9.5/10 - Good example and reasonable explanation. With room in the text, annoations would have been more clear. Several people used this example with various levels of explanation. Perhaps I can use this one in class sometime. Again, Good job David!

David Pick
CM 2403
Software Construction and Evolution

```
    public void run() {
        Map<String, FileDetail> fileDetails = new Hashtable<String, FileDetail>();
        try {
            BufferedReader reader = new BufferedReader(new
FileReader(MainFrame.WATCHED_DATA));
            String directoryName = reader.readLine();
            File fileData = new File(directoryName, ".vobas");
            while (directoryName != null) {
                if (!fileData.exists()) {
                    fileData.createNewFile();
                } else {
                    ObjectInputStream fileDetailsReader = new ObjectInputStream(new
FileInputStream(fileData));
                    FileDetail fileDetail = (FileDetail) fileDetailsReader.readObject();
                    while (fileDetail != null) {
                        fileDetails.put(fileDetail.getName(), fileDetail);
                        try {
                            fileDetail = (FileDetail) fileDetailsReader.readObject();
                        } catch (EOFException e) {
                            break;
                        }
                    }
                }
                File[] files = new File(directoryName).listFiles();
                for (File file : files) {
                    FileDetail fileDetail = fileDetails.get(file.getName());
                    if (fileDetail == null) {
                        ScpTo.send(directoryName + File.separatorChar + file.getName());
                        fileDetails.put(file.getName(), new FileDetail(new Date(), file.getName()));
                    } else if (file.lastModified() > fileDetail.getModificationDate().getTime()) {
                        ScpTo.send(directoryName + File.separatorChar + file.getName());
                        fileDetails.remove(file.getName());
                        fileDetails.put(file.getName(), new FileDetail(new Date(), file.getName()));
                    }
                }
                ObjectOutput objectOutput = new ObjectOutputStream(new
FileOutputStream(new File(directoryName, ".vobas")));
                for (FileDetail fileDetail : fileDetails.values()) {
                    objectOutput.writeObject(fileDetail);
                }
                objectOutput.close();
                directoryName = reader.readLine();
            }
            reader.close();
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

This code reads in files parses them and writes the results to a new file. The largest problem with this code, is the number of lines that could easily be extracted out to a method with a descriptive name, such as:

```
ObjectOutput objectOutput = new ObjectOutputStream(new FileOutputStream(new
File(directoryName, ".vobas")));
for (FileDetail fileDetail : fileDetails.values()) {
  objectOutput.writeObject(fileDetail);
}
```

Extracting this code and code like this into seperate methods has several advantages. First and foremost being readability. The original method is very long and it is difficult to fully understand what it is doing. By putting some of the code into methods with descriptive names, it becomes much easier to get a full understanding of what the code is doing. Writing shorter methods will also make it much easier to test and debug this code. This happens because tests can be written for each individual method and since they are short methods any errors and be found and fixed fairly easily.

Refactored Version:

In order to fix the problem several new methods were created including, sendModifiedFiles, readFileDetails, fileModified, eachFileIn, writeFileDetails. Code from the original method was moved into these new methods, thus making the run method much more readable and testable. Moving the code has also reduced the overall length of the code and it makes it much easier for the programmer to fully understand what the code is doing, since they do not need to get caught up with the details when trying to understand how the method works on a high level.

```
public class VobasBackupService implements Runnable {

    public void run() {
        Map<String, FileDetail> fileDetails = new Hashtable<String, FileDetail>();
        try {
            sendModifiedFiles(fileDetails);
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private void sendModifiedFiles(Map<String, FileDetail> fileDetails) throws
FileNotFoundException, IOException,
            ClassNotFoundException {
        BufferedReader reader = new BufferedReader(new
FileReader(MainFrame.WATCHED_DATA));
        String directoryName = reader.readLine();
        File fileData = new File(directoryName, ".vobas");
        while (directoryName != null) {
            readFileDetails(fileDetails, fileData);
            sendModifiedFiles(fileDetails, directoryName);
            writeFileDetails(fileDetails, directoryName);
            directoryName = reader.readLine();
        }
        reader.close();
    }

    private void readFileDetails(Map<String, FileDetail> fileDetails, File fileData) throws
IOException,
            FileNotFoundException, ClassNotFoundException {
        if (!fileData.exists()) {
            fileData.createNewFile();
        } else {
            ObjectInputStream fileDetailsReader = new ObjectInputStream(new
FileInputStream(fileData));
            FileDetail fileDetail = (FileDetail) fileDetailsReader.readObject();
            while (fileDetail != null) {
                fileDetails.put(fileDetail.getName(), fileDetail);
                try {
                    fileDetail = (FileDetail) fileDetailsReader.readObject();
                } catch (EOFException e) {
                    break;
                }
            }
        }
    }

    private void sendModifiedFiles(Map<String, FileDetail> fileDetails, String directoryName)
{
        for (File file : eachFileIn(directoryName)) {
            if (fileModified(file, fileDetails.get(file.getName()))) {
                ScpTo.send(directoryName + File.separatorChar + file.getName());
                fileDetails.put(file.getName(), new FileDetail(new Date(), file.getName()));
            }
        }
    }

    private boolean fileModified(File file, FileDetail fileDetail) {
        return fileDetail == null || file.lastModified() >
fileDetail.getModificationDate().getTime();
    }

    private File[] eachFileIn(String directoryName) {
        return new File(directoryName).listFiles();
    }
    private ObjectOutput writeFileDetails(Map<String, FileDetail> fileDetails, String
directoryName)
            throws IOException, FileNotFoundException {
        ObjectOutput objectOutput = new ObjectOutputStream(new FileOutputStream(new
File(directoryName, ".vobas")));
        for (FileDetail fileDetail : fileDetails.values()) {
            objectOutput.writeObject(fileDetail);
        }
        objectOutput.close();
        return objectOutput;
    }
}
```