Object Oriented Programming
Programming report
Assignment 3: Receipt

Corradini Matteo Berke Atac
S3051390 S3075168

May 17, 2016

1 Problem description

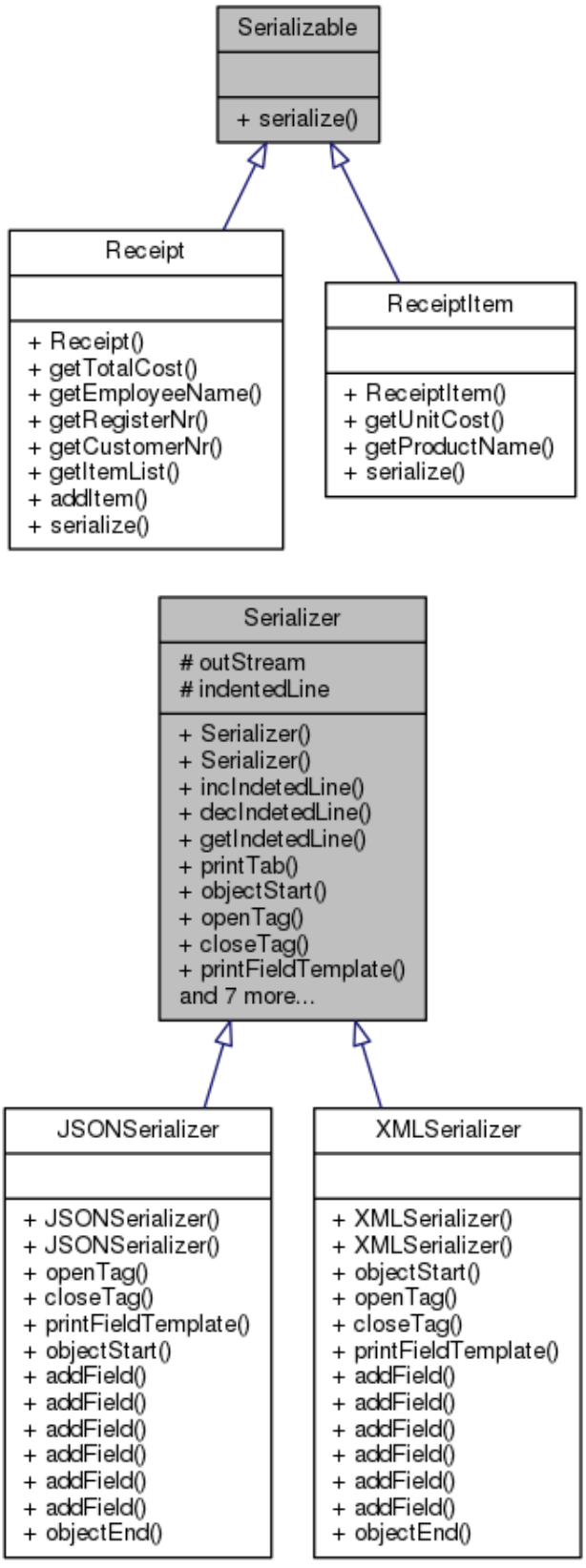
The problem was to serialize a set of objects that contains information, in JSON and XML data-interchange formats.

The problem is that we have a receipt with attributes where some are lists of objects. In this assignment, the input is represented by a receipt that contains a list of items and attributes. There are two receipts and the goal is to serialize these receipts into two data-interchange formats that are JSON and XML. The output has to be in line with the rules of these data serialization processes including correct use of tags and indentation.

2 Problem analysis

In the beginning we had the following Java source-files:
`Main.java`, `Receipt.java`, `ReceiptItem.java`, `Serializable.java`, `Serializer.java`.

