

Weekend Section -1

CSCI E97

Date 4th February 2017

Moshtaq Ahmed (ALM)

Agenda

- About the course
- HW 1 Grade sheet
- Warm up JAVA
- Design patterns



About this course

- This is not a programming course.
- “Design first” is the motto for this course.
- Except first assignment, all others are on software design.
- First assignment will test your Java skills.
- Keep it simple – don’t over do the implementation.
- Read the text book.

Functional Review

Criteria	Score	Comments
Code and documentation are packaged per specification	5	Please zip your code exactly specified in the assignment document. Please check before submitting.
Code compiles with the command <code>javac cscie97/asn1/knowledge/engine/ *.java cscie97/asn1/test/*.java</code>	5	Please make sure your code compiles and runs. If you have added any additional JAR, please make sure if you have included them in the submission. Please make a note, if you have done anything more than expected.
Code runs “out-of-the-box” with the correct java command and the two input files supplied with assignment. {The student might have an alternate command line for this}	5	Please make sure CLI supports two inputs file as params and it works out of the box. Any variations of your implementation, please don't forget to mention in your ReadMe file.
Additional features work as documented (e.g., student supplies additional input files that demonstrate error handling or other working features of program)	5	Please include additional inputs; e.g. error inputs, empty inputs, wrong queries. Error may occur; show the error in the command line. Java exceptions are not expected.
Other Comments		Any other comments on the design. Your idea and thoughts on the given design are expected.

Source Code Review

Criteria	Score	Comments
Written documentation provided describing any variations from design with justifications, and feedback on the Knowledge Graph Design.	5	Please submit a document if you have done any different implementation. A design can be implemented in many different ways.
General organization of code, use of packages as prescribed. Code is well commented	5	Please refactor your code before submission. Make two or three packages: knowledgeEngine , exceptions and tests. Please note too many packages can be complex to review. Please write code with comments. You can generate Javadoc (html format); it will give us a quick view of your code.
Appropriate use of exception handling (both on raising exceptions and catching exceptions)	5	Please consider raising and catching exceptions. Some cases you can even avoid raising exceptions. For example, if there is an empty line, you can just move to next line rather raising an exception.
General quality of implementation (clarity, readability, commenting)	5	Please check your code carefully. We must be able to read and understand your code.
Other comments (e.g., what improvements could the student make)		This is bonus score. If you have done something exceptional, we will appreciate it.



Code and documentation

Name	Date modified	Type	Size
cscie97	1/29/2017 12:17 AM	File folder	
inputTriples.nt	1/29/2017 12:17 AM	NT File	1 KB
run.sh	1/29/2017 12:17 AM	Shell Script	1 KB
sampleQuery.nt	1/29/2017 12:17 AM	NT File	1 KB

run.sh

```
javac cscie97/asn1/knowledge/engine/*.java
cscie97/asn1/test/*.java java cscie97.asn1.test.TestDriver
./inputTriples.nt ./sampleQuery.nt
```

By unzipping your code, we should find your input params in the folder.














- If possible, you add a run.sh for mac/linux or run.bat file for windows. (This is optional for lazy people)

Code and Documentation

Name	Date modified	Type	Size
cscie97	9/24/2015 7:45 PM	File folder	
doc	9/24/2015 7:45 PM	File folder	
testfiles	9/24/2015 7:45 PM	File folder	
hamcrest-core-1.3.jar	9/24/2015 7:45 PM	Executable Jar File	44 KB
inputQueries.txt	9/24/2015 7:45 PM	Text Document	1 KB
inputTriples.nt	9/24/2015 7:45 PM	NT File	1 KB
junit-4.11.jar	9/24/2015 7:45 PM	Executable Jar File	240 KB
README.txt	9/24/2015 7:45 PM	Text Document	2 KB
Review.pdf	9/24/2015 7:45 PM	Adobe Acrobat D...	123 KB
sampleQueryResult.txt	9/24/2015 7:45 PM	Text Document	1 KB

- Please add a ReadMe.txt file with your shell commands. So that we can copy and paste in the shell.
- For any additional .jar file, please make sure you added in the zip file and in your command ReadMe.txt
- Please include your design review documentation.

