EIU
TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

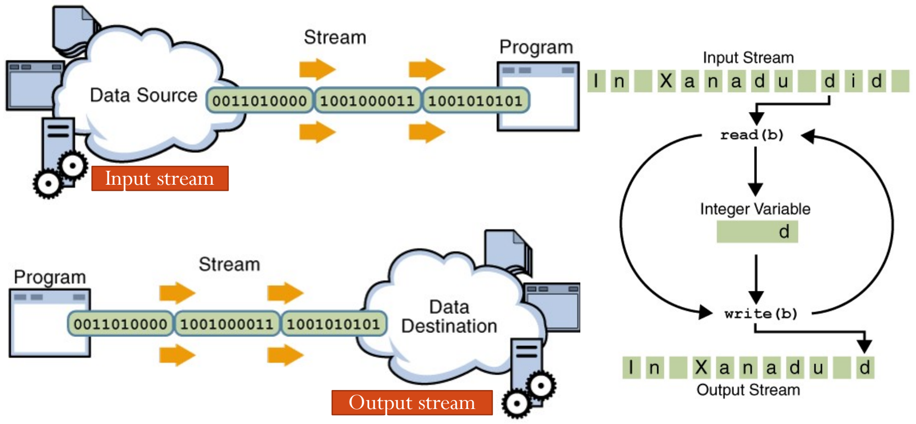
CSE203 – OBJECT ORIENTED PROGRAMMING

FILE (IO)

1

What are streams?

- A stream is an object managing a data source in which operations such as read data in the stream to a variable, write values of a variable to the stream associated with type conversions are performed automatically. These operations treat data as a chain of units (byte/character/data object) and data are processed in unit-by-unit manner.



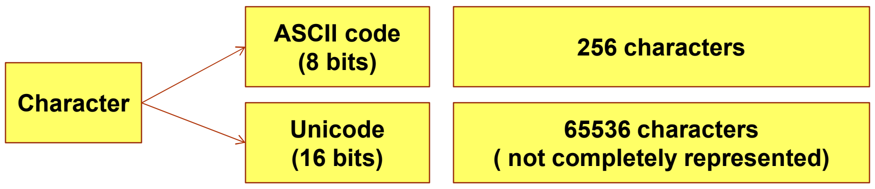
2

2

Objectives

- Distinguishing Text, UTF, and Unicode
- How to access directories and files?
- How to access text files.
- How to access binary files?
- How to read/write objects from/to files

1- Text, UTF, and Unicode



Unicode character: a character is coded using 16/32 bits
UTF: **U**niversal Character Set – UCS- **T**ransformation **F**ormat
UTF: *Unicode transformation format , a* Standard for compressing strings of Unicode text .
UTF-8: A standard for compressing Unicode text to 8-bit code units.
Refer to: http://www.unicode.org/versions/Unicode7.0.0/

Java :
- Uses UTF to read/write Unicode
- Helps converting Unicode to external 8-bit encodings and vice versa.

2- Introduction to the java.io Package

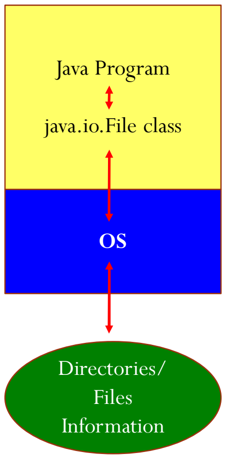
- Java treats all data sources (file, directory, IO devices,…) as streams
- The java.io package contains Java APIs for accessing to/from a stream.
- A stream can be a binary stream.
  - Binary low-level stream: data unit is a physical byte.
  - Binary high-level stream: data unit is primitive data type value or a string.
  - Object stream: data unit is an object.
- A stream can be a character stream in which a data unit is an Unicode character.

5

5

3- Accessing directories and files

The java.io.File Class
Class represents a file or a directory managed by operating system.



Constructor Summary
File(File parent, String child) Creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname) Creates a new File instance by converting the given pathname string into an abstract pathname.
File(String parent, String child) Creates a new File instance from a parent pathname string and a child pathname string.
File(URI uri) Creates a new File instance by converting the given file: URI into an abstract pathname.

6

6

Accessing directories and files…

The java.io.File Class…

Common Methods:

boolean canExecute(), canRead(), canWrite()
boolean exists(), isDirectory(), isFile()
String getAbsolutePath(), getCanonicalPath(),
getName(), getParent()
String[] list()
boolean delete(), createNewFile(), mkDir(),
rename(File newName)
long length()

This class helps accessing file/directory information only. It does not have any method to access data in a file.

Method Invoked	Returns on Microsoft Windows	Returns on Solaris (Unix)
getAbsolutePath()	c:\java\examples\examples\xanadu.txt	/home/cafe/java/examples/xanadu.txt
getCanonicalPath()	c:\java\examples\xanadu.txt	/home/cafe/java/examples/xanadu.txt

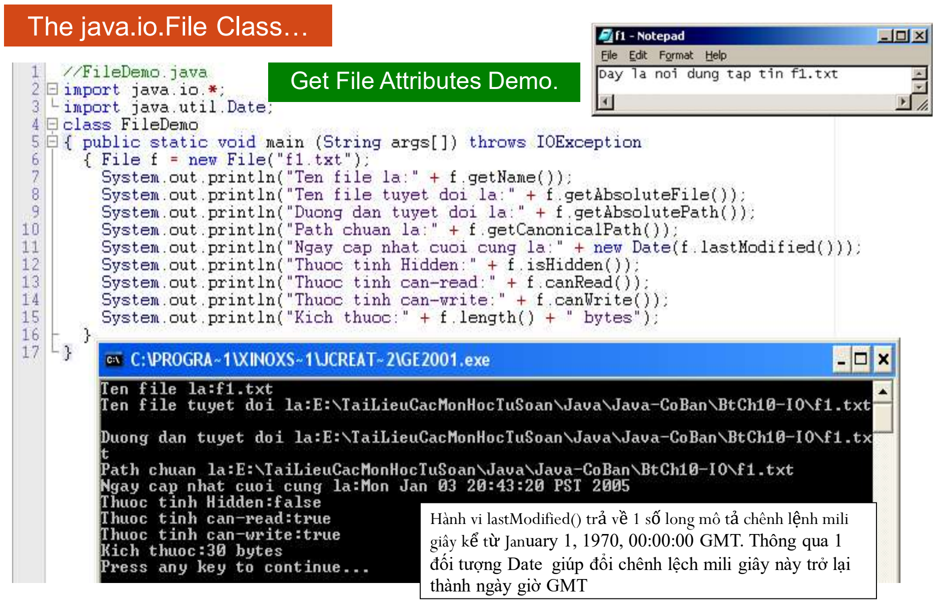
7

7

Accessing directories and files…

The java.io.File Class…

Get File Attributes Demo.



```java
//FileDemo.java
import java.io.*;
import java.util.Date;
class FileDemo
{ public static void main (String args[]) throws IOException
  { File f = new File("f1.txt");
    System.out.println("Ten file la:" + f.getName());
    System.out.println("Ten file tuyet doi la:" + f.getAbsoluteFile());
    System.out.println("Duong dan tuyet doi la:" + f.getAbsolutePath());
    System.out.println("Path chuan la:" + f.getCanonicalPath());
    System.out.println("Ngay cap nhat cuoi cung la:" + new Date(f.lastModified()));
    System.out.println("Thuoc tinh Hidden:" + f.isHidden());
    System.out.println("Thuoc tinh can-read:" + f.canRead());
    System.out.println("Thuoc tinh can-write:" + f.canWrite());
    System.out.println("Kich thuoc:" + f.length() + " bytes");
  }
}
```

```
Day la noi dung tap tin f1.txt
```

```
Ten file la:f1.txt
Ten file tuyet doi la:E:\TaiLieuCacMonHocTuSoan\Java\Java-CoBan\BtCh10-IO\f1.txt

Duong dan tuyet doi la:E:\TaiLieuCacMonHocTuSoan\Java\Java-CoBan\BtCh10-IO\f1.tx
t
Path chuan la:E:\TaiLieuCacMonHocTuSoan\Java\Java-CoBan\BtCh10-IO\f1.txt
Ngay cap nhat cuoi cung la:Mon Jan 03 20:43:20 PST 2005
Thuoc tinh Hidden:false
Thuoc tinh can-read:true
Thuoc tinh can-write:true
Kich thuoc:30 bytes
Press any key to continue...
```

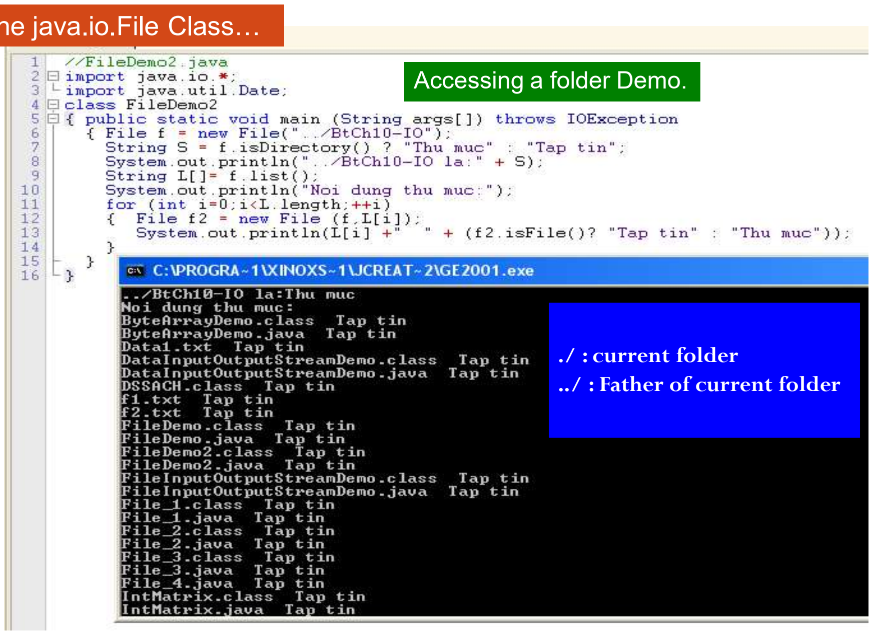
Hành vi lastModified() trả về 1 số long mô tả chênh lệnh mili giây kể từ January 1, 1970, 00:00:00 GMT. Thông qua 1 đối tượng Date giúp đổi chênh lệch mili giây này trở lại thành ngày giờ GMT

8

8

Accessing directories and files…

The java.io.File Class…



Accessing a folder Demo.

```
//FileDemo2.java
import java.io.*;
import java.util.Date;
class FileDemo2
{ public static void main (String args[]) throws IOException
   { File f = new File("../BtCh10-IO");
     String S = f.isDirectory() ? "Thu muc" : "Tap tin";
     System.out.println("../BtCh10-IO la:" + S);
     String L[]= f.list();
     System.out.println("Noi dung thu muc:");
     for (int i=0;i<L.length;++i)
     {  File f2 = new File (f,L[i]);
        System.out.println(L[i] +"   " + (f2.isFile()? "Tap tin" : "Thu muc"));
     }
   }
}
```

C:\PROGRA~1\XINOXS~1\JCREAT~2\GE2001.exe

```
../BtCh10-IO la:Thu muc
Noi dung thu muc:
ByteArrayDemo.class  Tap tin
ByteArrayDemo.java  Tap tin
Data1.txt  Tap tin
DataInputOutputStreamDemo.class  Tap tin
DataInputOutputStreamDemo.java  Tap tin
DSSACH.class  Tap tin
f1.txt  Tap tin
f2.txt  Tap tin
FileDemo.class  Tap tin
FileDemo.java  Tap tin
FileDemo2.class  Tap tin
FileDemo2.java  Tap tin
FileInputOutputStreamDemo.class  Tap tin
FileInputOutputStreamDemo.java  Tap tin
File_1.class  Tap tin
File_1.java  Tap tin
File_2.class  Tap tin
File_2.java  Tap tin
File_3.class  Tap tin
File_3.java  Tap tin
File_4.java  Tap tin
IntMatrix.class  Tap tin
IntMatrix.java  Tap tin
```

./ : current folder

../ : Father of current folder

9

9

4- Access Text Files

- Character Streams:
  - Two ultimate abstract classes of character streams are Reader and Writer.
  - Reader: input character stream will read data from data source (device) to variables (UTF characters).
  - Writer: stream will write UTF characters to data source (device).