The classes `Receipt.java` and `ReceiptItem.java` implements the class `Serializable.java`, while the classes `XMLSerializer.java` and `JSONSerializer.java` extends the class `Serializer.java`. We analyzed the code, we compiled it and we executed it. `Serializable` class represents the type of object that has to be serialized. While `Serializer` class stands as a template for other serializer classes. `Main.java` creates two receipts and serializers for both formats and then serializes the receipts into both formats.
We have to implement the abstract method in `Serializer` class: each of them has to add to the output a different type of information. For example, there is `addField(String fieldName, int i);` to add an integer field to the output translation, and there is also `addField(String fieldName, double d);` to add a double field. In this way, we divided the the whole problem in a sub-problem, with a particular and short job to do. Thus, the main idea is divided and conquer paradigm, and following this idea, we created three abstract methods in order to print a line in the correct output format for each serializer, and a new class called `IndenterLine` in order to print the correct indentation.


Serializable
+ serialize()
Receipt
+ Receipt()
+ getTotalCost()
+ getEmployeeName()
+ getRegisterNr()
+ getCustomerNr()
+ getItemList()
+ addItem()
+ serialize()
ReceiptItem
+ ReceiptItem()
+ getUnitCost()
+ getProductName()
+ serialize()
Serializer
outStream
indentedLine
+ Serializer()
+ Serializer()
+ incIndetedLine()
+ decIndetedLine()
+ getIndetedLine()
+ printTab()
+ objectStart()
+ openTag()
+ closeTag()
+ printFieldTemplate()
and 7 more...
JSONSerializer
+ JSONSerializer()
+ JSONSerializer()
+ openTag()
+ closeTag()
+ printFieldTemplate()
+ objectStart()
+ addField()
+ addField()
+ addField()
+ addField()
+ addField()
+ addField()
+ objectEnd()
XMLSerializer
+ XMLSerializer()
+ XMLSerializer()
+ objectStart()
+ openTag()
+ closeTag()
+ printFieldTemplate()
+ addField()
+ addField()
+ addField()
+ addField()
+ addField()
+ addField()
+ objectEnd()

3 Program design

We decided to provide new methods in the abstract class: there are two printing methods in order to print the correct close and open tag of each serializer (`openTag(String field)` and `closeTag(String field)`).
The third printing method is `printFieldTemplate(String field, Object x)`, and it uses the two previous methods to print the correct line for each serializer, where `field` is the field to print, and `x` is its value. For this method we used the late binding, thus the polymorphism: `x` is a general object, and it calls its own `toString()` method at runtime. With a general parameter we are able to use the same method for each type of information, using the same code only one times. Consequentially, each `addField()` method for a primitive type, is composed by one line. In this way, we wrote the features of each serializer in only three suitable methods.
In order to respect the rules of Object Oriented Programming, we add the getter methods in each class we needed to provide information (`Receipt` and `ReceiptItem`).
We implemented `serialize(Serializer serializer)` in both class with the same idea: we called `objectStart()` method, sequentially every pertinent `addField()` for each field of the object, and in the end we called `objectEnd()`.
In order to print a list of objects, we used the Java Generics creating a general iterator of objects that extend `Serializable`.

4 Evaluation of the program

The output of our program:

Listing 1: `Output`

```
JSON:
Receipt 1:
{
    Receipt: {
        registerNr:42,
        customerNr:1337,
        employeeName:"Peter␣Selie",
        totalCost:1358.34,
        itemList: [
            {
                ReceiptItem: {
                    productName:"Unicorn␣Meat",
                    unitCost:999.99
                }
            },
            {
                ReceiptItem: {
                    productName:"HandzOff",
                    unitCost:19.95
                }
            },
            {
                ReceiptItem: {
                    productName:"Diet␣Water",
                    unitCost:9.95
                }
            }
        ]
    }
}
Receipt 2:
```

```
{
    Receipt: {
        registerNr:11,
        customerNr:65536,
        employeeName:"Connie␣Veren",
        totalCost:182.76999999999998,
        itemList: [
            {
                ReceiptItem: {
                    productName:"Crib␣Dribbler",
                    unitCost:49.99
                }
            },
            {
                ReceiptItem: {
                    productName:"Bling␣Teeth",
                    unitCost:4.99
                }
            },
            {
                ReceiptItem: {
                    productName:"Pet␣Petter",
                    unitCost:39.95
                }
            },
            {
                ReceiptItem: {
                    productName:"AB-hancer",
                    unitCost:2.95
                }
            }
        ]
    }
}

XML:
Receipt 1:
<Receipt>
    <registerNr>42</registerNr>
    <customerNr>1337</customerNr>
    <employeeName>"Peter␣Selie"</employeeName>
    <totalCost>1358.34</totalCost>
    <itemList>
        <ReceiptItem>
            <productName>"Unicorn␣Meat"</productName>
            <unitCost>999.99</unitCost>
        </ReceiptItem>
        <ReceiptItem>
            <productName>"HandzOff"</productName>
            <unitCost>19.95</unitCost>
        </ReceiptItem>
        <ReceiptItem>
            <productName>"Diet␣Water"</productName>
            <unitCost>9.95</unitCost>
        </ReceiptItem>
    </itemList>
</Receipt>

