University College London
Department Of Electrical and Electronic Engineering
MSc in Telecommunications

Software for Networks and Services

1997-98 Assignment

Isaac Kokkinidis

January 1998

Contents

1 Java Coding **1**

1.1 Day of the Week 1

1.2 Networking Applications 2

1.2.1 The Client Side 2

1.2.2 The Single Client Echo Server 3

1.2.3 The Multi-threaded Server 4

1.3 Roots of a Quadratic Equation Application 7

2 Reverse Engineering **10**

2.1 The Day of the Week Application 10

2.2 The Networking Applications 10

2.3 The Roots of a Quadratic Equation Application 10

3 Java Today **14**

3.1 Introduction 14

3.2 Background 14

3.3 Arguments 14

3.4 Prospectives 15

3.5 Conclusion 15

References 15

1 Java Coding

This section contains the implementation of the applications, as described in the tutorial leaflet. A small description is preceding each program, and the fully commented source code is following.

1.1 Day of the Week

This introductory application makes use of the `Day` class in the `corejava` package. It just translates the `weekday()` method of the above class to a meaningful string, and prints it to the standard output stream. If a valid date is given, the program returns the day of the week for this date, otherwise the current day of the week is returned. The `java.util.Date` class could have been used, as well.

```
/*
 * DayOfWeek application
 * returns day of the week of a given date
 * arguments: date in the format dd mm yyyy (default: today)
 */
import corejava.*;

public class DayOfWeek
{
   public static void main(String[] args) {
      /* create a string array holding the days' names. */
      String DayOfWeekbuf[] = { "Sunday",   "Monday", "Tuesday", "Wednesday",
                                "Thursday", "Friday", "Saturday" };
      Day date=null;                   // Day class from the corejava package

      if (args.length == 3) {
         int d = Format.atoi(args[0]); // parse arguments
         int m = Format.atoi(args[1]); // Format class from the corejava package
         int y = Format.atoi(args[2]);
         try {
            date = new Day(y, m, d);
         }
         catch (IllegalArgumentException e) {
            System.err.println("Please give a valid date.");
            System.exit(-1);
         }
         System.out.print("The day of ");
      }
      else {                           // use current date
         date = new Day();
         System.out.print("Today ");
      }
      System.out.println( date.getDay() + "/" + date.getMonth() + "/" +
                          date.getYear() + " is a: " +
                          DayOfWeekbuf[date.weekday()]);
   }
}
```

1.2 Networking Applications

The text echoing application demonstrates the use of the `java.net` package, which implements among others the *BSD* sockets. A simple connection between two Internet hosts is established on a predefined port, and text is send through it.

The application comprises a client part and a server part, examined in the following paragraphs. A multiple client capable server is also presented.

1.2.1 The Client Side

The client side is straightforward. First, the server hostname is resolved, using standard name resolving routines in the `java.net` package. If a server listening on the given port is found, the connection is established, a command prompt with the name of the server is displayed, and input is awaited. If a new line character is encountered in the input, the text is sent through the connection, and the prompt is restored.

```
/*
 * SocketText application class
 * implements a client to the TextServer and MultiTextServer classes
 * arguments: hostname port
 */
import java.io.*;
import java.net.*;
import corejava.*;

class SocketText
{  public static void main(String[] args)
   {
       if (args.length == 2) {
         try {
            InetAddress ServerIPAddress = InetAddress.getByName(args[0]);
            int port = Format.atoi(args[1]);

            /* try to establish our connection */
            Socket ClientSock = new Socket(ServerIPAddress, port);
            Console ClientConsole = new Console();

            /* use 1.1 streams */
            PrintWriter send = new PrintWriter(ClientSock.getOutputStream());

            boolean more = true;
            int count = 0;           // just something to show on the prompt
            while (more) {
               count++;
               String str = ClientConsole.readString( count + ":" +
                                 ServerIPAddress.getHostName() + ">");
               send.println(str);
               send.flush();
               if (str == null || str.trim().equals("BYE") ) more = false;
            }
```

```
            ClientSock.close();
         }
         catch(Exception e) { System.err.println(e); }
      }
      else {
         System.out.println("Please give HostName and Port");
      }
   }
}
```

