Java Programming

CPT111 – Lecture 11

Erick Purwanto

Xi'an Jiaotong-Liverpool University

西交利物浦大学

CPT111 Java Programming

Lecture 11

File I/O and Lists

Welcome!

- Welcome to Lecture 11 !
- Last week, we have learned about OOP Principles and Exception
  - encapsulation, inheritance, and polymorphism
  - throwing and handling/catching exception
- In this lecture we are going to learn about
  - File I/O
    - learning Java File I/O mechanism
    - writing into a file
    - reading from a file
  - Lists
    - creating, accessing and manipulating a List as ArrayList and LinkedList
    - using list in method and in a class as an instance variable

Part 1: File I/O

- In the first part of lecture, we will learn about creating, writing to and reading from a file

File and File Class

- Data stored in a variable will be lost when a program terminates
- We need to save the data into a file, in a directory
  - the file is then stored into a storage device
- Absolute file name – depends on the OS – for example in Windows:
  - C:\Users\erick.purwanto\Documents\NetBeansProjects\Lecture11Demo\data\text1.txt
- Relative file name – relative to the current working directory
  - data\text1.txt
- In Java, we use a `File` object to represent a file
  - `File file = new File("C:\\Users\\erick.Purwanto\\Documents\\NetBeansProjects\\Lecture11Demo\\data\\text1.txt");`

File Object

- Since OS may vary, **do not** use absolute file name when creating `File` objects
- Use relative file name in Java (with forward slash) :
  ```
  File file = new File("data/text1.txt");
  ```
- Note that creating a File object **does not** actually create a file on the computer!

File Methods

- `File` class contains method to obtain the property of a file/directory

```
File file = new File("data/text1.txt");
file.exists();
file.isFile();
file.isDirectory();
file.isHidden();
file.length();
file.canRead();
file.canWrite();
file.getAbsolutePath();
```

in bytes

  - and to rename and delete a file/directory

    `delete(), renameTo(File), mkdir()`

however, to read/write into the file we need another class

Creating a new file using PrintWriter

- `PrintWriter` class is used to create a file and write data into a text file

```
File newFile = new File("data/text2.txt");
if (newFile.exists()) {
    System.out.println("File already exists!");
    System.exit(0);
}

try {
    PrintWriter output = new PrintWriter(newFile);
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
}
```

checking for existence first so that not overwritting

if so, print message and exit the program

otherwise, the new file will be created

Java forces us to handle this exception

Writing data into an existing file using PrintWriter

- `PrintWriter` class is used to create a file and write data into a text file

```
File file = new File("data/text1.txt");
try {
    PrintWriter output = new PrintWriter(file);
    output.println("Tanjiro");
    output.println("Zenitsu");
    output.println("Inosuke");
    output.close();
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
}
```

assuming file exists

then write similar to `System.out.println`

`close()` method must be used to write the data and close the file

In-Class Quiz 11.1

- What happens if we run the program again?

```
File file = new File("data/text1.txt");
try {
    PrintWriter output = new PrintWriter(file);
    output.println("Tanjiro");
    output.println("Zenitsu");
    output.println("Inosuke");
    output.close();
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
}
```

  - 3 more lines added
  - no more lines added
  - there is an error message printed

Improve efficiency using BufferedWriter

- `BufferedWriter` stores the strings in memory before writing into the file

```
try {
    FileWriter file = new FileWriter("data/text1.txt");
    BufferedWriter buffer = new BufferedWriter(file);
    buffer.write("Tanjiro");
    buffer.newLine();
    buffer.write("Zenitsu");
    buffer.newLine();
    buffer.flush();
    buffer.write("Inosuke");
    buffer.newLine();
    buffer.close();
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
}
```

use `flush()` method to force writing buffer content into file

otherwise write until buffer is full or closed

Reading text data using Scanner

- To read from a text file, create a Scanner for the file, read line-by-line:

```
File file = new File("data/text1.txt");
try {
    Scanner input = new Scanner(file);
    while (input.hasNextLine()) {
        String name = input.nextLine();
        Swordsman swordsman = new Swordsman(name);
        System.out.println(swordsman);
    }
    input.close();
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
}
```

Reading csv data using Scanner

- To read from a csv (comma-separated values) file, create a Scanner for the file, extract the values by first splitting them:

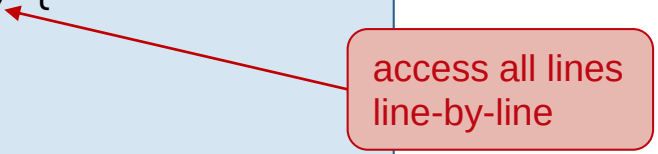
```
File file = new File("data/demonslayerdata.csv");
try {
    Scanner input = new Scanner(file);
    while (input.hasNextLine()) {
        String line = input.nextLine();
        String[] values = line.split(",");
        String name = values[0];
        int numDemonsKilled = Integer.parseInt(values[1]);
        Swordsman swordsman = new Swordsman(name, numDemonsKilled);
        System.out.println(swordsman);
    }
    input.close();
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
}
```

extract data from a line, line-by-line

Improve efficiency using BufferedReader

- BufferedReader reads once from file into memory to accessed later

```
File file = new File("data/text1.txt");
try {
    FileReader fileReader = new FileReader(file);
    BufferedReader reader = new BufferedReader(fileReader);
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
    reader.close();
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
}
```



Part 2: List, ArrayList and LinkedList

- In the second part of lecture, we will learn about lists
  - how to create a List
  - how to add and access its data
  - how to insert and remove its data
  - how to use it for problem-solving

Array Deficiencies

- Say you want to create an array of Swordsman objects
- What if you don't know how many objects to store beforehand?
  - arrays need to declared with a size!
- What if you want to store any numbers of objects?
  - arrays cannot grow arbitrarily!

