```java
package ts05.sensor;

import java.rmi.Remote;
import java.rmi.RemoteException;


/**
    Event object holding a single temperature reading broadcast
    from a temperature sensor

    @author Henrik Bærbak Christensen - (c) Imhotep 2003

*/


public interface TemperatureEvent extends java.io.Serializable {

  /** return the temperature reading that this event represents.
      @return the temperature reading measured in Celcius
  */
  public double getReading();
}
```

```java
package ts05.sensor;

import java.rmi.Remote;
import java.rmi.RemoteException;


/**
    Event object implementation holding a single temperature reading.
    This (immutable object) is sent to every listener that listens to
    a TemperatureSensor whenever the sensor has sampled a new
    temperature reading.

    @author (c) Henrik Bærbak Christensen - Imhotep 2003

*/

public class TemperatureEventImpl implements TemperatureEvent {

  /** the temperature reading */
  private double reading;

  /** construct an immutable temperature event holding the
      temperature given as parameter.
      @param reading the temperature in Celcius
  */
  public TemperatureEventImpl(double reading) {
    this.reading = reading;
  }

  public double getReading() { return reading; }
}
```

```
package ts05.sensor;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
    An object that is interested in receiving temperature data must
    implement this interface, and register itself as listener on
    a temperature sensor.

    <p>
    Any object implementing this interface acts as the 'observer'
    role from the GoF observer pattern.

    @author Henrik Bærbak Christensen / (c) Imhotep 2003

*/


public interface TemperatureListener extends Remote {

  /** this method is invoked every time a temperature sensor
      broadcasts a temperature
      @param t_event an object encapsulating the temperature value
      sampled.
  */
  public void temperatureSampled( TemperatureEvent t_event )
    throws RemoteException;
}
```

```
package ts05.sensor;

import java.rmi.Remote;
import java.rmi.RemoteException;


/**
    This interface represents the contract of a simple
    temperature sensor in the field.

    <p> The sensor is modelled as a separate, distributed, process
    that continuously broadcasts temperature values to any interested
    entity. It acts as the 'subject' role in the GoF observer pattern.

    <p>
    An entity register its interest in temperature data by
    adding itself as recipant of temperature data. It must
    do so by implementing the <tt>TemperatureListener</tt> contract,
    and register by calling the <tt>addTemperatureListener</tt> method.

    @author Henrik Bærbak Christensen / (c) Imhotep 2003

    @see TemperatureListener

*/


public interface TemperatureSensor extends Remote {
  /** register a listener to receive temperature data.
      @param tl the temperature listener instance to broadcast to.
  */
  public void addTemperatureListener( TemperatureListener tl )
    throws RemoteException;
}
```

```
package ts05.sensor;

import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

/**
    Temperature Sensor implementation.
    Implemented in its own thread.

    @author (c) Imhotep 2003 - Henrik Bærbak Christensen

*/

public class TemperatureSensorImpl extends UnicastRemoteObject
  implements TemperatureSensor, Runnable {

  private List listenerList;

  public TemperatureSensorImpl() throws RemoteException {
    super();
    listenerList = new Vector();
  }

  private double temperature = 80.0;

  public void run() {
    while ( true ) {
      // wait 0.1 sec
      try {
        Thread.sleep(100);
      } catch ( InterruptedException e ) {}
      // make a new temperature reading
      temperature += Math.random()-0.49;
      notifyAllListeners();
    }
  }

  public synchronized void addTemperatureListener( TemperatureListener tl )
    throws RemoteException {
    listenerList.add( tl );
  }

  private int notifications=0;
  private void notifyAllListeners() {
    TemperatureEvent te = new TemperatureEventImpl(temperature);
    Iterator i = listenerList.iterator();
    while ( i.hasNext() ) {
      TemperatureListener tl = (TemperatureListener) i.next();
      try {
        tl.temperatureSampled( te );
      } catch ( RemoteException exc ) {
        System.out.println( ""+exc );
        System.exit(1);
      }
    }
    System.out.println( "Broadcasted temperature "+temperature );

  }

}
```

```
package ts05.sensor;

import java.rmi.*;