10

10

Access Text Files … Character Streams

- java.io.Reader (implements java.io.Closeable, java.lang.Readable) (abstract)
  - java.io.BufferedReader
    - java.io.LineNumberReader
  - java.io.CharArrayReader
  - java.io.FilterReader
    - java.io.PushbackReader
  - java.io.InputStreamReader
    - java.io.FileReader
  - java.io.PipedReader
  - java.io.StringReader

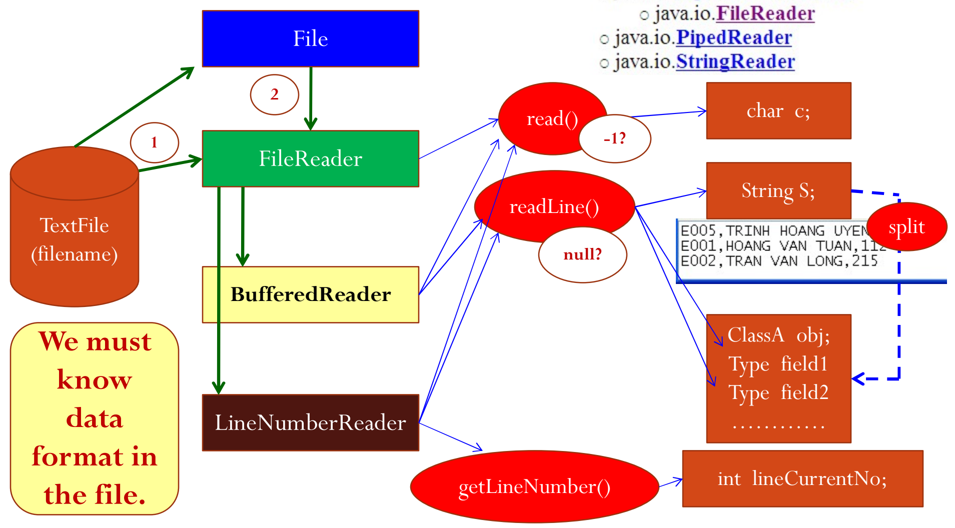
- java.io.Writer (implements java.lang.Appendable, java.io.Closeable, java.io.Flushable) (abstract)
  - java.io.BufferedWriter
  - java.io.CharArrayWriter
  - java.io.FilterWriter
  - java.io.OutputStreamWriter
    - java.io.FileWriter
  - java.io.PipedWriter
  - java.io.PrintWriter
  - java.io.StringWriter

11

11

Access Text Files … Reading Data

- java.io.Reader
  - java.io.BufferedReader
    - java.io.LineNumberReader
  - java.io.CharArrayReader
  - java.io.FilterReader
    - java.io.PushbackReader
  - java.io.InputStreamReader
    - java.io.FileReader
  - java.io.PipedReader
  - java.io.StringReader

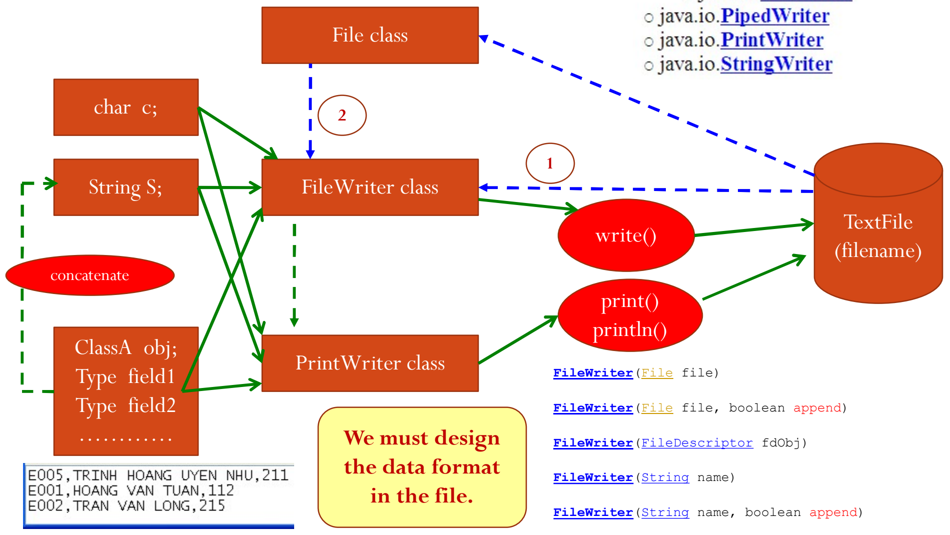


12

12

Access Text Files …
Writing Data

- java.io.Writer
  - java.io.BufferedWriter
  - java.io.CharArrayWriter
  - java.io.FilterWriter
  - java.io.OutputStreamWriter
    - java.io.FileWriter
  - java.io.PipedWriter
  - java.io.PrintWriter
  - java.io.StringWriter



```
E005,TRINH HOANG UYEN NHU,211
E001,HOANG VAN TUAN,112
E002,TRAN VAN LONG,215
```

```
FileWriter(File file)
FileWriter(File file, boolean append)
FileWriter(FileDescriptor fdObj)
FileWriter(String name)
FileWriter(String name, boolean append)
```

13

13

Access Text Files …
Case study 1

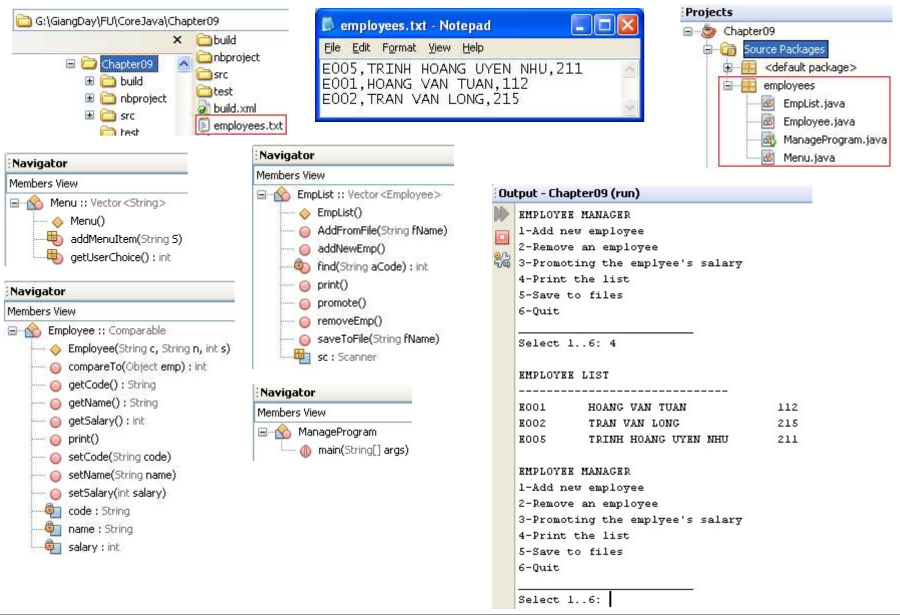
Problem

- Each employee details include: code, name, salary
- The text file, named employees.txt contains some initial employee details in the following line-by-line format
  code,name,salary
- Write a Java program having a simple menu that allows users managing a list of employees. Functions are supported:
  - Adding new employee
  - Removing employee.
  - Promoting the salary of an employee.
  - Listing employee details.
  - Save the list to file
  - Quit

14

14

Access Text Files …: Case study 1- Design



```
G:\GiangDay\FU\CoreJava\Chapter09
Chapter09
  build
  nbproject
  src
  test
build
nbproject
src
test
build.xml
employees.txt
```

employees.txt - Notepad

```
E005,TRINH HOANG UYEN NHU,211
E001,HOANG VAN TUAN,112
E002,TRAN VAN LONG,215
```

Projects

```
Chapter09
  Source Packages
    <default package>
    employees
      EmpList.java
      Employee.java
      ManageProgram.java
      Menu.java
```

Navigator – Members View

```
Menu :: Vector<String>
  Menu()
  addMenuItem(String S)
  getUserChoice() : int
```

Navigator – Members View

```
Employee :: Comparable
  Employee(String c, String n, int s)
  compareTo(Object emp) : int
  getCode() : String
  getName() : String
  getSalary() : int
  print()
  setCode(String code)
  setName(String name)
  setSalary(int salary)
  code : String
  name : String
  salary : int
```

Navigator – Members View

```
EmpList :: Vector<Employee>
  EmpList()
  AddFromFile(String fName)
  addNewEmp()
  find(String aCode) : int
  print()
  promote()
  removeEmp()
  saveToFile(String fName)
  sc : Scanner
```

Navigator – Members View

```
ManageProgram
  main(String[] args)
```

Output - Chapter09 (run)

```
EMPLOYEE MANAGER
1-Add new employee
2-Remove an employee
3-Promoting the emplyee's salary
4-Print the list
5-Save to files
6-Quit
_________________________
Select 1..6: 4

EMPLOYEE LIST
------------------------------
E001      HOANG VAN TUAN             112
E002      TRAN VAN LONG              215
E005      TRINH HOANG UYEN NHU       211

EMPLOYEE MANAGER
1-Add new employee
2-Remove an employee
3-Promoting the emplyee's salary
4-Print the list
5-Save to files
6-Quit
_________________________
Select 1..6:
```

15

15

Access Text Files …: Case study 1- Implementations

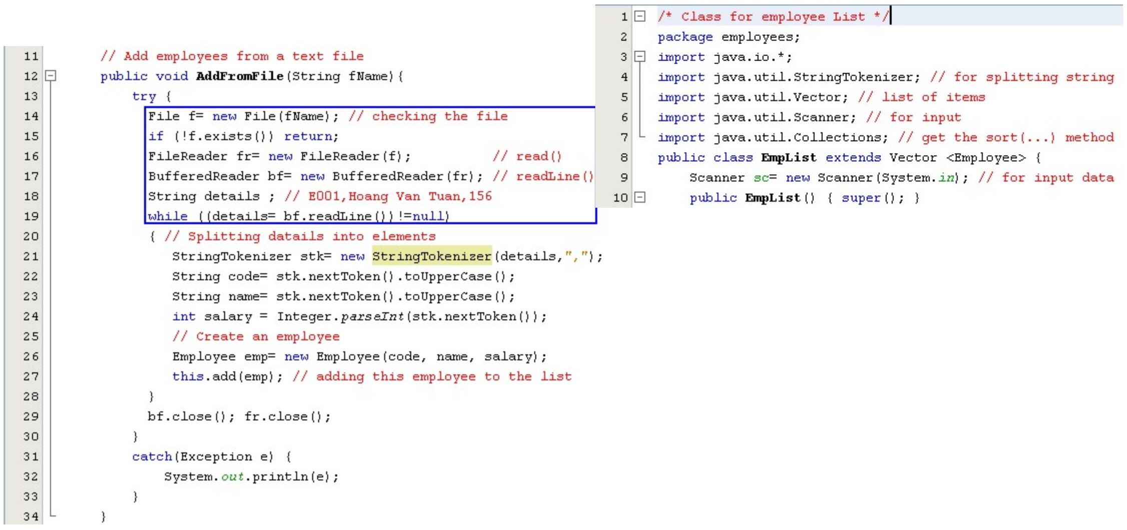
```java
/* Class for simple menu */
package employees;
import java.util.Vector;
import java.util.Scanner;
public class Menu extends Vector <String> {
  public Menu() { super(); }
  void addMenuItem(String S) { this.add(S); }
  // DO YOURSELF
  // Refer to the older case study
  int getUserChoice  (){...}
}
```

```java
/* Class for an employee */
package employees;
import java.lang.Comparable;
public class Employee implements Comparable   {
   private String code;
   private String name;
   private int salary;
   // DO YOURSELF
   public Employee(String c, String n, int s){...}
   // Print details to the screen
   public void print( ){...}
   // getters and setters - DO YOURSELF
    public String getCode() {...}
    public void setCode(String code) {...}
    public String getName() {...}
    public void setName(String name) {...}
    public int getSalary() {...}
    public void setSalary(int salary) {...}
    // Implement the Comparable interface for sorting operation
    public int compareTo(Object emp) {
       return this.getCode().compareTo(((Employee)emp).getCode());
    }
}
```