1.2.2 The Single Client Echo Server

The code listing of a simple server echoing the text from one only client, is presented below. The most important operation of the server is in the line:

```
Socket ClientSock = ServerSock.accept( );
```

which is binding ClientSock to ServerSock. This is done in a blocking way, forcing the whole application to wait idle until a connection request occurs. It is thus considered a bad implementation. A solution is proposed in the following paragraph.

```
/*
 * TextServer application class
 * implements a server echoing text
 * arguments: the listening port (default=8198)
 */
import java.io.*;
import java.net.*;
import corejava.*;

class TextServer
{
   public static void main(String[] args )
   {
      int port=8198;                        // our default listening port
      if (args.length == 1)                 // parse argument to change port No
         port = Format.atoi(args[0]);

      try {
         /* open listening port */
         ServerSocket ServerSock = new ServerSocket(port);

         /* show some initial information */
         InetAddress LocalIPAddress = InetAddress.getLocalHost();
         System.out.println("Server " + LocalIPAddress.getHostName() +
                            " listening on port: "
                            + ServerSock.getLocalPort());

         /* listen and wait until a client connects (blocking call) */
         Socket ClientSock = ServerSock.accept( );
```

```
        /* show us who connected */
        System.out.println( "Connection established with: "
                          + ClientSock.getInetAddress().getHostName());

        /* in captures the client input (1.1 streams) */
        BufferedReader in = new BufferedReader (
                               new InputStreamReader(ClientSock.getInputStream()));

        /* send is the output stream */
        PrintWriter send = new PrintWriter(ClientSock.getOutputStream());

        boolean done = false;
        while (!done) {
           String str = in.readLine();           // store client input
           System.out.println(str);              // show us client input
           send.println("Echo: " + str + "\r");  // echo back client input
           if (str == null || str.trim().equals("BYE"))
              done = true;
        }
        ClientSock.close();                      // close all connections
        in.close();                              // close all streams (the GC
        send.close();                            // would do this, anyway).
     }
     catch (Exception e) { System.err.println(e); }
  }
}
```

1.2.3 The Multi-threaded Server

There are two ways to override the blocking presented in the previous section. One is to use asynchronous messages and a `select()` method on the server socket, and the other is to use threads. The former was mostly used in *C* implementations, but the latter is inherently supported by Java, hence it is much more convenient to use here.

The following implementation provides a more practical application than that presented in the previous paragraph, allowing multiple clients to write text data on a server. Each client connection request initiates a server thread, which is bound with the specific connection during its lifetime. Then, the client is identified and its input is print on the system's standard output.