import java.rmi.server.*;


/**
    The Temperature server models a temperature sensor in the
    field. It instantiates a temperature sensor object, binds it
    to the name registry, and runs the sensor as a thread.

    @author Henrik Bærbak Christensen - (c) Imhotep 2003

*/

public class TemperatureSensorServer {

  public static void main(String[] args) {
    // define where the rmi registry is located...
    String registry_host = "//localhost/";
    System.out.println( "Using registry at "+registry_host );
    try {
      System.out.println( "Initializing ..." );
      if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
      }
      System.out.println( "SecurityManager installed..." );

      // create a temperature sensor object representing the TS-05
      TemperatureSensorImpl
        tss = new TemperatureSensorImpl();

      String name = registry_host+"section1";
      Naming.rebind(name, tss);
      System.out.println( "Sensor object has been bound to name: "+name );

      // make tss run in its own thead...
      Thread t = new Thread( tss );
      t.start();
      System.out.println( "Sensors are up and running..." );
    } catch (Exception e) {
      System.err.println( e );
      System.exit(1);
    }
  }
}
```

```java
package ts05.monitor;

import java.rmi.*;

/**
    Show names bound in the registry

    @author Henrik Bærbak Christensen - (c) Imhotep 2003

*/

public class ShowRegistry {

  public static void main(String[] args) {
    if (args.length != 1) {
      System.out.println( "Wrong number of parameters:" );
      System.out.println( " 1. parameter: host where registry is running" );
      System.exit(0);
    }

    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }
    try {
      String [] objs = Naming.list(args[0]);

      for ( int i = 0; i < objs.length; i++ ) {
        System.out.println( "- "+objs[i] );
      }
    } catch (Exception e) {
      System.err.println("ShowRegistry exception: " +
                         e.getMessage());
      e.printStackTrace();
    }
  }
}
```

```java
package ts05.monitor;

import java.rmi.*;
import java.rmi.server.*;

import ts05.sensor.*;

/** Implementation of a temperature listener that output
    received values on the console.

    @author Henrik Bærbak Christensen - (c) Imhotep 2003

*/

public class TemperatureListenerImpl extends UnicastRemoteObject
  implements Remote, TemperatureListener {

  public TemperatureListenerImpl() throws RemoteException {
    super();
  }

  /** outputs the temperature sampled on the console */
  public void temperatureSampled( TemperatureEvent t_event )
    throws RemoteException {
    double T = t_event.getReading();
    System.out.println( "Received reading="+T );
  }
}
```

```java
package ts05.monitor;

import java.rmi.*;
import java.io.*;

import ts05.sensor.*;

/**
    Temperature monitor attaches itself to a temperature sensor
    and monitors all readings by outputting on the console.

    @author Henrik Bærbak Christensen - (c) Imhotep 2003

*/

public class TemperatureMonitor {

  /** instantiate the monitor */
  public static void main(String[] args) {

    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }

    String registry_host = "//localhost/";

    System.out.println( "Using registry at "+registry_host );

    try {
      // get the temperature sensor object reference from the
      // RMI object request broker...
        String name = registry_host+"section1";
        System.out.println( "Looking up object reference: "+name );
        TemperatureSensor ts = (TemperatureSensor) Naming.lookup(name);

        System.out.println( "Located sensor object..." );

        // Create a local temperature listener object
        // (observer pattern 'observer'-role)
        // and register it at the temperature sensor.
        // The object refers to the counter object
        TemperatureListenerImpl tl = new TemperatureListenerImpl();
        System.out.println( "Created listener..." );
        ts.addTemperatureListener(tl);
        System.out.println( "Added listener object to sensor" );

        // wait for callbacks
        System.out.println( "Finished; awaiting callbacks..." );

        // tricks-of-the-trade way of waiting on incoming calls...
        java.lang.Object sync = new java.lang.Object();
        synchronized (sync) {
          sync.wait();
        }
    } catch (Exception e) {
      System.err.println("TemperatureMonitor exception: " +
                         e.getMessage());
      e.printStackTrace();
    }
  }
}
```