16

16

Access Text Files ...: Case study 1- Implementations

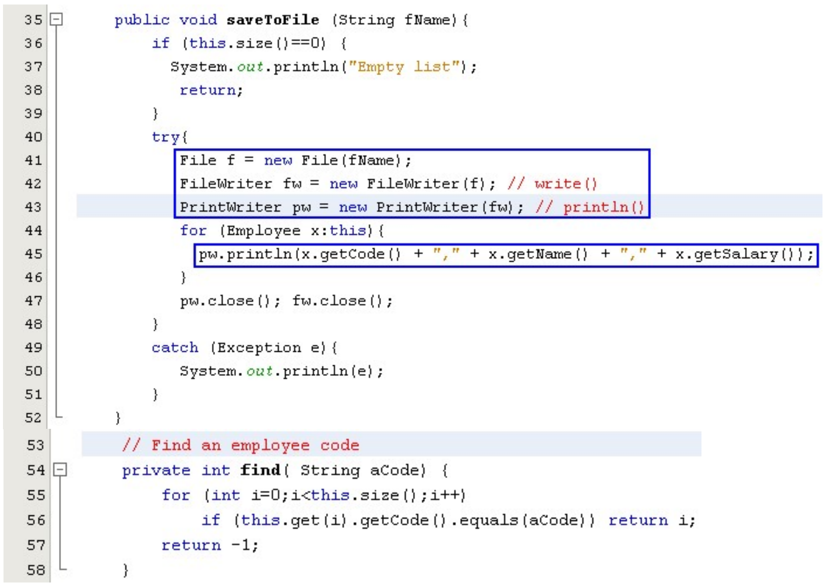


```
/* Class for employee List */
package employees;
import java.io.*;
import java.util.StringTokenizer; // for splitting string
import java.util.Vector; // list of items
import java.util.Scanner; // for input
import java.util.Collections; // get the sort(...) method
public class EmpList extends Vector <Employee> {
    Scanner sc= new Scanner(System.in); // for input data
    public EmpList() { super(); }
```

```
// Add employees from a text file
public void AddFromFile(String fName){
    try {
     File f= new File(fName); // checking the file
     if (!f.exists()) return;
     FileReader fr= new FileReader(f);           // read()
     BufferedReader bf= new BufferedReader(fr); // readLine()
     String details ; // E001,Hoang Van Tuan,156
     while ((details= bf.readLine())!=null)
      { // Splitting datails into elements
        StringTokenizer stk= new StringTokenizer(details,",");
        String code= stk.nextToken().toUpperCase();
        String name= stk.nextToken().toUpperCase();
        int salary = Integer.parseInt(stk.nextToken());
        // Create an employee
        Employee emp= new Employee(code, name, salary);
        this.add(emp); // adding this employee to the list
      }
     bf.close(); fr.close();
    }
    catch(Exception e) {
        System.out.println(e);
    }
}
```

17

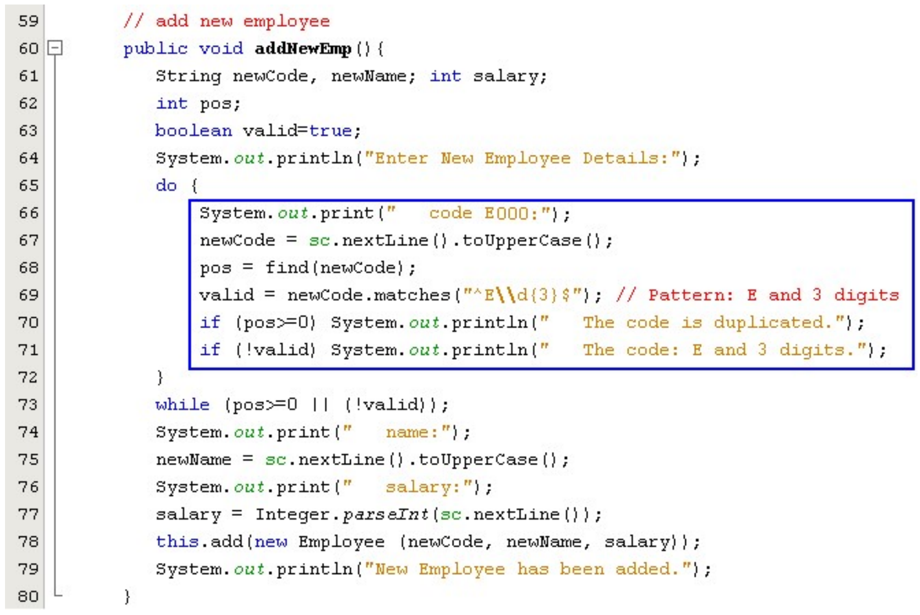
Access Text Files ...: Case study 1- Implementations



```
public void saveToFile (String fName){
    if (this.size()==0) {
      System.out.println("Empty list");
       return;
    }
    try{
       File f = new File(fName);
       FileWriter fw = new FileWriter(f); // write()
       PrintWriter pw = new PrintWriter(fw); // println()
       for (Employee x:this){
         pw.println(x.getCode() + "," + x.getName() + "," + x.getSalary());
       }
       pw.close(); fw.close();
    }
    catch (Exception e){
       System.out.println(e);
    }
}
 // Find an employee code
 private int find( String aCode) {
     for (int i=0;i<this.size();i++)
         if (this.get(i).getCode().equals(aCode)) return i;
     return -1;
 }
```

18

Access Text Files …: Case study 1- Implementations

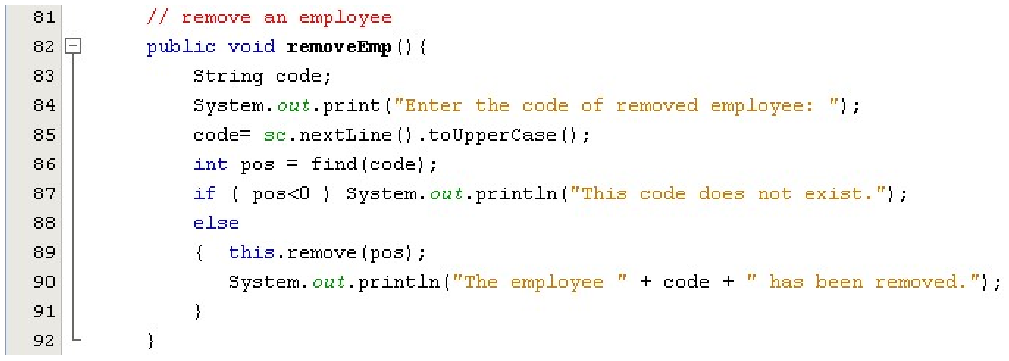


```java
// add new employee
public void addNewEmp(){
   String newCode, newName; int salary;
   int pos;
   boolean valid=true;
   System.out.println("Enter New Employee Details:");
   do {
      System.out.print("   code E000:");
      newCode = sc.nextLine().toUpperCase();
      pos = find(newCode);
      valid = newCode.matches("^E\\d{3}$"); // Pattern: E and 3 digits
      if (pos>=0) System.out.println("   The code is duplicated.");
      if (!valid) System.out.println("   The code: E and 3 digits.");
   }
   while (pos>=0 || (!valid));
   System.out.print("   name:");
   newName = sc.nextLine().toUpperCase();
   System.out.print("   salary:");
   salary = Integer.parseInt(sc.nextLine());
   this.add(new Employee (newCode, newName, salary));
   System.out.println("New Employee has been added.");
}
```

19

19

Access Text Files …: Case study 1- Implementations

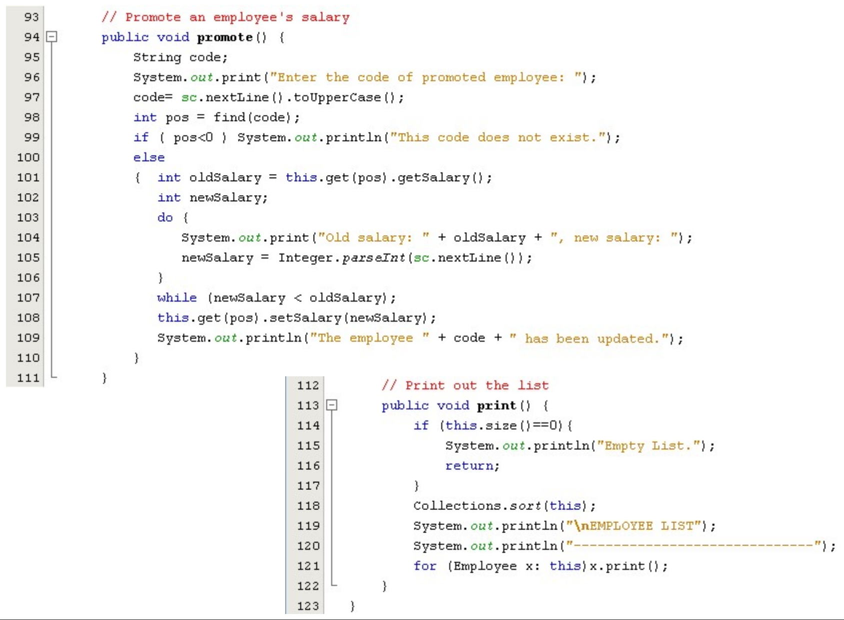


```java
// remove an employee
public void removeEmp(){
    String code;
    System.out.print("Enter the code of removed employee: ");
    code= sc.nextLine().toUpperCase();
    int pos = find(code);
    if ( pos<0 ) System.out.println("This code does not exist.");
    else
    {  this.remove(pos);
       System.out.println("The employee " + code + " has been removed.");
    }
}
```

20

20

Access Text Files …: Case study 1- Implementations



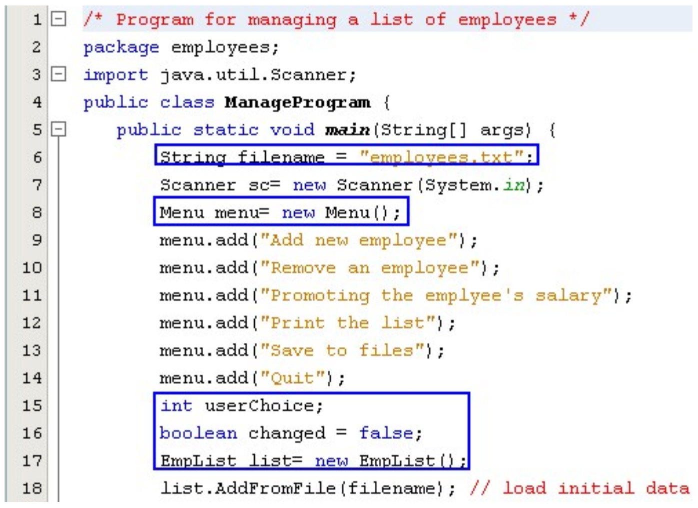
```java
    // Promote an employee's salary
    public void promote() {
        String code;
        System.out.print("Enter the code of promoted employee: ");
        code= sc.nextLine().toUpperCase();
        int pos = find(code);
        if ( pos<0 ) System.out.println("This code does not exist.");
        else
        {  int oldSalary = this.get(pos).getSalary();
           int newSalary;
           do {
              System.out.print("Old salary: " + oldSalary + ", new salary: ");
              newSalary = Integer.parseInt(sc.nextLine());
           }
           while (newSalary < oldSalary);
           this.get(pos).setSalary(newSalary);
           System.out.println("The employee " + code + " has been updated.");
        }
    }
```

```java
    // Print out the list
    public void print() {
        if (this.size()==0){
            System.out.println("Empty List.");
            return;
        }
        Collections.sort(this);
        System.out.println("\nEMPLOYEE LIST");
        System.out.println("------------------------------");
        for (Employee x: this)x.print();
    }
}
```

