Programmieren 2
Selbststudium Semesterwoche 6

1. Datenströme
(Basis: Handout PRG2_SW6_OOP)

1. Gehen Sie nochmals die Kontrollfragen A durch.
A.1. Ein JavaProgramm liest Daten aus einem ???strom.
➔ InputStream

A.2. Ein JavaProgramm schreibt Daten in einen ???strom.
➔ OutputStream

A.3. Welche 2 Arten von Datenströmen unterscheidet Java? Wie sind die entsprechenden Klassen benannt?
➔ Byte-Datenströme (InputStream, OutputStream)
➔ Zeichen-Datenströme (Reader, Writer)

A.4. Wie heissen die beiden Klassen, welche den I/O mit elementaren Datentypen unterstützen?
➔ DataInputStream
➔ DataOutputStream

A.5. Wie schaltet man bei Java programmiertechnisch Datenströme hintereinander?
➔ FileInputStream >>> DataInputStream >>> Programm (read() Methode)
➔ Programm >>> DataOutputStream >>> FileOutputStream (write() Methode)

A.6. In welcher Beziehung stehen die Klassen FileReader und InputStreamReader?
➔ FileReader erweitert InputStreamReader (Beziehung: Einfache Vererbung)
➔ FileReader implementiert zusätzliche Methoden zur Erleichterung des File Handlings.

2. Konsultieren Sie die J2SE API Dokumentation und schreiben Sie von folgenden Klassen je 3 wichtige Methoden auf:
➔ FileInputStream
➔ Wichtige Methoden: read(), close()
http://java.sun.com/j2se/1.4.2/docs/api/java/io/FileInputStream.html

➔ DataInputStream
➔ Wichtige Methoden: read(), readBoolean(), readDouble(), readChar(), readFloat(), readInt()…
http://java.sun.com/j2se/1.4.2/docs/api/java/io/DataInputStream.html

3. Konsultieren Sie die J2SE API Dokumentation und schreiben Sie von folgenden Klassen je 3 wichtige Methoden auf:
➔ BufferedWriter
➔ Wichtige Methoden: write(), close(), flush(), newLine()
http://java.sun.com/j2se/1.4.2/docs/api/java/io/BufferedWriter.html

➔ PrintWriter
➔ Wichtige Methoden: print(), close(), flush()
http://java.sun.com/j2se/1.4.2/docs/api/java/io/PrintWriter.html

➔ FileWriter
➔ Wichtige Methoden: Nur Konstruktoren implementiert; alle Methoden geerbt
http://java.sun.com/j2se/1.4.2/docs/api/java/io/FileWriter.html

2. Dateien

(Basis: Handout PRG2_SW6_OOP, Sourcecode zu den Beispielen)

4. Gehen Sie nochmals die Kontrollfragen B durch.
B.1. Inwiefern unterscheiden sich binäre Dateien und Text Dateien?
➔ Binäre Dateien: Folge von Bytes, beliebige Daten speicherbar.
➔ Text Dateien: Folge von Unicode-Zeichen. Das File ist mit einem Texteditor lesbar.

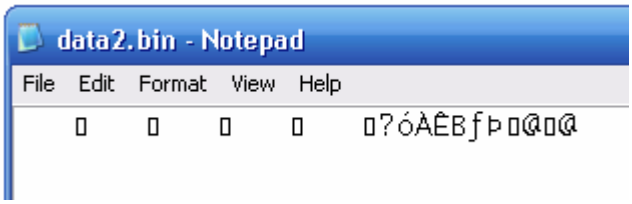
B.2. Was versteht man im Zusammenhang mit Dateien unter "Random Access"?
➔ Beim "wahlfreien Zugriff" kann jede beliebige Stelle einer Datei direkt angesprungen werden – ohne dabei die Datei "sequentiell" durchlaufen zu müssen.

B.3. Wie öffnet man in Java eine Datei?
➔
```
File aFile = new File("dateiname.txt");
FileInputStream aFileInputStream = new FileInputStream(aFile);
DataInputStream aDataInputStream = new DataInputStream(aFileInputStream);
```

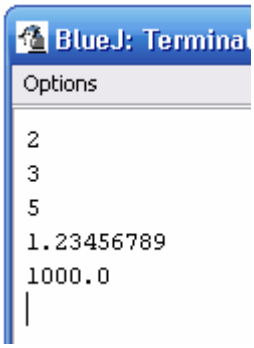
B.4. Beim Lesen einer binären Datei liefert read() ein Byte bzw. einen Wert 0… 255 zurück. Wie wird EOF signalisiert?
➔ Java Doku: "The next byte of data, or -1 if the end of the file is reached"

B.5. Was bewirkt die Methode flush()?
➔ Flush() fordert die sofortige Leerung aller Buffer in der Kette von Writers und OutputStreams.