Code and documentiton

Name	Date modified	Type	Size
 cscie97	9/24/2015 7:45 PM	File folder	
 allclasses-frame.html	9/24/2015 7:45 PM	Chrome HTML Do...	3 KB
 allclasses-noframe.html	9/24/2015 7:45 PM	Chrome HTML Do...	2 KB
 constant-values.html	9/24/2015 7:45 PM	Chrome HTML Do...	4 KB
 deprecated-list.html	9/24/2015 7:45 PM	Chrome HTML Do...	4 KB
 help-doc.html	9/24/2015 7:45 PM	Chrome HTML Do...	9 KB
 index.html	9/24/2015 7:45 PM	Chrome HTML Do...	3 KB
 index-all.html	9/24/2015 7:45 PM	Chrome HTML Do...	21 KB
 overview-tree.html	9/24/2015 7:45 PM	Chrome HTML Do...	7 KB
 package-list	9/24/2015 7:45 PM	File	1 KB
 script.js	9/24/2015 7:45 PM	JS File	1 KB
 serialized-form.html	9/24/2015 7:45 PM	Chrome HTML Do...	6 KB
 stylesheet.css	9/24/2015 7:45 PM	CSS File	14 KB

- Please generate your javaDoc . Html files. It will give us quick overview of your implementation. You will not lose any point, but it will be great to have.

Give some additional file to test your code

Name	Date modified	Type	Size
cscie97	9/24/2015 9:31 AM	File folder	
Results.pdf	9/24/2015 9:31 AM	Adobe Acrobat D...	33 KB
inputTriples.nt	9/24/2015 9:31 AM	NT File	1 KB
inputTriplesDouble.nt			
inputTriplesTooBig.nt			
inputQueries.txt			
inputQueriesTooBig.b			
InputQueriesTooBig.t			
inputTriplesDouble.txt			
outPutDoubleTriples.t			
outPutGivenFiles.txt			
outputInputTooBig.txt			

Name	Date modified	Type	Size
cscie97	9/24/2015 9:40 PM	File folder	
nobre_design_feedback.pdf	9/24/2015 9:40 PM	Adobe Acrobat D...	82 KB
bad_inputTriples.r*	9/24/2015 9:40 PM	NT File	1 KB
empty_inputTriple			
inputTriples.nt			
bad_inputQueries.			
inputQueries.txt			
Query_Results.txt			

Name	Date modified	Type	Size
cscie97	9/24/2015 9:56 PM	File folder	
Assignment1_JREED.pdf	9/24/2015 9:56 PM	Adobe Acrobat D...	22 KB
inputQueries.txt	9/24/2015 9:56 PM	Text Document	1 KB
inputQueries2.txt	9/24/2015 9:56 PM	Text Document	1 KB
inputTriples.nt	9/24/2015 9:56 PM	NT File	1 KB
inputTriples2.nt	9/24/2015 9:56 PM	NT File	1 KB
Results.txt	9/24/2015 9:56 PM	Text Document	3 KB

Other comments

- Did design document help you in writing code effectively ?
- Which are the areas you feel confusing during implementation ?
- What you did not like in the design document ? If any
- How the design can be improved – justify your point ?
- What are the limitations of current design ?
- Can you encapsulate the package for future re-use ?

Comment your code

- Class Comment

```
/**  
 * The Importer class is responsible for reading triples from input files using N-Triple format. The Importer class  
 * creates a Triple instance for each line read from the input file and passes the resulting Triples to the  
 * KnowledgeGraph.importTriples() method. Also, only fully qualified Triples (i.e. subject, predicate, object all  
 * have identifiers) should be added to the Knowledge Graph. Trim extra leading and trailing whitespace from  
 * identifier names. The importTripleFile method throws an ImportException on error processing the input file.  
 */
```

- Method comment

```
/**  
 * Public method for adding a list of Triples to the KnowledgeGraph. The following associations must be  
 * updated: nodeMap, tripleMap, queryMapSet, predicateMap to reflect the added Triple. There should be one Triple  
 * instance per unique Subject, Predicate, Object combination, so that Triples are not duplicated.  
 *  
 * @param subjectId subject identifier  
 * @param predicateId predicate identifier  
 * @param objectId object identifier  
 * @throws ImportException on error processing triple  
 */
```

Exception Handling

- Please make sure exceptions are really exceptions.
- Create specific class for particular type of exception . For example

```
public class ImportException extends Exception {  
}
```