21

21

Access Text Files …: Case study 1- Implementations

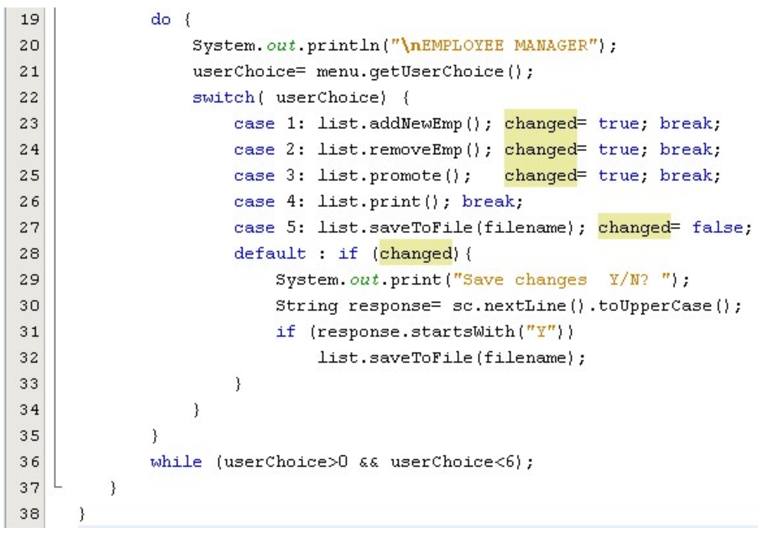


```java
/* Program for managing a list of employees */
package employees;
import java.util.Scanner;
public class ManageProgram {
   public static void main(String[] args) {
       String filename = "employees.txt";
       Scanner sc= new Scanner(System.in);
       Menu menu= new Menu();
       menu.add("Add new employee");
       menu.add("Remove an employee");
       menu.add("Promoting the emplyee's salary");
       menu.add("Print the list");
       menu.add("Save to files");
       menu.add("Quit");
       int userChoice;
       boolean changed = false;
       EmpList list= new EmpList();
       list.AddFromFile(filename); // load initial data
```

22

22

Access Text Files …: Case study 1- Implementations



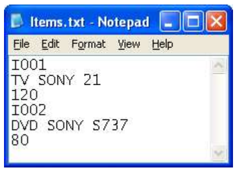
```
        do {
            System.out.println("\nEMPLOYEE MANAGER");
            userChoice= menu.getUserChoice();
            switch( userChoice) {
                case 1: list.addNewEmp(); changed= true; break;
                case 2: list.removeEmp(); changed= true; break;
                case 3: list.promote();   changed= true; break;
                case 4: list.print(); break;
                case 5: list.saveToFile(filename); changed= false;
                default : if (changed){
                    System.out.print("Save changes  Y/N? ");
                    String response= sc.nextLine().toUpperCase();
                    if (response.startsWith("Y"))
                        list.saveToFile(filename);
                }
            }
        }
        while (userChoice>0 && userChoice<6);
    }
}
```

23

Access Text Files …:
Case study 2.- Append File Demo.

Problem

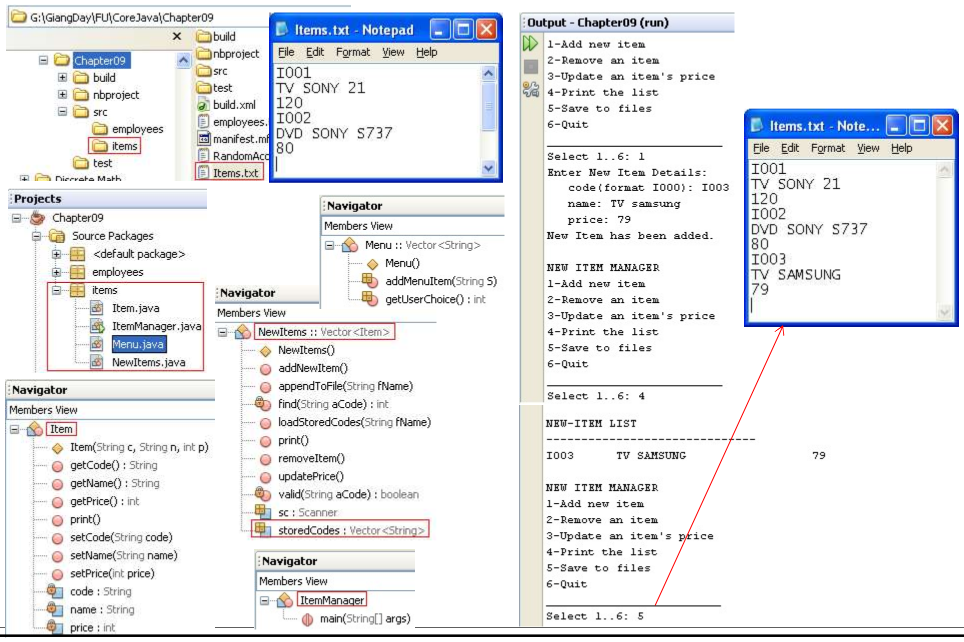
- Each item details include: code, name, price. The item's code can not be duplicated.
- An accountant cannot be allowed to view all stored items (in the text file, named items.txt) but he/she can add some new items to this file.
- Data format in this file (line by line):
  - Line for the code of item
  - Line for the name of item
  - Line for the price of item
- Write a Java program having a simple menu which allows users managing a item list through program's functions:
  - Add new item
  - Update an item
  - Delete an item
  - Save items(Appending items to this file)



```
I001
TV SONY 21
120
I002
DVD SONY S737
80
```

24

Access Text Files …: Case study 2.-Design



```
I001
TV SONY 21
120
I002
DVD SONY S737
80
```

```
1-Add new item
2-Remove an item
3-Update an item's price
4-Print the list
5-Save to files
6-Quit
_______________________
Select 1..6: 1
Enter New Item Details:
   code(format I000): I003
   name: TV samsung
   price: 79
New Item has been added.

NEW ITEM MANAGER
1-Add new item
2-Remove an item
3-Update an item's price
4-Print the list
5-Save to files
6-Quit
_______________________
Select 1..6: 4

NEW-ITEM LIST
------------------------------
I003      TV SAMSUNG                 79

NEW ITEM MANAGER
1-Add new item
2-Remove an item
3-Update an item's price
4-Print the list
5-Save to files
6-Quit
_______________________
Select 1..6: 5
```

```
I001
TV SONY 21
120
I002
DVD SONY S737
80
I003
TV SAMSUNG
79
```

25

Access Text Files …: Case study 2- Implementations

Refer to the case study 1.
DO YOURSELF

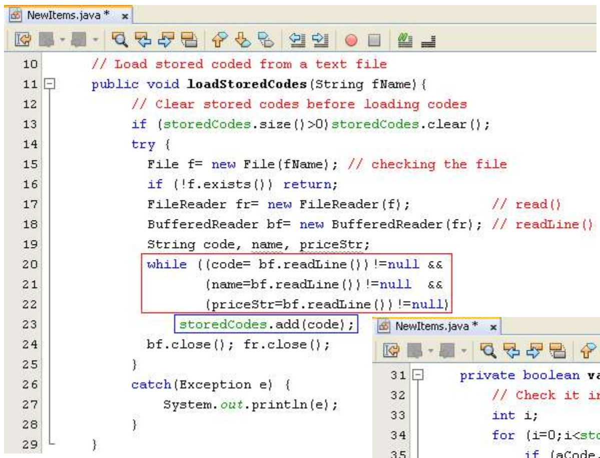
```java
/* Class for simple menu */
package items;
import java.util.Vector;
import java.util.Scanner;
public class Menu extends Vector <String> {
   public Menu() { super(); }
   void addMenuItem(String S) { this.add(S); }
   int getUserChoice () {...}
}
```

```java
/* Class for a product item */
package items;
public class Item {
   private String code;
    private String name;
    private int price;
   // Do yourself
   public Item (String c, String n, int p){...}
   // Print details to the screen
   public void print( ){...}
   //Getters and Setters
    public String getCode() {...}
   public void setCode(String code) {...}
   public String getName() {...}
   public void setName(String name) {...}
   public int getPrice() {...}
   public void setPrice(int price) {...}
}
```

```java
/* Class for new item list */
package items;
import java.util.Scanner;
import java.util.Vector;
import java.io.*;
public class NewItems extends Vector<Item> {
   Scanner sc= new Scanner(System.in); // for input data
   Vector <String> storedCodes = new Vector<String>(); // stored codes in file
   public NewItems() { super(); }
```

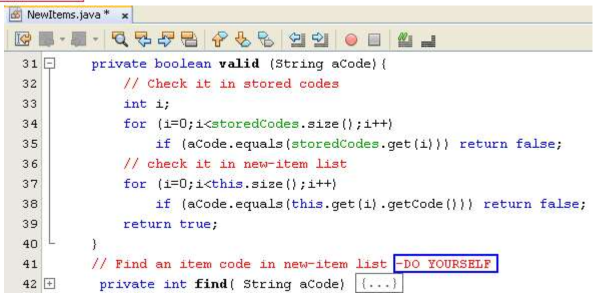
26

Access Text Files …: Case study 4.- Implementations



NewItems.java *

```java
    // Load stored coded from a text file
    public void loadStoredCodes(String fName){
        // Clear stored codes before loading codes
        if (storedCodes.size()>0)storedCodes.clear();
        try {
          File f= new File(fName); // checking the file
          if (!f.exists()) return;
          FileReader fr= new FileReader(f);           // read()
          BufferedReader bf= new BufferedReader(fr); // readLine()
          String code, name, priceStr;
          while ((code= bf.readLine())!=null &&
                 (name=bf.readLine())!=null  &&
                 (priceStr=bf.readLine())!=null)
              storedCodes.add(code);
          bf.close(); fr.close();
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
```

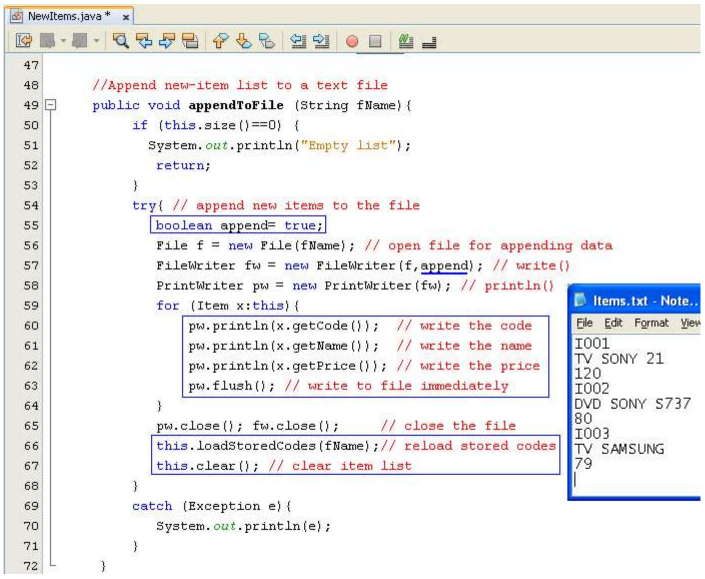


NewItems.java *

```java
    private boolean valid (String aCode){
        // Check it in stored codes
        int i;
        for (i=0;i<storedCodes.size();i++)
            if (aCode.equals(storedCodes.get(i))) return false;
        // check it in new-item list
        for (i=0;i<this.size();i++)
            if (aCode.equals(this.get(i).getCode())) return false;
        return true;
    }
    // Find an item code in new-item list -DO YOURSELF
     private int find( String aCode) {...}
```

