Programming 1

Lecture 10 – File Input/Output (2)

Faculty of Information Technology
Hanoi University

Contents

- Learn to print out formatted text.
  - `System.out.format(), String.format()`
- Learn to read text files with
  - `java.io.BufferedReader`
- Learn to write text files with
  - `java.io.BufferedWriter`
- Data serialization & de-serialization
  - `ObjectOutputStream`
  - `ObjectInputStream`

The `System.out.format()` method

- Prints multiple arguments based on a *format string*.
- The *format string* consists of static text and format specifiers.
- Examples:

```
System.out.format("Square root of %d is %f.%n", 2, 1.41);
```

Output:

```
The square root of 2 is 1.41.
```

%d	an integer
%f	a double or float
%n	new line character

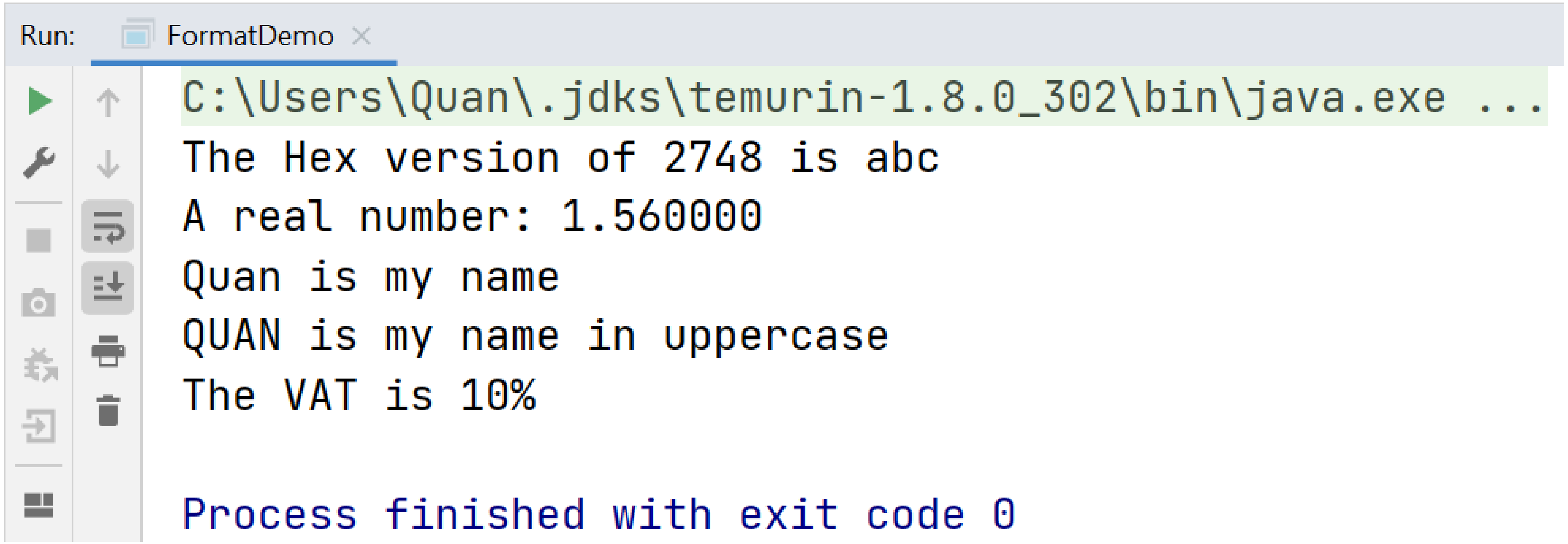
The `System.out.format()` method

- This statement is the same as `System.out.printf()`
- Basic format specifiers:

Format specifiers	Description
%d	An integer in decimal form
%x	An integer in hexadecimal form
%f	A floating point number (float, double)
%s	A String
%S	A String converted to UPPERCASE
%n	Platform-specific linebreak character
%%	The % character

Example: basic format specifiers

```java
int n = 2748;
System.out.format("The Hex version of %d is %x%n", n, n);
System.out.format("A real number: %f%n", 1.56);
System.out.format("%s is my name%n", "Quan");
System.out.format("%S is my name in uppercase%n", "Quan");
System.out.format("The VAT is %d%%%n", 10);
```



```
C:\Users\Quan\.jdks\temurin-1.8.0_302\bin\java.exe ...
The Hex version of 2748 is abc
A real number: 1.560000
Quan is my name
QUAN is my name in uppercase
The VAT is 10%

Process finished with exit code 0
```

Customizing integer format specifier

Example	Description	Example
%5d	An integer which takes up at least 5 character spaces, padded with spaces	`System.out.format("%5d", 15);` **Result:** `"   15"` (3 spaces) `System.out.format("%3d", 2457);` **Result:** `"2457"` (0 space)
%05d	An integer which takes up at least 5 character spaces, zero-padded	`System.out.format("%05d", 15);` **Result:** `"00015"`
%-5d	An integer which takes up 5 character spaces, padded with spaces, left-aligned	`System.out.format("%-5d", 15);` **Result:** `"15   "` (3 spaces)