5. Führen Sie die main()-Methode der Klasse WriteBinaryFile2 aus. Inspizieren Sie die Datei "data2.bin" mit einem einfachen TextEditor/Viewer. Führen Sie die main()-Methode der Klasse ReadBinaryFile2 aus.
➔ Inhalt der Datei data2.bin:



➔ Wird die Datei data2.bin mit ReadBinaryFile2 gelesen, sehen wir folgendes Terminal Output:



6. Schreiben Sie eine neue Klasse WriteBinaryFile3 (basierend auf WriteBinaryFile2), die NEU der Reihe nach 2 intWerte, 3 floatWerte und 4 charWerte in eine Datei "data3.bin" hineinschreibt.

```
import java.io.*;

public class WriteBinaryFile3
{
    public static void main(String[] args)
    {
        String fileName = "data3.bin";
        File aFile = new File(fileName);
        if (!aFile.exists()) {
            try {
                FileOutputStream aFileOutputStream =
                new FileOutputStream(aFile);

                // DataOutputStream unterstützt elementare Datentypen
                DataOutputStream aDataOutputStream =
                new DataOutputStream(aFileOutputStream);

                // 2 int Werte
                aDataOutputStream.writeInt(2);
                aDataOutputStream.writeInt(100);
                aDataOutputStream.writeInt(200);

                // 3 float Werte
                aDataOutputStream.writeInt(3);
                aDataOutputStream.writeFloat(100.53f);
                aDataOutputStream.writeFloat(200.64f);
                aDataOutputStream.writeFloat(300.78f);

                // 4 char Werte
                aDataOutputStream.writeInt(4);
                aDataOutputStream.writeChar('a');
                aDataOutputStream.writeChar('b');
                aDataOutputStream.writeChar('c');
                aDataOutputStream.writeChar('d');

                aFileOutputStream.close();
            }
            catch (IOException e)
            {
                // ...
            }
        }
    }
}
```

7. Schreiben Sie eine neue Klasse ReadBinaryFile3 (basierend auf ReadBinaryFile2), welche die Datei "data3.bin" bzw. dieses proprietäre FileFormat wieder ordentlich einlesen kann.

```
import java.io.*;
public class ReadBinaryFile3
{
    public static void main(String[] args) {
        String fileName = "data3.bin";
        File aFile = new File(fileName);
        if (aFile.exists()) {
            try {
                FileInputStream aFileInputStream =
                new FileInputStream(aFile);
                DataInputStream aDataInputStream =
                new DataInputStream(aFileInputStream);

                // Einlesen der int Werte
                int numberOfInt = aDataInputStream.readInt();
                for (int i = 0; i < numberOfInt; i++) {
                    System.out.println("readInt()="+aDataInputStream.readInt());
                }

                // Einlesen der float Werte
                int numberOfFloat = aDataInputStream.readInt();
                for (int i = 0; i < numberOfFloat; i++) {
                    System.out.println("readFloat()="+aDataInputStream.readFloat());
                }

                // Einlesen der char Werte
                int numberOfChar = aDataInputStream.readInt();
                for (int i = 0; i < numberOfChar; i++) {
                    System.out.println("readChar()="+aDataInputStream.readChar());
                }

                aFileInputStream.close();
            }
            catch (IOException e) {
                // ...
            }
        }
    }
}
```

3. Objekt Serialisierung

(Basis: Handout PRG2_SW6_OOP, Source Code zu den Beispielen)

8. Gehen Sie nochmals die Kontrollfragen C durch.
C.1. Wozu dient die Objekt-Serialisierung?
➔ Unter Serialisierung versteht man eine Abbildung von Objekten auf eine externe sequenzielle Darstellungsform. Serialisierung kann für das Erreichen von Persistenz für ein Objekt verwendet werden, aber auch in verteilten Softwaresystemen spielt Serialisierung eine bedeutende Rolle.

Übliche Speichermedien sind nur in der Lage, Datenströme zu speichern. Um Persistenz für ein Objekt zu erreichen, kann es serialisiert werden. Hier wird der komplette Zustand des Objektes, inklusive aller referenzierten Objekte, in einen Datenstrom umgewandelt, der anschließend auf ein Speichermedium geschrieben wird.

Nach der Serialisierung liegt ein Objekt mehrfach vor: als externe Darstellung (zum Beispiel als Datei) und im Arbeitsspeicher. Wird nach der Serialisierung eine Änderung am Objekt im Arbeitsspeicher vorgenommen, hat dieses keine Auswirkung auf das serialisierte Objekt in der externen Darstellung.

Die Umkehrung der Serialisierung, also die Umwandlung eines Datenstroms in Objekte, wird als Deserialisierung bezeichnet.

Eine Spezialität der Programmiersprache Java ist die Markierung von serialisierbaren Objekten über die Implementierung der Markierungsschnittstelle java.io.Serializable.
Quelle: http://de.wikipedia.org/wiki/Serialisierung

C.2. Was für Variablen werden nicht serialisiert?
➔ Statische Attribute (Klassenvariablen)
➔ Transiente Instanzvariablen (Modifier: `transient`)
➔ Konstanten

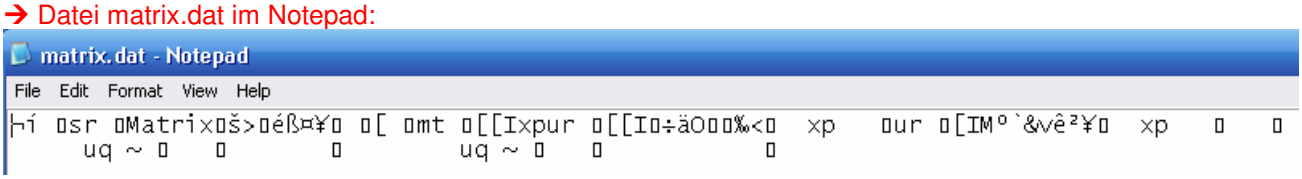
C.3. Welches Interface muss eine Klasse implementieren, damit ihre Objekte serialisierbar sind?
➔ Interface `java.io.Serializable`

C.4. Was ist das Besondere an diesem Interface?
➔ Normalerweise geben Interfaces gewisse Designvorgaben vor. Serializable ist jedoch nur ein Markierungsinterface und erzwingt deshalb keine Methoden. Es dient lediglich zur Markierung einer Klasse, dass diese fähig ist, serialisiert zu werden.

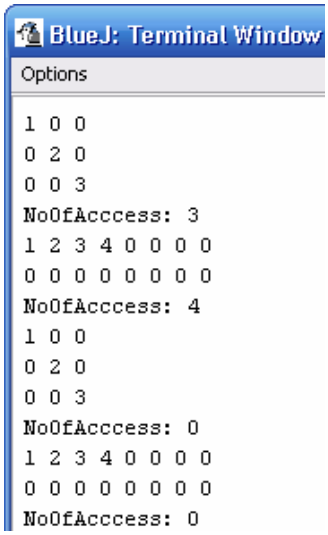
C.5. Was für Objekte nennt man persistent?
➔ Persistenz ist ein Begriff aus der Informatik, der die Fähigkeit bezeichnet, Daten (oder Objekte) in nicht-flüchtigen Speichermedien wie Dateisystemen oder Datenbanken zu speichern.
➔ Objekte, die ausserhalb eines Laufzeitsystems existieren (ausserhalb JVM), werden persistente Objekte genannt.
http://de.wikipedia.org/wiki/Persistenz_(Informatik)

C.6. Das Serialisieren eines bestimmten Objektes kann zahlreiche weitere Objekt-Serialisierungen nach sich ziehen. Erklären Sie.
➔ Wenn ein persistentes Objekt wieder in das Programm geladen wird (=Deserialisierung), müssen zwingend auch alle vom Objekt abhängigen Objekte mitgeladen werden!
➔ Beim Serialisieren wird demzufolge nicht nur das gewünschte Objekt – sondern auch alle indirekt referenzierten Objekte serialisert.

C.9. Spielen Sie das Beispiel in den Unterlagen selber bzw. führen Sie die main()-Methode der Klasse DemoSerialization aus und inspizieren Sie die Datei "matrix.dat" mit einem Editor.
➔ Datei matrix.dat im Notepad:



C.10. Erweitern Sie die main()-Methode so, dass NEU 2 Matrizen (unterschiedlich gross) serialisiert und wieder zurück gelesen werden. Hat's geklappt?




```
import java.io.*;
public class DemoSerialization
{
    public static void main(String[] args)
    {
        try {
            Matrix m1 = new Matrix(3, 3);
            m1.setElt(0, 0, 1);
            m1.setElt(1, 1, 2);
            m1.setElt(2, 2, 3);
            m1.print();

            Matrix m2 = new Matrix(2, 8);
            m2.setElt(0, 0, 1);
            m2.setElt(0, 1, 2);
            m2.setElt(0, 2, 3);
            m2.setElt(0, 3, 4);
            m2.print();

            FileOutputStream aFileOutputStream =
            new FileOutputStream("matrix.dat");
            ObjectOutputStream aObjectOutputStream =
            new ObjectOutputStream(aFileOutputStream);

            // Matrix Objekte in den Stream schrieben
            aObjectOutputStream.writeObject(m1);
            aObjectOutputStream.writeObject(m2);
            aFileOutputStream.close();

            ////////////////////////////////////////

            FileInputStream aFileInputStream =
            new FileInputStream("matrix.dat");
            ObjectInputStream aObjectInputStream =
            new ObjectInputStream(aFileInputStream);

            // Matrix Objekte aus dem Stream lesen
            Object o1 = aObjectInputStream.readObject();
            Object o2 = aObjectInputStream.readObject();
            aFileInputStream.close();

            Matrix mObj1 = (Matrix)o1;
            mObj1.print();
            Matrix mObj2 = (Matrix)o2;
            mObj2.print();
        }
        catch(Exception e)
        {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

4. Binäre und Text Dateien

(Basis: Handout PRG2_SW6_OOP, Sourcecode zu den Beispielen)

11. Studieren Sie die Programmier-Pattern im letzten Teil des Handouts.
12. Führen Sie die main()-Methoden aller Klassen aus. Inspizieren Sie zwischendurch die Dateien mit einem Editor.

5. Ordnung muss sein!

(Basis: Handout PRG2_SW6_DAT)

13. Gehen Sie nochmals die Kontrollfragen A, B und C durch.