27

Access Text Files …: Case study 2- Implementations



NewItems.java *

```java

    //Append new-item list to a text file
    public void appendToFile (String fName){
        if (this.size()==0) {
          System.out.println("Empty list");
           return;
        }
        try{ // append new items to the file
           boolean append= true;
           File f = new File(fName); // open file for appending data
           FileWriter fw = new FileWriter(f,append); // write()
           PrintWriter pw = new PrintWriter(fw); // println()
           for (Item x:this){
               pw.println(x.getCode());  // write the code
               pw.println(x.getName());  // write the name
               pw.println(x.getPrice()); // write the price
               pw.flush(); // write to file immediately
           }
           pw.close(); fw.close();      // close the file
           this.loadStoredCodes(fName);// reload stored codes
           this.clear(); // clear item list
        }
        catch (Exception e){
           System.out.println(e);
        }
     }
```

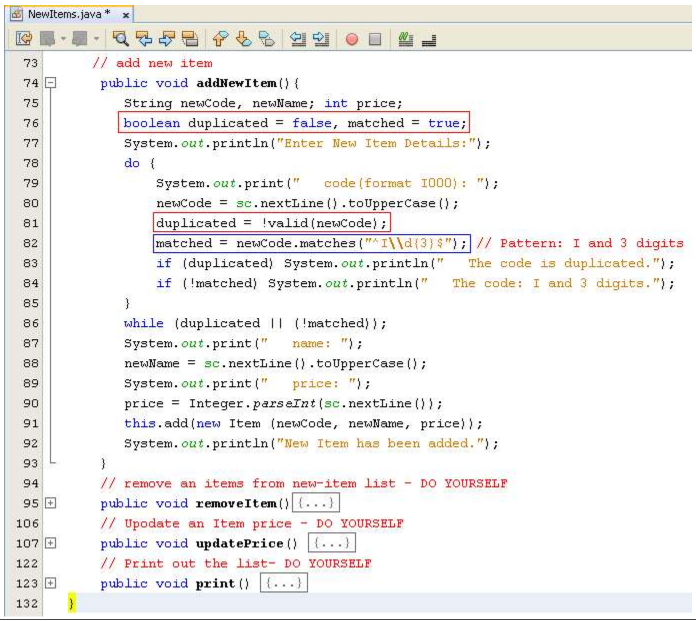
Items.txt - Note..

File Edit Format View

```
I001
TV SONY 21
120
I002
DVD SONY S737
80
I003
TV SAMSUNG
79
```

28

Access Text Files …: Case study 2- Implementations

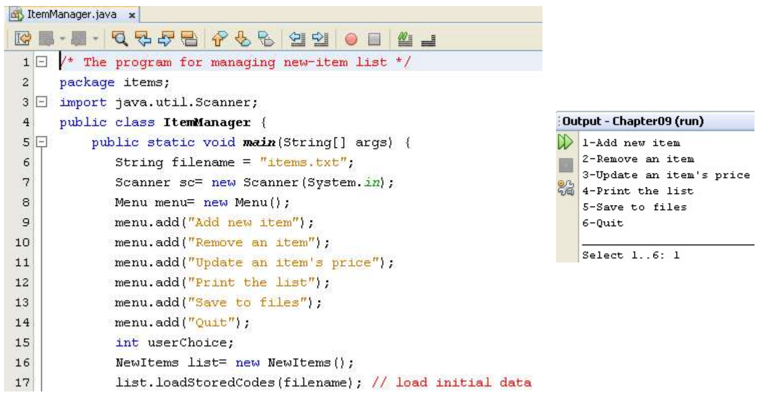


```java
    // add new item
     public void addNewItem(){
        String newCode, newName; int price;
        boolean duplicated = false, matched = true;
        System.out.println("Enter New Item Details:");
        do {
            System.out.print("   code(format I000): ");
            newCode = sc.nextLine().toUpperCase();
            duplicated = !valid(newCode);
            matched = newCode.matches("^I\\d{3}$"); // Pattern: I and 3 digits
            if (duplicated) System.out.println("   The code is duplicated.");
            if (!matched) System.out.println("   The code: I and 3 digits.");
        }
        while (duplicated || (!matched));
        System.out.print("   name: ");
        newName = sc.nextLine().toUpperCase();
        System.out.print("   price: ");
        price = Integer.parseInt(sc.nextLine());
        this.add(new Item (newCode, newName, price));
        System.out.println("New Item has been added.");
     }
     // remove an items from new-item list - DO YOURSELF
     public void removeItem(){...}
     // Upodate an Item price - DO YOURSELF
     public void updatePrice() {...}
     // Print out the list- DO YOURSELF
     public void print() {...}
}
```

29

29

Access Text Files …: Case study 2- Implementations



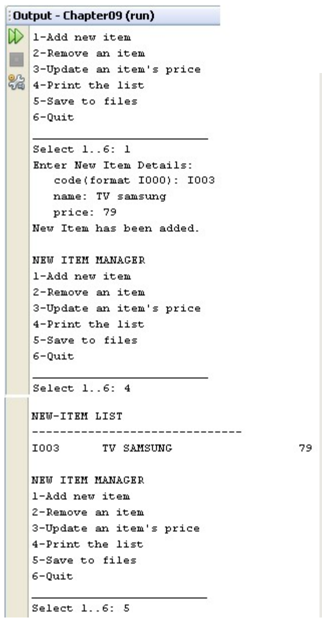
```java
/* The program for managing new-item list */
package items;
import java.util.Scanner;
public class ItemManager {
     public static void main(String[] args) {
         String filename = "items.txt";
         Scanner sc= new Scanner(System.in);
         Menu menu= new Menu();
         menu.add("Add new item");
         menu.add("Remove an item");
         menu.add("Update an item's price");
         menu.add("Print the list");
         menu.add("Save to files");
         menu.add("Quit");
         int userChoice;
         NewItems list= new NewItems();
         list.loadStoredCodes(filename); // load initial data
```

```
1-Add new item
2-Remove an item
3-Update an item's price
4-Print the list
5-Save to files
6-Quit
------------------------------
Select 1..6: 1
```

30

30

Access Text Files …: Case study 2- Implementations

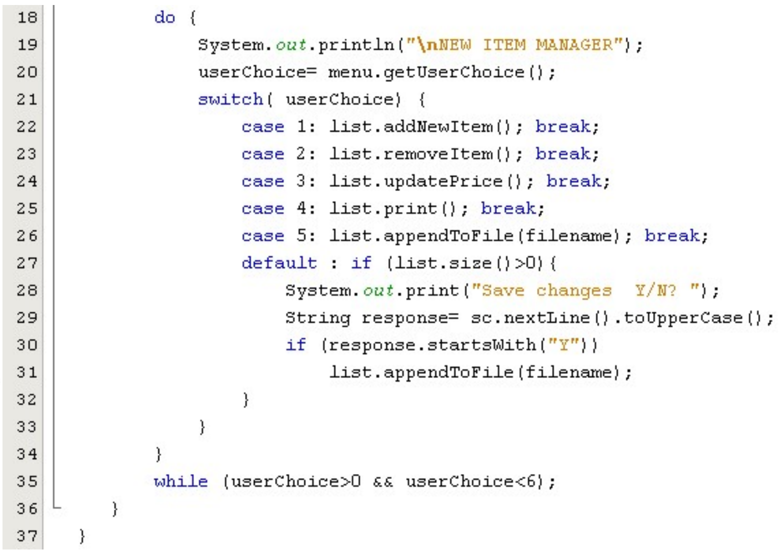


```
1-Add new item
2-Remove an item
3-Update an item's price
4-Print the list
5-Save to files
6-Quit
______________________
Select 1..6: 1
Enter New Item Details:
   code(format I000): I003
   name: TV samsung
   price: 79
New Item has been added.

NEW ITEM MANAGER
1-Add new item
2-Remove an item
3-Update an item's price
4-Print the list
5-Save to files
6-Quit
______________________
Select 1..6: 4

NEW-ITEM LIST
------------------------------
I003      TV SAMSUNG                        79

NEW ITEM MANAGER
1-Add new item
2-Remove an item
3-Update an item's price
4-Print the list
5-Save to files
6-Quit
______________________
Select 1..6: 5
```



```
        do {
            System.out.println("\nNEW ITEM MANAGER");
            userChoice= menu.getUserChoice();
            switch( userChoice) {
                case 1: list.addNewItem(); break;
                case 2: list.removeItem(); break;
                case 3: list.updatePrice(); break;
                case 4: list.print(); break;
                case 5: list.appendToFile(filename); break;
                default : if (list.size()>0){
                    System.out.print("Save changes  Y/N? ");
                    String response= sc.nextLine().toUpperCase();
                    if (response.startsWith("Y"))
                        list.appendToFile(filename);
                }
            }
        }
        while (userChoice>0 && userChoice<6);
    }
}
```

31

31

Access Text Files …: Read UTF-8 File content

UTF8 content is stored in compressed format➔ a character will be stored in 1 to 3 bytes.
Before reading UTF, decompressing is needed.

```
String content="";
FileInputStream f = new FileInputStream(filename);
InputStreamReader isr = new InputStreamReader(f, "UTF8");
int ch;
while ((ch = in.read()) > -1) content+=(char)ch;
```

For read bytes

For read a unicode character

Or "UTF-8"

```
String content="", s;
FileInputStream f = new FileInputStream(filename);
InputStreamReader isr = new InputStreamReader(f, "UTF8");
BufferedReader br = new BufferedReader (isr);
while ( (s= br.readline())!=null) content += s + "\n";
```

For read a unicode character or string.

32

32

5- Access binary files

- Binary streams.
  - Low-level streams: reading/writing data byte-by-byte.
  - High-level stream: reading/writing general-format data (primitives – group of bytes that store typed-values)

33

33

Access binary files…
The java.io.RandomAccessFile class

- It is used to read or modify data in a file that is compatible with the stream, or reader, or writer model
- It supports:
  - Get the file pointer
  - Get the length of the file
  - Seeking to any position within a file
  - Reading & writing single byte/groups of bytes, treated as higher-level data types
  - Close file.

34

34

Access binary files … java.io.RandomAccessFile class…

- Constructors
  RandomAccessFile(String *file*, String *mode*)
  RandomAccessFile(File *file*, String *mode*)
  - Mode "r" to open the file for reading only
  - Mode "rw" to open for both reading and writing
  - Mode "rws" is same as rw and any changes to the file's content or metadata (file attributes) take place **immediately**
  - Mode "rwd" is same as rw, and changes to the file content, but **not** its **metadata**, take place immediately. Its metadata are upadated only when the file is closed.

