SCHULICH
School of Engineering

ENSF 607

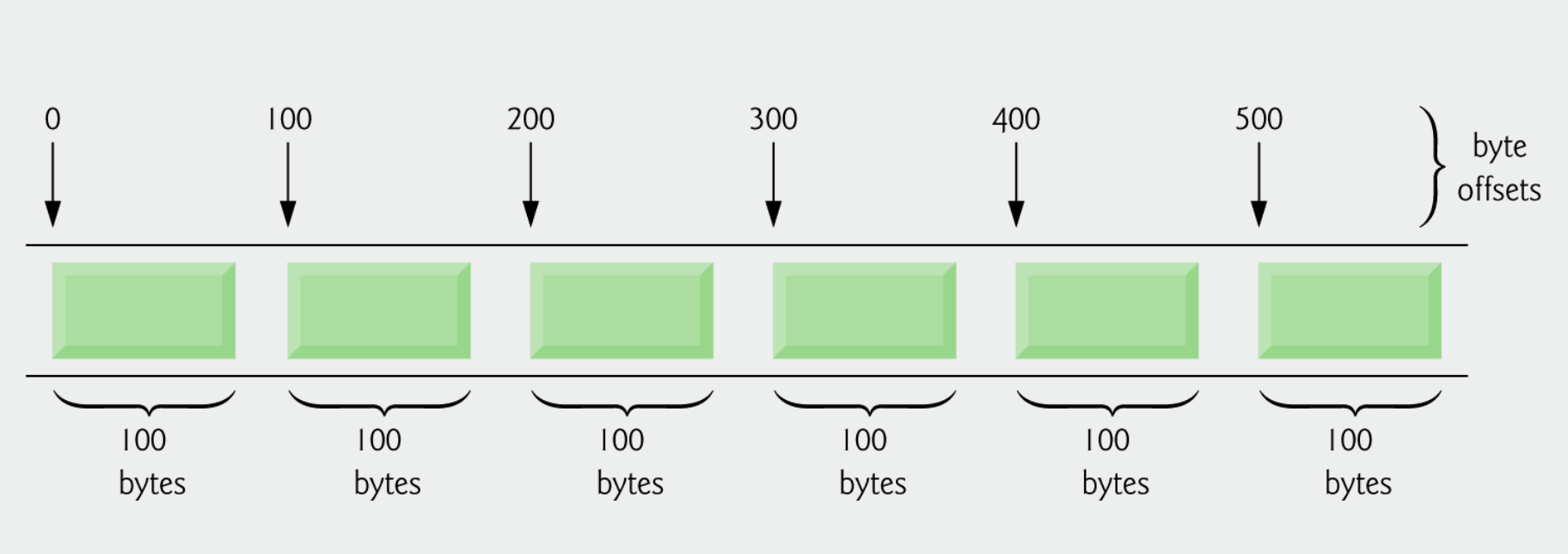
7 - Java Object Streams (Part II)

Random Access to the File

Random-Access Files

- Data in many sequential files cannot be modified without risk of destroying other data in file
  - Records in sequential-access files are not usually updated in place. Instead, entire file is usually rewritten.
- Sequential-access files is inappropriate for instant-access applications.
  - Instant-access applications are applications in which desired information must be located immediately
- Instant access is possible with random-access files (also called direct-access files) and databases
- Different techniques for creating random-access files
  - Simplest: Require that all records in file be same fixed length
    - Easy to calculate (as a function of record size and record key) exact location of any record relative to beginning of file

Java's view of a random-access file.



Creating a Random-Access File

- RandomAccessFile class
  - Using RandomAccessFile, program can read or write data beginning at location specified by file-position pointer
  - Methods readInt, readDouble, readChar used to read integer, double and character data from file
  - Methods writeInt, writeDouble, writeChars used to write integer, double and string data to file
  - File-open mode – specifies whether file is opened for reading ("r"), or for both reading and writing ("rw"). File-open mode specified as second argument to RandomAccessFile constructor

Create Random Access File

```
RandomAccessFile ra = null;


try
{
    ra = new RandomAccessFile( "clients.dat", "rw" );
 }
catch(...)
{
   ...
}
```

Reading from Random Access File

```
setAccount( ra.readInt() );
setBalance( ra.readDouble() );

char name[] = new char[ 15 ], temp;
for ( int count = 0; count < name.length; count++ )
{
 temp = ra.readChar();
 name[ count ] = temp;
}
setName(name);
```

Writing to the Random Access File

```
ra.writeInt( getAccount() );
ra.writeDouble( getBalance() );
StringBuffer buffer = null;
if ( name != null )
   buffer = new StringBuffer( name );
else
   buffer = new StringBuffer( 15 );

buffer.setLength( 15 );
ra.writeChars( buffer.toString() );
```

