

Basic Input/Output in Java

CS 4354
Summer II 2016

Jill Seaman

1

Reading from the screen (Input)

- Scanner class (in java.util)
 - ◆ Allows the user to read values of various types from a stream of characters.
 - ◆ There are two constructors that are particularly useful: one takes an `InputStream` object as a parameter and the other takes a `FileReader` object as a parameter.

```
Scanner in = new Scanner(System.in);  
// System.in is the InputStream associated with the keyboard
```

```
Scanner inFile = new Scanner(new FileReader("myFile"));  
// Creates a Scanner for a text file called myFile
```

2

Reading from the screen (Input)

- Useful Scanner methods:

- ◆ `int nextInt()` Returns the next token as an int. If the next token is not an integer, `InputMismatchException` is thrown.
- ◆ `long nextLong()` Similar
- ◆ `float nextFloat()` Similar
- ◆ `double nextDouble()` Similar
- ◆ `String nextLine()` Returns the rest of the current line, excluding any line separator at the end.
- ◆ `boolean hasNextInt()` Returns true if the next token in this scanner's input can be interpreted as an int value using the `nextInt()` method.
- ◆ `hasNextLong()`, `hasNextFloat()`, etc.

3

Reading from the screen (Input)

- Example using a Scanner with `System.in`:

```
Scanner sc = new Scanner(System.in);  
System.out.println("Enter the quantity: ");  
int i = sc.nextInt();  
System.out.println("Enter the price: ");  
double price = sc.nextDouble();  
System.out.println("Enter the name: ");  
sc.nextLine(); //skip to end of previous line, after price  
String name = sc.nextLine();
```

4

Writing to the screen (Output)

- System.out (in java.lang)
 - ◆ System.out is a PrintStream, used to print characters.
 - ◆ A PrintStream provides the ability to print **representations of various data values** conveniently.
- println(x) and print(x)
 - ◆ Methods of PrintStream (see API website for details)
 - ◆ Overloaded to print all the various data types.
 - ◆ Often uses the default toString() method of the wrapper classes.
 - for example, Integer.toString(int i) to print an int
 - ◆ The difference between print() and println() is that the latter adds a newline when it's done.

5

Wrapper classes

- Wrapper classes convert primitive type values to objects
 - ◆ **Byte, Short, Integer, Float, Double, Boolean**, etc.
 - ◆ In the java.lang package.
- Allows use of primitive values where Objects are required.
- Provides conversion functions between types.

```
int i = 50;
Integer mm = new Integer(i);
String k = mm.toString(); // k is now "50"
String k1 = Integer.toString(50); // the static toString method

int j = mm.intValue() + 5; // j is now 55
int y = Integer.parseInt(k); // converts string to int
```

- compare() and compareTo(Integer) are defined as well.

6

Writing to the screen: Formatting

- DecimalFormat class, used to format decimal numbers
 - ◆ DecimalFormat(String pattern) Creates a DecimalFormat using the given pattern.
 - ◆ format(x) produces a string by formatting an item (x) according to the objects pattern.
 - ◆ The following characters have special meaning in a pattern (other characters are taken literally, appearing in the string unchanged).

0	digit (left-padded with zeros)
#	digit, zero shows as absent (no 0 padding)
.	decimal separator
,	Grouping separator
E	Separates mantissa and exponent in scientific notation
%	Multiply by 100, show as percent

7

Formatting example

```
import java.text.*;

class FormatOut {
    public static void main(String args[]) {
        int [] iArray = {1, 12, 123};
        float [] fArray = {1.1F, 10.12F, 100.123F};
        double [] dArray = {1.1, 10.12, 100.1234, 1000.1239};

        DecimalFormat dfi = new DecimalFormat("#00");
        DecimalFormat dff = new DecimalFormat("#00.00 float");
        DecimalFormat dfd = new DecimalFormat("#000.000");

        for (int i = 0; i < iArray.length; i++)
            System.out.println(dfi.format(iArray[i]));

        for (int i = 0; i < fArray.length; i++)
            System.out.println(dff.format(fArray[i]));

        for (int i = 0; i < dArray.length; i++)
            System.out.println(dfd.format(dArray[i]));
    }
}
```

