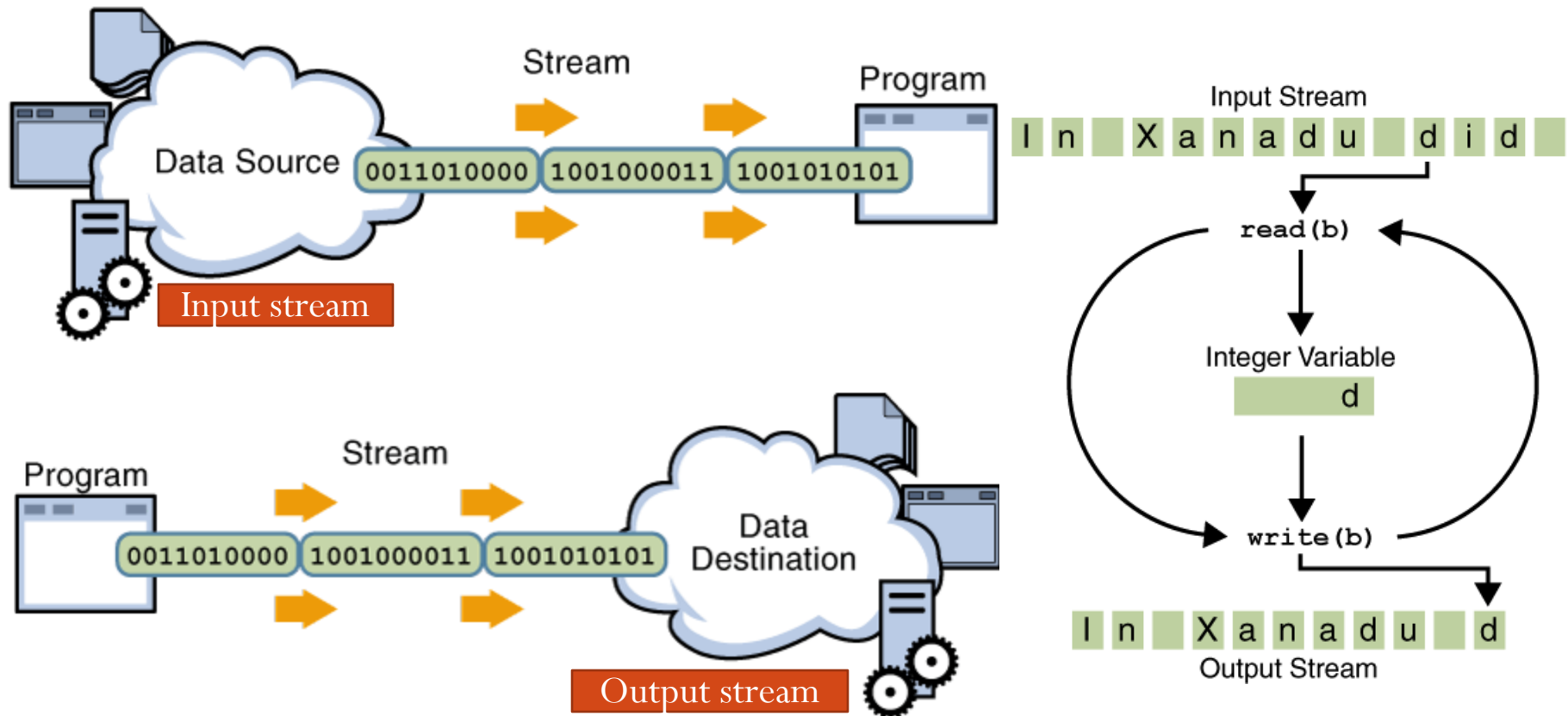


File I/O

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.



Why should you study this chapter?

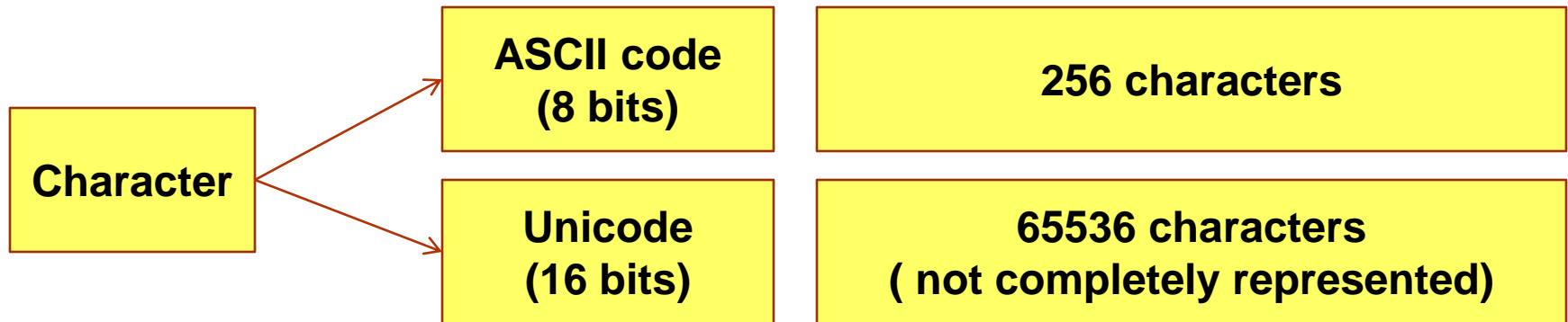
- Files can not be missing in large applications.
- Do you want to access a file in Java?
- How can we read/write data from/to a file?

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

- Text, UTF, and Unicode
- Introduction to the java.io package
- Accessing directories and files
- Accessing binary files
- Accessing text files.
- Read/write objects from/to files?

1- Text, UTF, and Unicode



Unicode character: a character is coded using 16/32 bits

UTF: Universal Character Set – UCS- Transformation Format

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.

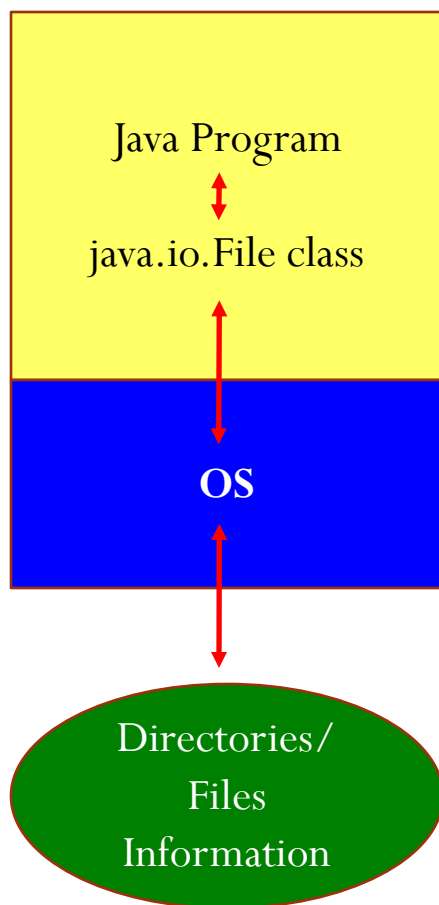
2- Introduction to the java.io Package

- 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.