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!--

Ant Build script for the TS-05 architectural prototype

(c) 2003  Imhotep
@author Henrik Bærbak Christensen.

In order to run the Server, you must

1) Unset the classpath, and start the rmiregistry.
2) ant runServer in one shell
3) ant runMonitor in a number of shells, all monitoring
-->

<project name="euc_sa" default="help" basedir=".">

  <target name="help">
    <echo message="Software Architecture in Practice."/>
        <echo message="Ant build script for TS-05"/>
    <echo message="Valid targets:"/>
    <echo message="  build_all:      Build everything"/>
    <echo message="  doc:            Build javadoc in directory 'docs'"/>
    <echo message="  runServer:      Run temperature sensor server"/>
    <echo message="  runMonitor:     Run temperature monitor"/>
    <echo message="  runList:        List rmiregistry contents"/>
    <echo message=""/>
    <echo message="Run the server before the monitor!"/>
    <echo message=""/>
    <echo message="(c) Imhotep / Henrik Bærbak Christensen 2003-2005"/>
  </target>


  <!-- Directory properties -->
  <property name="src" value="src"></property>
  <property name="resources" value="resources"></property>
  <property name="doc" value="docs"></property>
  <property name="build" value="build"></property>

  <property name="policy_file" value="java.policy"/>

  <!--
    Definitions of where the rmiregistry is running.
    We need both the 'short name' to set in a java property, and
    the 'full name' (including //'s) to give as parameter
  -->
  <property name="registry_host" value="localhost"/>
  <property name="registry_host_spec" value="//${registry_host}/"/>


  <!-- Classpath used for compilation - NOT used for execution! -->
  <path id="_classpath">
    <pathelement path="${build}"/>
  </path>

  <path id="_srcpath">
    <pathelement path="${src}"/>
  </path>

  <!-- Make the output building directory -->
  <target name="prepare">
```

```
    <mkdir dir="${build}"></mkdir>
  </target>

  <!-- === RESOURCE COPYING === -->
  <target name="copy_resources" depends="prepare">
    <copy todir="${build}" >
      <fileset dir="${resources}">
        <include name="java.policy"/>
      </fileset>
    </copy>
  </target>

  <!-- COMPILATION TARGETS -->
  <target name="compile_src" depends="prepare">
    <javac destdir="${build}" debug="on" deprecation="on">
      <src> <path refid="_srcpath"/> </src>
      <classpath> <path refid="_classpath"/> </classpath>
    </javac>
  </target>

  <target name="compile_all" depends="compile_src"/>

  <!-- RMIC targets -->
  <target name="rmic" depends="compile_all">

    <rmic base="${build}" stubversion="1.2" verify="on"
      classname="ts05.sensor.TemperatureSensorImpl">
      <classpath>
        <path refid="_srcpath"/>
        <path refid="_classpath"/>
      </classpath>
    </rmic>
    <rmic base="${build}" stubversion="1.2" verify="on"
      classname="ts05.monitor.TemperatureListenerImpl">
      <classpath>
        <path refid="_srcpath"/>
        <path refid="_classpath"/>
      </classpath>
    </rmic>

  </target>

  <!-- BUILD TARGETS -->
  <target name="build_all" depends="compile_all,rmic,copy_resources">
  </target>


  <!-- RUN TARGETS -->
  <target name="runServer" depends="build_all">
    <echo message="Running sensor, registry at ${registry_host}"/>
    <java classname="ts05.sensor.TemperatureSensorServer"
      dir="${build}" fork="yes">
      <classpath><path refid="_classpath"/> </classpath>
      <sysproperty key="java.security.policy"
        path="${build}/java.policy"/>
      <sysproperty key="java.rmi.server.hostname"
        value="${registry_host}"/>
      <sysproperty key="java.rmi.server.codebase"
        value="file:/${basedir}/${build}/"/>
    </java>
  </target>
```

```
  <target name="runMonitor" depends="build_all">
    <java classname="ts05.monitor.TemperatureMonitor"
      dir="${build}" fork="yes">
      <classpath><path refid="_classpath"/> </classpath>
      <sysproperty key="java.security.policy"
        path="${build}/java.policy"/>
    </java>
  </target>

  <target name="runList" depends="build_all">
    <java classname="euc.monitor.ShowRegistry"
      dir="${build}" fork="yes">
      <classpath><path refid="_classpath"/> </classpath>
      <sysproperty key="java.security.policy"
        path="${build}/java.policy"/>
      <arg value="${registry_host_spec}"/>
    </java>
  </target>

  <!-- HOUSEHOLDING TARGETS -->

  <target name="clean">
    <delete dir="${build}"></delete>
    <delete dir="${deploy}"></delete>
    <delete dir="${doc}"></delete>
  </target>

  <property name="backup" value="backup"/>

  <target name="mkdir_backup">
    <mkdir dir="${backup}"></mkdir>
    <tstamp/>
  </target>

  <target name="backup" depends="mkdir_backup">
    <zip
      zipfile="${backup}/${DSTAMP}.zip"
      update="true">
      <fileset
        dir="${basedir}"
        includes="${src}/**,${test}/**,build.xml,diary.txt,${resources}/**"
      </fileset>
    </zip>
  </target>

  <!-- DOC -->
  <target name="mkdirdoc">
    <mkdir dir="${doc}"></mkdir>
  </target>

  <target name="doc" depends="build_all,mkdirdoc">
    <javadoc
      packagenames=
      "euc.*"
      destdir="${doc}"
      package="true"
      doctitle="&lt;b&gt;Software Architecture in Practice: TS-05&lt;/b&gt;
      bottom="&lt;b&gt;Copyright &#169; Imhotep / Henrik Bærbak Christensen
/b&gt;"
      windowtitle="&#169; Imhotep / Henrik Bærbak Christensen"
      >
      <sourcepath>
```

```
        <pathelement path="${src}"/>
      </sourcepath>
      <classpath>
        <path refid="_classpath"/>
      </classpath>

    </javadoc>
  </target>

</project>
```