
```

- It separates all elements less than the pivot element, from those larger than pivot element.
- The implementation uses the `from` and `to` variables as cursors, iterating over the array, and exchanging (swapping) data where needed.

Partitioning

```
private int Partition(int[] array, int from, int to,
    int pivot) {
  int? pividx = null; int pivval = array[pivot];
  while (from<to) {
    if (array[from] > pivval) {
      Swap(array, from, to);     to--;
    } else {
      if (array[from] == pivval) { pividx = from; }
      from++;
    }
  }
  if (!pividx.HasValue) {
    if (array[from] == pivval) {
      return from;
    } else {
      throw new System.Exception(String.Format("
          Partition: pivot element {0} not found",
          pivval));
    }
```


Partitioning (cont'd)

```
  if (array[from] > pivval) {
    // bring pivot element to end of lower half
    Swap(array, (int)pividx, from-1);
    return from-1;
  } else {
    // done, bring pivot element to end of lower half
    Swap(array, (int)pividx, from);
    return from;
  }
}
```


Swapping

Swapping the i-th with the j-th element in the array:

```
private static void Swap(int[] array, int i, int j){
  if (i==j) {
    return;
  } else {
    int tmp = array[i];
    array[i] = array[j];
    array[j] = tmp;
  }
}
```

Delegate version: class definition

```
public class GenSort {

  // enumeration for results of a generic comparison
     function
  public enum Cmp { GT, EQ, LT };

  // this delegate defines a compaison operator over
     the list elements
  // see also (using System): public delegate int
     Comparison<in T>
  public delegate Cmp CmpDelegate(int x, int y);

  // the actual function to compare something
  private CmpDelegate the_cmp;

  public GenSort(CmpDelegate cmp) {
    the_cmp = cmp;
  }
```


Delegate version: partitioning

```
private int Partition(int[] array, int from, int to,
   int pivot) {
  int? pividx = null; int pivval = array[pivot];
  while (from<to) {
    if (the_cmp(array[from], pivval) == Cmp.GT) { //
       using DELEGATE
      Swap(array, from, to);
      to--;
    } else {
      if (the_cmp(array[from], pivval) == Cmp.EQ) {
         pividx = from; }
      from++;
  }  }
  if (!pividx.HasValue) {
    if (the_cmp(array[from], pivval) == Cmp.EQ) {  //
       using DELEGATE
      return from;
    } else {
      throw new System.Exception(String.Format("
         pivval));
  } }
```


Delegate version: paritioning (cont'd)

```
  if (!pividx.HasValue) {
    if (the_cmp(array[from], pivot_val) == Cmp.EQ ) {
      return from;
    } else {
      throw new System.Exception(String.Format("
         Partition: pivot element {0} not found",
         pivot_val));
  }  }
  if (the_cmp(array[from], pivot_val) == Cmp.GT) {
    // bring pivot element to end of lower half
    Swap(array, (int)pividx, from-1);
    return from-1;
  } else {
    // bring pivot element to end of lower half
    Swap(array, (int)pividx, from);
    return from;
  }
}
```


Using this version

```
public class Tester {
  ...
  // my sorting function: ascending order of integers
  public static GenSort.Cmp intCmp(int x, int y) {
    if (x>y)       return GenSort.Cmp.GT;
    else if (x==y) return GenSort.Cmp.EQ;
    else           return GenSort.Cmp.LT;
  }

  public static void Main(string []args) {
    ...
    // inst. of sorter class, using int comp.
    GenSort myGenSort = new GenSort(intCmp);
    ...
    myGenSort.QuickSort(arr);
    ...
```

[1] http://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/Samples/GenSort.cs

Generics version: class definition

```
public class GenSort<T> {

  // enumeration for results of a generic comparison function
  public enum Cmp { GT, EQ, LT };

  // this delegate defines a predicate over the list elemnts; mainly for testing
  // see also (using System): public delegate int Comparison<in T>
  public delegate Cmp CmpDelegate(T x, T y);

  // the actual function to compare something
  private CmpDelegate the_cmp;

  public GenSort(CmpDelegate cmp) {
    the_cmp = cmp;
  }
```


Generics version: partitioning

```
private int Partition(T[] array, int from, int to, int pivot) {
  int? pividx = null; T pivval = array[pivot];
  while (from<to) {
    if (the_cmp(array[from], pivval) == Cmp.GT) { // using DELEGATE
      Swap(array, from, to);
      to--;
    } else {
      if (the_cmp(array[from], pivval) == Cmp.EQ) {
        pividx = from; }
      from++;
    }
  }
  if (!pividx.HasValue) {
    if (the_cmp(array[from], pivval) == Cmp.EQ ) {
      return from;
    } else {
      throw new System.Exception(String.Format("
          pivval));
    }
```


Generics version: partitioning

```
    if (the_cmp(array[from], pivval) == Cmp.GT) {
      // bring pivot element to end of lower half
      Swap(array, (int)pividx, from-1);
      return from-1;
    } else {
      // done, bring pivot element to end of lower half
      Swap(array, (int)pividx, from);
      return from;
    }
  }
```


Using this Generics version

```
public class Tester {
 ...
 // my sorting function: ascending order of integers
 public static GenSort<int>.Cmp intCmp(int x, int y) {
   if (x>y)       return GenSort<int>.Cmp.GT;
   else if (x==y) return GenSort<int>.Cmp.EQ;
   else           return GenSort<int>.Cmp.LT;
 }
 public static void Main(string []args) {
  ...
  // inst of sorter class, using int comp.
  GenSort<int> myGenSort = new GenSort<int>(intCmp);
  ...
  myGenSort.QuickSort(arr);
```

[1] http://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/Samples/GenGenSort.cs

Summary

- Delegates and Generics are powerful mechanisms of abstraction.
- Delegates *abstract over code*, allowing to pass methods as arguments to other methods.
- The keyword `delegate` is used to define the interface for the parametric method.
- Generics *abstract over types*, allowing to define data-structures over different types.
- The notation `<T>` is used to specify a type argument to a class definition.