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.

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
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

Accessing directories and files...

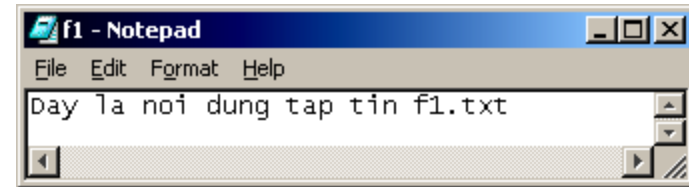
The java.io.File Class...

Get File Attributes Demo.

```

1 //FileDemo.java
2 import java.io.*;
3 import java.util.Date;
4 class FileDemo
5 { public static void main (String args[]) throws IOException
6 { File f = new File("f1.txt");
7   System.out.println("Ten file la:" + f.getName());
8   System.out.println("Ten file tuyet doi la:" + f.getAbsolutePath());
9   System.out.println("Duong dan tuyet doi la:" + f.getAbsolutePath());
10  System.out.println("Path chuan la:" + f.getCanonicalPath());
11  System.out.println("Ngay cap nhat cuoi cung la:" + new Date(f.lastModified()));
12  System.out.println("Thuoc tinh Hidden:" + f.isHidden());
13  System.out.println("Thuoc tinh can-read:" + f.canRead());
14  System.out.println("Thuoc tinh can-write:" + f.canWrite());
15  System.out.println("Kich thuoc:" + f.length() + " bytes");
16 }
17 }

```



```

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

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.txt
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 lệch 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

Accessing directories and files...

The java.io.File Class...

Accessing a folder Demo.

