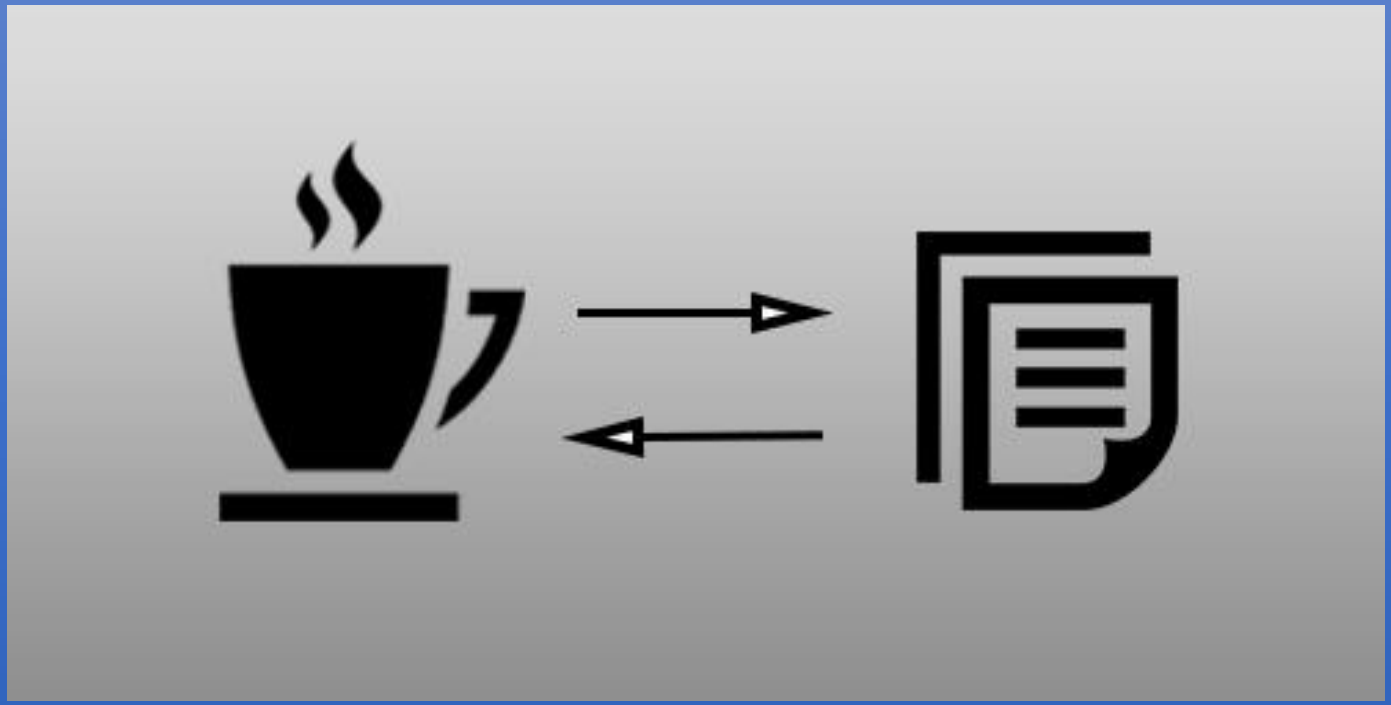


Lab 10
Java I.O

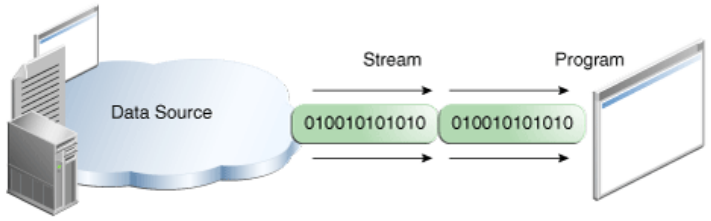
Goals

- Read and write text file
- Read and write object

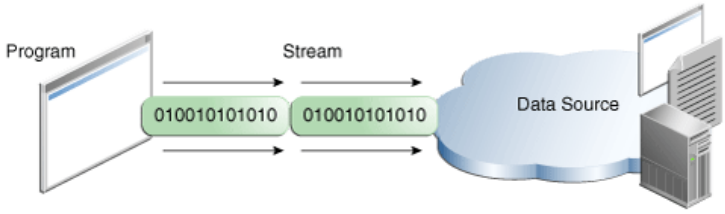
KEYNOTE

Stream

- An *I/O Stream* represents an input source or an output destination
- The streams that can handle all kinds of data, from primitive values to advanced objects.
- The data source and data destination pictured above can be anything that holds, generates, or consumes data:
  - disk files
  - another program,
  - a peripheral device,
  - a network socket,
  - or an array



Reading information into a program



Writing information from a program

Java I.O

- Table listing most Java IO classes divided by input, output, being byte based or character based, and any more specific purpose.

