Java IO programming

Yulia Newton, Ph.D.

CS151, Object Oriented Programming

San Jose State University

Spring 2020

IO programming in Java

- Java IO classes are available in *java.io* package
  - *import java.io.*;*
- Streams are sequence of data to be processed
  - Input stream
  - Output stream
- Types of streams
  - File streams – reading and writing data from and to files
    - Byte stream – input/output data as 8-bit bytes
  - Character stream – input/output data as 16-bit Unicode
  - Standard streams
    - Standard input – input device represented as *System.in* (e.g. keyboard)
    - Standard output – output device represented as *System.out* (e.g. terminal screen)
    - Standard error – output device represented as *System.err* (e.g. computer screen)

Useful IO classes

- File input/output streams
  - FileInputStream, FileOutputStream
- Character streams with file readers/writers
  - FileReader, FileWriter
- Standard input/output streams
  - InputStreamReader, OutputStreamWriter

Example: programming with byte streams

```
import java.io.*;
public class Test {
    public static void main(String args[]) throws IOException{
        FileInputStream myInput = null;
        FileOutputStream myOutput = null;

        try {
            myInput = new FileInputStream("input_test.txt");
            myOutput = new FileOutputStream("output_test.txt");

            int character;
            while ((character = myInput.read()) != -1) {
                myOutput.write(character);
            }
        }finally {
            if (myInput != null) {
                myInput.close();
            }
            if (myOutput != null) {
                myOutput.close();
            }
        }
    }
}
```

Example: programming with character streams (copying contents of one file to another)

```
import java.io.*;
public class Test {
    public static void main(String args[]) throws IOException{
        FileReader myInput = null;
        FileWriter myOutput = null;

        try {
            myInput = new FileReader("input_test.txt");
            myOutput = new FileWriter("output_test.txt");

            int character;
            while ((character = myInput.read()) != -1) {
                myOutput.write(character);
            }
        }finally {
            if (myInput != null) {
                myInput.close();
            }
            if (myOutput != null) {
                myOutput.close();
            }
        }
    }
}
```

Copies the contents of the input stream into output stream

Example: programming with character streams (create new file, read from newly created file)

```
import java.io.*;
public class Test {
    public static void main(String args[]){
        File myNewFile = null;
        FileWriter myFileWriter = null;
        FileReader myFileReader = null;

        try{
            // create a new file and output text to it:
            myNewFile = new File("new_test_input.txt");
            myFileWriter = new FileWriter(myNewFile);

            myFileWriter.write("This is my new contents");
            myFileWriter.flush();
            myFileWriter.close();

            // open and read newly created file:
            myFileReader = new FileReader(myNewFile);
            int data = myFileReader.read();
            while(data != -1){
                System.out.println((char)data);
                data = myFileReader.read();
            }
            myFileReader.close();
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Example: working with standard input stream

```java
import java.io.*;
public class Test {
    public static void main(String args[]) throws IOException{
        InputStreamReader myStandardIn = null;

        try {
            myStandardIn = new InputStreamReader(System.in);
            System.out.println("Enter 'x' to quit");

            char character;
            do {
                character = (char) myStandardIn.read();
                System.out.print(character);
            } while(character != 'x');
        }finally {
            if (myStandardIn != null) {
                myStandardIn.close();
            }
        }
    }
}
```

Output:

```
Enter 'x' to quit
a
a
b
b
x
```

Example: working with standard output stream

```java
import java.io.*;
public class Test {
    public static void main(String args[]) throws IOException{
        OutputStreamWriter myStandardOut = null;
        String test = "This is my test string\n";

        try {
            myStandardOut = new OutputStreamWriter(System.out);
            myStandardOut.write(test);
        }finally {
            if (myStandardOut != null) {
                myStandardOut.close();
            }
        }
    }
}
```

Output:

```
This is my test string
```

Example: working with file streams

```java
import java.io.*;
public class Test {
    public static void main(String args[]){
        FileInputStream myFileIn = null;
        FileOutputStream myFileOut = null;


        try {
            myFileIn = new FileInputStream("input_test.txt");
            myFileOut = new FileOutputStream("output_test.txt");

            int buffer;
            while  ((buffer=myFileIn.read()) != -1){
                myFileOut.write(buffer);
            }

            myFileIn.close();
            myFileOut.close();
        }catch(IOException e){
            System.out.print(e);
        }finally {
            if (myFileIn != null) {
                try{
                    myFileIn.close();
                }catch(IOException e){
                    System.out.print(e);
                }
            }
            if (myFileOut != null) {
                try{
                    myFileOut.close();
                }catch(IOException e){
                    System.out.print(e);
                }
            }
        }
    }
}
```

file streams

copy contents of the input file into output file

close the file streams

Example: using *File* object

```
import java.io.*;
public class Test {
    public static void main(String args[]){
        File myFile = new File("input_test.txt");

        if(myFile.exists()){
            System.out.println("File name :"+myFile.getName());
            System.out.println("Path: "+myFile.getPath());
            System.out.println("Absolute path:" +myFile.getAbsolutePath());
            System.out.println("Parent:"+myFile.getParent());
            System.out.println("Is writeable:"+myFile.canWrite());
            System.out.println("Is readable"+myFile.canRead());
            System.out.println("Is a directory:"+myFile.isDirectory());
            System.out.println("File Size in bytes "+myFile.length());
        }
    }
}
```

Example: working with directories

List the contents of specified directory:

```
import java.io.File;
public class Test {
    public static void main(String args[]){
        File myDir = null;

        try{
            myDir = new File(".");

            String[] dirContents = myDir.list();
            for(String f:dirContents) {
                System.out.println(f);
            }
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

specify dir path/name

Create/delete directory:

```
import java.io.File;
public class Test {
    public static void main(String args[]){
        File newDir = null;

        try{
            newDir = new File("./test_directory");

            // create new directory
            newDir.mkdir();
            if(newDir.exists()){
                System.out.println("New directory created successfully");
            }

            // remove newly created directory
            newDir.delete();
            if(newDir.exists()){
                System.out.println("Failed to remove this directory");
            }else{
                System.out.println("Successfully removed this directory");
            }
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

use *newDir.mkdirs()* to create parent directory structure

Example: working with directories

List the contents of specified directory:

```
import java.io.File;
public class Test {
    public static void main(String args[]){
        File myDir = null;

        try{
            myDir = new File(".");

            String[] dirContents = myDir.list();
            for(String f:dirContents) {
                System.out.println(f);
            }
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

specify dir path/name

Create/delete directory:

```
import java.io.File;
public class Test {
    public static void main(String args[]){
        File newDir = null;

        try{
            newDir = new File("./test_directory");

            // create new directory
            newDir.mkdir();
            if(newDir.exists()){
                System.out.println("New directory created successfully");
            }

            // remove newly created directory
            newDir.delete();
            if(newDir.exists()){
                System.out.println("Failed to remove this directory");
            }else{
                System.out.println("Successfully removed this directory");
            }
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

use *newDir.mkdirs()* to create parent directory structure

To recap

- Input/output streams
- Types of streams
  - File streams
    - FileInputStream, FileOutputStream
  - Character stream
    - FileReader, FileWriter
  - Standard streams
    - InputStreamReader, OutputStreamWriter
- File objects can be used to manipulate files as well as directories