```

1 //FileDemo2.java
2 import java.io.*;
3 import java.util.Date;
4 class FileDemo2
5 { public static void main (String args[]) throws IOException
6   { File f = new File("../BtCh10-IO");
7     String S = f.isDirectory() ? "Thu muc" : "Tap tin";
8     System.out.println("../BtCh10-IO la:" + S);
9     String L[] = f.list();
10    System.out.println("Noi dung thu muc:");
11    for (int i=0;i<L.length;++i)
12    { File f2 = new File (f.L[i]);
13      System.out.println(L[i] + " " + (f2.isFile()? "Tap tin" : "Thu muc"));
14    }
15  }
16 }

```

C:\PROGRA~1\XINXS~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
DataInputStreamDemo.class Tap tin
DataInputStreamDemo.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
FileInputStreamDemo.class Tap tin
FileInputStreamDemo.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

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).

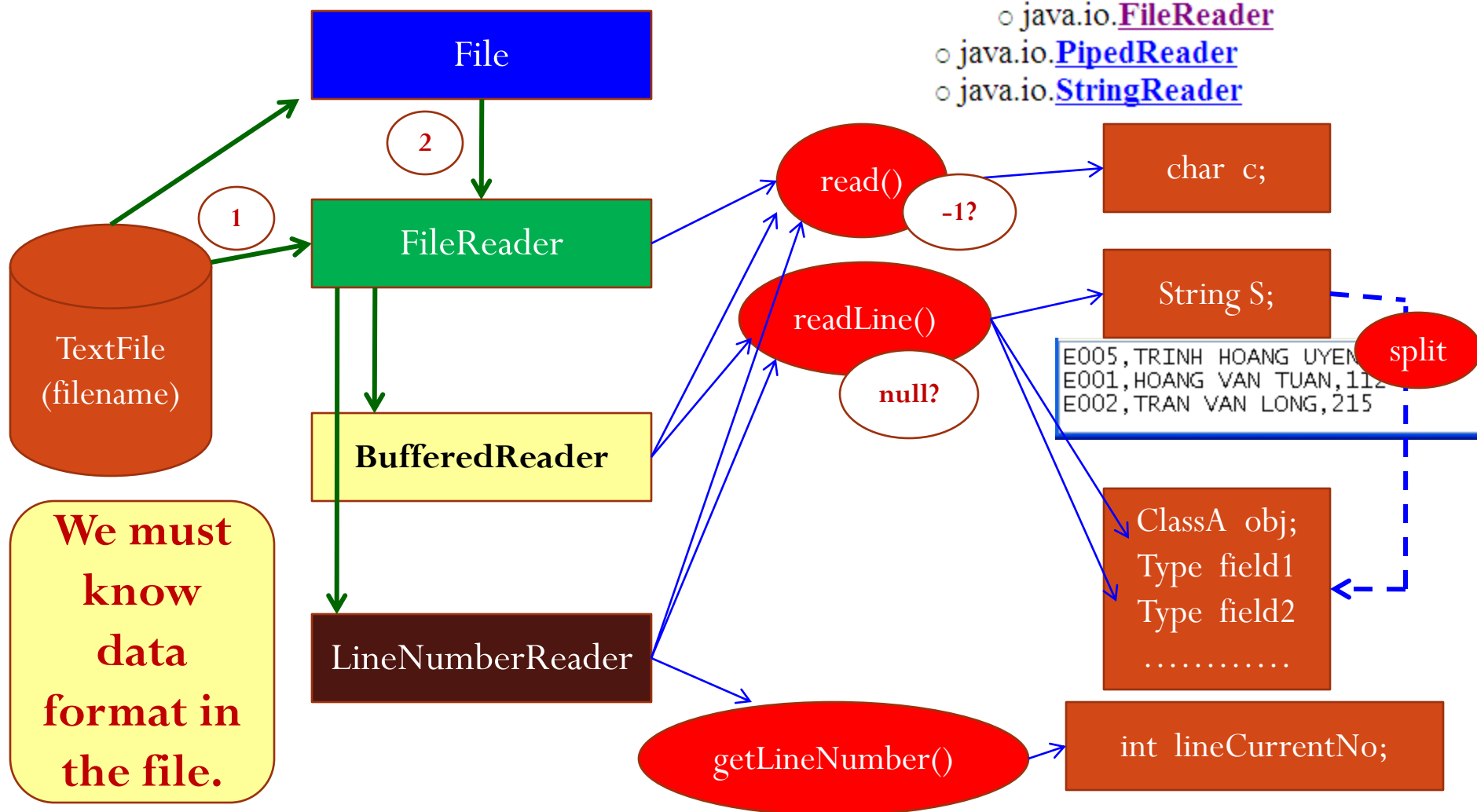
Access Text Files ...

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](#)

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

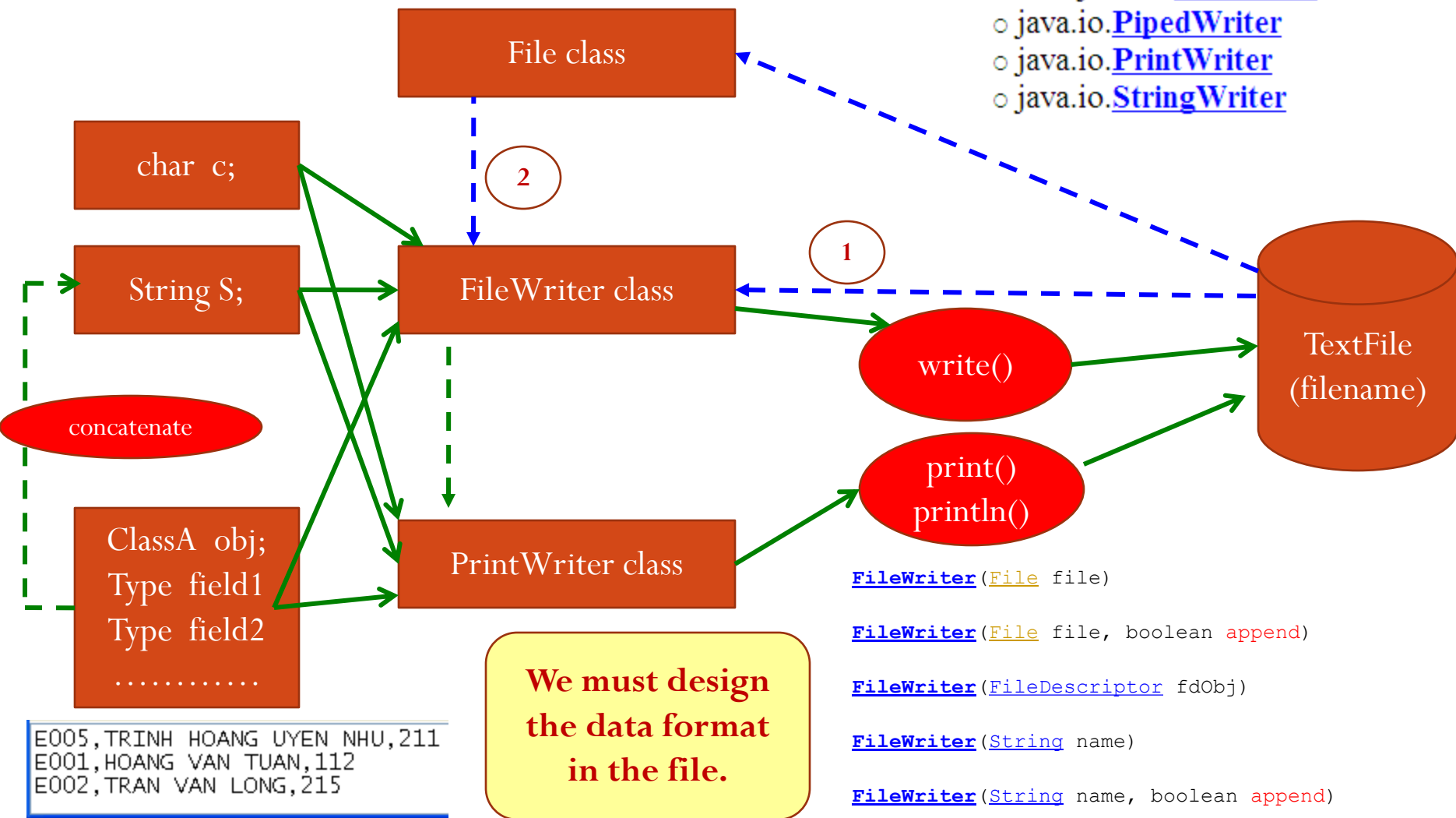
○ java.io.FileReader

○ java.io.PipedReader

○ java.io.StringReader

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](#)



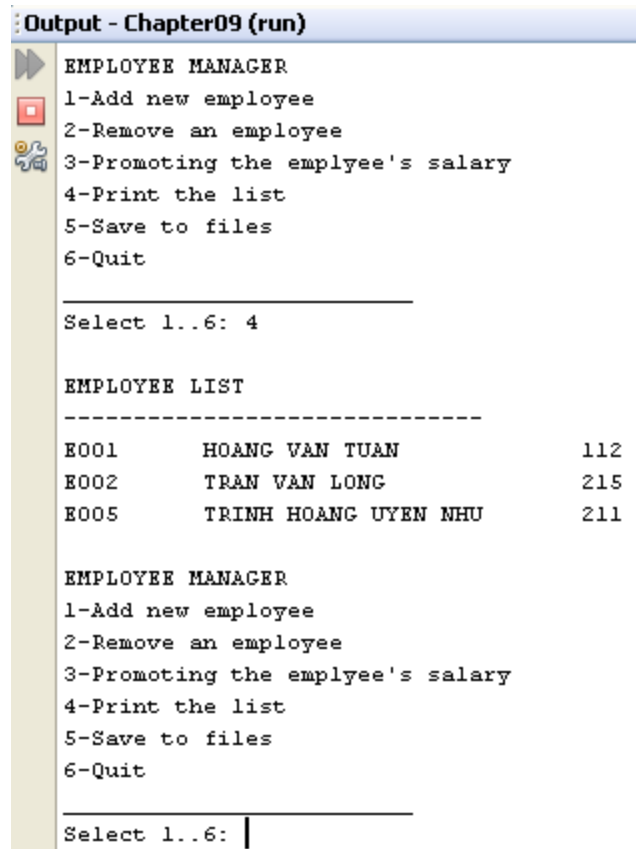
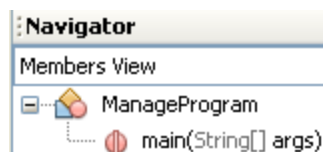
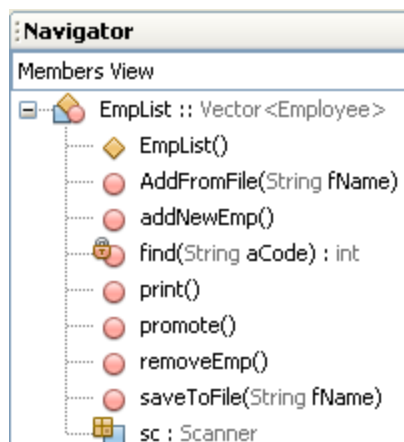
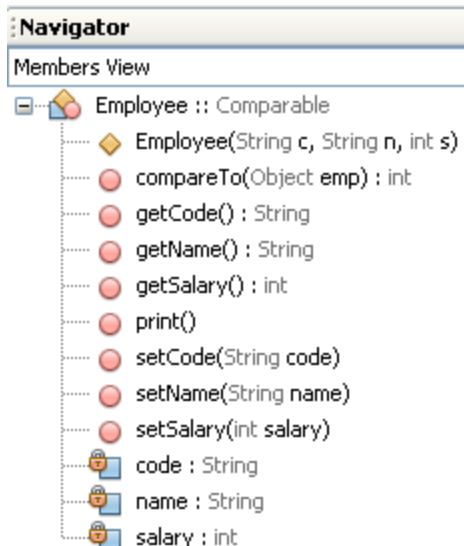
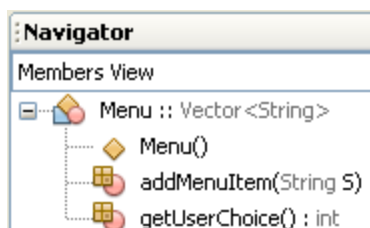
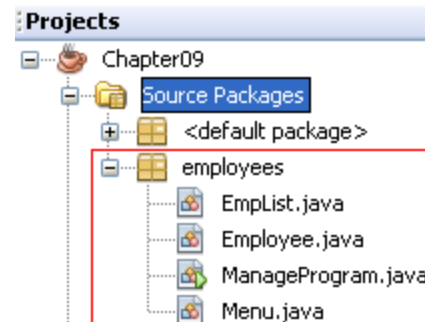
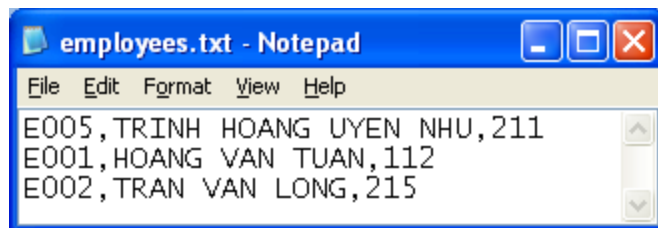
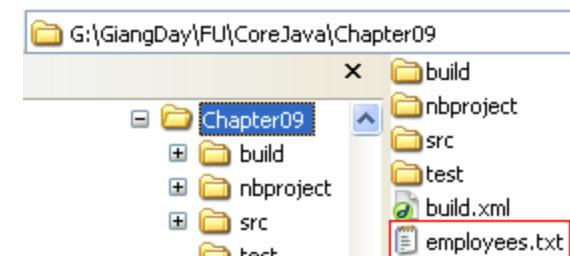
Access Text Files ...

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

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



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