	Byte Based		**Character Based**	
	Input	**Output**	**Input**	**Output**
Basic	InputStream	OutputStream	InputStreamReader	OutputStreamWriter
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	**FileInputStream RandomAccessFile**	**FileOutputStream RandomAccessFile**	FileReader	FileWriter
Buffering	BufferedInputStream	BufferedOutputStream	**BufferedReader**	**BufferedWriter**
Strings			StringReader	StringWriter
Data	DataInputStream	DataOutputStream		
Data - Formatted		PrintStream		PrintWriter
Objects	**ObjectInputStream**	**ObjectOutputStream**		

Buffered Writer/Reader

- Example

```
FileWriter fw = new FileWriter(“your_file.txt");
BufferedWriter bw = new BufferedWriter(fw);
bw.writeData(content);
bw.flush();
bw.close();
```

```
FileReader fr = new FileReader(“your_file.txt");
BufferedReader br = new BufferedReader(fr);

String sCurrentLine;
// read until the end of file
while ((sCurrentLine = br.readLine()) != null) {
        System.out.println(sCurrentLine);
}
br.close();
```

Object(Output/Input)Stream

- The class must implement the Serializable interface

```
public class YourClass implements Serializable{}
```

```
FileOutputStream fo = new FileOutputStream("your_file.data")
ObjectOutputStream out = new ObjectOutputStream(fo);
// make sure the object is serializable
out.writeObject(yourSerializableObject);
out.close();
```

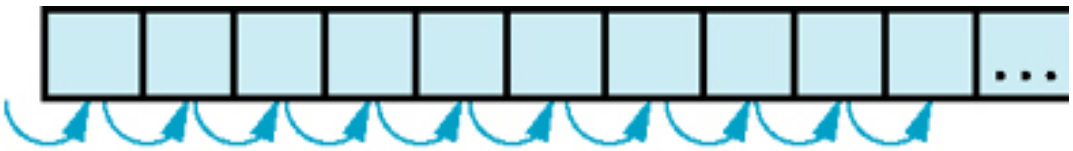
```
// create file stream and object stream
FileInputStream fi = new FileInputStream("your_file.data");
ObjectInputStream oistream = new ObjectInputStream(fi);

// read until the end of file input stream
while (istream.available() > 0){
        // casting here
        YourClass readObj = (YourClass)oistream.readObject();
}
oistream.close();
```

Need casting

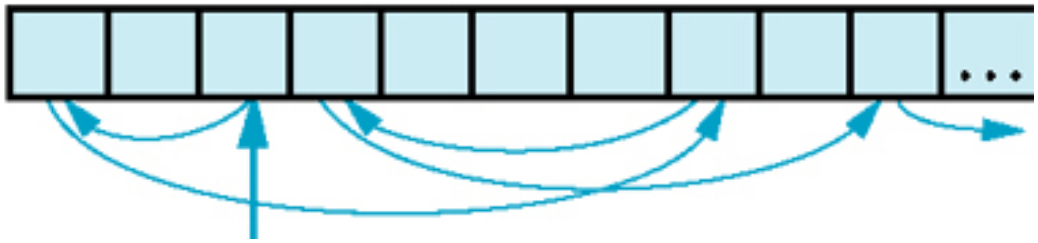
RandomAccessFile

```
RandomAccessFile raf = new RandomAccessFile("your_file.dat",
"rw");

int i = raf.readInt(); // read data as integer
raf.writeInt(i); // write an integer to file

raf.seek(position_in_file); // jump in any position in file
raf.skipByte(n_bytes); // skip n byte from current position

out.close();
```

