Java – Serializable

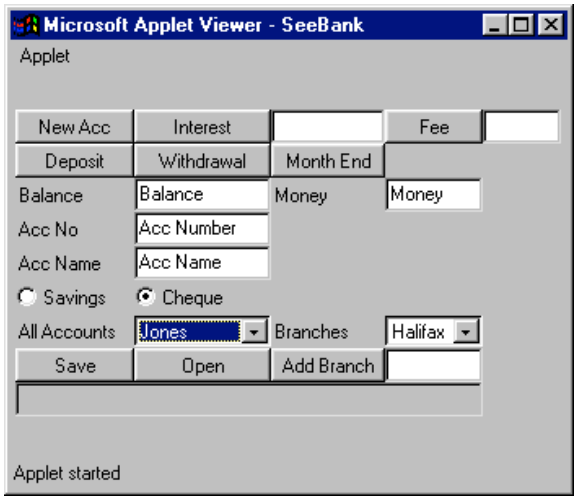
The Serializable (spelt sic or it won't work) interface is used to output classes to streams for saving to disk or to pass along network communications. Where an object contains a reference to other objects (a not uncommon occurrence) all the relevant objects will have to implement Serializable. Luckily it is not necessary to sub class String to implement Serializable. Hashtables and Vectors also can be saved without subclassing. All objects referenced within these collection classes will however need to implement Serializable.

Here the Bank system will be saved to disk as a single file that includes details of all accounts and branches. Bank and SeeBank will have to implement Serializable, this requires java.io.

```
import java.io.*; // for Serializable

public class Bank implements Serializable
{}
```

New buttons will be required to save and open the stored records. An additional TextField is to be used for system messages. All the branches and their account details are kept in a Hashtable. It is this object that will be written to disk.



This code is required to save objects:

```
Try //must try and catch this code
{
        // hard code the file name
        FileOutputStream fos = new FileOutputStream("Bank.dat");
        ObjectOutputStream oos= new ObjectOutputStream(fos);
        oos.writeObject(branches); //branches is the name of the Hashtable class
        oos.close();
        fos.close();
}
```

```
catch(IOException e)
{ //temporary error handling code – can devise a better output later
        System.err.println("Error in file saving " + e.getMessage());
}
```

Although not recommended the above code will work in an Applet.

For initial testing the file is stored in the same directory as the java files run from. Creating a few accounts and running the save routine should create a file Bank.dat. This can be opened in notepad. Much of the data is unreadable but there should be some familiar strings. Do not move on to coding to open a file until a suitable file with some sort of data has been created by the program.

The code for opening a stream will require more catch blocks as there are several things that can go wrong.

```
try
{
        FileInputStream fis = new FileInputStream("Bank.dat");
        ObjectInputStream ois = new ObjectInputStream(fis);
        //need to cast to Hashtable
        branches = (Hashtable)ois.readObject();
        //only want the keys of the Hashtable
        Enumeration list = branches.keys();
        // use the Enumeration to display the keys of the Hashtable
        //
        ois.close();
        fis.close();
}
catch(ClassNotFoundException c)
{ //one of the classes above does not exist
        System.err.println("Class not found " + c.getMessage());
}
catch(FileNotFoundException f)
{ // can't find the file
        System.err.println("File not found " + f.getMessage());
}
catch(IOException e)
{ //file reading problem
        System.err.println("Error reading file " + e.getMessage());
}
```

The program could be converted to an application allowing use of a FileDialog to search for the data file.