```
/* 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 () { ... }
}
```

```
/* 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());
    }
}
```

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

```

11 // Add employees from a text file
12 public void AddFromFile(String fName) {
13     try {
14         File f= new File(fName); // checking the file
15         if (!f.exists()) return;
16         FileReader fr= new FileReader(f); // read()
17         BufferedReader bf= new BufferedReader(fr); // readLine()
18         String details ; // E001,Hoang Van Tuan,156
19         while ((details= bf.readLine()) !=null)
20         { // Splitting details into elements
21             StringTokenizer stk= new StringTokenizer(details,",");
22             String code= stk.nextToken().toUpperCase();
23             String name= stk.nextToken().toUpperCase();
24             int salary = Integer.parseInt(stk.nextToken());
25             // Create an employee
26             Employee emp= new Employee(code, name, salary);
27             this.add(emp); // adding this employee to the list
28         }
29         bf.close(); fr.close();
30     }
31     catch(Exception e) {
32         System.out.println(e);
33     }
34 }

```

```

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

```

TRƯỜNG ĐẠI HỌC FPT Access Text Files ...: Case study 1- Implementations

```

35 public void saveToFile (String fName) {
36     if (this.size()==0) {
37         System.out.println("Empty list");
38         return;
39     }
40     try{
41         File f = new File(fName);
42         FileWriter fw = new FileWriter(f); // write()
43         PrintWriter pw = new PrintWriter(fw); // println()
44         for (Employee x:this) {
45             pw.println(x.getCode() + "," + x.getName() + "," + x.getSalary());
46         }
47         pw.close(); fw.close();
48     }
49     catch (Exception e) {
50         System.out.println(e);
51     }
52 }

53 // Find an employee code
54 private int find( String aCode) {
55     for (int i=0;i<this.size();i++)
56         if (this.get(i).getCode().equals(aCode)) return i;
57     return -1;
58 }

```

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

```

59      // add new employee
60      public void addNewEmp() {
61          String newCode, newName; int salary;
62          int pos;
63          boolean valid=true;
64          System.out.println("Enter New Employee Details:");
65          do {
66              System.out.print("    code E000:");
67              newCode = sc.nextLine().toUpperCase();
68              pos = find(newCode);
69              valid = newCode.matches("^E\\d{3}$"); // Pattern: E and 3 digits
70              if (pos>=0) System.out.println("    The code is duplicated.");
71              if (!valid) System.out.println("    The code: E and 3 digits.");
72          }
73          while (pos>=0 || (!valid));
74          System.out.print("    name:");
75          newName = sc.nextLine().toUpperCase();
76          System.out.print("    salary:");
77          salary = Integer.parseInt(sc.nextLine());
78          this.add(new Employee (newCode, newName, salary));
79          System.out.println("New Employee has been added.");
80      }

```

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

```

81      // remove an employee
82      public void removeEmp() {
83          String code;
84          System.out.print("Enter the code of removed employee: ");
85          code= sc.nextLine().toUpperCase();
86          int pos = find(code);
87          if ( pos<0 ) System.out.println("This code does not exist.");
88          else
89          {   this.remove(pos);
90              System.out.println("The employee " + code + " has been removed.");
91          }
92      }
  
```

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

