Lecture 13
Project Discussion
Working with large datasets

Atul Prakash

IMDB movies list

- Contains over 1 million lines.
- We want to run queries on it, such as finding movies that contain a keyword or movies that were released in a given year
- Takes around 30 seconds on my machine to read the data in and do basic parsing on it
- Difficult to know if the answers to queries are right

First Step

- Produce a smaller dataset for testing
- Write a Java program to read in the movies.list file and write out (after stripping out the irrelevant lines at the beginning), say, first 100 movies.
- For a better sample, you could also do a random subset of lines (e.g., output a line with probability 0.0001).
- **Hint:** The lines with movies contain a tab character. Others do not.

I/O

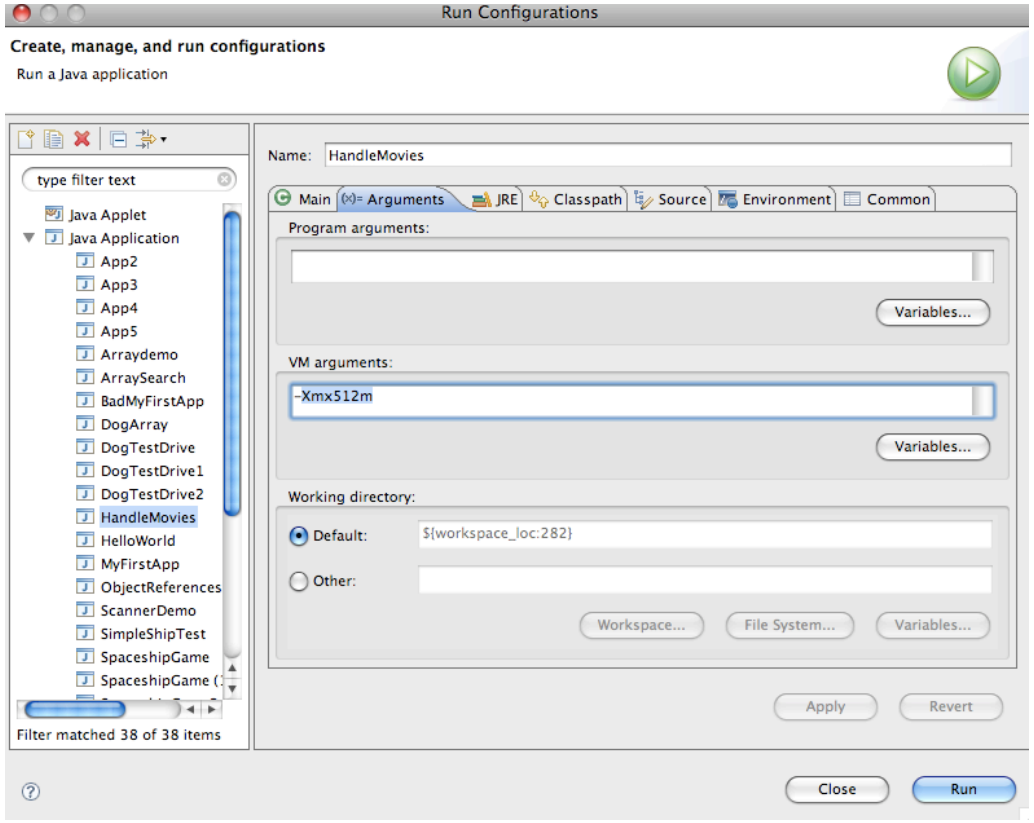
- You can use the Scanner class to read in the file line by line.
- For writing out to a file, you create a file reference and make a PrinterWriter object from it. Now you can write to it just like to System.out.

```
File out = new File("outputfile.txt");
PrintWriter fop = new PrintWriter(out);
fop.println("output string");
 …
fop.close();
```

Note: Also, need to declare or catch FileNotFoundException

Java Memory

- Java allows up to 32 MB of memory to programs by default.
- This may not be enough.
- You can increase the limit. Give "-Xmx512m" option to java to increase memory to 512MB.



Measuring Performance

- How much time does your program take to do some work, e.g., reading a file in?
- Use System.getCurrentTimeMillis() to retrieve the current time in milliseconds since Jan. 1, 1970.

```
long start = System.currentTimeMillis();

… your code that you want to measure …

long endread = System.currentTimeMillis();
System.out.println("Read time: " + (endread-start)/1000.0 + " s");
```

Anecdotes: History of Computer Time

- Jan. 1, 1970 reference point comes from Unix operating system, which originated around that time.

- Y2000 bug: many computer programs failed in 2000 because year was stored as a 2-digit number in many programs

- Year 2038 problem or "Unix millenium bug": many computer programs will fail in 2038 because they use 32-bit integers to represent time. If time is measured in seconds, time will rollover to 0 on Jan. 19th, 2038.

http://en.wikipedia.org/wiki/Year_2038_problem

Good Strategies for Measuring Time

- Computers are fast, but the clock resolution may be at the interval of 1/60 sec or millisecond
- Strategy: Measure time for n operations and divide by n, rather than averaging t1, t2, ..., tn, if ti are too small as compared to clock resolution

Processing User Queries

- File I/O is very slow (around 30 seconds on my computer to read in the entire movies.list).
- Better to read the file only once, store it in memory and then answer queries using the in-memory data

```
Read movies.list
Store data in a list of movies
while (true) {
    read search query
    (if quit command) break;
    search through the list and
    output answer
}
```

Lists

- We showed how to build our own lists using arrays earlier.
- Java provides a standard implementation as well:
  - ArrayLists

ArrayList

- ArrayList in Java: array-based lists.
- Suppose you have a class called Movie that contains information about a movie (e.g., title, year)

```
ArrayList<Movie> a = new ArrayList<Movie>();
```

creates a reference to a list of objects a, which must all be of class "Movie".

Generics in Java

- This notion of making a class parameterized by type is called "generics". The classes are "generic" for multiple types.
- Rather than defining different list classes for different types of objects, one can use generic classes.

Defining your own generic list

- See attached file MyGenericList.java and its test.

```
public class MyGenericList<T> { // list of values of type T
    private T[] data; // list itself. null values at the end
    int capacity;  // maximum capacity of the list
    int size;  // current size of the list
    static final int DEFAULT_CAPACITY = 100;

 public T getElement(int index) {
     // find the element at given index
     return getData()[index];
 }

public void addElementv1(T a) {
    // add an object a to the end of the list
        getData()[size] = a;
        size++;
}
```

ArrayList<T> methods

- Lookup Java Documentation. Key methods:
  - add(T element): add an element to the list
  - T get(int index): get an element
  - int size(): get the size of the list

Example

```
import java.util.ArrayList;

public class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<Integer> intarray;
        intarray = new ArrayList<Integer>();
        intarray.add(4);
        intarray.add(10);
        System.out.println("size of the list = " + intarray.size());
        for (int i = 0; i < intarray.size(); i++) {
            System.out.println(i + "th element: " + intarray.get(i));
        }
    }
}
```

```
size of the list = 2
0th element: 4
1th element: 10
```