CS4/MSc Distributed Systems
Tutorial 6: Leader Elections in Java

Stephen Gilmore
stg@dcs.ed.ac.uk

Tutorial: M 12, Room 5325
Office hour: Th 2, Room 1608

Yussuf Abu-Shaaban
yna@dcs.ed.ac.uk

Tutorial: M 12, Room 6207
Office hour: M 4, Room 3311

Leader Elections in Java

- The leader election simulator in Java provides classes which are needed to complete the implementation of the practical.
- We discuss the classes `Node` and `Message` below.
- The network can be seen as Java vectors of nodes which can be compared. We discuss the `java.util.Vector` class and the `java.lang.Comparable` interface.

The Node Interface

```java
import java.util.Vector;
interface NodeInterface {
    public void receiveMessage(Message msg);
    public Vector neighbours();
    public Vector allNodes();
    public int getId();
    public boolean isFailed();
    public void changeGraphicalState();
    public void discardMessage(Message msg);
    public void setFailed();
}
```

- The simulator's `Node` class implements the interface above.

The methods in the `Node` class

- **int** `getId()`, returns the Id of this `Node` object.
- **Vector** `neighbours()`, returns all the `Node` objects that this `Node` can send messages to (i.e. there is a unidirectional channel from this `Node` to each of the returned `Node` objects.
- **Vector** `allNodes()`, returns all the `Node` objects in the network.
- **void** `receiveMessage(`**Message** `msg)`, called whenever the `Node` is sent a message. Currently this method contains an empty statement. *You should override this method in your implementation.*
- **void** `discardMessage(`**Message** `msg)`, removes a `Message` object from the network.
- **void** `changeGraphicalState()`, changes the colour of the graphical representation of this `Node` to red.
- **boolean** `isFailed()`, returns **true** if this `Node` object is in a *failed* state.

Vectors in Java

```java
import java.util.*;
class VectorExample {
    public static void main (String[] args) {
        Vector v = new Vector();
        for (int i = 0 ; i < args.length ; i++) {
            v.addElement(new Integer(i));
            v.addElement(args[i]);
        }
        for (Enumeration e = v.elements();
             e.hasMoreElements(); ) {
            Object o = e.nextElement();
            if (o instanceof Integer)
                System.out.println((Integer) o);
            if (o instanceof String)
                System.out.println((String) o);
        }
    }
}
```


The Comparable interface

- Nodes are *comparable*. The interface `java.lang.Comparable` imposes a total ordering on the objects of each class that implements it. This ordering is referred to as *the class's natural ordering*, and the class's `compareTo()` method is referred to as its *natural comparison method.*
- Lists (and arrays) of objects that implement this interface can be sorted automatically by `Collections.sort` (and `Arrays.sort`). Objects that implement this interface can be used as keys in a sorted map or elements in a sorted set, without the need to specify a comparator.
- The natural ordering of class `C` is said to be *consistent with equals* if and only if `(e1.compareTo((Object)e2)==0)` has the same boolean value as `e1.equals((Object)e2)` for every `e1` and `e2` of class `C`.

The Message Interface

```java
interface MessageInterface {
    public String getType();
    public int getValue();
    public Node getSender();
    public Node getReceiver();
    public void send();
    public int getId();
}
```


The methods in the `Message` class

- **public** `Message(`**String** `type,` **int** `value,` **Node** `sender,` **Node** `receiver)`, a constructor to create a `Message` object specifying the `type` of the message, its `value`, its `sender` and its `receiver`.
- **String** `getType()`, returns the type of this `Message`.
- **int** `getValue()`, returns the value stored in this `Message`.
- **Node** `getSender()`, returns the sender `Node` object.
- **Node** `getReceiver()`, returns the receiver `Node` object.
- **void** `send()`, causes the `receiveMessage()` method of the `Node` object `receiver` to be invoked. Graphically, the message will travel from the sender node to the receiver node.