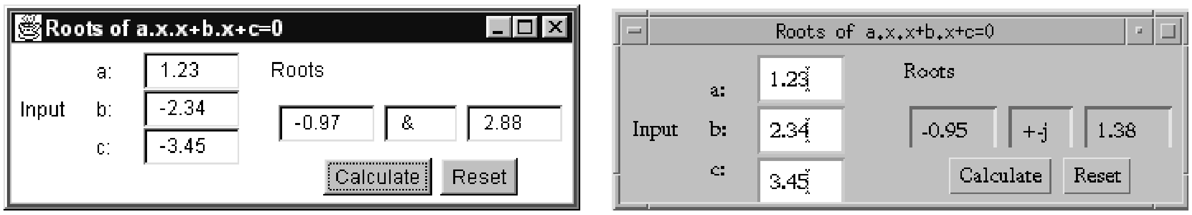
```
/*
 * MultiTextServer application class
 * implements a server echoing text from multiple clients
 * arguments: the initial listening port (default=8198)
 */

import java.io.*;
import java.net.*;
import corejava.*;
```

```
class MultiTextServer
{
   public static void main(String[] args )
   {
      int port = 8198;
      ServerSocket ServerSock = null;

      if (args.length == 1)
         port = Format.atoi(args[0]);

      try {
         ServerSock = new ServerSocket(port);
         InetAddress LocalIPAddress = InetAddress.getLocalHost();
         System.out.println("Server " + LocalIPAddress.getHostName()
                            + " listening on port: "
                            + ServerSock.getLocalPort());
      }
      catch (Exception e) { System.err.println(e); }

      boolean always = true;                           // infinite loop
      while(always) {
         Socket ClientSock = null;
         try {
            ClientSock = ServerSock.accept( );
         }
         catch (IOException e) {                       // we may have other clients
            System.err.println("Accept failed: " + e); // keep on serving!
         }
         System.out.println( "New connection established with: "
                             + ClientSock.getInetAddress().getHostName());
         new MultiTextServerThread(ClientSock).start();
      }

      try {                                            // code never reached
         ServerSock.close();
      }
      catch (IOException e) { System.err.println(e); }
   }
}


/*
 * MultiTextServerThread class
 * implements a thread of the MultiTextServer
 * constructor argument: socket of the new client connection
 */

import java.io.*;
import java.net.*;
import corejava.*;
```

```
class MultiTextServerThread extends Thread
{
   private Socket ClientSock = null;

   MultiTextServerThread(Socket newsock)
   {
        super("MultiTextServerThread");  // create our thread
        this.ClientSock = newsock;       // store the socket requesting service
   }

   public void run()                     // override Thread.run() for our task
   {
      try {
         BufferedReader in = new BufferedReader (       // 1.1 API streams used
                                 new InputStreamReader(ClientSock.getInputStream()) );
         PrintWriter  send = new PrintWriter(ClientSock.getOutputStream());

         boolean done = false;
         while (!done) {
            String str = in.readLine();
            if (str == null) done = true;
            else {
               System.out.println(ClientSock.getInetAddress().getHostName()
                                   + " says: " + str);
               send.println("Echo: " + str + "\r");
               if (str.trim().equals("BYE")) done = true;
            }
         }
         send.println("Goodbye");
         System.out.println("Client "
                             + ClientSock.getInetAddress().getHostName()
                             + " dropped the connection");
         ClientSock.close();             // try to close everything gracefully
         in.close();
         send.close();
      }
      catch (Exception e) { System.err.println(e); }
   }
}
```

1.3 Roots of a Quadratic Equation Application

The following program is a demonstration of the `java.awt` package. An elementary graphical user interface (GUI) is built, in order to calculate the roots of a user-defined quadratic equation. Most of the code is GUI-related, while the calculation itself is done in the `action()` method. A convenient reset function is also supplied. The 1.0 API event model has been used, as well as some methods of the `Format` class of the `corejava` package. Two sample instants of the application are shown in Figure 1.



(a) (b)

