Advanced Computer Programming

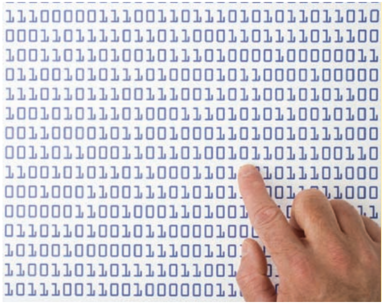
[Lecture 11]

Saeed Reza Kheradpisheh

kheradpisheh@ut.ac.ir

Department of Computer Science
Shahid Beheshti University
Spring 1397-98

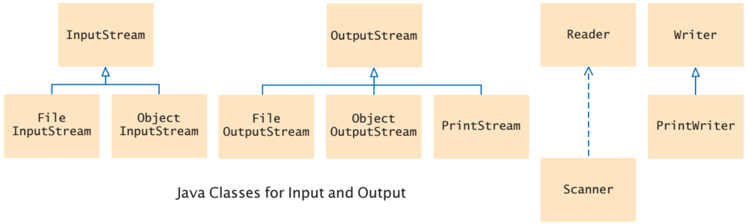
STREAMS



By learning about streams, we will learn more about how to write Java programs that interact with files and other sources of bytes and characters.

Input/Ouput Streams

- There are two fundamentally different ways to store data: in text format or binary format.
- In text format, data items are represented in human-readable form, as a sequence of characters. E.g., ***12,345*** is stored as ***'1' '2' '3' '4' '5'***.
- In binary form, data items are represented in bytes. E.g., ***12,345*** is stored as a sequence of four bytes: ***0 0 48 57***.
- The Java library provides two sets of classes for handling input and output. Streams handle binary data. Readers and writers handle data in text form.



Java Classes for Input and Output

Binary File Streams

- To read binary data from a disk file, you create a `FileInputStream` object:

```
InputStream inputStream = new FileInputStream("input.bin");
```

- Similarly, you use `FileOutputStream` objects to write data to a disk file in binary form:

```
OutputStream outputStream = new FileOutputStream("output.bin");
```

Reading Binary Files

- The `InputStream` class has a method, `read`, to read a single byte at a time.
- The `FileInputStream` class overrides this method to read the bytes from a disk file.
- The `InputStream.read` method returns an **int**, not a byte.
- It returns the byte read as an integer **between 0 and 255** or, when it is at the end of the input, it returns -1.

```
InputStream in = . . .;
int next = in.read();
if (next != -1)
{
   Process next   // A value between 0 and 255
}
```

Writing Binary Files

- The `OutputStream` class has a `write` method to write a single byte.
- The parameter variable of the write method has type **int**, but only the **lowest eight bits** of the argument are written to the stream.

```
OutputStream out = . . .;
int value = . . .; // Should be between 0 and 255
out.write(value);
```

When you are done writing to the file, you should close it:

```
out.close();
```

Writing Binary Files

- The `OutputStream` class has a `write` method to write a single byte.
- The parameter variable of the write method has type **int**, but only the **lowest eight bits** of the argument are written to the stream.

```
OutputStream out = . . .;
int value = . . .; // Should be between 0 and 255
out.write(value);
```

When you are done writing to the file, you should close it:

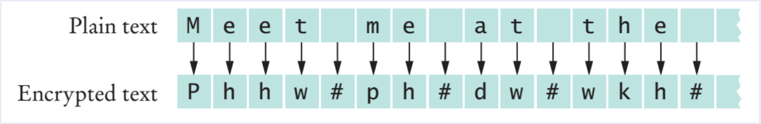
```
out.close();
```

- The job of an input stream is to get bytes, not to analyze them.
- If you want to read numbers, strings, or other objects, you have to combine the class with other classes whose responsibility is to group individual bytes or characters together into numbers, strings, and objects.

The Caesar Cipher

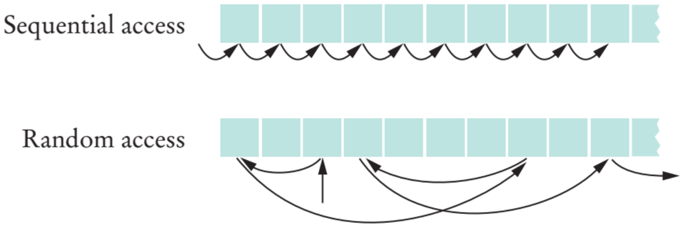
Exercise (`The Caesar Cipher`)

In a more complex encryption program, you would read a block of bytes, encrypt the block, and write it out.