- Raise exception where it is needed

```
throw importException;
```

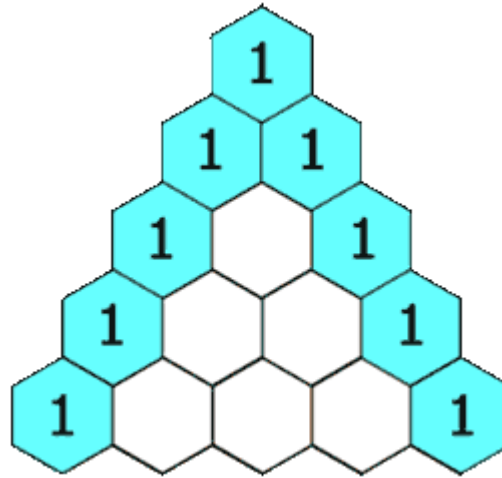
Exception Handling

- Catch exception where it is needed

```
} catch (ImportException e) {  
    e.setLineNumber(lineNumber);  
    e.setErrorContent(sCurrentLine);  
    throw e;  
}  
} catch (IOException e) {  
    e.printStackTrace();  
    ImportException ie = new ImportException();  
    ie.setFilename(inputFileName);  
    ie.setErrorContent(e.getMessage());  
    throw ie;  
}
```

- Any IllegalArgumentException ?

```
throw new IllegalArgumentException("null Node identifier");
```



Patterns in this assignment

Singleton Pattern

```
1 package test;
2
3
4 public class test {
5
6     /**
7      * Private default constructor
8      */
9     private test() {
10
11     }
12
13     /**
14      * Private static singleton holder
15      */
16     private static class testSingletonHolder {
17         public static final test INSTANCE = new test();
18     }
19
20     /**
21      * This method returns a reference to the single static instance of the Test.
22      *
23      * @return single instance of Test.
24      */
25     public static test getInstance() {
26         return testSingletonHolder.INSTANCE;
27     }
28
29
30 }
```

- Get some idea from https://en.wikipedia.org/wiki/Singleton_pattern

Flyweight pattern

Example in Java [\[edit\]](#)

```
import java.util.List;
import java.util.Map;
import java.util.Vector;
import java.util.concurrent.ConcurrentHashMap;

// Instances of CoffeeFlavour will be the Flyweights
class CoffeeFlavour {
    private final String name;

    CoffeeFlavour(final String newFlavor) {
        this.name = newFlavor;
    }

    @Override
    public String toString() {
        return name;
    }
}

// Menu acts as a factory and cache for CoffeeFlavour flyweight objects
class Menu {
    private Map<String, CoffeeFlavour> flavours = new ConcurrentHashMap<String, CoffeeFlavour>();

    CoffeeFlavour lookup(final String flavorName) {
        if (!flavours.containsKey(flavorName))
            flavours.put(flavorName, new CoffeeFlavour(flavorName));
        return flavours.get(flavorName);
    }

    int totalCoffeeFlavoursMade() {
        return flavours.size();
    }
}

// Order is the context of the CoffeeFlavour flyweight.
class Order {
    private final int tableNumber;
    private final CoffeeFlavour flavour;

    Order(final int tableNumber, final CoffeeFlavour flavor) {
        this.tableNumber = tableNumber;
        this.flavour = flavor;
    }

    void serve() {
```

```
        System.out.println("Serving " + flavour + " to table " + tableNumber);
    }
}

public class CoffeeShop {
    private final List<Order> orders = new Vector<Order>();
    private final Menu menu = new Menu();

    void takeOrder(final String flavourName, final int table) {
        CoffeeFlavour flavour = menu.lookup(flavourName);
        Order order = new Order(table, flavour);
        orders.add(order);
    }

    void service() {
        for (Order order : orders)
            order.serve();
    }

    String report() {
        return "\ntotal CoffeeFlavour objects made: "
            + menu.totalCoffeeFlavoursMade();
    }

    public static void main(final String[] args) {
        CoffeeShop shop = new CoffeeShop();

        shop.takeOrder("Cappuccino", 2);
        shop.takeOrder("Frappe", 1);
        shop.takeOrder("Espresso", 1);
        shop.takeOrder("Frappe", 897);
        shop.takeOrder("Cappuccino", 97);
        shop.takeOrder("Frappe", 3);
        shop.takeOrder("Espresso", 3);
        shop.takeOrder("Cappuccino", 3);
        shop.takeOrder("Espresso", 96);
        shop.takeOrder("Frappe", 552);
        shop.takeOrder("Cappuccino", 121);
        shop.takeOrder("Espresso", 121);

        shop.service();
        System.out.println(shop.report());
    }
}
```

- https://en.wikipedia.org/wiki/Flyweight_pattern

Immutable Objects

```
public final class Node {  
    /**  
     * Immutable classes to stop promoting object proliferation. If we want safety we have to  
prevent the user from changing the object setter.  
     * String class is immutable as it does not provide any setter to change its content  
     */  
  
    private final String identifier;  
  
    /**  
     * Default private constructor will ensure no unplanned construction of class  
     * @param identifier  
     */  
    public Node(String identifier) {  
        // set the private property value  
        this.identifier=identifier.toLowerCase();  
    }  
    /**
```