```

93 // Promote an employee's salary
94 public void promote() {
95     String code;
96     System.out.print("Enter the code of promoted employee: ");
97     code= sc.nextLine().toUpperCase();
98     int pos = find(code);
99     if ( pos<0 ) System.out.println("This code does not exist.");
100    else
101    { int oldSalary = this.get(pos).getSalary();
102      int newSalary;
103      do {
104          System.out.print("Old salary: " + oldSalary + ", new salary: ");
105          newSalary = Integer.parseInt(sc.nextLine());
106      }
107      while (newSalary < oldSalary);
108      this.get(pos).setSalary(newSalary);
109      System.out.println("The employee " + code + " has been updated.");
110    }
111 }
  
```

```

112 // Print out the list
113 public void print() {
114     if (this.size()==0) {
115         System.out.println("Empty List.");
116         return;
117     }
118     Collections.sort(this);
119     System.out.println("\nEMPLOYEE LIST");
120     System.out.println("-----");
121     for (Employee x: this)x.print();
122 }
123 }
  
```

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

```

1  /* Program for managing a list of employees */
2  package employees;
3  import java.util.Scanner;
4  public class ManageProgram {
5      public static void main(String[] args) {
6          String filename = "employees.txt";
7          Scanner sc= new Scanner(System.in);
8          Menu menu= new Menu();
9          menu.add("Add new employee");
10         menu.add("Remove an employee");
11         menu.add("Promoting the employee's salary");
12         menu.add("Print the list");
13         menu.add("Save to files");
14         menu.add("Quit");
15         int userChoice;
16         boolean changed = false;
17         EmpList list= new EmpList();
18         list.AddFromFile(filename); // load initial data
  
```


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

```

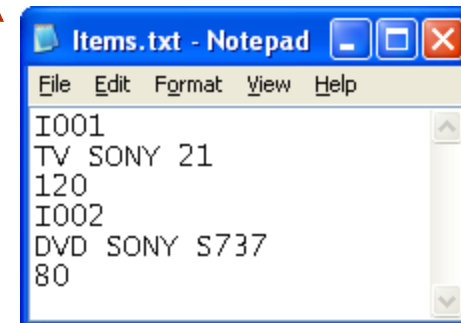
19         do {
20             System.out.println("\nEMPLOYEE MANAGER");
21             userChoice= menu.getUserChoice();
22             switch( userChoice) {
23                 case 1: list.addNewEmp(); changed= true; break;
24                 case 2: list.removeEmp(); changed= true; break;
25                 case 3: list.promote(); changed= true; break;
26                 case 4: list.print(); break;
27                 case 5: list.saveToFile(filename); changed= false;
28                 default : if (changed) {
29                     System.out.print("Save changes Y/N? ");
30                     String response= sc.nextLine().toUpperCase();
31                     if (response.startsWith("Y"))
32                         list.saveToFile(filename);
33                 }
34             }
35         }
36         while (userChoice>0 && userChoice<6);
37     }
38 }
  
```

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 can not 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)



Access Text Files ...: Case study 2.-Design

The screenshot displays an IDE environment for a Java project named 'Chapter09'. The file explorer on the left shows the project structure, with the 'items' folder and 'Items.txt' file highlighted. The 'Items.txt' file is open in a Notepad window, showing a list of items: I001, TV SONY 21, 120, I002, DVD SONY S737, 80. The project navigator on the right shows the 'items' package containing 'Item.java', 'ItemManager.java', 'Menu.java', and 'NewItems.java'. The 'ItemManager' class is selected, showing its methods: 'NewItems()', 'addNewItem()', 'appendToFile(String fName)', 'find(String aCode) : int', 'loadStoredCodes(String fName)', 'print()', 'removeItem()', 'updatePrice()', 'valid(String aCode) : boolean', 'sc : Scanner', and 'storedCodes : Vector<String>'. The output window on the right shows the program's execution, including a menu of options (1-Add new item, 2-Remove an item, 3-Update an item's price, 4-Print the list, 5-Save to files, 6-Quit) and the resulting list of items: I001, TV SONY 21, 120, I002, DVD SONY S737, 80, I003, TV SAMSUNG, 79.

Items.txt - Notepad

```
File Edit Format View Help
I001
TV SONY 21
120
I002
DVD SONY S737
80
```

Projects

- Chapter09
 - Source Packages
 - <default package>
 - employees
 - items
 - Item.java
 - ItemManager.java
 - Menu.java
 - NewItems.java

Navigator

Members View

- Menu :: Vector<String>
 - Menu()
 - addMenuItem(String S)
 - getUserChoice() : int
- NewItems :: Vector<Item>
 - NewItems()
 - addNewItem()
 - appendToFile(String fName)
 - find(String aCode) : int
 - loadStoredCodes(String fName)
 - print()
 - removeItem()
 - updatePrice()
 - valid(String aCode) : boolean
 - sc : Scanner
 - storedCodes : Vector<String>
- ItemManager
 - main(String[] args)

Output - Chapter09 (run)

```
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
```

Items.txt - Note...

```
File Edit Format View Help
I001
TV SONY 21
120
I002
DVD SONY S737
80
I003
TV SAMSUNG
79
```

Refer to the case study 1.
DO YOURSELF

```
1  /* Class for simple menu */
2  package items;
3  import java.util.Vector;
4  import java.util.Scanner;
5  public class Menu extends Vector <String> {
6      public Menu() { super(); }
7      void addMenuItem(String S) { this.add(S); }
8      int getUserChoice () {...}
16 }
```

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

```
1  /* Class for a product item */
2  package items;
3  public class Item {
4      private String code;
5      private String name;
6      private int price;
7      // Do yourself
8      public Item (String c, String n, int p) {...}
11     // Print details to the screen
12     public void print() {...}
15     //Getters and Setters
16     public String getCode() {...}
19     public void setCode(String code) {...}
22     public String getName() {...}
25     public void setName(String name) {...}
28     public int getPrice() {...}
31     public void setPrice(int price) {...}
34 }
```

```
NewItems.java * x
// 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 * x
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) {...}
```

```

47
48 //Append new-item list to a text file
49 public void appendToFile (String fName) {
50     if (this.size()==0) {
51         System.out.println("Empty list");
52         return;
53     }
54     try{ // append new items to the file
55         boolean append= true;
56         File f = new File(fName); // open file for appending data
57         FileWriter fw = new FileWriter(f,append); // write()
58         PrintWriter pw = new PrintWriter(fw); // println()
59         for (Item x:this) {
60             pw.println(x.getCode()); // write the code
61             pw.println(x.getName()); // write the name
62             pw.println(x.getPrice()); // write the price
63             pw.flush(); // write to file immediately
64         }
65         pw.close(); fw.close(); // close the file
66         this.loadStoredCodes(fName); // reload stored codes
67         this.clear(); // clear item list
68     }
69     catch (Exception e) {
70         System.out.println(e);
71     }
72 }

```

Items.txt - Note..

File	Edit	Format	View
I001			
TV SONY 21			
120			
I002			
DVD SONY S737			
80			
I003			
TV SAMSUNG			
79			