Figure 1: Sample `Quad` instances in (a) the MS-Windows and (b) the Solaris environment.

```
/*
 * Quad application
 * finds and displays the roots of a quadratic function
 */

import java.awt.*;
import corejava.*;

public class Quad extends Frame
{
    private TextField aField;                   // user input coefficient A box
    private TextField bField;                   // user input coefficient B box
    private TextField cField;                   // user input coefficient C box
    private TextField rootAField;               // root #1 output box
    private TextField rootBField;               // root #2 output box
    private TextField rootLabelField;           // just some visual aid...

    public Quad()                               // here is the GUI setup code
    {
        setTitle("Roots of a.x.x+b.x+c=0");

        Panel coefLabelPanel = new Panel();     // the coefficients labels grid
        coefLabelPanel.setLayout(new GridLayout(3,1));
        coefLabelPanel.add(new Label("a:"));
        coefLabelPanel.add(new Label("b:"));
        coefLabelPanel.add(new Label("c:"));
```

```
    Panel coefPanel = new Panel();          // the coefficients grid
    coefPanel.setLayout(new GridLayout(3,1));
    aField = new TextField("0.00", 5);
    bField = new TextField("0.00", 5);
    cField = new TextField("0.00", 5);
    coefPanel.add(aField);
    coefPanel.add(bField);
    coefPanel.add(cField);

    Panel inPanel = new Panel();            // the input panel
    inPanel.setLayout(new FlowLayout());
    inPanel.add(new Label("Input"));
    inPanel.add(coefLabelPanel);
    inPanel.add(coefPanel);
    add("West", inPanel);

    Panel rootPanel = new Panel();          // the roots panel
    rootPanel.setLayout(new FlowLayout());
    rootAField = new TextField("0.00",5);
    rootAField.setEditable(false);
    rootLabelField = new TextField("&",3);
    rootLabelField.setEditable(false);
    rootBField = new TextField("0.00",5);
    rootBField.setEditable(false);
    rootPanel.add(rootAField);
    rootPanel.add(rootLabelField);
    rootPanel.add(rootBField);

    Panel commandPanel = new Panel();       // the command panel
    commandPanel.setLayout(new FlowLayout());
    commandPanel.add(new Button("Calculate"));
    commandPanel.add(new Button("Reset"));

    Panel outPanel = new Panel();           // the output panel
    outPanel.setLayout(new GridLayout(3,1));
    outPanel.add(new Label("Roots"));
    outPanel.add(rootPanel);
    outPanel.add(commandPanel);
    add("East", outPanel);

}



public boolean handleEvent(Event evt)
{
    if (evt.id == Event.WINDOW_DESTROY) System.exit(0);
    return super.handleEvent(evt);
}
```

```
    public boolean action(Event evt, Object arg) // override action() for our events
    {
        if (arg.equals("Calculate")) {
            double a = Format.atof(aField.getText());
            double b = Format.atof(bField.getText());
            double c = Format.atof(cField.getText());
            double d = b*b - 4*a*c;

            if( a == 0 ) {
               if( b == 0 ) {                    // ambiguous, but possible user input
                  this.action(evt, "Reset");     // take care of it.
                  return false;
               }
               rootAField.setText(new Format("%2.2f").form(-c/b));
               rootLabelField.setText("");
               rootBField.setText("single");
            }
            else if( d == 0 ) {                  // one double real root
               rootAField.setText(new Format("%2.2f").form(-b/(2*a)));
               rootLabelField.setText("");
               rootBField.setText("double");
            }
            else if( d < 0 ) {                   // two complex conjugate roots
               rootAField.setText(new Format("%2.2f").form(-b/(2*a)));
               rootLabelField.setText("+-j");
               rootBField.setText(new Format("%2.2f").form(Math.sqrt(-d)/(2*a)));
            }
            else {                               // two real roots
               rootAField.setText(new Format("%2.2f").form(-b/(2*a)-Math.sqrt(d)/(2*a)));
               rootLabelField.setText("&");
               rootBField.setText(new Format("%2.2f").form(-b/(2*a)+Math.sqrt(d)/(2*a)));
            }
        }
        else if (arg.equals("Reset")) {
                aField.setText("0.00");
                bField.setText("0.00");
                cField.setText("0.00");
                rootAField.setText("0.00");
                rootLabelField.setText("&");
                rootBField.setText("0.00");
        }
        else return false;
        return true;
    }

    public static void main(String[] args)
    {
        Frame f = new Quad();
        f.resize(350, 125);                      // should use system/display-
        f.show();                                // independent metrics here.
    }
}
```

2 Reverse Engineering

The small scale of the applications implemented, makes them unsuitable for object oriented analysis. Some aspects of such analyses, as well as the Unified Modelling Language (UML) notation, will be demonstrated in this section. There will not be a global treatment of the applications in respect to the depth the analysis will proceed, *i.e.* in some cases it will be very detailed, while in others not. This will be done according to where there are important concepts, which are worth to be illustrated.

2.1 The Day of the Week Application

The `DayOfWeek` class is just a wrapper for the main function, which performs the required task. It has no role in the object oriented hierarchy, as it is very unlikely that it will be re-used or extended. Hence, anything more complicated than the implementation in paragraph 1.1, would be over-engineering practice.

Due to the simplicity of the tasks performed by the application, the use case diagram, Figure 2a, does not provide a lot of information. This is the case for the activity and static class diagrams, which are also depicted in Figure 2. There is no information to include in a state transition or a collaboration diagram for the specific class.

2.2 The Networking Applications

The applications developed for the networking examples provide a better basis for demonstrating the use of object-oriented analysis. The reverse engineering process starts with the use case diagram, which is depicted for the multiple client capable server in Figure 3a. The single server equivalent is straightforward. Of the remaining diagrams, the most interesting is the collaboration sequence diagram, which is illustrated for the single server in Figure 3c. The multiple server diagram collaboration sequence diagram, Figure 3d, has been based on the respective single server in order to be more readable.