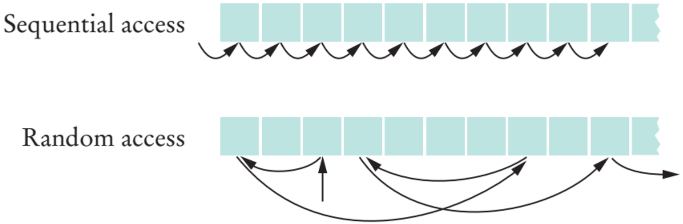
Random Access

- Consider a file that contains a set of bank accounts and we want to change the balances of some of the accounts.
- Random access is needed when we want to access specific locations in a file and change only those locations.



Random Access

- Consider a file that contains a set of bank accounts and we want to change the balances of some of the accounts.
- Random access is needed when we want to access specific locations in a file and change only those locations.



- Only disk files support random access; the `System.in` and `System.out` streams, which are attached to the **keyboard** and the **terminal window**, do not.

File Pointer

- Each disk file has a special file pointer position and the next read command starts reading input at the file pointer location.
- In Java, you use a `RandomAccessFile` object to access a file and move a file pointer.
- To open a random access file, you supply a ***file name*** and a string to specify the ***open mode***.
- You can open a file either for reading only `("r")` or for reading and writing `("rw")`:

```
RandomAccessFile f = new RandomAccessFile("bank.dat", "rw");
```

- The `seek` method moves the pointer to a given postion:

```
f.seek(position);
```

- To find the current position of the file pointer:

```
long position = f.getFilePointer();
```

- To determine the number of bytes in a file :

```
long fileLength = f.length();
```

Read and Write in Random Access Files

- The `readInt` and `writeInt` methods read and write integers as four-byte quantities.
- The `readDouble` and `writeDouble` methods process double-precision floatingpoint numbers as eight-byte quantities.

```
double x = f.readDouble();
f.writeDouble(x);
```

- E.g., If we save the account number as an integer and the balance as a double value, then each bank account record consists of 12 bytes.

Exercise (`Bank Account`)

Write a program to restore and update bank account records in a binary file with random access.

Object Streams

- The `ObjectOutputStream` class can save entire objects out to disk.

```
BankAccount b = . . .;
ObjectOutputStream out = new ObjectOutputStream(
      new FileOutputStream("bank.dat"));
out.writeObject(b);
```

- The `ObjectInputStream` class can read the objects from disk files back to memory.

```
ObjectInputStream in = new ObjectInputStream(
      new FileInputStream("bank.dat"));
BankAccount b = (BankAccount) in.readObject();
```

- The `readObject` method can throw a `ClassNotFoundException`, it is a checked exception, so you need to catch or declare it.

Read and write Arrays/ArrayLists

- You can store a whole bunch of objects in an array list or array, or inside another object, and then save that object:

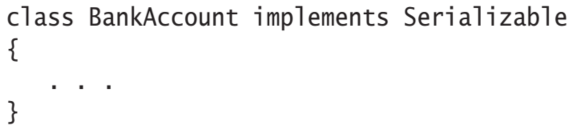
```
ArrayList<BankAccount> a = new ArrayList<BankAccount>();
// Now add many BankAccount objects into a
out.writeObject(a);
```

- With one instruction, you can save the array list and all the objects that it references:

```
ArrayList<BankAccount> a = (ArrayList<BankAccount>) in.readObject();
```

Serialization

- To place objects of a particular class into an object stream, the class must implement the `Serializable` interface.



```
class BankAccount implements Serializable
{
   . . .
}
```

- Each object is assigned a serial number on the stream.
- If the same object is saved twice, only the serial number is written out the second time.
- When the objects are read back in, duplicate serial numbers are restored as references to the same object.

Exercise (`Bank Account`)

Write a program to restore and update bank account objects in a binary file using serialization.