Customizing floating-point specifier

Example	Description	Example
%10f	A real number which takes up at least 10 spaces (width) and has 6 digits (precision) after the decimal point (6 is default)	`System.out.format("%10f", 1.5);` **Result:** `"  1.500000"` (2 spaces)
%10.1f	Width = 10, precision = 1	`System.out.format("%10.1f", 1.5);` **Result:** `"       1.5"` (7 spaces)
%-10.2f	Width = 10, precision = 2, left-aligned	`System.out.format("%-10.2f", 1.5);` **Result:** `"1.50      "` (6 spaces)
%010.3f	Width = 10, precision = 3, zero-padded	`System.out.format("%010.3f", 1.5);` **Result:** `"000001.500"`
%.4f	Width = auto, precision = 4	`System.out.format("%.4f", 1.5);` **Result:** `"1.5000"`

Customizing String format specifier

Example	Description	Example
%9s	A String which takes up at least 9 spaces (width), right-aligned	`System.out.format("%9s", "abc");` **Result:** `"      abc"` (6 spaces)
%9.2s	Width = 9, limited to 2 characters (right-truncated)	`System.out.format("%9.2s", "abc");` **Result:** `"       ab"` (7 spaces)
%-9s	Width = 9, left-aligned	`System.out.format("%-9s", "abc");` **Result:** `"abc      "` (6 spaces)

%.3s	Width = auto, limited to 3 characters (right-truncated)	`System.out.format("%.3s", "abcd");`
		Result: `"abc"`
		
		`System.out.format("%.3s", "ab");`
		Result: `"ab"`

The System.out.format() method

Syntax:

```
System.out.format(format_string, arg1, arg2…)
```

- Except for %% and %n, all format specifiers must match an argument.
- We can specify which format specifier to use with which argument by adding the argument index to the format specifier.

```
System.out.format("%2$s %1$s", "FIT", "HANU");
```

Output:

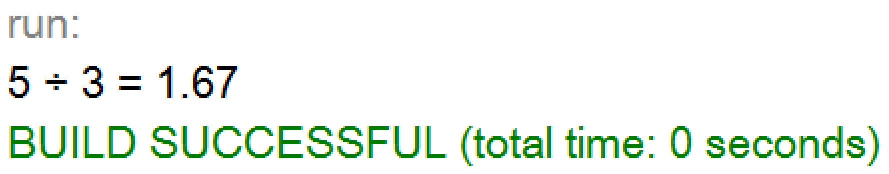
```
HANU FIT
```

The `String.format()` method

- It is similar to `System.out.format()` but returns a formatted `String` instead of printing it.
- Example:

```
String m = String.format("%d ÷ %d = %.2f", 5, 3, 1.6667);
System.out.println(m);
```

Output (NetBeans):



```
run:
5 ÷ 3 = 1.67
BUILD SUCCESSFUL (total time: 0 seconds)
```

Example 1

- Print a data table of students with ID, name, class and scores.

```java
public class Example1 {
    public static void main(String[] args) {
        int[] ids = {15, 151, 152};
        String[] names = {"Alex", "Nguyen Van A", "Michael"};
        String[] classes = {"1C18", "2C18", "3C18"};
        double[] scores = {6.5, 7, 7.5};
        System.out.println("---------------------------------------");
        for (int i = 0; i < ids.length; i++) {
            System.out.format(
                    "| %05d | %-15s|%5s |%4.1f |%n",
                    ids[i], names[i], classes[i], scores[i]
            );
        }
        System.out.println("---------------------------------------");
        System.out.close();
    }
}
```

Example 1 (OOP)

```java
public class Example1OOP {
    public static void main(String[] args) {
        Student[] students = {
                new Student(15, "Alex", "1C18", 6.5),
                new Student(151, "Nguyen Van A", "2C18", 7),
                new Student(152, "Michael", "3C18", 7.5)
        };
        System.out.println("---------------------------------------");
        for (Student std : students) {
            System.out.format(
                    "| %05d | %-15s|%5s |%4.1f |%n",
                    std.id, std.name, std.className, std.score
            );
        }
        System.out.println("---------------------------------------");
        System.out.close();
    }
}
```

