Basic Input/Output in Java

CS 4354
Summer II 2015

Jill Seaman

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"));
```


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.

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: ");
price = sc.nextDouble();
System.out.println("Enter the name: ");
sc.nextLine();  //skip to end of previous line
name = sc.nextLine();
```

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.

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

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]));
    }
}
```


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
```

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.

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.

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);
    }
}
```


Serialization example: ObjFIO.java

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

        try {
            FileOutputStream fos = new FileOutputStream("zStudentFile");
            ObjectOutputStream oos = new ObjectOutputStream(fos);

            for (int i = 0; i < 5; i++) {
                oos.writeObject(zstudents[i]);// to write 1 obj at a time
            }
            fos.close();
        } catch (IOException e) {
            System.out.println("Problem with file output");
        }
```

Serialization example: ObjFIO.java cont.

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

            // to read in student records from a file
            for (int i = 0; i < 5; i++) {
                stud = (ZStudent)ois.readObject();  // explicit cast reqd
                System.out.println(stud.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.");
        }
    }
}
```


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
```

- Note: Arrays are objects, and may be serialized as a whole:

```
oos.writeObject(zstudents);
```

```
ZStudent [] newStudents = (ZStudent[])ois.readObject();
for (int i=0; i<5; i++) {
    System.out.println(newStudents[i].display());
}
```