



This *JAVANAISE* library was realized as part of the *JAVANAISE Project* for the course of *Advanced Distributed Programming* in *M2PGL*.
The authors are **Johan GIRARD** and **Pierre ODIN**.

Man JAVANAISE

Distributed object cache management

1 Add JAVANAISE in a Project

To use JAVANAISE, the first thing to do is to add the library and several elements to the Project :

- Add JAVANAISE.jar to the *Build Path* of the Project.
- Create a `save/` directory at the root of the Project.
- Create a `java.policy` file at the root of the Project with the following code :

```
grant{  
    permission java.security.AllPermission "", "";  
};
```

2 Run the coordinator

Next, create the class `RunCoordinator`. This class will be use to start and configure the coordinator :

```
import jvn.JvnCoordImpl;  
  
public class RunCoordinator {  
    public static void main(String[] args) {  
        JvnCoordImpl.start(false);  
    }  
}
```

The `start(...)` method take a boolean as input :

- `TRUE` → The coordinator will restore his old tables (list of objects, servers, locks, ...) when starting.
- `FALSE` → The coordinator will reset all his tables when starting.

3 Create an Object

The creation of an object requires 3 steps. Note that the number of different objects created is not limited.

3.1 Interface

Start by create an Interface extending `JvnTransactionObject`. This Interface will contain the definition of all the methods of the object. Each of these methods has to be annotate with `@JvnWriteMethod` if it's a method which can change the state of the object or `@JvnReadMethod` in the other case.

```
import jvn.JvnTransactionObject;
import jvn.annots.JvnReadMethod;
import jvn.annots.JvnWriteMethod;

public interface IMyObject extends JvnTransactionObject {

    @JvnWriteMethod
    public void write(String text);

    @JvnReadMethod
    public String read() ;

}
```

3.2 Class

Create a class extending `JvnTransactionObjectAbstract` and implementing the previous interface. This class will provide the implementation of all the methods of the object.

```
import jvn.JvnTransactionObjectAbstract;

public class MyObject extends JvnTransactionObjectAbstract implements IMyObject {

    ...

    public MyObject(){}

    ...

}
```

3.3 Instanciation

Finally, instantiate the object with the following code :

```
IMyObject o = (IMyObject) JvnObjectProxy.instanceJvn(new MyObject(), "MYOBJECT");
```

The second argument of the `instanceJvn(...)` method is the id of the object on the coordinator.

4 Use an Object in a client application

After the instantiation, the object is ready to use. Every methods define in the interface can be called.

```
o.read();
o.write("Hello");
o.read();
o.write("Bye");
```