8

Formatting example

- Output from running FormatOut:

```
01
12
123
01.10 float
10.12 float
100.12 float
001.100
010.120
100.123
1000.124
```

9

Object serialization

- A process of transforming an object into a stream of bytes, to be saved in a file.
- Object serialization allows you to implement persistence:
- Persistence: when an object's lifetime is not determined by whether a program is executing; the object exists in between invocations of the program.
- The object's class must implement the `Serializable` interface.

```
public class Circle implements Serializable { ...
```

- ◆ If not, you get an exception: `java.io.NotSerializableException: theClass`
- ◆ Note: there are no required methods to override
- ◆ The field object types must be serializable too.

10

Object serialization: streams

- Java provides two object streams for serialization.
 - ◆ These are both initialized given a `FileOutputStream` and a `FileInputStream` (respectively). The example shows how to initialize these given a file name.
- `ObjectOutputStream`
 - ◆ The `writeObject()` method writes an object to the output stream, converting all the data in the object to bytes.
 - ◆ All the field objects in the class must also be serializable
- `ObjectInputStream`
 - ◆ The `readObject()` method reads an object from the input stream.
 - ◆ The object was most likely written using `writeObject`
 - ◆ You must cast the result to the correct object.

11

Serialization example: ZStudent.java

```
import java.io.*;

// Simple student class
class ZStudent implements Serializable {
    int no;
    String first, mid, last; // Note these are serializable objects
    float ave;

    ZStudent() {} // default constructor
    ZStudent(int no, String first, String mid, String last, float ave) {
        this.no = no;
        this.first = first;
        this.mid = mid;
        this.last = last;
        this.ave = ave;
    }

    public String display() {
        return (no + " " + first + " " + mid + " " + last + " " + ave);
    }
}
```

12

Serialization example: ObjFIO.java

```
import java.io.*;
import java.util.*;

class ObjFIO {
    public static void main(String[] args) {
        ArrayList<ZStudent> zstudents = new ArrayList<ZStudent>();
        zstudents.add(new ZStudent(50, "Blue ", "M", "Monday ", 50.0F));
        zstudents.add(new ZStudent(100, "Gray ", "G", "Tuesday ", 60.0F));
        zstudents.add(new ZStudent(150, "Green", "G", "Wednesday", 70.0F));
        zstudents.add(new ZStudent(200, "Pink ", "P", "Thursday ", 80.0F));
        zstudents.add(new ZStudent(300, "Red ", "R", "Friday ", 90.0F));

        try {
            FileOutputStream fos = new FileOutputStream("zStudentFile");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(zstudents); //ArrayList & contents are serializable
            fos.close();
        } catch (IOException e) {
            System.out.println("Problem with file output");
        }
    }
}
```

13

Serialization example: ObjFIO.java cont.

```
try {
    FileInputStream fis = new FileInputStream("zStudentFile");
    ObjectInputStream ois = new ObjectInputStream(fis);

    ArrayList<ZStudent> students =
        (ArrayList<ZStudent>)ois.readObject(); // explicit cast reqd
    for (ZStudent zs : students)
        System.out.println(zs.display());

    fis.close();
} catch (FileNotFoundException e) {
    System.out.println("Cannot find datafile.");
} catch (IOException e) {
    System.out.println("Problem with file input.");
} catch (ClassNotFoundException e) {
    System.out.println("Class not found on input from file.");
}
}
```

14

Serialization example

- Output from the example:

```
50 Blue M Monday 50.0
100 Gray G Tuesday 60.0
150 Green G Wednesday 70.0
200 Pink P Thursday 80.0
300 Red R Friday 90.0
```

15