Software Workshop 1 - Lecture 1

Ossama Edbali

January 17, 2015

1 Module outline

In this module we are going to cover the theoretical aspects taught in the FOCS module and implement them using Java.

There will be 3 exercise in week 2, week 3 and week 6. Some material is mandatory in order to understand topics explained the Robot Programming module such as A* search.

2 Lists using the composite pattern

In FOCS 1 we defined lists as follows:

```
type 'a list = Nil | Cons of ('a * 'a list)
```

In Java we have the same pattern (though not strictly recursive) but using interfaces and classes (for polymorphism there are generics). Note that we are dealing with **immutable lists**.

Here I will just define the interface for the two implementations (Nil and Cons):

```
1  package src;
2
3  public interface MList<E>
4  {
5          public boolean isEmpty();
6          public int size();
7          public MList<E> reverse();
8          public MList<E> append(MList<E> l);
9          public MList<E> append(E el);
10         public boolean has(E el);
11 }
```

The implementation of the two constructors (composite pattern) is left as an exercise (see *Lec1* folder).

3 Notes on memory management

Java handles its memory in two areas: the *heap* and the *stack*. Java objects reside in an area called the heap. The heap is created when the JVM starts up and may increase or decrease in size while the application runs. When the heap becomes full, garbage is collected. During the garbage collection objects that are no longer used are cleared, thus making space for new objects.

When we create an object we are indeed creating a reference to a memory location. Thus, when comparing two objects we are actually comparing their addresses and not internal structure.

Stack is where the method invocations and the local variables are stored. If a method is called then its stack frame is put onto the top of the call stack. The stack frame holds the state of the method including which line of code is executing and the values of all local variables. The method at the top of the stack is always the current running method for that stack.