2.3 The Roots of a Quadratic Equation Application

The most interesting diagram of this application is the activity diagram, depicted in Figure 4b. This illustrates the concurrency introduced by the graphical environment and the messages structure. It should be noted that the collaboration sequence diagram, Figure 4c, is for a sample use of the application, as in such environments there is no predefined sequence of the user actions.

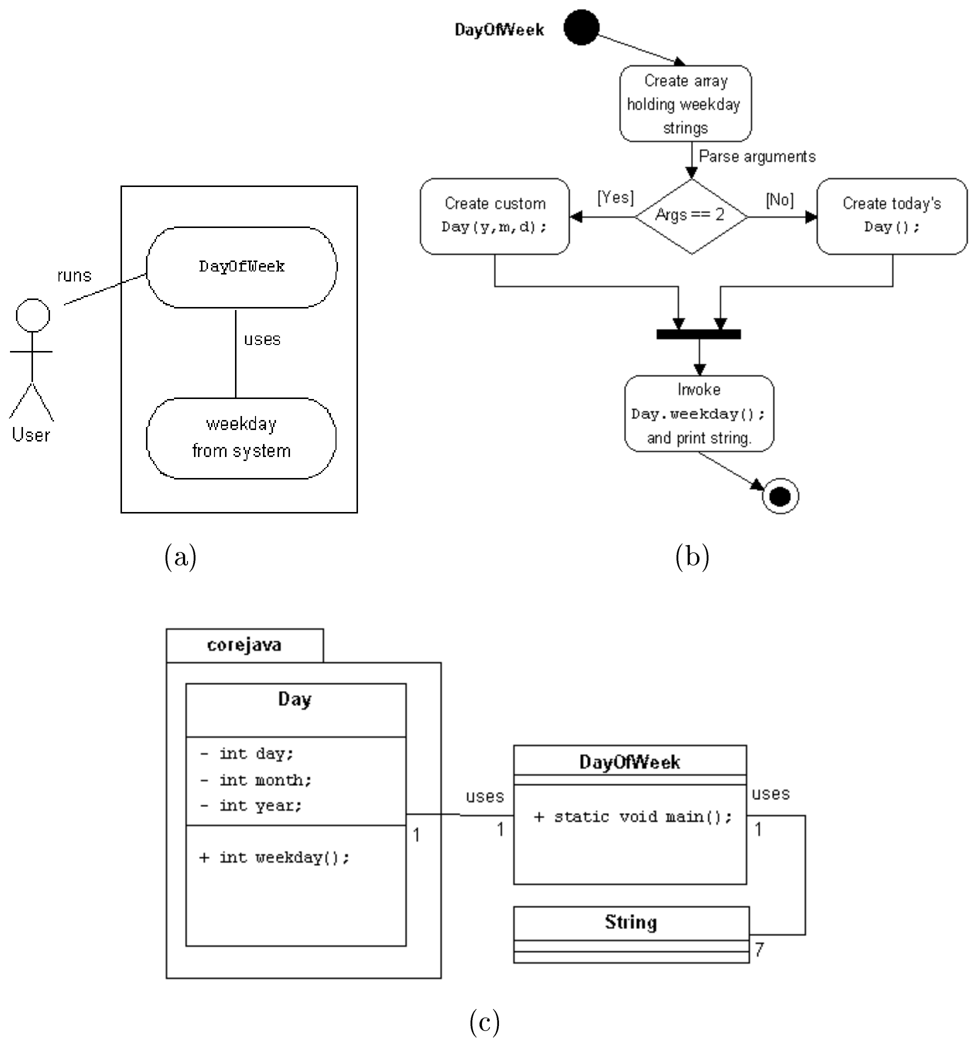


Figure 2: (a) Use Case, (b) Activity, and (c) Static Class diagrams for `DayOfWeek`.

