CSE3040 Java Language

Lecture 12: File Input Output

Dept. of Computer Engineering,
Sogang University

class Object: method equals

- The method equals checks whether two objects are the same.
  - returns true if two objects are the same objects.
  - return false if two objects are different or the compared object is null.

- The following code produces false, because the two objects are different.

```
Employee empl1 = new Employee("John", 50000);
Employee empl2 = new Employee("John", 50000);
System.out.println(empl1.equals(empl2));
```

- The method equals is often overridden to compare the "equality" of the two objects.
  - The definition of equality is up to the class designer.
  - For example, the String class overrides the method equals which compares the content of the two String objects.
  - The following code produces true.

```
String s1 = new String("hello");
String s2 = new String("hello");
System.out.println(s1.equals(s2));
```

class Object: method equals

- We can override method equals in class Employee.
  - The equals method returns true if two Employees have the same name and salary.

```
class Employee {
    private String name;
    protected int salary;
    public Employee(String name, int salary) { this.name = name; this.salary = salary; }
    public String toString() { return getClass().getName() + "[name=" + name + ",salary=" + salary + "]"; }
    public boolean equals(Object otherObject) {
        if(this == otherObject) return true;
        if(otherObject == null) return false;
        if(getClass() != otherObject.getClass()) return false;
        Employee other = (Employee)otherObject;
        return (this.name == other.name && this.salary == other.salary);
    }
    public String getName() { return this.name; }
    public int getSalary() { return this.salary; }
}
```

```
Employee empl1 = new Employee("John", 50000);
Employee empl2 = new Employee("John", 50000);
System.out.println(empl1.equals(empl2));
```

class Object: method equals

- Notes when overriding method equals
  - First check whether the two objects are the same object and return true if so.
  - Then check if the otherObject is null and return false if so.
  - Then check whether the two compared objects are of the same class. Otherwise, return false.
    - You can implement equals to work with different classes, but it is not a typical use.
  - Cast the otherObject to the class.
  - Compare instance variables. Be careful what "==" means for each instance variable.
    - Sometimes you may need to use equals() method to compare the two instance variables.

```
class Employee {
    private String name;
    protected int salary;
    public Employee(String name, int salary) { this.name = name; this.salary = salary; }
    public String toString() { return getClass().getName() + "[name=" + name + ",salary=" + salary + "]"; }
    public boolean equals(Object otherObject) {
        if(this == otherObject) return true;
        if(otherObject == null) return false;
        if(getClass() != otherObject.getClass()) return false;
        Employee other = (Employee)otherObject;
        return (this.name == other.name && this.salary == other.salary);
    }
    public String getName() { return this.name; }
    public int getSalary() { return this.salary; }
}
```

class Object: method hashCode

- A hash code is an integer value that is calculated from the object.
  - The hash function in Java is defined so that if x and y are two different objects, they have different hash code with high probability. (Not 100% due to possibility of hash collision.)

- Assume we did not override method equals yet.

```
class Employee {
    private String name;
    protected int salary;
    public Employee(String name, int salary) { this.name = name; this.salary = salary; }
    public String toString() { return getClass().getName() + "[name=" + name + ",salary=" + salary + "]"; }
    public String getName() { return this.name; }
    public int getSalary() { return this.salary; }
}
```

- Hash code of empl1 and empl2 are different, because they are different objects.

```
Employee empl1 = new Employee("John", 50000);
Employee empl2 = new Employee("John", 50000);
System.out.println(empl1.hashCode() + " " + empl2.hashCode());
```

class Object: method hashCode

- If you override method equals, then you should also override method hashCode to match the definition of equality.
  - Otherwise, problems occurs when using data structures such as HashMap.

- For example, class String overrides both methods equals and hashCode so that if two Strings are equal, their hashCode are the same.

```
String s1 = new String("hello");
String s2 = new String("hello");
System.out.println(s1.equals(s2));
System.out.println(s1.hashCode() + " " + s2.hashCode());
```

- The hashCode method of class String calculates hash code of a String object as follows.
  - If the two objects represent the same string, then their hash codes are the same.

```
int hash = 0;
for(int i = 0; i < length(); i++)
    hash = 31 * hash + charAt(i);
```

File output: FileOutputStream

- Frequently, we need to write the output of a program to a file.
- We can use the library class FileOutputStream to achieve the goal.