Java Collections and List

- Java provides a number of more powerful and flexible tools for managing collections of objects: the **Java Collections Framework**
- It includes List (this week) and Set, Map (next week)
  - *both would be very useful for your CW3*

- A List contains an ***ordered*** collection of zero or more objects, where the same object might appear multiple times
  - we can add and remove items to and from the List, which will grow and shrink to accommodate its contents

Creating Lists (1)

- Java helps us distinguish between the *specification* of a type – *what does it do?* – and the *implementation – what is the code?*
- List, Set and Map are all *interfaces*: they define **how** these respective types work, but they **don't** provide implementation code
  - *will learn more in CPT204 Advanced OOP later*
- One advantage is that we, the users of these types, get to choose different implementations in different situations
- To create Lists, specify the object/generic type inside diamond <> operator:
  - `List<String> list1 = new ArrayList<String>();`
  - `List<String> list2 = new LinkedList<String>();`

Creating Lists (2)

- If the generic type parameters are the **same** on the left and right, Java can infer what's going on and save us some typing:
  - `List<String> list1 = new ArrayList<>();`
  - `List<String> list2 = new LinkedList<>();`

- `ArrayList` and `LinkedList` are two implementations of List
  - both provide all the operations of List, and those operations must work as described in the documentation for List
  - `list1` and `list2` will behave the same way,
    i.e. if we swap which one used `ArrayList` vs `LinkedList`,
    our code will **not** break

In-Class Quiz 11.2

- What do you think the resulting list would be after these list operations?

```
List<String> list = new ArrayList<>();
list.add("A");
list.add("B");
list.add("C");
list.add(1, "X");
list.set(2, "Y");
System.out.println(list);
```

  - [A X Y C]
  - [A B X Y C]
  - [A X B Y C]
  - [A X B C Y]

List Operations

- To declare a List variable, for example of integers, with an ArrayList:
  - `List<Integer> list = new ArrayList<>();`
- Some common operations:
  - adding `list.add(5)`
  - inserting `list.add(1, 3)`
  - assigning `list.set(2, 7)`
  - retrieving `list.get(2)`
  - length `list.size()`
  - remove `list.remove(1)`
  - remove all `list.clear()`

wrapper class for primitives

adding to the back of list

replace the value at that index

- Create a list from an array

```
List<Integer> list = Arrays.asList(10, 20, 30)
```

List and ArrayList

```
List<Integer> list = new ArrayList<Integer>();
```

- `List` is an **interface**, a type that can't be constructed directly with `new`, but that instead *specifies the operations* that a List must provide
  - `ArrayList` is a **class**, a concrete type that provides *implementations of those operations*
  - `ArrayList` isn't the only implementation of the `List` type, though it's the most commonly used one
    - `LinkedList` is another implementation

Object type / generic type parameter

```
List<Integer> list = new ArrayList<Integer>();
```

- We wrote `List<Integer>` instead of `List<int>`
  Lists only know how to deal with *object types*, not *primitive types*
  - In Java, each of the primitive types (lowercase, abbreviated) has an equivalent object type (capitalized, fully spelled out)
  - Java requires us to use these object type equivalents when we **parameterize** a type with diamond operator **< >** angle brackets

ArrayList vs LinkedList

- Two possible implementations of a List:

	`ArrayList`	`LinkedList`
Actual data structure	Resizing Array	Doubly Linked List
Storing and accessing	Fast	Slow
Inserting and deleting	Slow	Fast

you will actually create both yourself from scratch in CPT204!

Iterate through List

- You can use the **enhanced for loops** to iterate through a list
  - for example:

```
for (int x : list) {
    System.out.print(x + " ");
}
```

- Demo examples:
  - Write a function `findMaxInt` to find the maximum element in an input `List` of integers using the enhanced for loop
  - Write a function `readFile` that takes a file name and returns a `List` of `Swordsman` objects generated from the file
  - Write a `DemonSlayerCorps` class that uses a `List` to store a troops of `Swordsman` objects

FindMaxInt

- Write a function `findMaxInt` to find the maximum element in an input `List` of integers using the enhanced for loop

```
public static int findMaxInt(List<Integer> list) {
    int max = list.get(0);
    for (int num : list) {
        if (num > max) {
            max = num;
        }
    }
    return max;
}
```

Reading from a file and returning List of Swordsman

- Write a function `readFile()` that takes a file name and returns a `List` of `Swordsman` objects generated from the file!
  - the file contains names of the `Swordsman`

```
public static List<Swordsman> readFile(String fileName) {
    File file = new File(fileName);
    List<Swordsman> list = new ArrayList<>();
    try {
        Scanner input = new Scanner(file);
        while (input.hasNextLine()) {
            String name = input.nextLine();
            Swordsman swordsman = new Swordsman(name);
            list.add(swordsman);
        }
        input.close();
    } catch (IOException ioe) {
        System.out.println(ioe.getMessage());
    }
    return list;
}
```

DemonSlayerCorps Class

- Write a `DemonSlayerCorps` class that uses a `List` to store a troops of `Swordsman` objects!
  - complete it with constructors and instance methods such as `heal()` that heals the whole troops

Thank you for your attention !

- In this lecture, you have learned:
  - how to create, read and write files
  - how to create, access and manipulate a List implementation with ArrayList and LinkedList

- Please continue to Lab 11 to complete Lab Tasks, and then solve
  - Exercise and CW1 of Week #11