Brock University

Lab Test 3, Winter 2025 **Number of Pages: 3**
Course: COSC 3P91 **Number of Hours: 2**
Instructor: M. Winter

Instructions

1) **Write a program as indicated below using Java and NetBeans.**
2) **Zip your project folder and name it with your name and student number, e.g., MichaelWinter123456.zip**
3) **Send the file to your lab instructor by email.**
4) **Wait for a confirmation email.**
5) **Log out of the lab computer and hand back the question sheets to the lab instructor. You can now leave the lab.**
6) **The test is worth a total of 20 marks.**

In this test you are going to use XML, multiple threads and the design patterns **Producer-Consumer** and **Singleton**. Implement the following:

1. Add the `COSC3P91.jar` (available on the course webpage) file as a library to your project so that you can use the classes and methods discussed in class. Add the XSD and XML folder that you will find in a zip file on the course webpage to your project, i.e., add the folders to the project folder (same folder `src`).
2. For all classes in this test make fields `private` resp. `final` if possible. Implement getter and setter methods **only** when needed.
3. Whenever you implement `toXMLString()` you can use plain strings, i.e., you do not need to use a `StringBuilder`.
4. (1 mark) Implement a class `Employee` that has the following fields `firstName`, `lastName` of type `String`, and `age` and `hourlyRate` of type `int`. In addition to a constructor receiving values for the fields make the class an instance of XMLObject. Refer to the file `Employee.xml` and the XML schema `Employee.xsd` for the syntax of an employee in XML. Add a method `int computeID()` that compute an ID of an employee. The ID is the sum of the employee's age, the first character of his/her first name casted to an `int`, and the first character of his/her last name casted to an `int`. Add a static method `Employee read(Reader source)` to the class that returns an employee based on the XML description in `source` (refer to 7.).
5. (1 mark) Implement a class `SalaryData` that has the fields `id` and `hourlyRate` of type `int`. In addition to a constructor receiving values for the fields make the

class an instance of XMLObject. Refer to the XML schema `SalaryData.xsd` and the example below for the syntax of `SalaryData` in XML.
`<SalaryData id="193" checkSum="1" hourlyRate="2346"/>`
Add methods `int computeCheckSum()` and `boolean check(int checksum)` that compute the checksum of an ID and check that the parameter `checkSum` equals the checksum of this ID. The checksum of an ID is `id % 16`. Add a static method `SalaryDate read(Reader source)` to the class that returns a `SalaryData` based on the XML description in `source` (refer to 8.).

6. (1 mark) Implement a class `WorkData` that has the fields `id`, and `hoursWorked` of type `int`. In addition to a constructor receiving values for the fields make the class an instance of XMLObject. Refer to the file `WorkData.xml` and the XML schema `WorkData.xsd` for the syntax of an employee in XML. Add a method `void addHours(int amount)` that increases the `hoursWorked` by `amount`. Add a static method `WorkData read(Reader source)` to the class that returns a `WorkData` based on the XML description in `source` (refer to 9.).
7. (2 marks) Implement a class `XMLNodeConverterEmployee` that is an `XMLNodeConverter` for `Employee`. The `convertXMLNode(Node node)` method should return `null` if the `node` is not a node of a `Employee`. Make this class a **Singleton**, i.e., use the **Singleton** design pattern for this class.
8. (2 marks) Implement a class `XMLNodeConverterSalaryData` that is an `XMLNodeConverter` for `SalarayData`. The `convertXMLNode(Node node)` method should return `null` if the `node` is not a node of a `SalaryData`. Make this class a **Singleton**, i.e., use the **Singleton** design pattern for this class.
9. (2 marks) Implement a class `XMLNodeConverterWorkData` that is an `XMLNodeConverter` for `WorkData`. The `convertXMLNode(Node node)` method should return `null` if the `node` is not a node of a `WorkData`. Make this class a **Singleton**, i.e., use the **Singleton** design pattern for this class.
10. (2 mark) Implement a generic class `Queue<E>` of a queue storing elements from `E`. This class is supposed to be thread safe, i.e., can be used safely in a multiple thread environment. The class has methods `void push(E element)` and `E pull()` for adding and getting the next element from the queue. The `pull()` method is supposed to be blocking, i.e., if `pull()` is called and the queue is empty, then the thread calling `pull()` is supposed to wait until an element becomes available.
11. (3 marks) Implement a class `EmployeeThread` that is a thread. The class has a `Queue<Integer> queue` that is passed in the constructor. The `run()` method the `EmployeeThread` reads the employees from the file `Employee.xml` (each

line is one employee) using a `BufferedReader`, creates a `SalarayData` from the employee, and pushes this into the `queue`. If there is no employee left in the file, push `null` to the `queue` as an indication that all data has been processed.

12. (3 marks) Implement a class `WorkDataThread` that is a thread. The class has an `int bonusHours` and a `Queue<Integer> queue` that are passed in the constructor. The `run()` method the `WorkDataThread` reads the `WorkData` from the file `WorkData.xml` (each line is one `WorkData`) using a `BufferedReader`, adds the `bonusHours` to the `WorkData`, and pushes this into the `queue`. If there is no `WorkData` left in the file, push `null` to the `queue` as an indication that all data has been processed.
13. (3 marks) In the `main()` method of the program set the XSD path of the `XMLReader` to your XSD folder, i.e., add `XMLReader.setXSDPath("./XSD/");`. Then implement the **Producer-Consumer** design pattern for the `EmployeeThread`, the `WorkDataThread`, and the main thread (the thread of the method `main()`). Therefore, create two queues, an `EmployeeThread` and a `WorkDataThread` with `bonusHours=10` and one of the queues, respectively, and start the two threads. Then receive the `SalaryData` from the `EmployeeThread` via the first queue and add the data to a `Map<Integer, Integer>` that maps an `id` to its `hourlyRate`. Finally, receive the `WorkData` from the `WorkDataThread` via the second queue and print the employee's salary following output format as listed below, where the salary is the `hoursWorked * hourlyRate`.
    Employee 193 gets $3096.72.
    Employee 210 gets $3319.34.
    Employee 182 gets $2788.45.