Example 2

Use PrintWriter to export the data table in the previous example to a text file.

```java
import java.io.PrintWriter;
public class Example2 {
    public static void main(String[] args) throws Exception {
        int[] ids = {15, 151, 152};
        String[] names = {"Alex", "Nguyen Van A", "Michael"};
        String[] classes = {"1C18", "2C18", "3C18"};
        double[] scores = {6.5, 7, 7.5};
        PrintWriter pw = new PrintWriter("data.txt");
        pw.println("----------------------------------------------");
        for (int i = 0; i < ids.length; i++) {
            pw.format(
                    "| %05d | %-15s|%5s |%4.1f |%n",
                    ids[i], names[i], classes[i], scores[i]
            );
        }
        pw.println("----------------------------------------------");
        pw.close();
    }
}
```

Example 2 (OOP)

```java
import java.io.PrintWriter;
public class Example2b {
    public static void main(String[] args) throws Exception {
        Student[] students = {
                new Student(15, "Alex", "1C18", 6.5),
                new Student(151, "Nguyen Van A", "2C18", 7),
                new Student(152, "Michael", "3C18", 7.5)
        };
        PrintWriter pw = new PrintWriter("data.txt");
        pw.println("--------------------------------------------");
        for (Student std : students) {
            pw.printf(
                    "%-4d %-30s %-5s %4.1f%n",
                    std.id, std.name, std.className, std.score
            );
        }
        pw.println("--------------------------------------------");
        pw.close();
    }
}
```

Read files with BufferedReader

- This example uses StringBuilder instead of ordinary String.

```java
import java.io.FileReader;
import java.io.IOException;
import java.io.BufferedReader;
public class BufferedReading {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("text"));
        StringBuilder sb = new StringBuilder();
        String line = "";
        while (line != null) {
            line = br.readLine();
            sb.append(line);
        }
        System.out.println(sb.toString());
    }
}
```

java.io.FileReader

- This class provides the necessary methods for reading characters from a file.
- Methods:
  - `int` **`read()`**: return the integer representation of the next character in the file.

```java
import java.io.FileReader;
public class FileReaderDemo {
    public static void main(String[] args) throws Exception {
        FileReader fr = new FileReader("file1");
        fr.read();
        int c = fr.read();
        System.out.println(c);
    }
}
```

`java.io.BufferedReader`

- This class provides the methods for reading text lines from a `FileReader` object.
- It performs better than `Scanner` for large files.
- Methods:
  - `String` **`readLine()`**: return the next line from the file as a String. Return `null` if it reaches the end of the file.

java.io.BufferedWriter

- Contains methods necessary to efficiently write simple Strings or char arrays to a file.
- Syntax to create a BufferedWriter object:

File name

```
FileWriter fw = new FileWriter("data.txt");
BufferedWriter out = new BufferedWriter(fw);
```

- **Attention:** If the file exists, its content will be erased.
- The statement to create a `BufferedWriter` object may throw `IOException` and needs to be surrounded with `try…catch` (or declared to be thrown)

`java.io.BufferedWriter`

- Methods in the BufferedWriter class:
  - `void` **`write()`**: write a char, a char array or a String to the output file. Throws IOException.
  - `void` **`newLine()`**: write a newline character to the output file. Throws IOException.
  - `void` **`flush()`**: transfer the content from the buffer to the file. Throws IOException.
    - Without flushing, texts are not written to the file.
  - `void` **`close()`**: flush and close the connection to the file, as well as release any system resources being used by the BufferedWriter object.

Example 2

Use BufferedWriter to create a text file containing the name and age entered by a user.

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class example3 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BufferedWriter out = null;
        try {
            FileWriter fw = new FileWriter("data.txt");
            out = new BufferedWriter(fw);
        } catch (IOException ex) {
            System.out.println("Error!");
            System.exit(1);
        }
        System.out.print("Please enter your name: ");
        String name = sc.nextLine();
        System.out.print("Please enter your age: ");
        int age = sc.nextInt();
        try {
            out.write("Name: " + name);
            out.newLine();
            out.write("Age: " + age);
            out.close();
        } catch (IOException ex) {
            System.out.println("Write error!");
            System.exit(2);
        }
    }
}
```

Problem: append content to a file

- How to open a text file for writing while not erasing the file's content?
- Solution with `FileWriter`:

```
FileWriter fw = new FileWriter("data.txt", true);
fw.write("Hello!");
```

- Solution with `PrintWriter`:

```
FileWriter fw = new FileWriter("data.txt", true);
PrintWriter pw = new PrintWriter(fw);
pw.println("Hello");
```

Data serialization

- What is it?
  - **Serialization:** The conversion of an object into a sequence of `bytes`.
  - **De-serialization:** The conversion from a sequence of `bytes` to an object (opposite of Serialization).
- The sequence (array) of bytes can be saved to a file (to be loaded to the program later).

Data serialization

- Why?
  - Save program data to files.
  - Send data to other systems or through network without losing data or data integrity.
- The programmer does not have to manually handle how data are organized in the file.
  - It is convenient but it does not give the programmer full control of the data file.
  - Serialized data must be deserialized using the same programming language.

Data serialization

- The FileOutputStream class:
  - Contains the necessary methods to write single bytes or `byte` arrays to a `File`.
- Methods:
  - `void` **`write()`**: writes one byte or a byte array to the output file.
  - `void` **`close()`**: ends connection to the file and releases system resources.

