Lecture 1

A program for keeping track of countries of the world.

- `Country`: has 3 instance variables (name, population & currency); a constructor & 3 getters
- Class `MainGUI`: runs a `JFileChooser`; shows 2 buttons (“Enter Countries” & “View Countries”); their handlers create further windows
- Exceptions (checked)
  - `CountryDataInputException`: if the user is entering data about a new country but supplies illegal data (e.g. non-numeric population)
  - `NoSuchCountryException`: if the user searches the file for a country which is not in the file
  - `CountryFileIOException`: if the file cannot be opened, read/written or closed

The `EnterGUI` class

This GUI is a modal dialog (using `JDialog`).

It contains a `CountryPanel` with a button below it.
This is the button’s handler:

```
public void actionPerformed(ActionEvent ae)
{  try
   {  Country c = countryPanel.getCountry();
      theCFile.insert(c);
      countryPanel.clear();
   }
   catch (CountryDataInputException cdie)
   {  // Show error message
   }
   catch (CountryFileIOException nsce)
   {  // Show error message
   }
}
```

Errors messages are also done using dialogs: the `JOptionPane` class offers some standard dialogs for error messages, etc.

The `ViewGUI` class

This GUI is also a modal `JDialog`.

It contains a `CountryPanel` with a textfield below it.
This is the textfield’s handler:

```
public void actionPerformed(ActionEvent ae)
{  try
   {  countryPanel.clear();
      String cname = findTF.getText().trim()
      Country c = theCFile.retrieve(cname);
      countryPanel.setCountry(c);
   }
   catch (NoSuchCountryException nsce)
   {  // Show information message
   }
   catch (CountryFileIOException nsce)
   {  // Show error message
   }
}
```


The `CountryPanel` class

A subclass of `JPanel` with 3 labels & 3 textfields.

```
public Country getCountry()
 throws CountryDataInputException
{ try
  { return new Country(
     nameTF.getText().trim(),
     Integer.parseInt(popTF.getText().trim()
     currTF.getText().trim());
  }
  catch (NumberFormatException nfe)
  { throw new CountryDataInputException();
  }
}

public void setCountry(Country theCountry)
{ nameTF.setText(theCountry.getName());
  popTF.setText("" + theCountry.getPopulatio
  currTF.setText(theCountry.getCurrency());
}
```


The skeleton of the `CountryFile` class

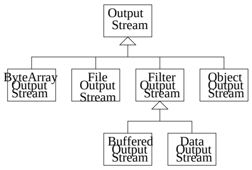
```
public class CountryFile
{  public CountryFile(File theFile)
   {   file = theFile;
   }

   public Country retrieve(String theTarget)
      throws NoSuchCountryException,
             CountryFileIOException
   {  // Open file, find country (or throw
      // exception), close file
   }

   public void insert(Country theCountry)
      throws CountryFileIOException
   {  // Open file (in append mode), write
      // country, close file
   }

   private File file;
}
```


File I/O

- A *sequential-access* file is a stream of data: a continuous *one-way* flow
- A *random-access* file allows *reads and/or writes* at any position within the file
- *Binary files* store data using the same formats used to store data in main memory
- *Text files* contain data that is to be interpreted as sequences of characters

	Sequential	Random
Binary	Subclasses of abstract classes *`InputStream`* & *`OutputStream`*	Class `RandomAccess-File`
Text	Subclasses of abstract classes *`Reader`* & *`Writer`*	N/A

Writing binary data: `OutputStream`



Writing bytes to binary files: `FileOutputStream`

- You write *bytes* to a `FileOutputStream`, so writing an `int` will write only the least sig 8 bits:

```
stream = new FileOutputStream(...);
int i = 23;
stream.write(i);
```

- To write a complete `int` takes, e.g., 4 `write`s and 3 shifts

Writing data to binary files: `DataOutputStream`

- Streams can be 'pipelined' together. E.g. attach a `DataOutputStream`, can write primitive types to another output stream: `writeBoolean`, `writeByte`, `writeChar` (16 bits), `writeDouble`, `writeFloat`, `writeInt`, `writeLong`, `writeShort`

- So form a 'pipeline'; then can write an `int` easily:

```
stream = new DataOutputStream(
      new FileOutputStream(...));
int i = 23;
stream.writeInt(i);
```

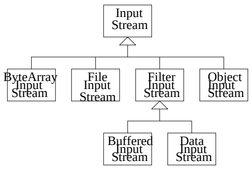
- Can even write strings: `stream.writeUTF("Hello")` writes the string in a special encoding called UTF (Unicode Transmission Format), designed for compact transmission of Unicode characters

- Attach a `BufferedOutputStream` for efficiency

Using this in `CountryFile`

```
public void insert(Country theCountry)
   throws CountryFileIOException
{  DataOutputStream dos = null;
   try
   {  dos = new DataOutputStream(new
        BufferedOutputStream(new
          FileOutputStream(file)));
      dos.writeUTF(theCountry.getName());
      dos.writeInt(theCountry.getPopulation(
      dos.writeUTF(theCountry.getCurrency())
   }
   catch (IOException ioe)
   {  throw new CountryFileIOException();
   }
   finally
   {  // try to close
   }
}
```


Reading binary data: `InputStream`



Reading bytes from binary files: `FileInputStream`

- You read *bytes* from a `FileInputStream`, so the following reads just one byte:

```
stream = new FileInputStream(...);
int i = stream.read();
```

- To read a complete `int` takes, e.g., 4 `read`s and 3 shifts

Reading data from binary files: `DataInputStream`

- So attach a `DataInputStream`, can read primitive types from another input stream: `readBoolean`, `readByte`, `readChar` (16 bits), `readDouble`, `readFloat`, `readInt`, `readLong`, `readShort`, `readUTF()`

- So form a 'pipeline'; then can read an `int` easily:

```
stream = new DataInputStream(
            new FileInputStream(...));
int i = stream.readInt();
```

- These read methods throw `EOFException` if end of file is reached (including if it is reached when trying to read in the appropriate number of bytes)

- Attach a `BufferedInputStream` for efficiency

Using this in `CountryFile`

```
public Country retrieve(String theTarget)
throws NoSuchCountryException,CountryFileIOE
{  DataInputStream dis = null;
   try
   {  dis = new DataInputStream(
        new BufferedInputStream(
          new FileInputStream(file)));
      while (true)
      {  String n = dis.readUTF();
         int p = dis.readInt();
         String c = dis.readUTF();
         if (n.equals(theTarget))
         {  return new Country(n, p, c);}
      }
   }
   catch (EOFException eofe)
   {  throw new NoSuchCountryException();}
   catch (IOException ioe)
   {  throw new CountryFileIOException();}
   finally
   {  // try to close }
}
```