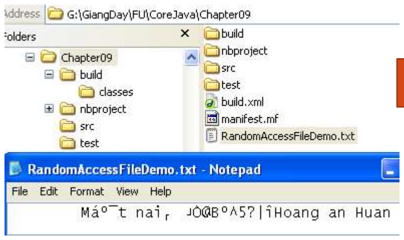
35

35

Access binary files … java.io.RandomAccessFile class…

A demo. for write data to a file then read data from the file

The try…catch statement must be used when accessing file – checked exception

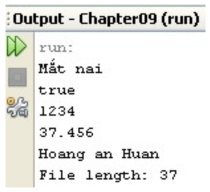


G:\GiangDay\FU\CoreJava\Chapter09

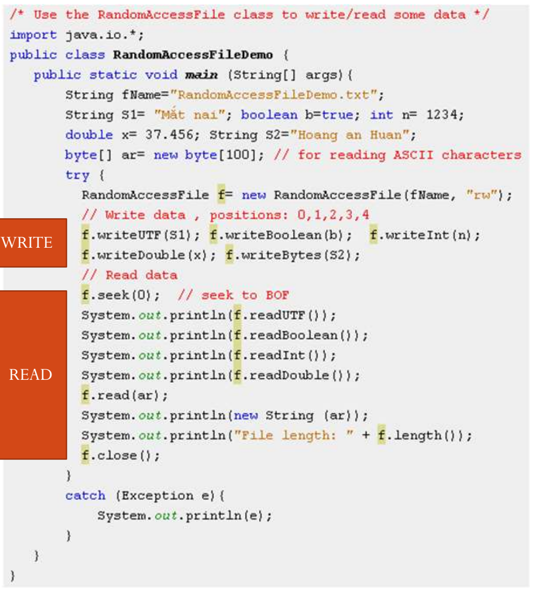
build
nbproject
src
test
build.xml
manifest.mf
RandomAccessFileDemo.txt

RandomAccessFileDemo.txt - Notepad

```
Máº¯t nai, ╜Ò@BºΛ5?|îHoang an Huan
```


Output - Chapter09 (run)

```
run:
Mắt nai
true
1234
37.456
Hoang an Huan
File length: 37
```



```
/* Use the RandomAccessFile class to write/read some data */
import java.io.*;
public class RandomAccessFileDemo {
   public static void main (String[] args){
       String fName="RandomAccessFileDemo.txt";
       String S1= "Mắt nai"; boolean b=true; int n= 1234;
       double x= 37.456; String S2="Hoang an Huan";
       byte[] ar= new byte[100]; // for reading ASCII characters
       try {
         RandomAccessFile f= new RandomAccessFile(fName, "rw");
         // Write data , positions: 0,1,2,3,4
         f.writeUTF(S1); f.writeBoolean(b);  f.writeInt(n);
         f.writeDouble(x); f.writeBytes(S2);
         // Read data
         f.seek(0);  // seek to BOF
         System.out.println(f.readUTF());
         System.out.println(f.readBoolean());
         System.out.println(f.readInt());
         System.out.println(f.readDouble());
         f.read(ar);
         System.out.println(new String (ar));
         System.out.println("File length: " + f.length());
         f.close();
       }
       catch (Exception e){
           System.out.println(e);
       }
   }
}
```

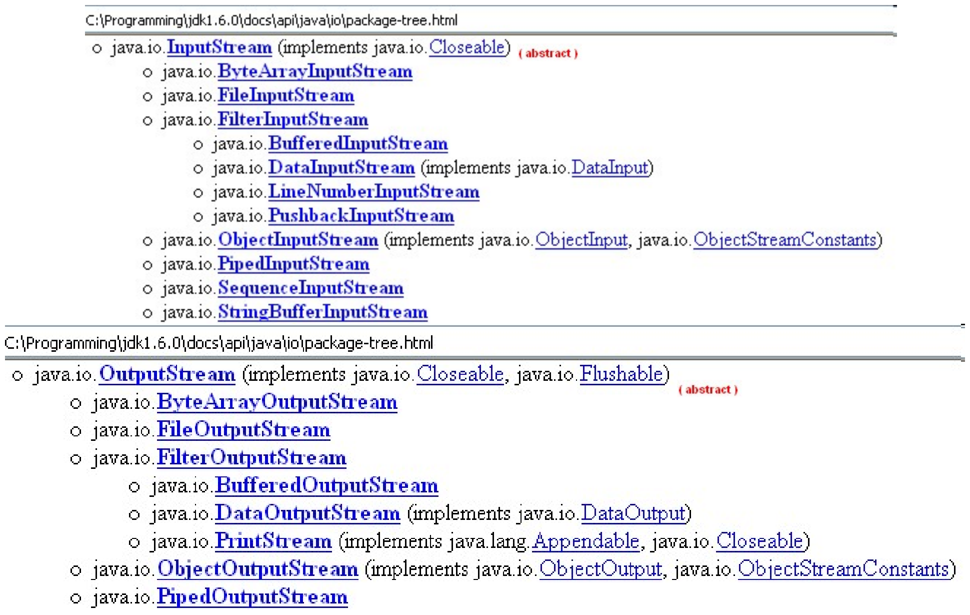
WRITE

READ

36

36

Access binary files…
Binary Streams



C:\Programming\jdk1.6.0\docs\api\java\io\package-tree.html

- java.io.InputStream (implements java.io.Closeable) (abstract)
  - java.io.ByteArrayInputStream
  - java.io.FileInputStream
  - java.io.FilterInputStream
    - java.io.BufferedInputStream
    - java.io.DataInputStream (implements java.io.DataInput)
    - java.io.LineNumberInputStream
    - java.io.PushbackInputStream
  - java.io.ObjectInputStream (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
  - java.io.PipedInputStream
  - java.io.SequenceInputStream
  - java.io.StringBufferInputStream

C:\Programming\jdk1.6.0\docs\api\java\io\package-tree.html

- java.io.OutputStream (implements java.io.Closeable, java.io.Flushable) (abstract)
  - java.io.ByteArrayOutputStream
  - java.io.FileOutputStream
  - java.io.FilterOutputStream
    - java.io.BufferedOutputStream
    - java.io.DataOutputStream (implements java.io.DataOutput)
    - java.io.PrintStream (implements java.lang.Appendable, java.io.Closeable)
  - java.io.ObjectOutputStream (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
  - java.io.PipedOutputStream

37

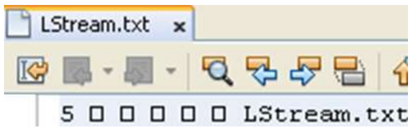
37

Access binary files…
Low-Level Binary Stream Demo.1

```
public class LowLevelStreamDemo {
    /**...*/
    public static void main(String[] args) {
        final char BLANK=32;
        final String fileName="LStream.txt";
        int[] a ={1, 2, 3, 4, 5};
        char n = '5';
        try {
            FileOutputStream os = new FileOutputStream(fileName);
            os.write(n);//begin writing
            os.write(BLANK);
            for(int i=0; i<5; i++){
                os.write(a[i]);
                os.write(BLANK);
            }
            for(int i=0; i<fileName.length(); i++){
                os.write(fileName.charAt(i));
            }
            os.close();
```

These values can not be greater than 127 because only the lower bytes are written to the file.

Write data to file



LStream.txt

```
5 □ □ □ □ □ LStream.txt
```

We can not read these number in the file because of binary file. However, we can see characters.

38

38

Access binary files…
Low-Level Binary Stream Demo.1…

Read data from the file then print them out.

```
    FileInputStream is = new FileInputStream(fileName);
    int count = is.available();
    System.out.println("The size of file is " + count + " bytes");
    System.out.println("The content of file: ");
    //read first char
    byte[] bytes = new byte[1];
    is.read(bytes);
    System.out.print(new String(bytes));
    //read blank
    is.read(bytes);
    System.out.print(new String(bytes));
    //read int number
    for(int i=0; i<5; i++){
        int tmp = is.read();
        is.read(bytes);
        System.out.print(tmp + new String(bytes));
    }
    bytes = new byte[11];
    is.read(bytes);
    System.out.println(new String(bytes));
    is.close();
}catch(IOException e){
    e.printStackTrace();
}
    }
}
```

Read a byte: '5'

Read the blank

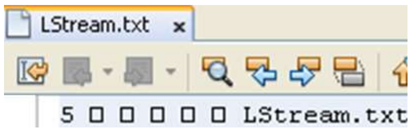
Read the blank

Read a number

Read filename stored at the end of the file

Convert array of characters to string for printing them easier.

```
The size of file is 23 bytes
The content of file:
5 1 2 3 4 5 LStream.txt
```



```
5 □ □ □ □ □ LStream.txt
```

39

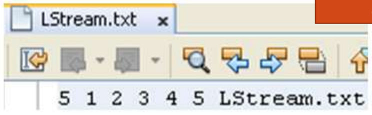
Access binary files…
Low-Level Binary Stream Demo.2

```
public class LowLevelStreamDemo {
    /**...*/
    public static void main(String[] args) {
        final char BLANK=32;
        final String fileName="LStream.txt";
        int[] a ={1, 2, 3, 4, 5};
        char n = '5';
        try {
            FileOutputStream os = new FileOutputStream(fileName);
            os.write(n);//begin writing
            os.write(BLANK);
            for(int i=0; i<5; i++){
                os.write(Character.forDigit(a[i],10));
                os.write(BLANK);
            }
            for(int i=0; i<fileName.length(); i++){
                os.write(fileName.charAt(i));
            }
            os.close();
```

This demo. Is the same as the previous one. But, all small number will be converted to digits then write them to the file

Write data to file

Now, we can see all the file content because they are characters



```
5 1 2 3 4 5 LStream.txt
```

40

Access binary files…
Low-Level Binary Stream Demo.2…

Read data from the file

```
        FileInputStream is = new FileInputStream(fileName);
        int count = is.available();
        System.out.println("The size of file is " + count + " bytes");
        byte[] bytes = new byte[count];
        int readCount = is.read(bytes);
        System.out.println("The content of file: ");
        System.out.println(new String(bytes));
        System.out.println("Number of read bytes: " + readCount);
        is.close();
    }catch(IOException e){
        e.printStackTrace();
    }
}
```

```
The size of file is 23 bytes
The content of file:
5 1 2 3 4 5 LStream.txt
Number of read bytes: 23
```

41

41

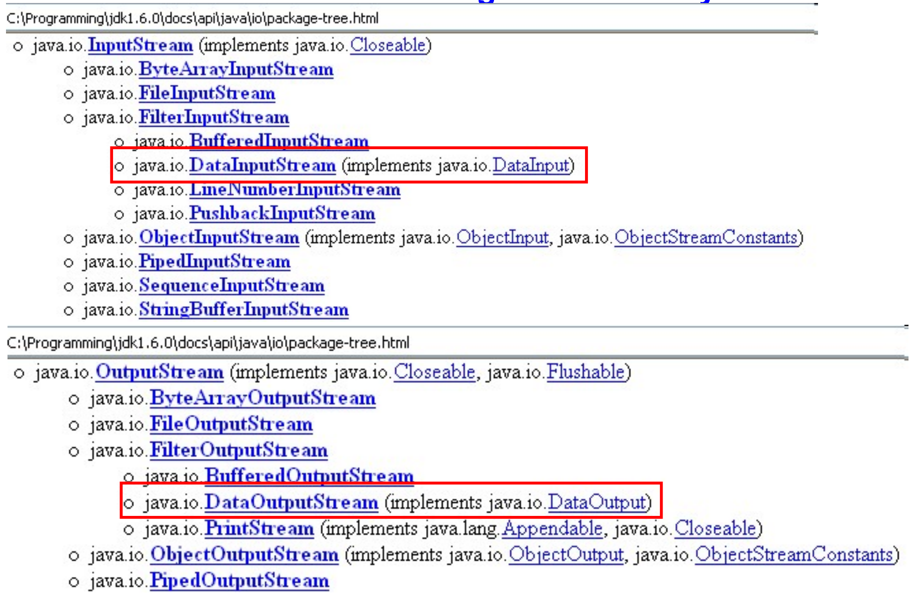
Access binary files
High-Level Binary Stream

- More often than not bytes to be read or written constitute higher-level information (int, String, …)
- The most common of high-level streams extend from the super classes FilterInputStream and FilterOutputStream.
- Do not read/write from input/output devices such as files or sockets; rather, they read/write from other streams
  - DataInputStream/ DataOutputStream
    - Constructor argument: InputStream/ OutputStream
    - Common methods: readXXX, writeXXX
  - BufferedInputStream/ BufferedOutputStream: supports read/write in large blocks
  - ….