```

NewItems.java * x
73 // add new item
74 public void addNewItem() {
75     String newCode, newName; int price;
76     boolean duplicated = false, matched = true;
77     System.out.println("Enter New Item Details:");
78     do {
79         System.out.print("    code(format I000): ");
80         newCode = sc.nextLine().toUpperCase();
81         duplicated = !valid(newCode);
82         matched = newCode.matches("^I\\d{3}$"); // Pattern: I and 3 digits
83         if (duplicated) System.out.println("    The code is duplicated.");
84         if (!matched) System.out.println("    The code: I and 3 digits.");
85     }
86     while (duplicated || (!matched));
87     System.out.print("    name: ");
88     newName = sc.nextLine().toUpperCase();
89     System.out.print("    price: ");
90     price = Integer.parseInt(sc.nextLine());
91     this.add(new Item (newCode, newName, price));
92     System.out.println("New Item has been added.");
93 }
94 // remove an items from new-item list - DO YOURSELF
95 public void removeItem() {...}
106 // Update an Item price - DO YOURSELF
107 public void updatePrice() {...}
122 // Print out the list- DO YOURSELF
123 public void print() {...}
132 }

```

```

ItemManager.java x
1  /* The program for managing new-item list */
2  package items;
3  import java.util.Scanner;
4  public class ItemManager {
5      public static void main(String[] args) {
6          String filename = "items.txt";
7          Scanner sc= new Scanner(System.in);
8          Menu menu= new Menu();
9          menu.add("Add new item");
10         menu.add("Remove an item");
11         menu.add("Update an item's price");
12         menu.add("Print the list");
13         menu.add("Save to files");
14         menu.add("Quit");
15         int userChoice;
16         NewItems list= new NewItems();
17         list.loadStoredCodes(filename); // load initial data

```

Output - Chapter09 (run)

```

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

```


Output - Chapter09 (run)

```

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

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

```

```

18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

```

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);
}

```

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.

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)

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.

Access binary files ... java.io.RandomAccessFile class...

- Constructors

`RandomAccessFile(String file, String mode)`

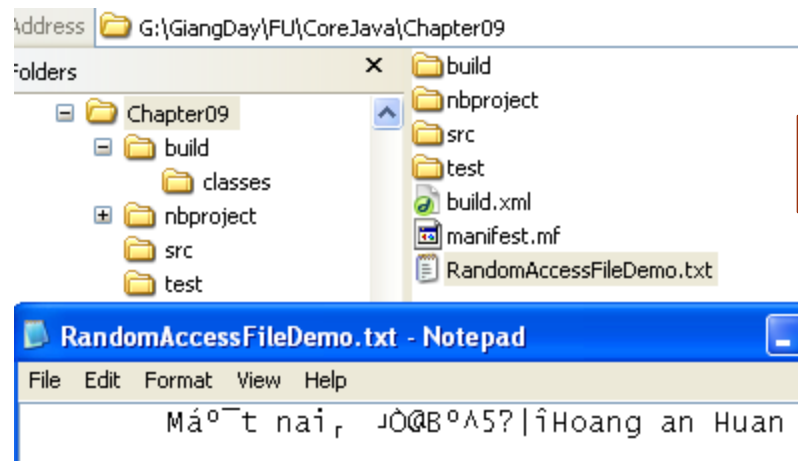
`RandomAccessFile(File file, String mode)`

- 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 updated only when the file is closed.

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



Output - Chapter09 (run)

```
run:
Mát 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= "Mát 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

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](#)

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++){
```

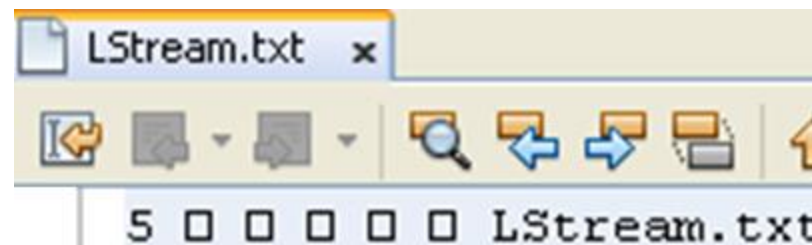
```
                os.write(fileName.charAt(i));
```

```
            }
```

```
            os.close();
```

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

Write
data to file



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

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'

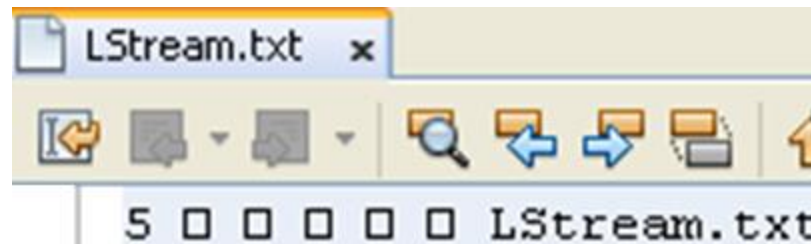
Read the blank

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



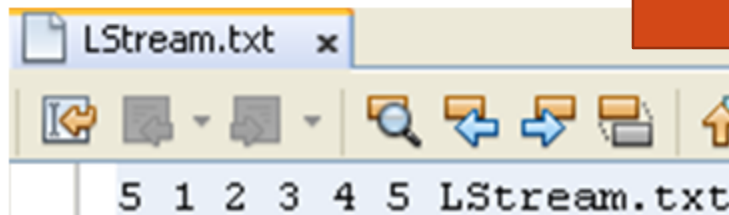
Access binary files... Low-Level Binary Stream Demo.2

```
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();
        }
    }
}
```

Write
data to file

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

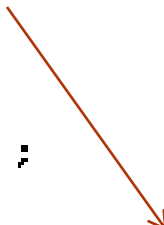
Now, we can see all the file content because they are characters



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
```

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
 -

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](#)

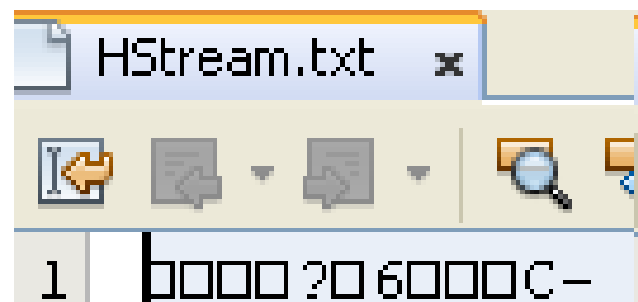
C:\Programming\jdk1.6.0\docs\api\java\io\package-tree.html

- 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](#)

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();
        }
    }
  
```



DataOutputStream
(int, string,...)

