Assignment 8

Exercise 9.1

Consider the following C# method from file Selsort.cs:

```
public static void SelectionSort(int[] arr) {
    for (int i = 0; i < arr.Length; i++) {
      int least = i;
      for (int j = i+1; j < arr.Length; j++)
        if (arr[j] < arr[least])
          least = j;
      int tmp = arr[i]; arr[i] = arr[least]; arr[least] = tmp;
    }
}
```

(i)

Description

From a Visual Studio Command Prompt, compile it using Microsoft's C# compiler with the optimize flag (/o), then disassemble it, saving the output to file Selsort.il:

```
 csc /o Selsort.cs
  ildasm /text Selsort.exe > Selsort.il
```

Open Selsort.il in a text editor, find method SelectionSort and its body (bytecode), and delete everything else. Now try to understand the purpose of each bytecode instruction. Write comments to the right of the instructions (or between them) as you discover their purpose. Also describe which local variables in the bytecode (local 0, 1, ...) correspond to which variables in the source code.

To see the precise description of a .NET Common Language Infrastructure bytecode instruction such as ldc.i4.0, consult the Ecma-335 standard [10], find Partition III (PDF pages 324-471 in the December 2010 version) of that document, and search for ldc.

Solution

The solution can be found in *Selsort.il*

(ii)

Description

Now do the same with the corresponding Java method in file Selsort.java. Compile it, then disassemble the Selsort class: javac Selsort.java javap -verbose -c Selsort > Selsort.jvmbytecode Then investigate and comment Selsort.jvmbytecode as suggested above. For the precise description of JVM bytecode instructions, see [17, Chap. 6].

Hand in the two edited bytecode files with your comments.

Solution

The solution can be found in *Selsort.jvmbytecode*

Exercise 9.3

Description

For this exercise we have to find an issue in the `QueueWithMistake.java"

Solution

The solution can be found in QueueWithMistake.java

The main problem was identified in the SentinelLockQueue.get() method, which was previously

```
public synchronized int get() {
    if (head.next == null)
        return -999;
    Node first = head;
    head = first.next;
    return head.item;
}
```

We have however modified it to be:

```
public synchronized int get() {
    if (head.next == null)
        return -999;
    Node first = head; //Current head
    head = head.next; //Set head to the next head
    first.next = null; //Ensure that the "previous" head does not continue
pointing to the new head
    return head.item; //Return new head.item
}
```