42

42

Access binary files…
High-Level Binary Streams



C:\Programming\jdk1.6.0\docs\api\java\io\package-tree.html

- java.io.InputStream (implements java.io.Closeable)
  - java.io.ByteArrayInputStream
  - java.io.FileInputStream
  - java.io.FilterInputStream
    - java.io.BufferedInputStream
    - java.io.DataInputStream (implements java.io.DataInput)
    - java.io.LineNumberInputStream
    - java.io.PushbackInputStream
  - java.io.ObjectInputStream (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
  - java.io.PipedInputStream
  - java.io.SequenceInputStream
  - java.io.StringBufferInputStream

C:\Programming\jdk1.6.0\docs\api\java\io\package-tree.html

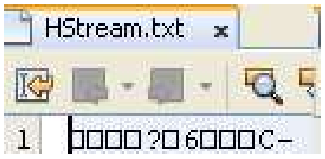
- java.io.OutputStream (implements java.io.Closeable, java.io.Flushable)
  - java.io.ByteArrayOutputStream
  - java.io.FileOutputStream
  - java.io.FilterOutputStream
    - java.io.BufferedOutputStream
    - java.io.DataOutputStream (implements java.io.DataOutput)
    - java.io.PrintStream (implements java.lang.Appendable, java.io.Closeable)
  - java.io.ObjectOutputStream (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
  - java.io.PipedOutputStream

43

43

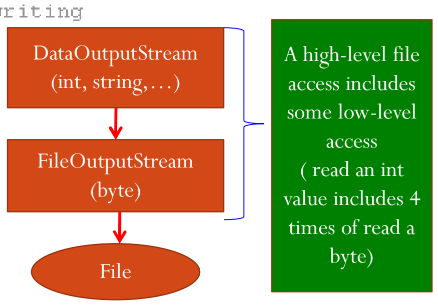
Access binary files…
High-Level Binary Stream Demo.

```
public class HighLevelStreamDemo {
    /**...*/
    public static void main(String[] args) {
        final char BLANK=32;
        final String fileName="HStream.txt";
        int[] a ={1, 2, 3, 4, 5};
        char n = '5';
        try {
            FileOutputStream os = new FileOutputStream(fileName);
            DataOutputStream ds = new DataOutputStream(os);
            ds.writeChar(n);//begin writing
            ds.writeChar(BLANK);
            for(int i=0; i<5; i++){
                ds.writeInt(a[i]);
                ds.writeChar(BLANK);
            }
            ds.writeUTF(fileName);
            ds.close();
            os.close();
```



HStream.txt

```
1 |□□□□ ?□ 6□□□C−
```



DataOutputStream (int, string,…)
FileOutputStream (byte)
File
A high-level file access includes some low-level access (read an int value includes 4 times of read a byte)

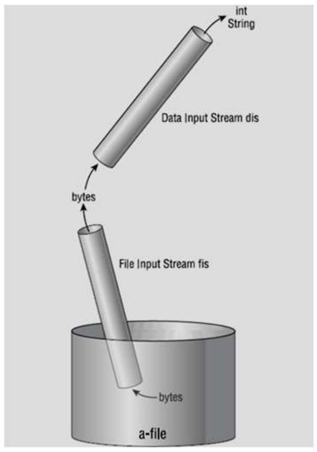
44

44

Access binary files…
High-Level Binary Stream Demo. …

```
            FileInputStream is = new FileInputStream(fileName);
            DataInputStream dis = new DataInputStream(is);
            int count = dis.available();
            System.out.println("The size of file is " + count + " bytes");
            System.out.println("The content of file: ");
            System.out.print(dis.readChar());
            System.out.print(dis.readChar());
            for(int i=0; i<5; i++){
                System.out.print(dis.readInt());
                System.out.print(dis.readChar());
            }
            System.out.println(dis.readUTF());
            dis.close();
            is.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

```
The size of file is 47 bytes
The content of file:
5 1 2 3 4 5 HStream.txt
```



45

45

6- Access Object Files

- 2 Object streams :Object Input stream, Object Output stream

- java.lang.Object
  - java.io.InputStream (implements java.io.Closeable)
    - java.io.ByteArrayInputStream
    - java.io.FileInputStream
    - java.io.FilterInputStream
    - java.io.ObjectInputStream (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
  - java.io.OutputStream (implements java.io.Closeable, java.io.Flushable)
    - java.io.ByteArrayOutputStream
    - java.io.FileOutputStream
    - java.io.FilterOutputStream
    - java.io.ObjectOutputStream (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)

Serialization is a task which will concate all data of an object to a byte stream then it can be written to a datasource. **Static and transient data can not be serialized.**
De-serialization is a task which will read a byte stream from a datasourse , split the stream to fields then assign them to data fields of an object appropriately.
Transient fields are omitted when an object is serialized.

46

46

Serialization

- The process of writing an object is called *serialization.*
- Use java.io.ObjectOutputStream to serialize an object.
- It is only an object's data that is serialized, not its class definition.
- When an object output stream serializes an object that contains references to other object, every referenced object is serialized along with the original object.
- Not all data is written.
  - static fields are not
  - transient fields are also not serialized

47

47

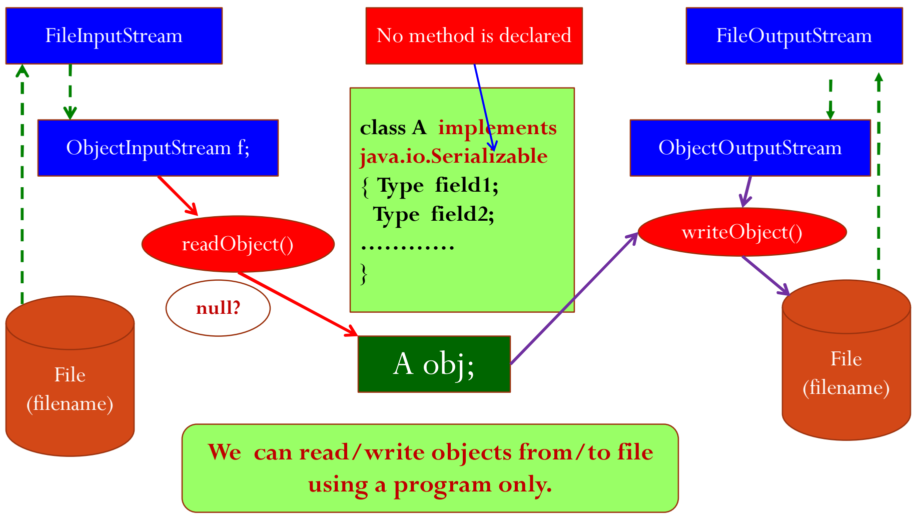
De-serialization

- De-serialization is to convert a serialized representation into a replicate of the original object.
- Use java.io.ObjectInputStream to deserialize an object.
- When an object is serialized, it will probably be deserialized by a different JVM.
- Any JVM that tries to deserialize an object must have access to that object's class definition.

48

48

Access Object Files…: How to?



49

49

Access Object Files…: Case study 3 - Object Streams Demo.

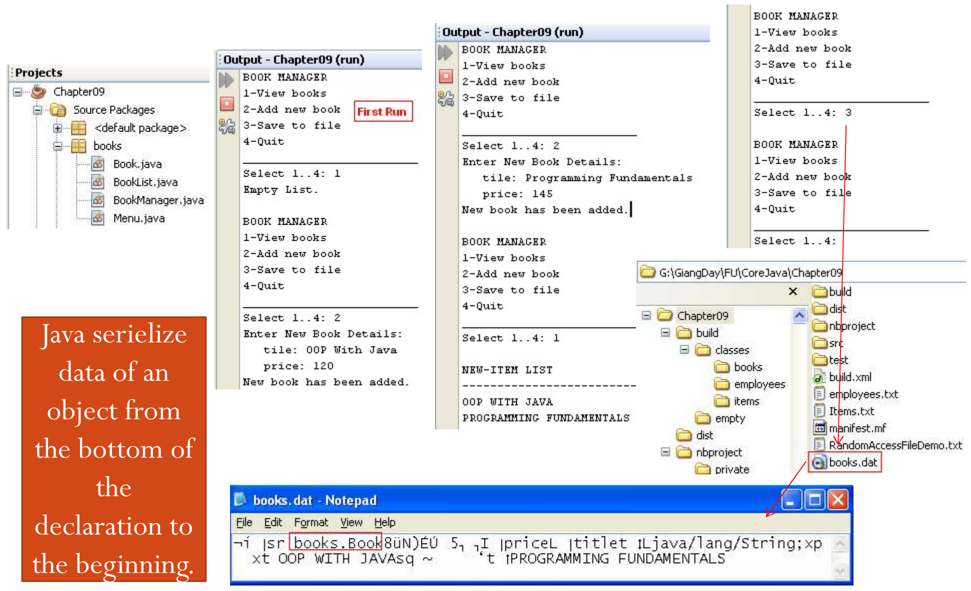
Problem

- Book <title, price>
- Write a Java program that allows user:
  - View books in the file books.dat
  - Append a book to the file
- Read/ Write books as binary objects from/to the file.

50

50

Access Object Files…: Case Study 3 - Design



```
BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit

Select 1..4: 1
Empty List.

BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit

Select 1..4: 2
Enter New Book Details:
    tile: OOP With Java
    price: 120
New book has been added.
```

```
BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit

Select 1..4: 2
Enter New Book Details:
    tile: Programming Fundamentals
    price: 145
New book has been added.

BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit

Select 1..4: 1

NEW-ITEM LIST
---------------------------
OOP WITH JAVA
PROGRAMMING FUNDAMENTALS
```

```
BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit

Select 1..4: 3

BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit

Select 1..4:
```

```
¬í ¦sr books.Book8üN)ÉÚ 5┐ ┐I ¦priceL ¦titlet ↨Ljava/lang/String;xp
  xt OOP WITH JAVAsq ~     ‛t ¦PROGRAMMING FUNDAMENTALS
```

Java serielize data of an object from the bottom of the declaration to the beginning.

51

51

Access Object Files…: Case Study 3- Implementations

Refer to the case study 1, 2.
DOYOURSELF

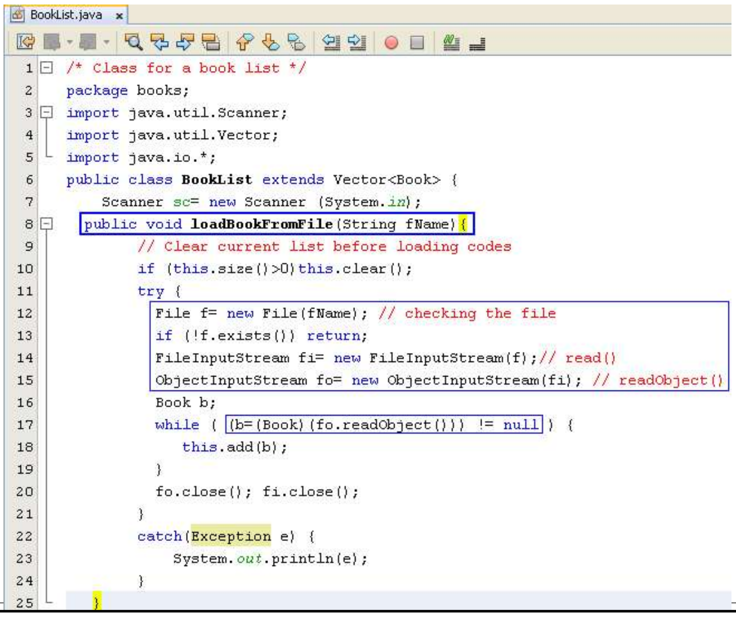
```
/* Class for a simple menu */
package books;
import java.util.Vector;
import java.util.Scanner;
public class Menu extends Vector <String> {
    public Menu() { super(); }
    void addMenuItem(String S) { this.add(S); }
    int getUserChoice () {...}
}
```