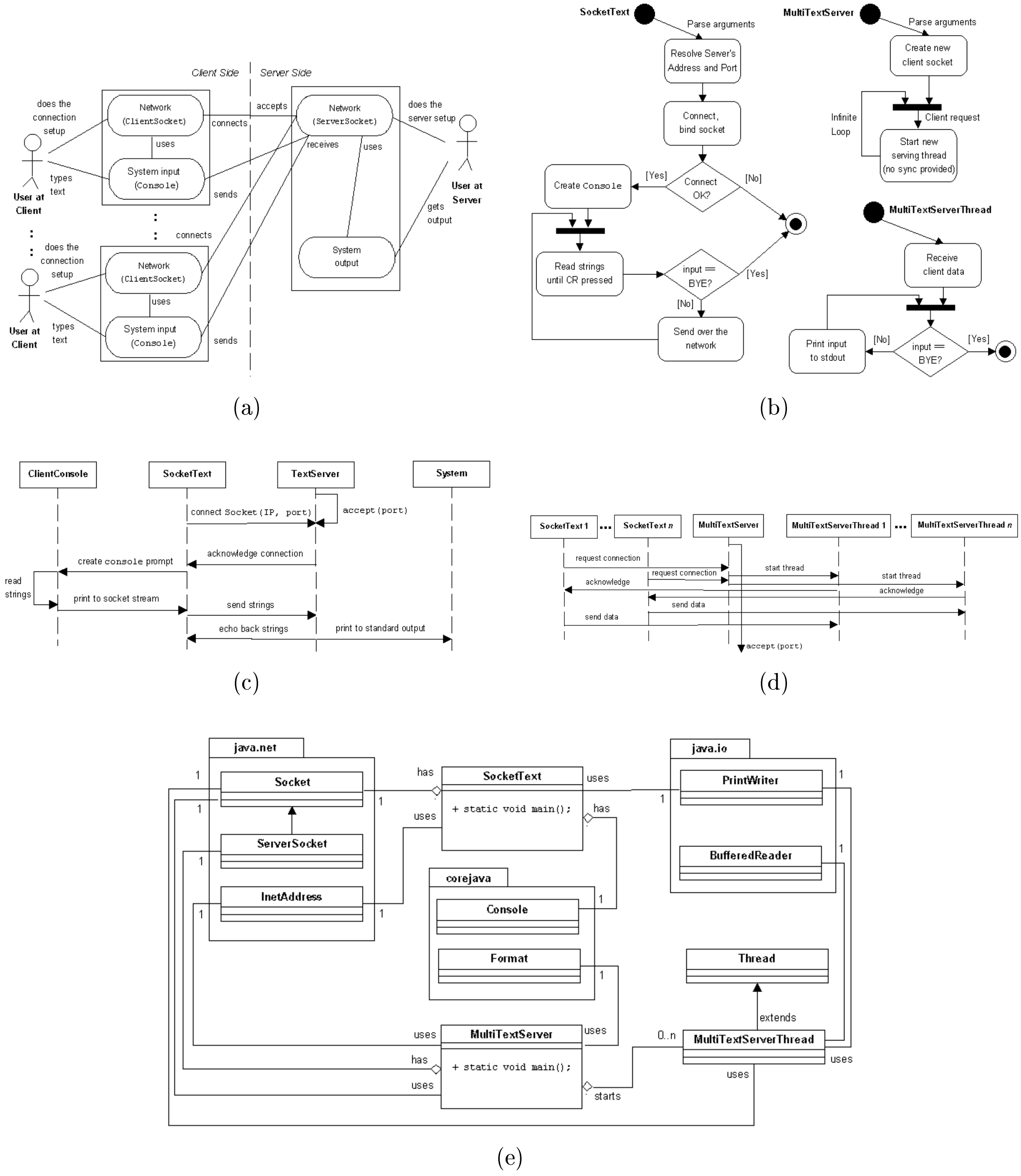


Figure 3: (a) Use Case, (b) Activity, (c) Collaboration for single server, (d) Collaboration for multiple server, (e) Static Class diagrams for the networking applications.