Sequential access



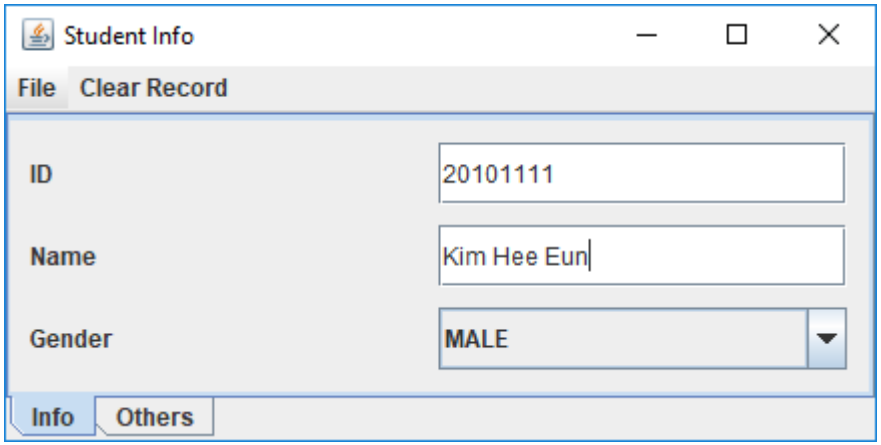
Random access



PRACTICE

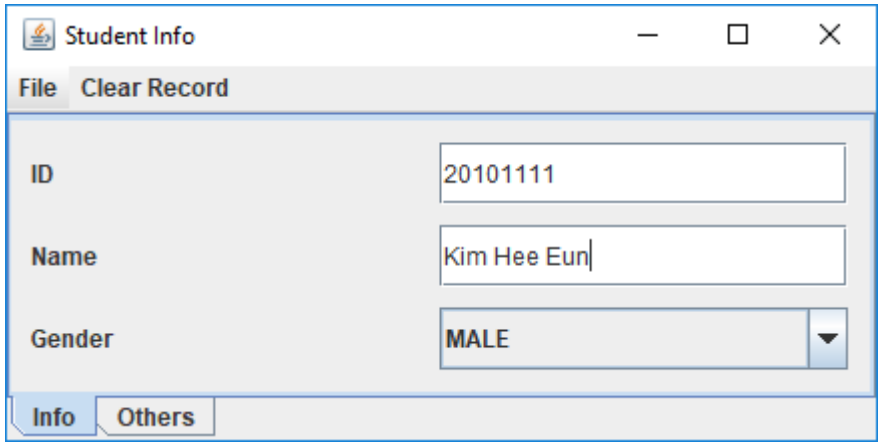
1. Problems

- Create text editor which can
  - Open file
  - Display text
  - Save file
    - [Character Base] BufferedReader & BufferedWriter
    - [Binary Base] ObjectInputStream & ObjectOutputStream
    - RandomAccessFile