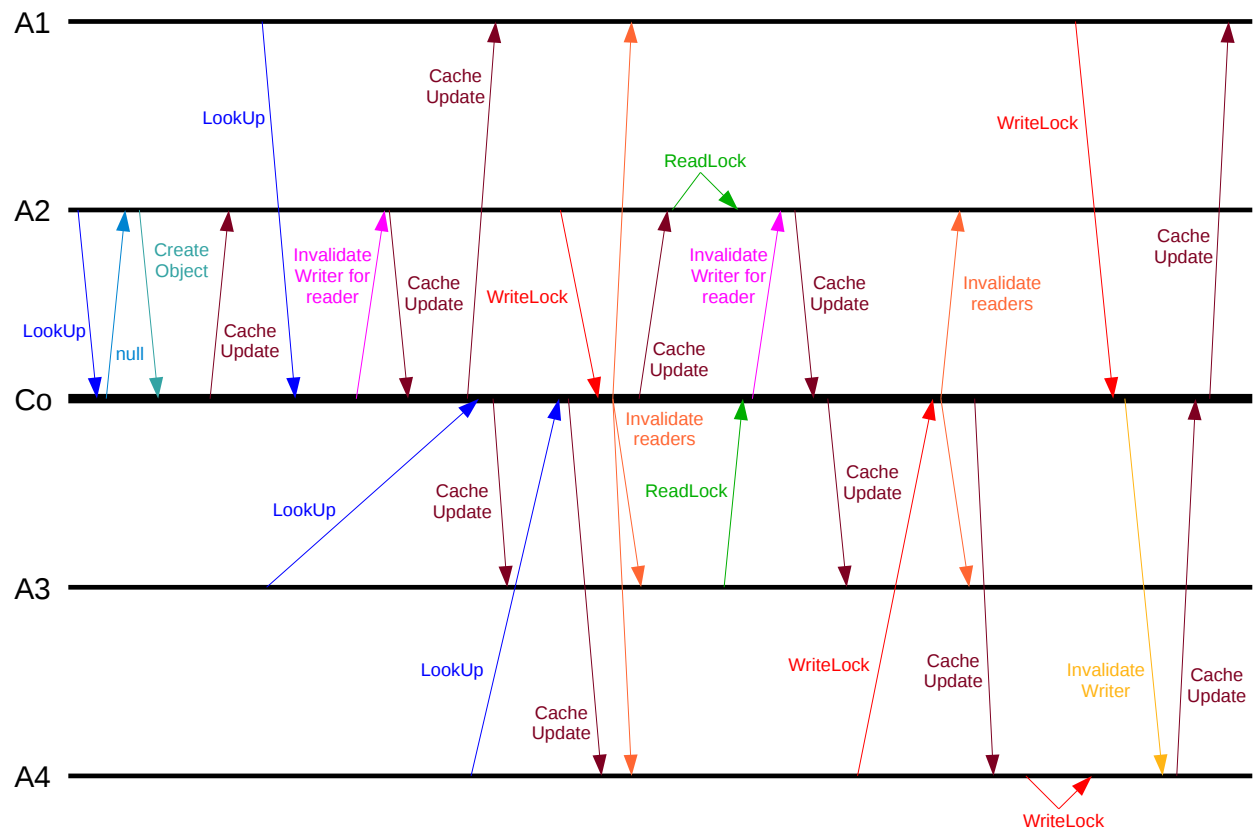
5 Exit the application

At the end of the execution of the application, the coordinator needs to be notify in order to remove the server from his tables. Use the following instruction before ending the execution :

```
JvnServerImpl.jvnGetServer().jvnTerminate();
```

6 Simplified diagram of how it works

The following diagram show how the cache and the coordinator are used.

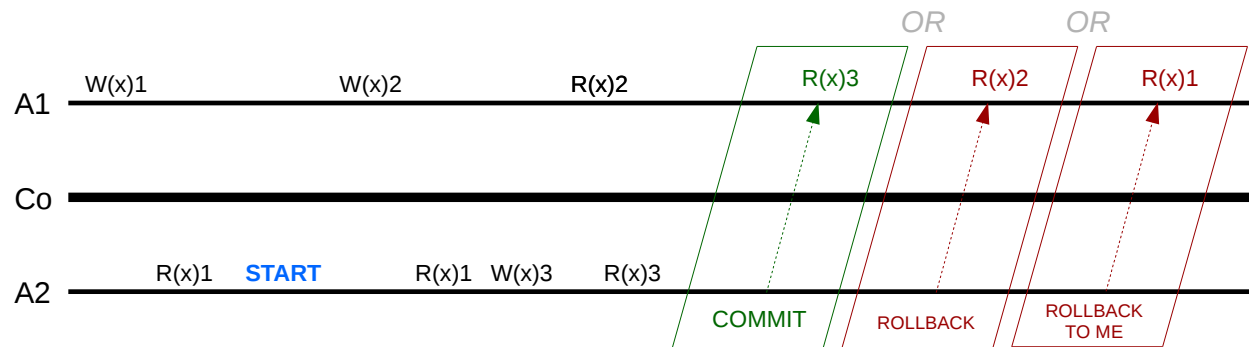


Additional features

7 Transaction

The JAVANAISE library include a basic transaction system. The purpose is to be able to realize multiple call of the functions of the object without any modification coming from the other servers. Four functions can be call for each object :

- `start()` → This function starts the transaction. After the call to `start()`, the updated state of the object will not be send to the coordinator until the end of the transaction.
- `commit()` → This function validates the transaction. The current state of the object is send to the coordinator.
- `rollback()` → This function cancels the transaction. The current state is forgotten and replaced by the state known by the coordinator.
- `rollbackToMe()` → This function cancels the transaction. The current state is forgotten and replaced by the state saved at the moment of the call to `start()`. This state is also send to the coordinator.



8 Coordinator breakdown handling

8.1 No coordinator available at launch

If the application can't get the coordinator when starting, a mechanism of wait is set up. Every 5 seconds, the application test again if the coordinator is reachable. The execution starts again normally when the test is successfull.

8.2 Breakdown of the coordinator during execution

At any moment of the execution, if the application can't reach the coordinator, the message *"Breakdown of the coordinator"* is shown and the application starts to wait the coordinator. A notification is send to the application when the coordinator is restart and the execution starts again normally.

- ⚠ The coordinator needs to be restart with `start(TRUE)`.
- ⚠ The updated state of an object can be lost because of a breakdown.