CS 580 Client-Server Programming
Spring Semester, 2005
Assignment 2 & 3 Comments
Contents

Assignment 3.1 Comments
Class & Objects

A class represents a single abstraction

A class has both

- Methods (actions)
- Fields (data)

Without both a class is not complete

- Actions and data not together
- Need different abstraction

```
public class Read
    {
    private static final char CARRAGE_RETURN = (char) 13;

    public  byte[] readMessage(Socket connection) throws
            IOException
        {
        UpToFilterInputStream in =
            new UpToFilterInputStream(
                new BufferedInputStream(connection.getInputStream()));
        return in.upTo(CARRIAGE_RETURN);
        }
    }
```

Constructors as Main Methods

Constructors are used to

- Initialize fields of the object

Constructors are not main methods

```
public class Server {
    public Server(int port ) {
        ServerSocket server = new ServerSocket(port);
        while (true) {
            Socket connection = server.accept();
            Reader in = new BufferedReader( new InputStreamReader(
                                connection.getInputStream()));
            Writer out = new OutputStreamWriter(
                                connection.getOutputStream());
            etc.
        }
    }
```

Separate Class for Main

What is the point in having a separate class for main?

Particularly a short main?

```
public class TimeDateServer {
    blah
    blah
}


public class TimeDateServerMain {
    public static void  main(String[] args) {
        new TimeDateServer().run();
    }
}
```

Functionality verse Usage in Program

UpToInputStream can work on any input stream

Your program only passes in a BufferedInputStream

Why restrict UpToInputStream?

```
public class UpToInputStream {
    private BufferedInputStream in;

    public UpToInputStream( BufferedInputStream input) {
        in = input;
    }

    blah
}
```

```
public class UpToInputStream {
    private InputStream in;

    public UpToInputStream( InputStream input) {
        in = input;
    }

    blah
}
```

More Functionality verse Usage in Program

What do the two methods do?

What is the dfference?

```
public class MetaDataEncoding {
    //method used by client to encode meta data
    public String metaDataEncoder(String filePath, Vector keywords) {
        blah
    }

    //method used by server to encode meta data
    public String metaDataEncoder(String filePath, String id) {
        blah
    }

    etc.
```

Avoid Polling

Polling consumes cycles

The following have the same effect

The first consumes more cycles

```
while (in.ready() != true) ;
message = readMessage(in);
```

verses

```
message = readMessage(in);
```

Assignment 2 Decoder

Separate abstractions

Without error checking

```
public class Decoder {
    private static final char  INTEGER = 'i';
    private static final char LIST = 'l';
    private static final char DICTIONARY = 'd';
    private static final char END_OF_TOKEN = 'e';

    public Object decode(String bEncoded) {
        return decode(new UpToStream( bEncoded) );
    }

    public Object decode(UpToStream bEncoded) {
        Char typeIndicator = bEncoded.peek();

        switch( typeIndicator) {
            case INTEGER:
                return decodeInteger(bEncoded);
            case LIST:
                return decodeList(bEncoded);
            case DICTIONARY:
                return decodeDictionary(bEncoded);
            case '0': case '1': case '2': case '3': case '4':
            case '5': case '6': case '7': case '8': case '9':
                return decodeString(bEncoded);
        }
         new Exception( 'Invalid type ' + typeIndicator );
    }

    private Integer decodeInteger(UpToStream bEncoded) {
        bEncoded.skip();
        String integer = bEncoded.upTo(END_OF_TOKEN);
        return new Integer(integer);
    }
```

```
private String decodeString(UpToStream bEncoded) {
    String stringSize = bEncoded.upTo(':');
    int size = Integer.parseInt(stringSize).intValue();
    return bEncoded.next(size);
}

private List decodeList(UpToStream bEncoded) {
    ArrayList list = new ArrayList();
    bEncoded.skip();

    Char typeIndicator = bEncoded.peek();
    while (typeIndicator != END_OF_TOKEN) {
        list.add(decode(bEncoded));
        typeIndicator = bEncoded.peek();
    }
    bEncoded.skip();
    return list;
}

private Map decodeDictionary(UpToStream bEncoded) {
    HashMap map = new HashMap();
    bEncoded.skip();

    Char typeIndicator = bEncoded.peek();
    while (typeIndicator != END_OF_TOKEN) {
        String key = decodeString(bEncoded);
        Object value = decode(bEncoded);
        map.put(key, value);
        typeIndicator = bEncoded.peek();
    }
    bEncoded.skip();
    return map;
}
```

How much error checking is needed?

UpToStream

What methods are needed?

How hard will it be to write?

Doing two simple separate things is much easier than doing them together

Can be reused other places

Software is Hard

Student programs are too hard for me to write

Software is hard

Find ways to make it easy

Empty Constructors

Why?

```
public class Bencode {
    public Bencode() {
    }
    etc.
}
```