- The following example creates a file names "out.txt" under directory "src/cse3040".
  - In eclipse, the base directory would be the project folder.
  - The file contains the string "hello world".

```
package cse3040;

import java.io.FileOutputStream;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           FileOutputStream output = new FileOutputStream("src/cse3040/out.txt");
           String str = "hello world";
           byte[] bytes = str.getBytes();
           output.write(bytes);
           output.close();
     }
}
```

File output: FileOutputStream

- Two library classes should be imported.
  - FileOutputStream and IOException

- The main method throws IOException.
  - We will learn about exceptions later.

```
package cse3040;

import java.io.FileOutputStream;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           FileOutputStream output = new FileOutputStream("src/cse3040/out.txt");
           String str = "hello world";
           byte[] bytes = str.getBytes();
           output.write(bytes);
           output.close();
     }
}
```

File output: FileOutputStream

- First, we create a FileOutputStream object.
- The argument to the constructor is the file name including the path.
  - The path could be either absolute or relative.
  - Absolute path example: "c:/out.txt"
  - Relative path example: "src/cse3040/out.txt"

```
package cse3040;

import java.io.FileOutputStream;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           FileOutputStream output = new FileOutputStream("src/cse3040/out.txt");
           String str = "hello world";
           byte[] bytes = str.getBytes();
           output.write(bytes);
           output.close();
     }
}
```

File output: FileOutputStream

- Once we create a FileOutputStream object, we can write to the file using the write method.
- The write method takes a byte array as its argument.

```
void write(byte buf[]) throws IOException
```

- We can convert a String to a byte array using getBytes, an instance method of class String.

```
package cse3040;

import java.io.FileOutputStream;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           FileOutputStream output = new FileOutputStream("src/cse3040/out.txt");
           String str = "hello world";
           byte[] bytes = str.getBytes();
           output.write(bytes);
           output.close();
     }
}
```

File output: FileOutputStream

- Once we are done with the file, it is good to close the output file stream using method close.
- A file stream needs to be closed before it can be opened by someone else.

```
package cse3040;

import java.io.FileOutputStream;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           FileOutputStream output = new FileOutputStream("src/cse3040/out.txt");
           String str = "hello world";
           byte[] bytes = str.getBytes();
           output.write(bytes);
           output.close();
     }
}
```

File output: FileWriter

- Since FileOutputStream writes bytes to files, we cannot directly write Strings to the file. We need to convert the Strings to byte arrays.
- If we use FileWriter instead of FileOutputStream, we can directly write Strings.

```
package cse3040;

import java.io.FileWriter;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           FileWriter fw = new FileWriter("src/cse3040/out.txt");
           for(int i=0; i<10; i++) {
              String str = "This is line number " + i + "\r\n";
              fw.write(str);
           }
           fw.close();
     }
}
```

- FileOutputStream is meant for writing streams of raw bytes, whereas FileWriter is meant for writing stream of characters.

File output: PrintWriter

- PrintWriter is a more convenient class that can be used for file output.
- PrintWriter lets you use methods like print, println, and printf.
  - Similar to System.out.
  - Using println, we do not need to insert "\r\n" at the end of the string.

```
package cse3040;

import java.io.PrintWriter;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           PrintWriter pw = new PrintWriter("src/cse3040/out.txt");
           for(int i=0; i<10; i++) {
              String str = "This is line number " + i + ".";
              pw.println(str);
           }
           pw.close();
     }
}
```

File output: appending to a file

- When you create a FileWriter object to write to a file, a new empty file is opened.
  - If the file already existed, it will be erased.
- If we want to append to a file, we need to create a FileWriter with the second argument which indicates the file open mode.

```
package cse3040;

import java.io.FileWriter;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           FileWriter fw = new FileWriter("src/cse3040/out.txt");
           for(int i=0; i<10; i++) fw.write("This is line number " + i + ".\r\n");
           fw.close();

           FileWriter fw2 = new FileWriter("src/cse3040/out.txt", true);
           for(int i=10; i<20; i++) fw2.write("This is line number " + i + ".\r\n");
           fw2.close();
     }
}
```

File output: appending to a file

- PrintWriter constructor does not have the append parameter, but PrintWriter can be created using a FileWriter.
- We can implement the append mode using PrintWriter as the following code.

```
package cse3040;

import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;

public class Lecture {
     public static void main(String[] args) throws IOException {
           PrintWriter pw = new PrintWriter("src/cse3040/out.txt");
           for(int i=0; i<10; i++) pw.println("This is line number " + i + ".");
           pw.close();

           PrintWriter pw2 = new PrintWriter(new FileWriter("src/cse3040/out.txt", true));
           for(int i=10; i<20; i++) pw2.println("This is line number " + i + ".");
           pw2.close();
     }
}
```