```

```
Receipt 2:
<Receipt>
    <registerNr>11</registerNr>
    <customerNr>65536</customerNr>
    <employeeName>"Connie␣Veren"</employeeName>
    <totalCost>182.76999999999998</totalCost>
    <itemList>
        <ReceiptItem>
            <productName>"Crib␣Dribbler"</productName>
            <unitCost>49.99</unitCost>
        </ReceiptItem>
        <ReceiptItem>
            <productName>"Bling␣Teeth"</productName>
            <unitCost>4.99</unitCost>
        </ReceiptItem>
        <ReceiptItem>
            <productName>"Pet␣Petter"</productName>
            <unitCost>39.95</unitCost>
        </ReceiptItem>
        <ReceiptItem>
            <productName>"AB-hancer"</productName>
            <unitCost>2.95</unitCost>
        </ReceiptItem>
    </itemList>
</Receipt>
```

As it is possible to see, the indentation and comma rules are respected.

5 Extension of the program

An important feature is the indentation: we created the new class `IndenterLine`, in order to provide a new suite that is able to count how many `\t` (tabulation) has to be printed before the field. The class contains a private integer counter (`indentedLine`) and methods to increase, decrease and get the value the counter. Furthermore, there is also the method to print an amount of tabulation, in according with `indentedLine` counter. We decided to create a new class, because it has to perform only one job, different from the serialization of the object. On the other hand, the `Serializer` class is abstract, thus it is not logically correct that it has implement methods. An instance of `IndenterLine` is present in `Serializer` abstract class, and it is inherited from each serializer.
Another important feature, it is about the `JSONSerializer` class. The problem with JSON was about the comma: it has to be printed after each field, less than the last one. Thus, we had this problem before the close tag of the object, in `objectEnd()` method. Our solution is print a special character (`\b`) in order to remove the last one (a comma) from the output stream. We implemented this feature in `objectEnd()`, before to print any character. Thus, if there is a character in the output stream, it has to be the comma which we want to remove.

6 Conclusions

Our program solves the assignment requests. The most challenging feature to implement was how to print the correct indentation and how to remove the comma from the last field.
Firstly, we implemented the code for the indentation feature in the serializer class. However, we noticed the huge mistake and we created a new class.
Another problem was to manage propriately the special character `\n` in `JSONSerializer` in order to create a correct output, combining this solution with the comma feature. After few attempts, we adjust each printing function to work in according with the standard output.

This assignment helped us to understand the late binding, and also the already implemented overloading of the method `addField()`, thus Java is able to understand that methods with the same name are different from the signature. Our program knows which add `addField()` methods has to call checking the second parameter, and there is always a perfect match between that one and the one in one of `addField()` method.
We also learned how to managed an output stream, with the special character `\n`, `\b` and `\t`.