FileOutputStream
(byte)

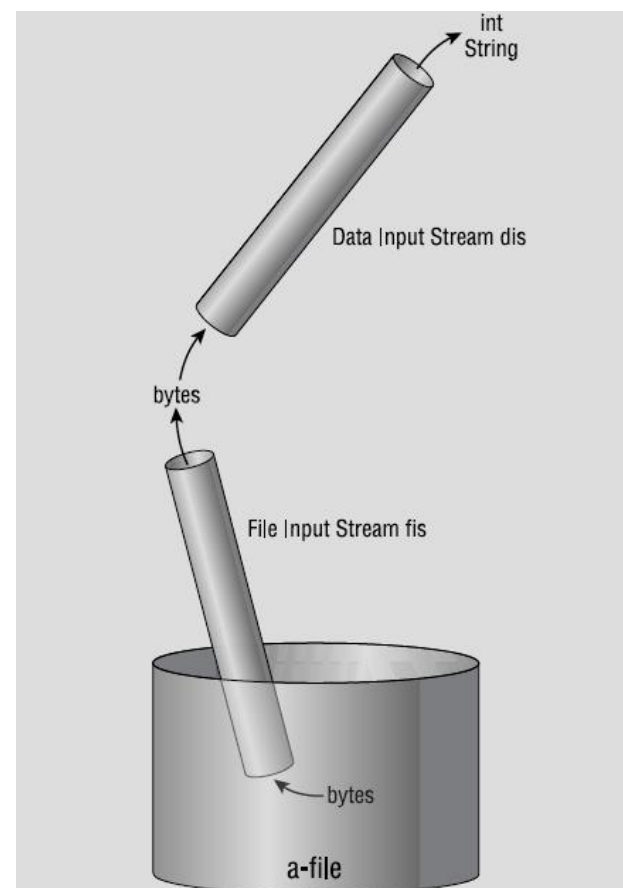
File

A high-level file
 access includes
 some low-level
 access
 (read an int
 value includes 4
 times of read a
 byte)

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
```



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 datasource , split the stream to fields then assign them to data fields of an object appropriately.

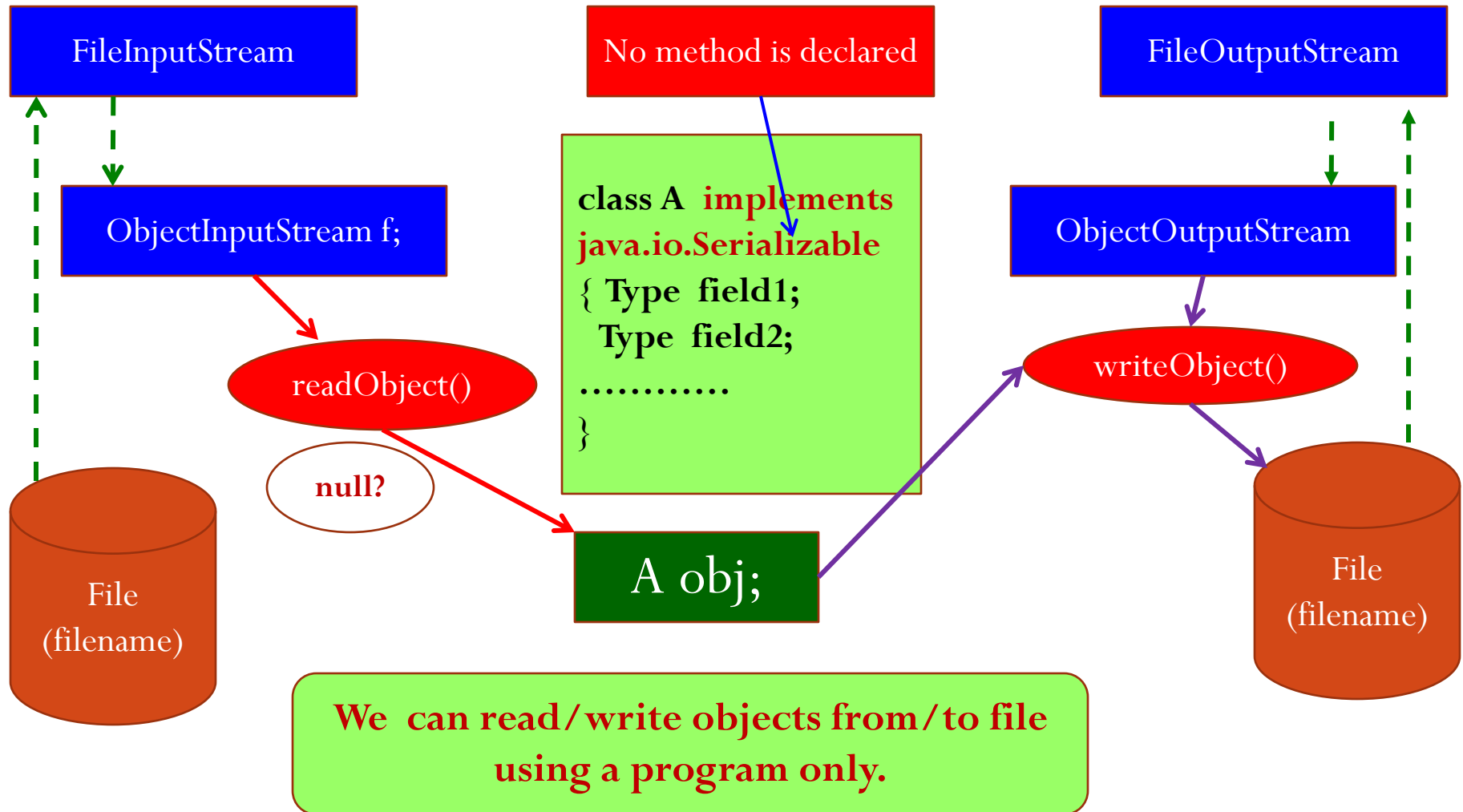
Transient fields are omitted when an object is serialized.

- 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

De-serialization

- De-serialization is to convert a serialized representation into a replica 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.

Access Object Files...: How to?



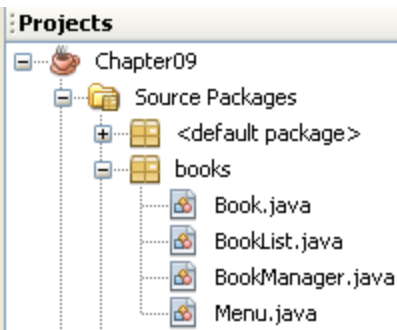
Access Object Files...:

Case study 3 - Object Streams Demo.

Problem

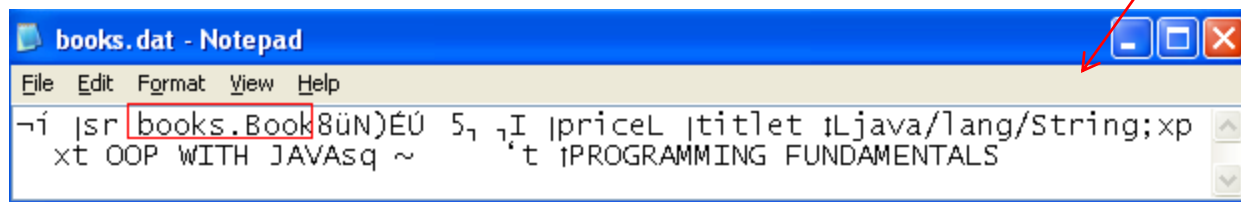
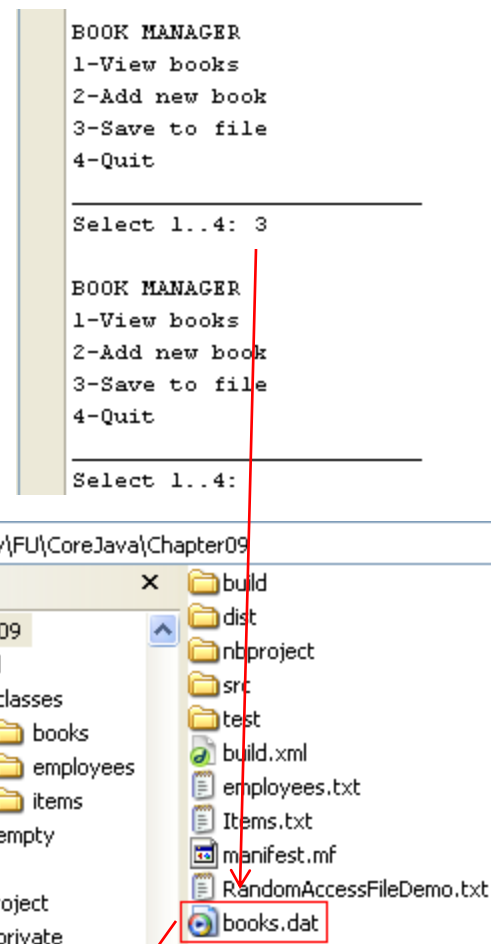
- Book <title, price>
- Write a Java program that allows user:
 - View books in the file books.dat
 - Append a book to the file
- Read/ Write books as binary objects from/to the file.

Access Object Files....: Case Study 3 - Design