```
/* Class for a book */
package books;
import java.io.Serializable;
public class Book implements Serializable {
  private String title;
     private int price;
   public Book(String title, int price) {...}
    // Print details to the screen
    public void print( ) {...}
    // Getters and Setters
     public String getTitle() {...}
     public void setTitle(String title) {...}
     public int getPrice() {...}
     public void setPrice(int price) {...}
}
```

52

52

Access Object Files…: Case Study 3– Implementations…



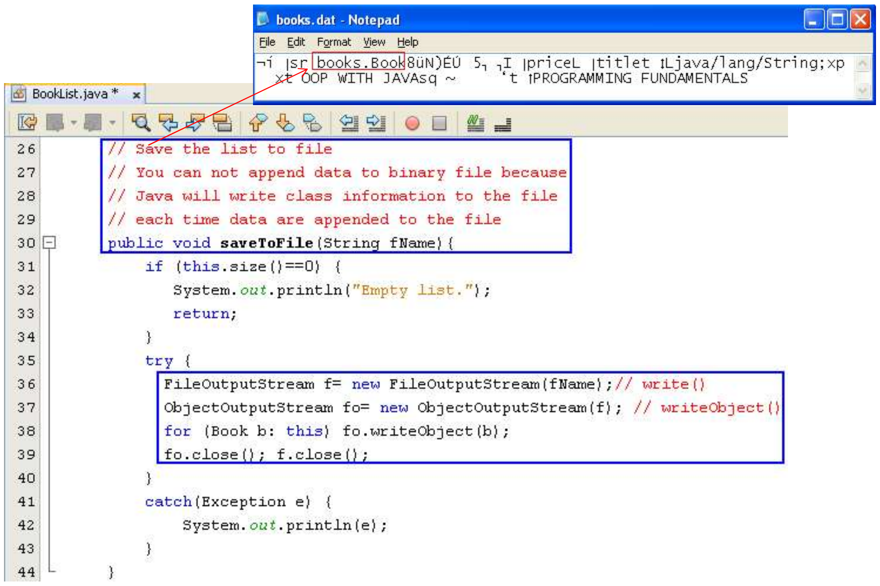
BookList.java

```java
/* Class for a book list */
package books;
import java.util.Scanner;
import java.util.Vector;
import java.io.*;
public class BookList extends Vector<Book> {
    Scanner sc= new Scanner (System.in);
  public void loadBookFromFile(String fName) {
        // Clear current list before loading codes
        if (this.size()>0)this.clear();
        try {
          File f= new File(fName); // checking the file
          if (!f.exists()) return;
          FileInputStream fi= new FileInputStream(f);// read()
          ObjectInputStream fo= new ObjectInputStream(fi); // readObject()
          Book b;
          while ( (b=(Book)(fo.readObject()))  != null) ) {
             this.add(b);
          }
          fo.close(); fi.close();
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
```

53

53

Access Object Files…: Case Study 3– Implementations…



books.dat - Notepad

File Edit Format View Help

```
¬í ╷sr books.Book8üN)ÉÚ 5╕ ╕I ╷priceL ╷titlet ↕Ljava/lang/String;xp
  xt OOP WITH JAVAsq ~     ʻt ↕PROGRAMMING FUNDAMENTALS
```

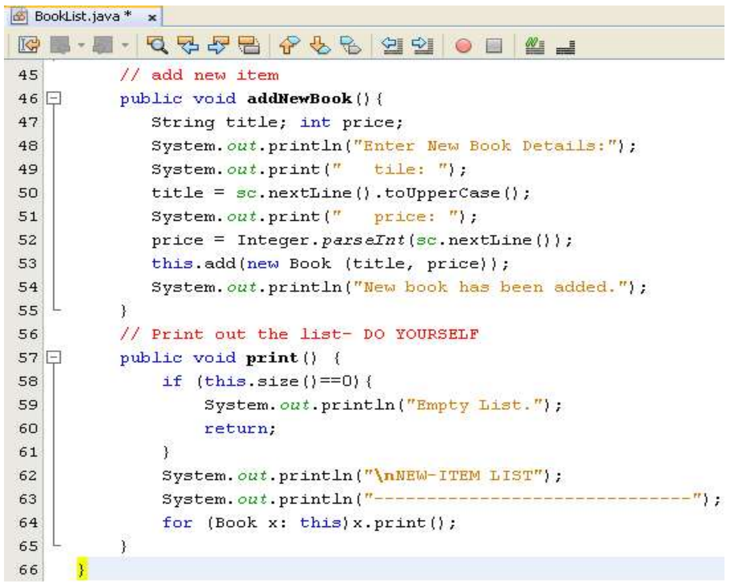
BookList.java *

```java
  // Save the list to file
  // You can not append data to binary file because
  // Java will write class information to the file
  // each time data are appended to the file
  public void saveToFile(String fName){
        if (this.size()==0) {
           System.out.println("Empty list.");
           return;
        }
        try {
          FileOutputStream f= new FileOutputStream(fName);// write()
          ObjectOutputStream fo= new ObjectOutputStream(f); // writeObject()
          for (Book b: this) fo.writeObject(b);
          fo.close(); f.close();
        }
        catch(Exception e) {
           System.out.println(e);
        }
    }
```

54

54

Access Object Files…: Case Study 3– Implementations…

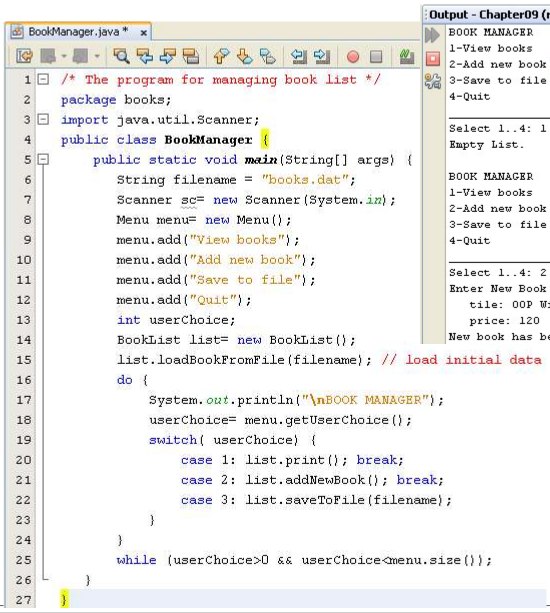

BookList.java *

```java
        // add new item
        public void addNewBook(){
            String title; int price;
            System.out.println("Enter New Book Details:");
            System.out.print("    tile: ");
            title = sc.nextLine().toUpperCase();
            System.out.print("    price: ");
            price = Integer.parseInt(sc.nextLine());
            this.add(new Book (title, price));
            System.out.println("New book has been added.");
        }
        // Print out the list- DO YOURSELF
        public void print() {
             if (this.size()==0){
                 System.out.println("Empty List.");
                 return;
             }
             System.out.println("\nNEW-ITEM LIST");
             System.out.println("------------------------------");
             for (Book x: this)x.print();
        }
    }
```

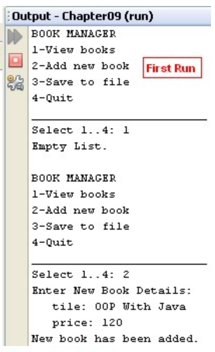
55

55

Access Object Files…: Case Study 5 – Implementations…


BookManager.java *

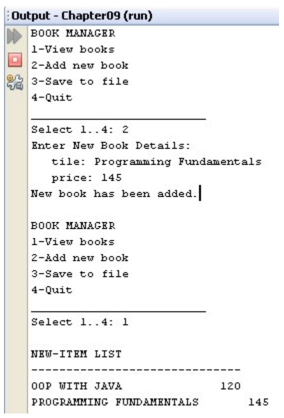
```java
/* The program for managing book list */
package books;
import java.util.Scanner;
public class BookManager {
    public static void main(String[] args) {
        String filename = "books.dat";
        Scanner sc= new Scanner(System.in);
        Menu menu= new Menu();
        menu.add("View books");
        menu.add("Add new book");
        menu.add("Save to file");
        menu.add("Quit");
        int userChoice;
        BookList list= new BookList();
        list.loadBookFromFile(filename); // load initial data
        do {
            System.out.println("\nBOOK MANAGER");
            userChoice= menu.getUserChoice();
            switch( userChoice) {
                case 1: list.print(); break;
                case 2: list.addNewBook(); break;
                case 3: list.saveToFile(filename);
            }
        }
        while (userChoice>0 && userChoice<menu.size());
    }
}
```


Output - Chapter09 (run)

First Run

```
BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit
______________________________
Select 1..4: 1
Empty List.

BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit
______________________________
Select 1..4: 2
Enter New Book Details:
   tile: OOP With Java
   price: 120
New book has been added.
```


Output - Chapter09 (run)

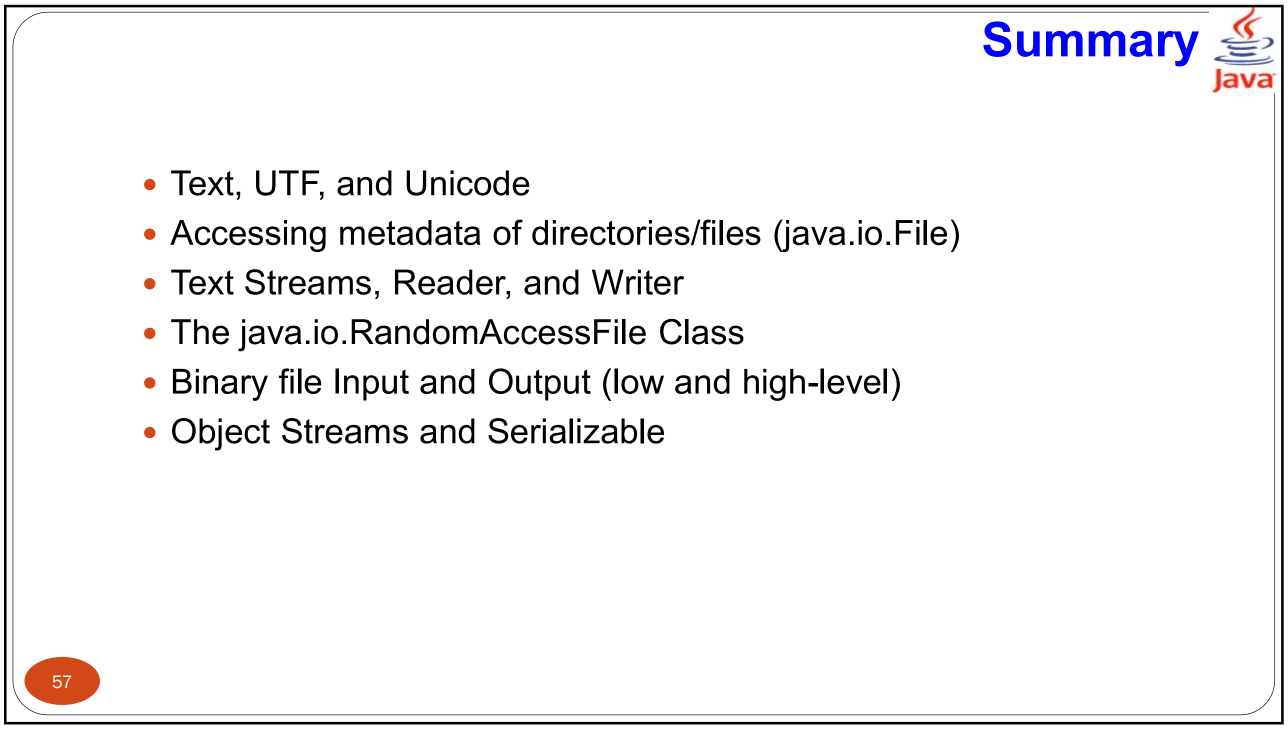
```
BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit
______________________________
Select 1..4: 2
Enter New Book Details:
   tile: Programming Fundamentals
   price: 145
New book has been added.

BOOK MANAGER
1-View books
2-Add new book
3-Save to file
4-Quit
______________________________
Select 1..4: 1

NEW-ITEM LIST
------------------------------
OOP WITH JAVA                  120
PROGRAMMING FUNDAMENTALS          145
```

56

56



Summary

- Text, UTF, and Unicode
- Accessing metadata of directories/files (java.io.File)
- Text Streams, Reader, and Writer
- The java.io.RandomAccessFile Class
- Binary file Input and Output (low and high-level)
- Object Streams and Serializable

57

57