Writing Data Randomly

- RandomAccessFile method seek positions file-position pointer to a specific location in a file relative to beginning of file
- You may get the position of the pointer by method: getFilePointer
- Size of each record is known, so location in file of a specific record can be found by multiplying size of record with number of record
- Once location known, new record data can be written without worrying about rest of file, as each record is always same size

Class File

Class File

- Class File useful for retrieving information about files and directories from disk
- Objects of class File do not open files or provide any file-processing capabilities
- Class File provides four constructors:
  1. Takes String specifying name and path (location of file on disk)
  2. Takes two Strings, first specifying path and second specifying name of file
  3. Takes File object specifying path and String specifying name of file
  4. Takes URI object specifying name and location of file

File methods.

- `boolean canRead()`
- `boolean canWrite()`
- `boolean exists()`
- `boolean isDirectory()`
- `boolean isAbsolute()  // is path an absolute path?`
- `String getAbsoultePath()`
- `String getName()`
- `String getPath()`
- `String geParent()`
- `Long length()`
- `long lastModified()`
- `String [ ] list()  //returns the list of files in a directory`
- `boolean isFile()`
  - to determine whether a File object represents a file (not a directory) before attempting to open the file.

Access to Information

```
import java.io.File;
public class FileDemo
{
  public static void main( String [] args)
  {
    String path =  "/Users/mahmood";
    File name = new File(path );
    if ( name.exists() )
    {
      System.out.printf( ”%s”, name.getName());
```

Access to directory

```
      if ( name.isDirectory() ) {
         String directory[] = name.list();
         System.out.println( "\n\nDirectory contents:\n" );
         for ( String directoryName : directory )
             System.out.printf( "%s\n", directoryName );
       }
     }
  else {
     System.out.printf( "%s %s", path, "does not exist." );
    }
  } // end of main
} // end class definition
```

java.util Classes fo Read/Write file

- Scanner – can be used to easily read data from a file
- Formatter – can be used to easily write data to a file

Formatter Class

- Formatter class can be used to open a text file for writing
  - Pass name of file to constructor
  - If file does not exist, will be created
  - If file already exists, contents are truncated (discarded)
  - Use method format to write formatted text to file
  - Use method close to close the Formatter object (if method not called, OS normally closes file when program exits)

Formatter Class Exceptions

- Possible exceptions
  - SecurityException – occurs when opening file using Formatter object, if user does not have permission to write data to file
  - FileNotFoundException – occurs when opening file using Formatter object, if file cannot be found and new file cannot be created
  - NoSuchElementException – occurs when invalid input is read in by a Scanner object
  - FormatterClosedException – occurs when an attempt is made to write to a file using an already closed Formatter object

Example: Using Class Formatter

```
private Formatter output;
try {
   output = new Formatter( ”myfile.txt" );
    }
    catch ( SecurityException securityException )
    {   System.err.println( "You do not have write access to this file." );
       System.exit( 1 );
     } // end catch
     catch ( FileNotFoundException filesNotFoundException )
      {
         System.err.println( "Error creating file." );
         System.exit( 1 );
       } // end catch
```

Writing to the file

```
try{
      output.format( "%d %s %s %.2f\n”,  data1, data2, data3, data4 );
   }
 catch ( FormatterClosedException formatterClosedException )
    {
      System.err.println( "Error writing to file." );
     } // end catch

 output.close();
```

Scanner Class

- We normally like to read the data as int or double, which neatly formatted.
  - Scanner make it possible
- Scanner object can be used to read data sequentially from a text file in the form of tokens, separated by a delimiter, such as whitespace.
  - Pass File object representing file to be read to Scanner constructor
  - FileNotFoundException occurs if file cannot be found
  - Data read from file using same methods as for keyboard input – nextInt, nextDouble, next, etc.
  - IllegalStateException occurs if attempt is made to read from closed Scanner object

Using Scanner

```
Scanner sc = new Scanner(new File("myNumbers"));
while (sc.hasNextLong()) {
        long aLong = sc.nextLong();
 }
```

- You can also use buffered input streams wrapped within Scanner constructor. See the example next page:

Using Scanner for Buffered Input

```
import java.io.*;
import java.util.Scanner;
public static void main(String[] args) throws IOException {
  Scanner s = null;
  try {
     s = new Scanner(new BufferedReader(new FileReader("xanadu.txt")));
        while (s.hasNext()) {
           System.out.println(s.next());
        }
   } finally {
      if (s != null) {
           s.close();
      }
   }
  }
}
```