7 Appendix: program text

Listing 2: `Main`

```
1
2  public class Main {
3
4      public static void main(String[] args) {
5          // Create receipts
6          Receipt receipt1 = new Receipt("Peter␣Selie", 42, 1337);
7          Receipt receipt2 = new Receipt("Connie␣Veren", 11, 65536);
8
9          // Populate receipts
10         receipt1.addItem(new ReceiptItem("Unicorn␣Meat", 999.99), 1);
11         receipt1.addItem(new ReceiptItem("HandzOff", 19.95), 3);
12         receipt1.addItem(new ReceiptItem("Diet␣Water", 9.95), 30);
13
14         receipt2.addItem(new ReceiptItem("Crib␣Dribbler", 49.99), 1);
15         receipt2.addItem(new ReceiptItem("Bling␣Teeth", 4.99), 2);
16         receipt2.addItem(new ReceiptItem("Pet␣Petter", 39.95), 3);
17         receipt2.addItem(new ReceiptItem("AB-hancer", 2.95), 1);
18
19         //TODO: Construct Serializer subclasses.
20         Serializer jsonSerializer = new JSONSerializer();
21         Serializer xmlSerializer = new XMLSerializer();
22
23         // Print serialized receipts. The OutputStream of jsonSerializer and
               xmlSerializer should
24         //   be set to System.out, which will cause printing to occur during
               serialization.
25         System.out.println("JSON:");
26         System.out.println("Receipt␣1:");
27         receipt1.serialize(jsonSerializer);
28         System.out.println("\nReceipt␣2:");
29         receipt2.serialize(jsonSerializer);
30
31         System.out.println("\n\nXML:");
32         System.out.println("Receipt␣1:");
33         receipt1.serialize(xmlSerializer);
34         System.out.println("\nReceipt␣2:");
35         receipt2.serialize(xmlSerializer);
36     }
37 }
```

Listing 3: `Serializer`

```
1
2  import java.util.List;
3  import java.io.OutputStream;
4  import java.io.PrintStream;
```

```
5
6  public abstract class Serializer {
7      protected PrintStream outStream;
8      protected IndenterLine iLine;
9
10     public Serializer(OutputStream out) {
11         this.outStream = new PrintStream(out);
12         this.iLine = new IndenterLine();
13     }
14     public Serializer() {
15         this.outStream = System.out;
16         this.iLine = new IndenterLine();
17     }
18
19     /**
20      * Begin serializing a new object. Should always be eventually followed by
            a call to objectEnd
21      *   with the identical objectName.
22      * @param objectName Name of the object to begin serializing.
23      */
24     public abstract void objectStart(String objectName);
25
26     /**
27      *  This couple of utility functions are used to print the specific tag of
            the
28      *  different serializer class.
29      * @param field name of the field has to be written in the tag.
30      * @return String object with the correct tag of the serializer.
31      */
32     public abstract String openTag (String field);
33     public abstract String closeTag (String field);
34
35     /**
36      * General template to print a field, used by addField for primitive types
            .
37      * @param fieldName name of the field to print
38      * @param x value of the field
39      * @return String object with a template of the correct serializer
40      */
41     public abstract String printFieldTemplate(String fieldName, Object x);
42
43     /**
44      * Add a field/value pair to the serialization.
45      * @param fieldName Name of the field to be added.
46      */
47     public abstract void addField(String fieldName, int i);
48     public abstract void addField(String fieldName, double d);
49     public abstract void addField(String fieldName, boolean b);
50     public abstract void addField(String fieldName, String s);
51     public abstract void addField(String fieldName, List<? extends
           Serializable> l);
52     //Should we do remove it?
53     public abstract void addField(String fieldName, Serializable object);
54     /**
55      * Ends the serialization of an object, potentially continuing
           serialization on other
56      *   containing objects. Should always be proceeded by a call to
           objectStart with the identical
```

```
57        *   objectName.
58        * @param objectName Name of the object to finish serializing.
59        */
60       public abstract void objectEnd(String objectName);
61   }
```

