Thread Signaling and M2M Communication

Luca Arrotta and Riccardo Presotto

EWLab – Università degli studi di Milano

Professor: Claudio Bettini

These slides are based on previous versions created by Letizia Berolaja, Sergio Mascetti, Dario Freni, Claudio Bettini and Gabriele Civitarese

Copyright

Some slides for this course are partly adapted from the ones distributed by the publisher of the reference book for this course (Distributed Systems: Principles and Paradigms, A. S. Tanenbaum, M. Van Steen, Prentice Hall, 2007).

Every Ware Lab

Outline

- Thread Signaling – Wait & Notify
- Machine to Machine Communication – XML and JSON

Thread Signaling

- We have different methods to coordinate the executions of more threads
  - *wait()*
  - *notify()*
  - *join()*
  - *sleep()*
  - …
- You can see something more in this Java Tutorial

Every Ware Lab

Thread.join()

- The *join()* method is used to wait until the end of a thread
  - The thread that called the *join()* method stops until the end of the thread associated with the object on which the method is called
- It can be invoked with an optional parameter that represents the timeout (in milliseconds)
  - In this case, even if the thread does not end before the timeout, the calling thread go on with its next instruction
- The state of a thread can be controlled with the *isAlive()* method
  - It returns *true* if the thread is active, *false* if the thread is ended

Thread.join()

```java
public class Main {
    public static void main(String arg[]) throws Exception {
        Random r = new Random();
        ArrayList<Thread> threads = new ArrayList<Thread>();

        //create some threads
        for (int i=0; i<100; i++) {
            MyThread mt = new MyThread(r);
            threads.add(mt);
        }

        //start all the threads
        for (Thread t: threads) {
            t.start();
        }

        //wait all the thread to finish
        for (Thread t : threads) {
            t.join();
        }

        System.out.println("...All thread finished!");
    }
}
```

Every Ware Lab

Thread.sleep()

- The *sleep()* method temporarily stops the execution of the current thread
  - When should we use it?
    - To perform periodic actions
    - To allow the execution of other threads (useful for debugging)
  - It is a static method, usually it is called directly from the *Thread* class
    - It has a time parameter expressed in milliseconds
    - *Thread.sleep(10000) // wait for 10 seconds*

Object.wait() and Object.notify()

- They are methods of the *Object* class, so they can be applicable to any object
- To any object is associated a list of waiting threads
  - Each thread that calls a *wait()* on a specific object waits until another thread awakens it by calling a *notify()* on the same object
- These methods are used to coordinate two or more threads
- Also the *wait()* method can be invoked with an optional timeout parameter
  - Useful to avoid deadlocks (e.g., a node of a network crashed, we don't want to wait for its answer forever)

Observations on wait() and notify()

- More threads can be waiting on the same object
- Calling *notify()* will awaken only **one** thread
  - **Be careful**: it is not granted that the order in which the threads are awakened is the same order in which they have called the *wait()* method!
- *notifyAll()* awakens **all** the waiting threads

The Use of wait() and notify()

- We create a class to allow two threads to wait for each other before going on with the execution

- Why the *meetUp()* method is *synchronized*?

- How to adapt this code to more than two threads?

```
2  public class CheckPoint {
3      boolean hereFirst = true;
4
5      synchronized void meetUp () {
6          if (hereFirst) {
7              hereFirst = false;
8              wait();
9          } else {
10             notify();
11             hereFirst = true;
12         }
13     }
14 }
```

Every Ware Lab

wait() and Intrinsic Lock

- But…the last example should not work! Why?
  - The first thread enters in the critical region, acquires the lock and calls *wait()*
  - The second thread could not take the lock on the object → Deadlock!
- Why does it work?

wait() and Intrinsic Lock

- *wait()* releases the intrinsic lock associated with the object on which it is called
  - The lock must be acquired before calling *wait()*, otherwise an exception will be raised
- So, the second thread can execute the method
  - and then it awakens the first thread
- When the first thread is awakened, it has to acquired again the lock before going on
  - So it has to wait until the second thread leaves the method

The Use of wait() and notifyAll()

- In this case, we want to coordinate 10 threads

```
21  public class CheckPoint {
22      int n_threads = 10;
23      int counter = 0;
24
25      synchronized void meetUp () {
26          if (counter < n_threads-1) {
27              counter++;
28              wait();
29          } else {
30              notifyAll();
31              counter = 0;
32          }
33      }
34  }
```