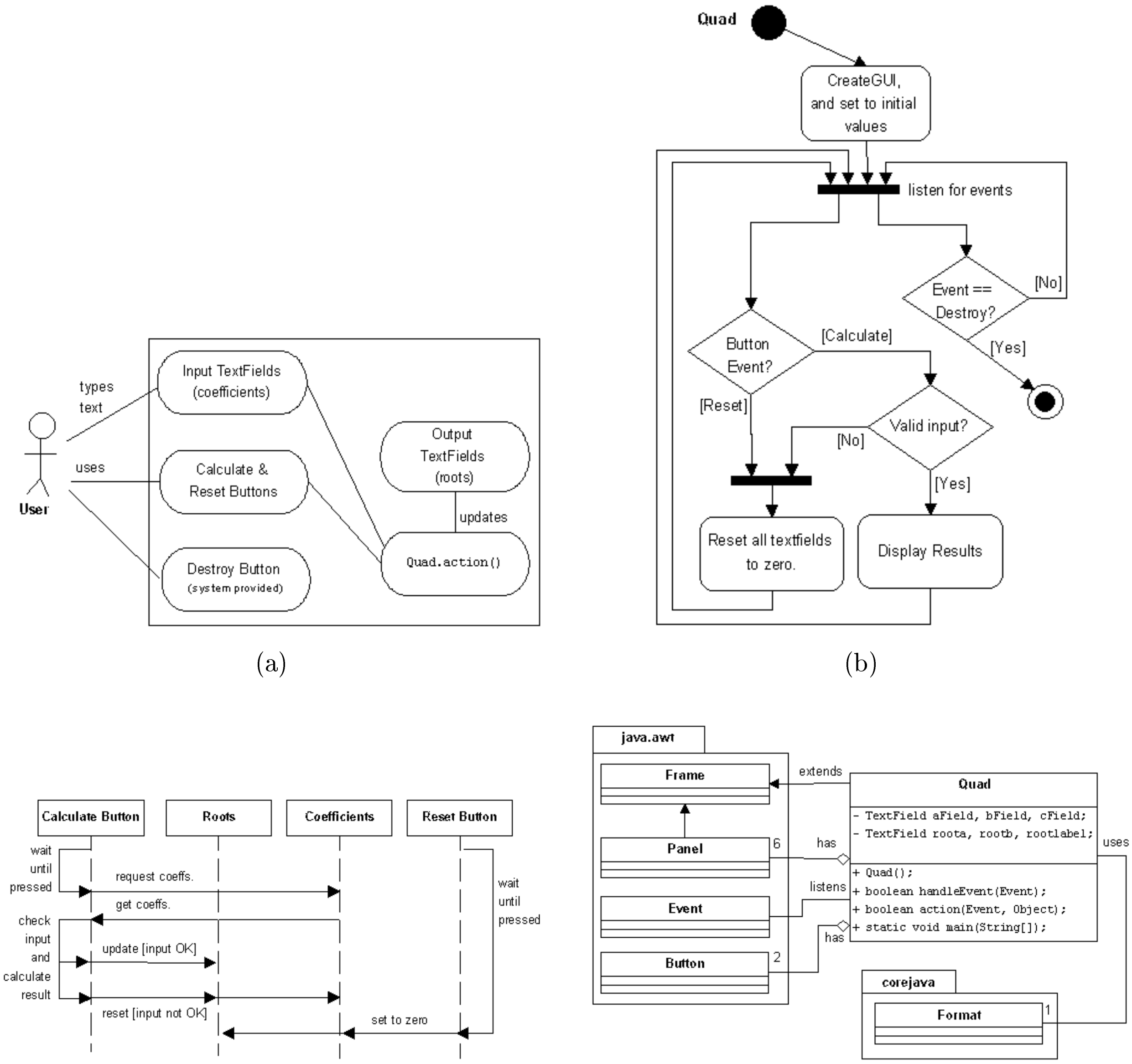


Figure 4: (a) Use Case, (b) Activity, (c) Collaboration, and (d) Static Class diagrams for the `Quad` application.

3 Java Today

3.1 Introduction

The continually increasing proportion of interconnected Personal Computers (PCs) has necessitated the creation of multi-platform interworking environments. Java claims to offer such a software platform, and not just a plain programming facility. It was conceived initially as an operating system for smart home appliances, but Sun Microsystems has developed a simple Application Programming Interface (API), and has offered free development tools to attract developers.

However, what really boosted Java was its adoption by Netscape in the same-named World Wide Web browser. Currently, the platform is widely used for powerful tasks in the web context, but the goal, according to Sun, is Java everywhere. This goal does not cause positive feelings to the virtual, but legal, monopoly in PC Operating Systems (OSes), Microsoft. The latter is seeing a potential danger that competitive OSes will benefit from the write once, run anywhere concept, or even that Java alone could offer a novel alternative OS.