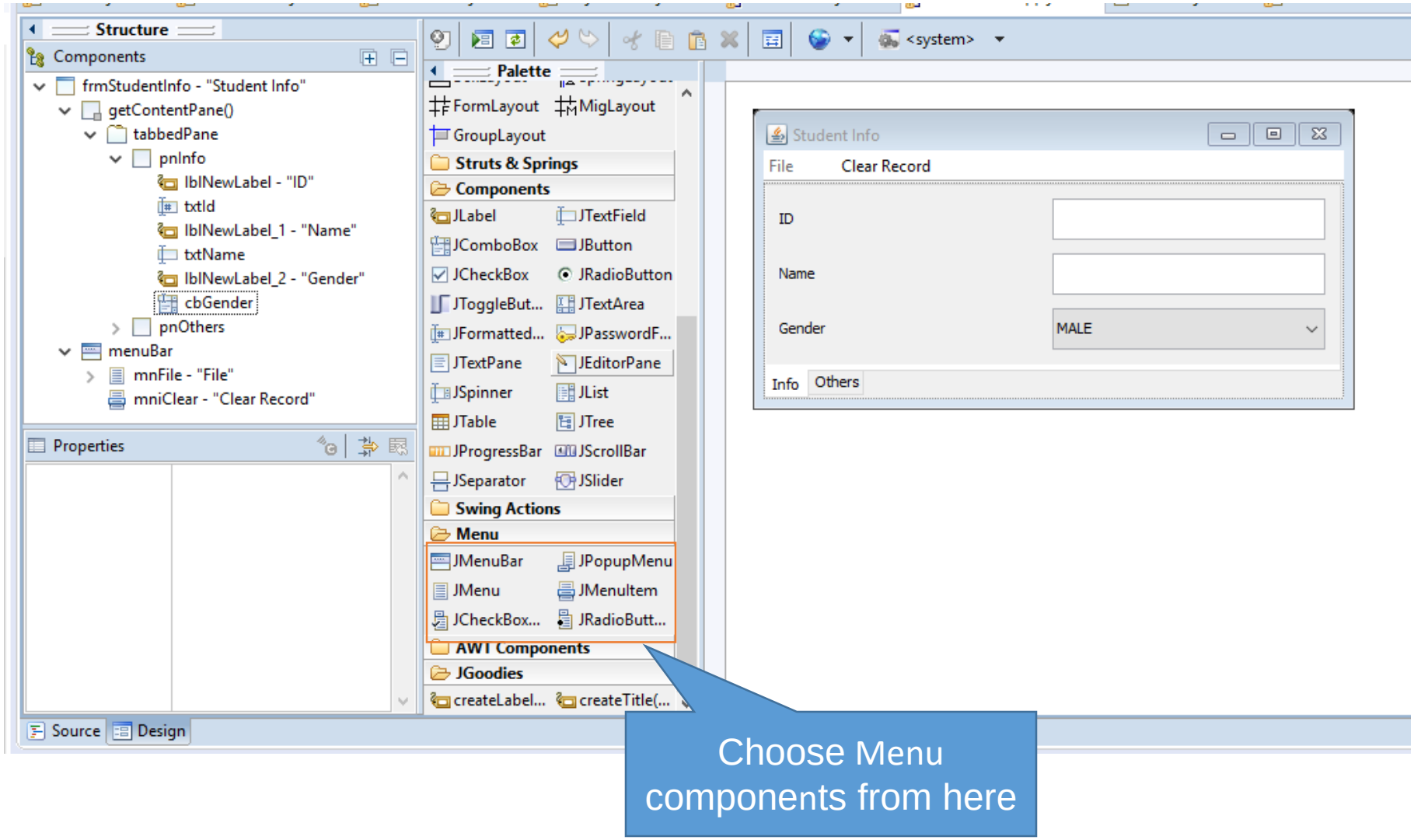


2. Design - MVC pattern

- Model:
  - Student student
- View:
  - JTextArea txtArea.
  - JTextField txtId.
  - JTextField txtName.
  - JComboBox cbGender
- Controller:
  - JMenuBar menu.
  - JMenuItem mniOpen, mniSave, mniExit, mniClear.



3. Implements
Create the components



3. Implements
Add action listener for Buttons

- Declare event handler class and implements the methods in listener interface.

```
public class FileIoDemo extends JFrame implements ActionListener {
     // Declare fields
     // Implement methods
     // ...

     public void actionPerformed(ActionEvent e) {
     //Handle open button action.
     if (e.getSource() == mnbtOpen) {
          // Open file here
     }else if (e.getSource() == mnbtSave) {
          // Save file here
   }else if (e.getSource() == mnbtExit){
        FileIoDemo.this.dispose();
   }
}
```

- Register an instance of the event handler class as a listener on menu item components.

```
mnbtOpen = new JMenuItem("Open");
mnbtOpen.addActionListener(this);

mnbtSave = new JMenuItem("Save");
mnbtSave.addActionListener(this);
```

3. Implements
Event handling

- Open selected file:

```
if (e.getSource() == mnbtOpen) {
   // create file chooser and show it
   JFileChooser fc = new JFileChooser();
   int returnVal = fc.showOpenDialog(FileIoDemo.this);

   // check if user action
   if (returnVal == JFileChooser.APPROVE_OPTION) {
      File file = fc.getSelectedFile();
      Path path = Paths.get(file.getAbsolutePath());

      // put your code to ready from file

   }
}
```

3. Implements
Event handling

- Save to selected file:

```
else if (e.getSource() == mnbtSave) {
   // create file chooser and show it
   JFileChooser fc = new JFileChooser();
   int returnVal = fc.showSaveDialog(FileIoDemo.this);

   // check if user action
   if (returnVal == JFileChooser.APPROVE_OPTION) {
      File file = fc.getSelectedFile();
      Path path = Paths.get(file.getAbsolutePath());

      // put your code to write to file

   }
}
```