Every Ware Lab

Busy Waiting

- This is an example of Busy Waiting

```
...
while (true) {
    if (eventHappened){
        doSomething();
    }

    Thread.sleep(SOME_TIME_IN_MILLISECONDS);
}
...
```

Every Ware Lab

Busy Waiting

- This is the evil!
- The CPU is busy for nothing while it could be used for the computations of other threads
- This happens even if we use the *sleep* method!
- The solution is to use *wait()* and *notify()*

Use this in your project only
if your goal is to do the exam again

```
...
while (true) {
    if (eventHappened){
        doSomething();
    }

    Thread.sleep(SOME_TIME_IN_MILLISECONDS);
}
...
```

Every Ware Lab

Busy Waiting

- To avoid the Busy Waiting
  - Instead of continuing to iterate waiting for the event, the thread calls the *wait()* method
  - Another thread will *notify()* the first thread when the event occurs, awakening it

```
...
while (!eventHappened) {
    wait();
    if(eventHappened)
        doSomething();
}
...
```

Every Ware Lab

Busy Waiting and Synchronized Statements

- The *wait* and *notify* methods are always related to the synchronization object
- You should always think in terms of the intrinsic lock!

```
...
synchronized(obj){
        ...
        obj.wait();
        ...
}
...
synchronized(obj){
        ...
        obj.notify();
        ...
}
```

Every Ware Lab

Exercise – The Veterinarian

- A veterinarian has a waiting room that can contain only dogs and cats
  - A cat can't enter in the waiting room if there is already a dog or a cat
  - A dog can't enter in the waiting room if there is a cat
  - There can't be more than four dogs together
- The animals stay in the waiting room for a random interval of time
- The animals that can't enter in the waiting room have to wait as necessary
- The problem must be solved by using *synchronized*, *wait* and *notify*
  - You have to develop two methods: *enterRoom* and *exitRoom*
  - Each animal is represented by a randomly generated thread from which you will call these methods

The java.util.concurrent Library

- Java provides useful libraries to make the concurrent programming easier
- There are several tools
  - High-level synchronization primitives (semaphores, locks, …)
  - Data structures that support concurrency
  - Atomic data types (e.g., *AtomicInteger*)
- You **MUST NOT** use these tools in the project!
  - The purpose of this lab is to learn in depth how concurrency works in Java

Producer-Consumer Pattern

- It’s a multi-process communication pattern
- The Producer is the entity (in our case, a thread) that sends some data
- The Consumer is the entity (another thread) that receives these data
- Data are transmitted through a shared memory buffer, typically a queue
- It is possible to consider more producers and/or consumers
- What’s the main problem?

The Consumer shouldn’t periodically check if there are data in the buffer

Queue Implementation Example

```
1   public class Queue {
2       public ArrayList<String> buffer = new ArrayList<String>();
3
4       public synchronized void put(String message) {
5           buffer.add(message);
6           notify();
7       }
8
9       public synchronized String take() {
10          String message = null;
11
12          while(buffer.size() == 0) {
13              try { wait(); }
14              catch (InterruptedException e) { e.printStackTrace(); }
15          }
16
17          if(buffer.size() > 0) {
18              message = buffer.get(0);
19              buffer.remove(0);
20          }
21          return message;
22      }
23  }
```

Every Ware Lab

Exercise – The Chat Service

- Build a *chat* service through sockets
- Each user writes the messages to be sent with the keyboard
- The communication is asynchronous: each user can send messagges indipendently from the other users
- The threads must communicate through a shared buffer

- Two possible implementations:
  1. Chat between only two users (one is the server, the other is the client)
  2. Chat-room: a server receives connections from several users and forwards the messages to all the participants

Machine to Machine Communication

- In a distributed system, processes that runs on different machines connected through a network have to communicate
- The communication occurs through messages exchange
- The coordination between processes in a network consists of two operations:
  - Messages transmission
  - Messages reception
- Messages are exchanged through sockets

How to Represent Messages?

- At the Application Layer, data are stored in proper data structures
- At the Network Layer, data are transmitted as streams of bytes
- The problem: the processes can be heterogeneous and store data in different ways (Big Endian vs Little Endian, different float representations, ASCII vs Unicode, …)
- It is necessary to agree on a common format to represent data!