3.2 Background

The war on Java has revealed that the real debate is on the *objects*. These are the building blocks of every software application, from a web browser to a word processing package. Whoever gains control of the object architecture, will probably reign in the extremely profitable software business.

Due to the increasing number of networked computers, distributed architectures have emerged, the most popular of those being the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), and Microsoft's Distributed Component Object Model (DCOM). It is not a coincidence that among the companies backing CORBA are IBM, Sun, Novell, Oracle and Netscape, all Microsoft competitors. Java supports its own distributed object architecture, Remote Method Invocation (RMI), but the roads of Java and OMG have converged and it has been announced that the next Java specification release will support inherently CORBA [1].

Another point of rivalry between the two companies, is the wrapper suites under which they offer foundation objects, usually referred to as *controls*. Sun endorses JavaBeans for Java applets, while Microsoft has defined ActiveX [2], which is a set of reusable controls supported by DCOM. The latter have a clear performance advantage in Microsoft OSes, as they are native applications, and not interpreted as the Java equivalents.

3.3 Arguments

The Java Application Programming Interface (API), currently at version 1.1, has undergone significant changes from its initial specification. These were essential to enhance the integrity and security of Java, but were seen as setbacks from many developers. Microsoft claims that some issues in the new standard should not be adopted by the industry. As a Java Virtual Machine (VM) manufacturer for its own OSes, it claims that it has rights on the specification of Java, and due to the reasons presented in the previous paragraph, does not support RMI. It does support Java, but only as a neat programming language for accessing native controls, which are much more performance effective.

The response of Sun, recently recognised as an official submitter of Java to the International Standards Organisation (ISO), is that since Microsoft does not comply to the Pure Java specification, it should remove the logo from all its products. Sun claims that it could provide support for Java developers and users under the Microsoft OSes, but this is very difficult without the support of Microsoft. Sun is counting on the support of other OS manufacturers, which considers that will increase their market positions through Java, but so does Microsoft with competitors of Sun in the workstation field [3].

3.4 Prospectives

The fears of Microsoft over Java are purely commercial and totally justified. The company's place in the market, makes it ignore all the other platforms, and its stance against third-party standards is definite. Eventually, it is very likely that it will acquire the Java concept, and create a proprietary version suiting its own technologies, as it has done in similar past cases. Therefore, the future of Java in the PC industry does not seem bright, especially if Sun and Microsoft do not reach an agreement for the benefit of all developers and users.

On the other hand, the Internet is currently considered multi-platform, although the vast majority of the end users are using Microsoft OSes. Java with CORBA Internet Interoperable ORB Protocol (IIOP) deliver now advanced services, and have the support of many large Microsoft-competitive companies. Furthermore, the telecommunications industry is not dependent on Microsoft architectures, and Java's prospectives as a tool for the development of platform independent services are very good. Towards that contribute the definition of the Java Telephony API (JTAPI) for Computer Telephony Integration (CTI), the concept of thin clients providing flexibility and centralised control to operators, and Intelligent Networking platforms, which make extended use of distributed architectures as CORBA [4].

3.5 Conclusion

The Java platform is now having infancy problems, as Windows and MacOS had in their first versions. If both sides reach a compromise, it could become the next PC software development standard. However, Sun is porting it to many other small and large scale hardware products, as telecommunications terminal devices, and its opportunity might be there.

References

[1] R. Orfali, D. Harkey, J. Edwards, "CORBA, Java and the Object Web," *BYTE Magazine*, McGraw-Hill Publications, Vol. 22, No 10, pp. 95-100, October 1997.

[2] D. Chappell and D. S. Linthicum, "ActiveX Demystified," *BYTE Magazine*, McGraw-Hill Publications, Vol. 22, No 9, pp. 56-64, September 1997.

[3] M. Ricciuti, "MS licenses COM to SGI, Iona," *CNet news.com*, at `http://www.news.com/News/Item/0,4,18467,00.html`, January 1998.

[4] M. Lambert, "Java: Shaping the future of the telecoms market," *Java White Paper*, Sun Microsystems UK, at `http://www.sun.co.uk/publication/whitep/javawptelecom2.html`