Listing 4: JSONSerializer

```
1   import java.io.OutputStream;
2   import java.util.Iterator;
3   import java.util.List;
4
5   public class JSONSerializer extends Serializer {
6
7       public JSONSerializer(){
8           super();
9       }
10      public JSONSerializer(OutputStream out){
11          super(out);
12      }
13
14      @Override
15      public String openTag(String field) {
16          // TODO Auto-generated method stub
17          String buff = this.iLine.printTab() + "{\n";
18          this.iLine.incIndetedLine();
19          buff += this.iLine.printTab() + field + ":␣{␣";
20          this.iLine.incIndetedLine();
21          return buff;
22      }
23
24      @Override
25      public String closeTag(String field) {
26          // TODO Auto-generated method stub
27          String buff = "\b␣\n" + this.iLine.printTab() + "}\n";
28          this.iLine.decIndetedLine();
29          buff += this.iLine.printTab() + "}";
30          return buff;
31      }
32
33      @Override
34      public String printFieldTemplate(String fieldName, Object x){
35          return "\n"+ this.iLine.printTab() + fieldName + ":" + x + ",";
36      }
37
38      @Override
39      public void objectStart(String objectName) {
40          // TODO Auto-generated method stub
41          this.outStream.print(this.openTag(objectName));
42      }
43
44      @Override
45      public void addField(String fieldName, int i) {
46          // TODO Auto-generated method stub
47          this.outStream.print(this.printFieldTemplate(fieldName, i));
48      }
49
50      @Override
51      public void addField(String fieldName, double d) {
```

```
52          // TODO Auto-generated method stub
53          this.outStream.print(this.printFieldTemplate(fieldName, d));
54      }
55
56      @Override
57      public void addField(String fieldName, boolean b) {
58          // TODO Auto-generated method stub
59          this.outStream.print(this.printFieldTemplate(fieldName, b ? "true" : "
                false"));
60      }
61
62      @Override
63      public void addField(String fieldName, String s) {
64          // TODO Auto-generated method stub
65          this.outStream.print(this.printFieldTemplate(fieldName, '"' + s + '"')
                );
66      }
67
68      //TODO: Add indentetion
69      @Override
70      public void addField(String fieldName, List<? extends Serializable> l) {
71          // TODO Auto-generated method stub
72          this.outStream.print("\n" + this.iLine.printTab() + fieldName +": [\n"
                );
73          Iterator<? extends Serializable> iter = l.iterator();
74
75          this.iLine.incIndetedLine();
76
77          while (iter.hasNext()){
78              iter.next().serialize(this);
79              if (iter.hasNext()) {this.outStream.print(",\n");}
80          }
81
82          this.iLine.decIndetedLine();
83
84          this.outStream.print("\n" + this.iLine.printTab() + "] ");
85      }
86
87      @Override
88      public void addField(String fieldName, Serializable object) {
89          // TODO Auto-generated method stub
90      }
91
92      @Override
93      public void objectEnd(String objectName) {
94          // TODO Auto-generated method stub
95          this.iLine.decIndetedLine();
96          this.outStream.print(this.closeTag(objectName));
97      }
98
99  }
```

Listing 5: `XMLSerializer`

```
1   import java.io.OutputStream;
2   import java.util.Iterator;
3   import java.util.List;
4
5   public class XMLSerializer extends Serializer {
```

```
6
7       public XMLSerializer(){
8           super();
9       }
10      public XMLSerializer(OutputStream out){
11          super(out);
12      }
13
14      @Override
15      public void objectStart(String objectName) {
16          // TODO Auto-generated method stub
17          this.outStream.print(this.openTag(objectName) + "\n");
18          this.iLine.incIndetedLine();
19      }
20
21      @Override
22      public String openTag (String field){
23          return this.iLine.printTab() + "<" + field + ">";
24      }
25
26      @Override
27      public String closeTag (String field){
28          return "</" + field + ">\n";
29      }
30
31      @Override
32      public String printFieldTemplate(String fieldName, Object x){
33          return this.openTag(fieldName) + x + this.closeTag(fieldName);
34      }
35
36      @Override
37      public void addField(String fieldName, int i) {
38          // TODO Auto-generated method stub
39          this.outStream.print(this.printFieldTemplate(fieldName, i));
40      }
41
42      @Override
43      public void addField(String fieldName, double d) {
44          // TODO Auto-generated method stub
45          this.outStream.print(this.printFieldTemplate(fieldName, d));
46      }
47
48      @Override
49      public void addField(String fieldName, boolean b) {
50          // TODO Auto-generated method stub
51          this.outStream.print(this.printFieldTemplate(fieldName, b ? "true" : "
                false"));
52      }
53
54      @Override
55      public void addField(String fieldName, String s) {
56          // TODO Auto-generated method stub
57          this.outStream.print(this.printFieldTemplate(fieldName, '"' + s + '"')
                );
58      }
59
60      @Override
61      public void addField(String fieldName, List<? extends Serializable> l) {
```

```
62          // TODO Auto-generated method stub
63          this.objectStart(fieldName);
64
65          Iterator<? extends Serializable> iter = l.iterator();
66
67          while (iter.hasNext()){
68              iter.next().serialize(this);
69          }
70
71          this.objectEnd(fieldName);
72      }
73
74      @Override
75      public void addField(String fieldName, Serializable object) {
76          // TODO Auto-generated method stub
77      }
78
79      @Override
80      public void objectEnd(String objectName) {
81          // TODO Auto-generated method stub
82          this.iLine.decIndetedLine();
83          this.outStream.print(this.iLine.printTab() + this.closeTag(objectName)
                );
84          this.outStream.flush();
85      }
86
87  }
```