Marshalling and Unmarshalling

- Who transmits the data performs the **marshalling** process: the data to be transmitted are assembled in a proper format, suitable for the transmission
- Who receives the data performs the **unmarshalling** process: the received data are translated in proper data structures
- The processes have to agree on a common external data format to grant interoperability
- We will see three formats
  - XML
  - JSON
  - Protocol Buffers

XML (eXtensible Markup Language)

- An encoding information standard
- A generalized markup language (it’s possible to create dialects)
- Both human- and machine-readable
- There are several tools for validation (*XMLSchema*) and query (*XPath/XQuery*)

Every Ware Lab

XML (eXtensible Markup Language)

```xml
<actor>
    <name>Christian</name>
    <surname>Bale</surname>
    <filmography>
        <movie>
            <title>The Prestige</title>
            <year>2006</year>
        </movie>
        <movie>
            <title>The Dark Knight</title>
            <year>2008</year>
        </movie>
    </filmography>
</actor>
```

Every Ware Lab

JSON (JavaScript Object Notation)

- A more recent standard
- Born for Javascript, but independent of it
- Both human- and machine- readable
- More compact and readable than XML
- Can encode arrays
- The transmission is quicker for a given message
- It has almost replaced XML for network communications

JSON (JavaScript Object Notation)

```
{
    "name": "Christian",
    "surname": "Bale",
    "filmography": [
        {"title": "The Prestige", "year": 2006},
        {"title": "The Dark Knight", "year": 2008}
    ]
}
```

More details about JSON objects definition here

Every Ware Lab

JSON: Encoding with Java

```
public class Actor {
...
    public String toJSONString() throws JSONException {
        JSONObject actor = new JSONObject();
        actor.put(" name ", this.name);
        actor.put(" surname ", this.surname);
        JSONArray movies = new JSONArray()
        for (Movie m : this.filmography) {
            JSONObject movie = new JSONObject();
            movie.put(" title ", m.getTitle());
            movie.put(" year ", m.getYear());
            movies.put(movie);
        }
        actor.put(" filmography ", movies);
        return actor.toString()
    }
...
```

Every Ware Lab

JSON: Decoding with Java

```
public class Actor {
...
    public Actor(String jsonString) throws JSONException {
        JSONObject input = new JSONObject(jsonString);
        this.setName(input.getString(" name "));
        this.setSurname(input.getString(" surname "));
        JSONArray array = input.getJSONArray(" filmography ");
        for (int i=0; i<array.length(); i++) {
            JSONObject current = array.getJSONObject(i);
            String title = current.getString(" title ");
            int year = current.getInt(" year ");
            Movie m = new Movie(title, year);
            this.filmography.add(m);
        }
    }
...
```

Every Ware Lab

The Gson Library

- A Google library used to convert Java objects into JSON representations and viceversa
- It works with any type of pre-existing Java object (even objects we don’t have the code of)
- It doesn’t require annotations and supports generic types
- Easy to use:
  - *toJson()* method to convert an object to a JSON string
  - *fromJson()* method to convert a JSON string to an instance of an object
- More info here

How Data are Transmitted?

- Data are represented in their text-based coding
  - No problem with strings, numbers and booleans
- We need a proper coding for binary data. For example, **Base64**
  - Used to convert a string of bits in a string of ASCII characters
  - The string of bits is divided into 6-bits blocks
  - Each block can contain a value from 0 to 63, and so it can be represented through an ASCII character

Exercise – Students Stats with JSON

- Build a server application *University* that receives data from a client application *Student*
- The *Student* sends through sockets to *University* data related to a specific student:
  - Personal details: Name, Surname, Year of Birth, Place of Residence (multi-attributes field)
  - List of passed exams: for each exam, store Exam Name, Mark and Date of Verbalization
- The *University* receives the message from the socket and prints the student's stats
- The solution must be built by using JSON as data interchange format (you can use the GSON library!)

Every Ware Lab

References

- Code Examples: https://ewserver.di.unimi.it/gitlab/luca.arrotta/lab2-examples

- Exercises Setup: https://ewserver.di.unimi.it/gitlab/riccardopresotto/setup_test_sdp

Every Ware Lab

Contact

- Contact the tutors via email for any clarification or meeting:

tutorsdp@di.unimi.it

Every Ware Lab