```
Output - Chapter09 (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:
  title: 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:
  title: 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
```



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

Access Object Files...: Case Study 3- Implementations

Refer to the case study 1, 2.
DO YOURSELF

```

- /* 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) {...}
  }

```

Access Object Files...: Case Study 3– Implementations...

```
BookList.java x
1  /* Class for a book list */
2  package books;
3  import java.util.Scanner;
4  import java.util.Vector;
5  import java.io.*;
6  public class BookList extends Vector<Book> {
7      Scanner sc= new Scanner (System.in);
8      public void loadBookFromFile(String fName) {
9          // Clear current list before loading codes
10         if (this.size() > 0) this.clear();
11         try {
12             File f= new File(fName); // checking the file
13             if (!f.exists()) return;
14             FileInputStream fi= new FileInputStream(f); // read()
15             ObjectInputStream fo= new ObjectInputStream(fi); // readObject()
16             Book b;
17             while ( (b=(Book) (fo.readObject())) != null ) {
18                 this.add(b);
19             }
20             fo.close(); fi.close();
21         }
22         catch (Exception e) {
23             System.out.println(e);
24         }
25     }
}
```

Access Object Files...: Case Study 3– Implementations...

The screenshot shows an IDE with a file named `BookList.java` open. The code in the IDE is as follows:

```

26 // Save the list to file
27 // You can not append data to binary file because
28 // Java will write class information to the file
29 // each time data are appended to the file
30 public void saveToFile(String fName) {
31     if (this.size()==0) {
32         System.out.println("Empty list.");
33         return;
34     }
35     try {
36         FileOutputStream f= new FileOutputStream(fName); // write()
37         ObjectOutputStream fo= new ObjectOutputStream(f); // writeObject()
38         for (Book b: this) fo.writeObject(b);
39         fo.close(); f.close();
40     }
41     catch (Exception e) {
42         System.out.println(e);
43     }
44 }

```

A Notepad window titled `books.dat - Notepad` is also open, showing the contents of the `books.dat` file. The text in the Notepad window is:

```

-i |sr books.Book8ÜN)ÉÜ 5, 7I |priceL |titlet iLjava/lang/String;xp
xt OOP WITH JAVAsq ~ 't |PROGRAMMING FUNDAMENTALS

```

A red arrow points from the `books.dat` file in the Notepad window to the `saveToFile` method in the `BookList.java` file, indicating the relationship between the file and the code that writes to it.

Access Object Files...: Case Study 3– Implementations...

```

BookList.java * x
// add new item
public void addNewBook() {
    String title; int price;
    System.out.println("Enter New Book Details:");
    System.out.print("    title: ");
    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();
}

```

Access Object Files...:

Case Study 5 – Implementations...

```

BookManager.java * x
1  /* The program for managing book list */
2  package books;
3  import java.util.Scanner;
4  public class BookManager {
5      public static void main(String[] args) {
6          String filename = "books.dat";
7          Scanner sc= new Scanner(System.in);
8          Menu menu= new Menu();
9          menu.add("View books");
10         menu.add("Add new book");
11         menu.add("Save to file");
12         menu.add("Quit");
13         int userChoice;
14         BookList list= new BookList();
15         list.loadBookFromFile(filename); // load initial data
16         do {
17             System.out.println("\nBOOK MANAGER");
18             userChoice= menu.getUserChoice();
19             switch( userChoice) {
20                 case 1: list.print(); break;
21                 case 2: list.addNewBook(); break;
22                 case 3: list.saveToFile(filename);
23             }
24         }
25         while (userChoice>0 && userChoice<menu.size());
26     }
27 }
  
```

Output - Chapter09 (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:
    title: 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:
    title: 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
  
```

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