```
          File f = new File("data.bin");
FileOutputStream os = new FileOutputStream (f);
```

Data serialization

- The ObjectOutputStream class:
  - Contains the necessary methods to serialize and write Java primitive values and objects to an output stream (e.g. `FileOutputStream`).
- Methods:
  - `void` **`writeObject()`**: serializes and writes an object (including arrays) to the output stream.
  - `void` **`writeChars()`**: serializes and writes a String as a `char` array to the output stream.
  - `void` **`writeInt()`**: serializes and writes a 32-bit `int` to the output stream.

Data serialization

- Methods:
  - `void` **`write()`**: writes a `byte` to the output.
  - `void` **`writeFloat()`**: serializes and writes a 32-bit `float` to the output stream.
  - `void` **`writeDouble()`**: serializes and writes a 64-bit `double` to the output stream.
  - `void` **`writeLong()`**: serializes and writes a 64-bit `long` to the output stream.
  - `void` **`writeBoolean()`**: serializes and writes a `boolean` value to the output stream.

Example 3

Saving products data in a menu-driven program to a file.

```java
import java.io.File;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.util.Scanner;

public class example4 {
    public static void main(String[] args) throws Exception {
        String[] names = new String[3];
        double[] prices = new double[3];
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.println("1. Add products\n2. Display products");
            System.out.println("3. Save products\n4. Load products");
            System.out.print("5. Quit\nYour choice:");
            int n = sc.nextInt();
            if (n == 1) {
                names = new String[]{"Dell", "HP", "Lenovo"};
                prices = new double[]{305.1, 402, 292.2};
            } else if (n == 2) {
                for (int i = 0; i < names.length; i++) {
                    System.out.format("%20s | %10.2f |%n", names[i], prices[i]);
                }
            } else if (n == 3) {
                File f = new File("data.bin");
                FileOutputStream fos = new FileOutputStream(f);
                ObjectOutputStream oos = new ObjectOutputStream(fos);
                oos.writeObject(names);
                oos.writeObject(prices);
                oos.close();
            } else if (n == 5) {
                System.out.println("Goodbye!");
                break;
            } else {
                System.out.println("Invalid choice!");
            }
        }
    }
}
```

Data de-serialization

- The FileInputStream class:
  - Contains the necessary methods to read bytes from a file.
- Methods:
  - `void` **`read()`**: reads one byte or a byte array from the input file.
  - `void` **`skip(n)`**: discards n bytes of data.
  - `void` **`close()`**: ends connection to the file…

```
FileInputStream os = new FileInputStream("data.bin");
```

- **Attention:** the statement to create this object may throw a `FileNotFoundException`.

Data de-serialization

- The ObjectInputStream class:
  - Contains the necessary methods to read and de-serialize primitive values and objects from an input stream (e.g. `FileInputStream`).
- Methods:
  - `Object` **`readObject()`**: reads and de-serializes an object (including arrays) from the input stream.
  - `int` **`read()`**: reads one `byte` or a `byte` array.
  - `int` **`readInt()`**: reads a 32-bit integer.

Data de-serialization

- Methods:
  - `float` **`readFloat()`**: reads a 32-bit `float` from the input stream.
  - `double` **`readDouble()`**: reads a 64-bit `double` from the input stream.
  - `long` **`readLong()`**: reads a 64-bit `long` from the input stream.
  - `boolean` **`readBoolean()`**: reads a `boolean` value from the input stream.
  - `void` **`close()`**: closes the file and release system resources.

Example 4

Loading products data from file.

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.Scanner;

public class example5 {
    public static void main(String[] args) throws Exception {
        String[] names = new String[3];
        double[] prices = new double[3];
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.println("1. Add products\n2. Display products");
            System.out.println("3. Save products\n4. Load products");
            System.out.print("5. Quit\nYour choice:");
            int n = sc.nextInt();
            if (n == 1) {
                names = new String[]{"Dell", "HP", "Lenovo"};
                prices = new double[]{305.1, 402, 292.2};
            } else if (n == 2) {
                for (int i = 0; i < names.length; i++) {
                    System.out.format("%20s | %10.2f |%n", names[i], prices[i]);
                }
            } else if (n == 4) {
                File f = new File("data.bin");
                FileInputStream fis = new FileInputStream(f);
                ObjectInputStream ois = new ObjectInputStream(fis);
                names = (String[]) ois.readObject();
                prices = (double[]) ois.readObject();
                ois.close();
            } else if (n == 5) {
                System.out.println("Goodbye!");
                break;
            } else {
                System.out.println("Invalid choice!");
            }
        }
    }
}
```