File input: FileInputStream

- The basic approach to reading data from a file is to use FileInputStream.
- We create a new FileInputStream object with the filename including the path.
- Then, we use method read to read the bytes from the file into a byte array.
- In order to print the data, we convert the data into a String.
- Since the byte array size is 1024, the read method reads 1024 bytes (or until the end of file).
  - The read method returns the number of bytes. If EOF is reached, the method returns -1.

```
package cse3040;

import java.io.FileInputStream;
import java.io.IOException;

public class Lecture {
    public static void main(String[] args) throws IOException {
        byte[] b = new byte[1024];
        FileInputStream input = new FileInputStream("src/cse3040/out.txt");
        input.read(b);
        System.out.println(new String(b));
        input.close();
    }
}
```

File input: FileInputStream

- If the file is larger than the byte array, only the number of bytes equal to the array size will be read from one call to method read.
- In this case, we can use while loop to read the whole file, which continues until the read method returns -1.

```
package cse3040;

import java.io.FileInputStream;
import java.io.IOException;

public class Lecture {
    public static void main(String[] args) throws IOException {
        byte[] b = new byte[100];
        FileInputStream input = new FileInputStream("src/cse3040/out.txt");
        while(input.read(b) != -1) {
            System.out.println(new String(b));
        }
        input.close();
    }
}
```

- This code still has problem. Can you identify and solve the problem?

File input: FileReader and BufferedReader

- A more convenient way of reading from a file is to use FileReader and BufferedReader to read a line at once.
- FileInputStream is meant for reading raw bytes, whereas FileReader is meant for reading characters.
- BufferedReader provides method readLine which reads one line from the file.

```
package cse3040;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Lecture {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("src/cse3040/out.txt"));
        while(true) {
            String line = br.readLine();
            if(line == null) break;
            System.out.println(line);
        }
        br.close();
    }
}
```

Programming Lab #12

12-01. equals()

- Override method equals() so that the main method produces correct results.
  - If the two employees have the same name and the same salary, the equals() method should return true.

```
class Employee {
    private String name;
    protected int salary;
    public Employee(String name, int salary) { this.name = name; this.salary = salary; }
    public String getName() { return this.name; }
    public int getSalary() { return this.salary; }
}

public class Ex12_01 {
    public static void main(String[] args) {
        Employee e1 = new Employee(new String("John"), 50000);
        Employee e2 = new Employee(new String("John"), 50000);
        System.out.println(e1 == e2);
        System.out.println(e1.equals(e2));
    }
}
```

```
false
true
```

12-02. hashCode()

- Override method hashCode() so that it matches the semantics of equals()
  - If the two employees have the same name and the same salary, their hash codes should be the same. Otherwise, their hash code should be different with high probability.

```
class Employee {
    private String name;
    protected int salary;
    public Employee(String name, int salary) { this.name = name; this.salary = salary; }
    public String getName() { return this.name; }
    public int getSalary() { return this.salary; }
}

public class Ex12_02 {
    public static void main(String[] args) {
        Employee e1 = new Employee(new String("John"), 50000);
        Employee e2 = new Employee(new String("John"), 50000);
        System.out.println(e1.equals(e2));
        System.out.println(e1.hashCode() == e2.hashCode());
    }
}
```

```
true
true
```

12-03. Writing to a File

- Write the following text to a file.

```
Hello, my name is Harry Potter.
I am a student in the Java Language class.
I am trying to write this text to a file.
Thank you.
```

- First, use FileOutputStream class to write this text to a file named "myfile1.txt".
- Second, use FileWriter class to write this text to a file named "myfile2.txt".
- Third, use PrintWriter class to write this text to a file named "myfile3.txt".
- Check if all files were written correctly.

```
public class Ex12_03 {
    public static void main(String[] args) {
        String str = "Hello, my name is Harry Potter.\nI am a student in the Java Language class.\nI am trying
     to write this text to a file.\nThank you.\n";

        // write str to the files.
    }
}
```

12-04. Reading from a File

- Suppose we have a file containing the following text. Read the file and print the text on the screen.

```
Hello, my name is Harry Potter.
I am a student in the Java Language class.
I am trying to write this text to a file.
Thank you.
```

- First, use FileInputStream class to read the file.
- Second, use FileReader and BufferedReader to read the file.
- Assume the file exists in the given path.

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr