

# File IO

---

- Binary Files
- Reading and Writing Binary Files
- Writing Objects to files
- Reading Objects from files

# Binary Files

- Files that are designed to be read by programs and that consist of a sequence of binary digits are called *binary files*
- Binary files store data in the same format used by computer memory to store the values of variables
  - No conversion needs to be performed when a value is stored or retrieved from a binary file
- Java binary files, unlike other binary language files, are portable
  - A binary file created by a Java program can be moved from one computer to another
  - These files can then be read by a Java program, but only by a Java program

# Writing to a Binary File

- The class **ObjectOutputStream** is a stream class that can be used to write to a binary file
  - An object of this class has methods to write strings, values of primitive types, and objects to a binary file
- An **ObjectOutputStream** object is created and connected to a binary file as follows:

```
ObjectOutputStream outputStreamName = new  
    ObjectOutputStream(new  
        FileOutputStream(fileName));
```

# Writing to a Binary File (continued)

- After opening the file, **ObjectOutputStream** methods can be used to write to the file
  - Methods used to output primitive values include **writeInt**, **writeDouble**, **writeChar**, and **writeBoolean**
- *UTF* is an encoding scheme used to encode Unicode characters that favors the ASCII character set
  - The method **writeUTF** can be used to output values of type **String**
- The stream should always be closed after writing

# Example 1

```
1. import java.io.*;
2. public class BinaryFiles
3. {
4.     public static void main(String args[])
5.     {
6.         try{
7.             ObjectOutputStream outputStreamName = new
8.                 ObjectOutputStream(new
9.                     FileOutputStream("output.txt"));
10.             int i = 45; double j = 3.4; char k = 'a';
11.             outputStreamName.writeInt(i);
12.             outputStreamName.writeDouble(j);
13.             outputStreamName.writeChar(k);
14.             outputStreamName.close();
15.         }catch(IOException e){ }
```

File data if opened  
through any editor

```
-í w -@
333333 a
```

# Reading from a Binary File

- The class **ObjectInputStream** is a stream class that can be used to read from a binary file
  - An object of this class has methods to read strings, values of primitive types, and objects from a binary file
- An **ObjectInputStream** object is created and connected to a binary file as follows:

```
ObjectInputStream inStreamName = new  
    ObjectInputStream(new  
        FileInputStream(fileName));
```

# Reading From a Binary File (continued)

- After opening the file, **ObjectInputStream** methods can be used to read to the file
  - Methods used to input primitive values include **readInt**, **readDouble**, **readChar**, and **readBoolean**
  - The method **readUTF** is used to input values of type **String**
- If the file contains multiple types, each item type must be read in exactly the same order it was written to the file
- The stream should be closed after reading

# Example 2

```
1. public class BinaryFiles
2. {
3.     public static void main(String args[])
4.     {
5.         try{
6.             ObjectInputStream inStreamName = new
7.                 ObjectInputStream(new
8.                 FileInputStream("output.txt"));
9.             int i = inStreamName.readInt();
10.            double j = inStreamName.readDouble();
11.            char k = inStreamName.readChar();
12.            System.out.println(i+" "+j+" "+k);
13.        }catch(IOException e){ }
14.    } }
```

Output:  
45 3.4 a

# Checking for the End of a Binary File

- All of the **ObjectInputStream** methods that read from a binary file throw an **EOFException** when trying to read beyond the end of a file
  - This can be used to end a loop that reads all the data in a file
- Note that different file-reading methods check for the end of a file in different ways
  - Testing for the end of a file in the wrong way can cause a program to go into an infinite loop or terminate abnormally

# Objects IO to Binary File

- Objects can also be input and output from a binary file
  - Use the **writeObject** method of the class **ObjectOutputStream** to write an object to a binary file
  - Use the **readObject** method of the class **ObjectInputStream** to read an object from a binary file
  - In order to use the value returned by **readObject** as an object of a class, it must be type cast first:

```
SomeClass someObject =  
    (SomeClass)objectInputStream.readObject();
```

# Object IO to Binary Files

- In addition, the class of the object being read or written must implement the ***Serializable*** interface
- In order to make a class serializable, simply add **`implements Serializable`** to the heading of the class definition  

```
public class SomeClass implements Serializable
```
- When a serializable class has instance variables of a class type, then all those classes must be serializable also
  - A class is not serializable unless the classes for all instance variables are also serializable for all levels of instance variables within classes

# Example

- A simple Student Class

```
1. class Student implements Serializable
2. {
3.     private String Name;
4.     private int Age;
5.     private String ID;
6.     public Student(String Name, int Age, String ID)
7.     {
8.         this.Name = Name;
9.         this.Age = Age;
10.        this.ID = ID;
11.    }
12.    public String toString()
13.    {
14.        return Name + ":" + ID + ":" + Age;
15.    }
16. }
```

# Write Object to Binary File

```
1. public class BinaryFiles
2. {
3.     public static void main(String args[])
4.     {
5.         try{
6.             ObjectOutputStream outputStreamName = new
7.             ObjectOutputStream(new
8.             FileOutputStream("output.txt"));
9.             Student s = new Student("Ahmed",21,"232323");
10.            outputStreamName.writeObject(s);
11.            outputStreamName.close();
12.        }catch(IOException e){ }
```

# Reading Objects

```
1. public class BinaryFiles
2. {
3.     public static void main(String args[])
4.     {
5.         try{
6.             ObjectInputStream inStreamName = new
7.                 ObjectInputStream(new
8.                 FileInputStream("output.txt"));
9.             Student s = (Student) inStreamName.readObject();
10.            System.out.println(s);
11.        }catch(IOException e){ }
12.        catch(ClassNotFoundException e){ }
13.    }}
```

Output:

Ahmed:232323:12

# Introduction to Scanner Class

- Java Introduced a new class called Scanner in jdk 1.5
- This class can be used to read Data from the keyboard or from a file
- The following code will read an integer from the user

```
Scanner sc = new Scanner(System.in);  
int i = sc.nextInt();
```

- The following code will read and integer from a file

```
Scanner sc = new Scanner(new  
    FileInputStream(fileName));  
int i = sc.nextInt();
```

# Scanner Class (contd)

- Some Useful functions in the Scanner class
  - **byte** [nextByte\(\)](#) reads the next Byte
  - **double** [nextDouble\(\)](#) reads the next Double
  - **int** [nextInt\(\)](#) reads the next Integer
  - **String** [nextLine\(\)](#) reads a complete Line
- Checkout <http://java.sun.com/j2se/1.5.0/docs/api/> for complete documentation