Listing 6: `Serializable`

```
1
2   public interface Serializable {
3       public void serialize(Serializer module);
4   }
```

Listing 7: `Receipt`

```
1
2   import java.util.ArrayList;
3   import java.util.List;
4
5   public class Receipt implements Serializable {
6       private List<ReceiptItem> itemList;
7       private double totalCost;
8       private String employeeName;
9       private int registerNr;
10      private int customerNr;
11
12      public Receipt(String employeeName, int registerNr, int customerNr) {
13          this.employeeName = employeeName;
14          this.registerNr = registerNr;
15          this.customerNr = customerNr;
16          itemList = new ArrayList<>();
17          totalCost = 0;
18      }
19
20      public double getTotalCost() {
21          return totalCost;
22      }
```

```
23
24      public String getEmployeeName() {
25          return employeeName;
26      }
27
28      public int getRegisterNr() {
29          return registerNr;
30      }
31
32      public int getCustomerNr() {
33          return customerNr;
34      }
35
36      public List<ReceiptItem> getItemList() {
37          return itemList;
38      }
39
40      public void addItem(ReceiptItem item, int amount) {
41          itemList.add(item);
42          totalCost += item.getUnitCost() * amount;
43      }
44
45      @Override
46      public void serialize(Serializer serializer) {
47          //TODO: serialize all fields using serializer.
48          serializer.objectStart("Receipt");
49          serializer.addField("registerNr", this.getRegisterNr());
50          serializer.addField("customerNr", this.getCustomerNr());
51          serializer.addField("employeeName", this.getEmployeeName());
52          serializer.addField("totalCost", this.getTotalCost());
53          serializer.addField("itemList", this.getItemList());
54          serializer.objectEnd("Receipt");
55      }
56  }
```

Listing 8: `ReceiptItem`

```
1
2   public class ReceiptItem implements Serializable {
3       private String productName;
4       private double unitCost;
5
6       public ReceiptItem(String productName, double unitCost) {
7           this.productName = productName;
8           this.unitCost = unitCost;
9       }
10
11      public double getUnitCost() {
12          return unitCost;
13      }
14
15      public String getProductName() {
16          return productName;
17      }
18
19      @Override
20      public void serialize(Serializer serializer) {
21          //TODO: serialize all fields using serializer.
22          serializer.objectStart("ReceiptItem");
```

```
23          serializer.addField("productName", this.getProductName());
24          serializer.addField("unitCost", this.getUnitCost());
25          serializer.objectEnd("ReceiptItem");
26      }
27  }
```

8 Appendix: extended program text

Listing 9: `IndenterLine`

```
1
2   public class IndenterLine {
3       private int indentedLine;
4
5       public IndenterLine(){
6           this.indentedLine = 0;
7       }
8
9       public void incIndetedLine(){
10          this.indentedLine++;
11      }
12
13      public void decIndetedLine(){
14          this.indentedLine--;
15      }
16
17      public int getIndetedLine(){
18          return this.indentedLine;
19      }
20
21      public String printTab(){
22          String buff = "";
23          for (int i = 0; i<this.getIndetedLine() ; i++) buff += "\t";
24          return buff;
25      }
26
27  }
```