# Code Inspection

Software Engineering 2

Authors:

Hasan Aliyev
Madat Mustafayev

Milan —2015
Keynote

# Contents

1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.1 Name Convention . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2 Indention . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.3 Braces . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.4 File Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.5 Wrapping Lines. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.6 Commands. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.7 Java Source Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.8 Package and Import Statements. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.9 Class and Interface Declarations . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.10 Method Calls. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.11 Arrays . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.12 Object Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.13 Output Format . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

# 1. Introduction

- ***Classes that were assigned to the group***

The name of class which is assigned to us is **StatefullSessionContainer**

- ***Functional role of assigned set of classes***

**StatefullSessionContainer**

This class provides container functionality specific to Stateful SessionBeans. At deployment time, one instance of the StatefulSessionContainer is created for each stateful  SessionBean type (i.e. deployment descriptor) in a JAR.

There are 5 states of a Stateful EJB:

PASSIVE State - the container can passivate and activate the session bean instance. This usually occurs when the number of instances reaches a certain limit specified by the developer in the deployment descriptor. During this process, the container calls the session
bean's ejbPassivate and ejbActivate methods.

READY State - When a bean instance is in the ready state, it can service client request that is, execute component methods.

INVOKING - A session bean represents a single client inside the access an application that is deployed on the server, the client invokes the session bean's methods. The session bean performs work for its client, shielding the client from complexity by executing business tasks inside the server.

INCOMPLETE_TX : ready for invocations, transaction in progress

DESTROYED - Like the entity bean and stateless session bean, when a bean instance is in the Does Not Exist state, it is not an instance in the memory of the system. In other words, it has not been instantiated yet.

# 2. List of issues found by applying the checklist

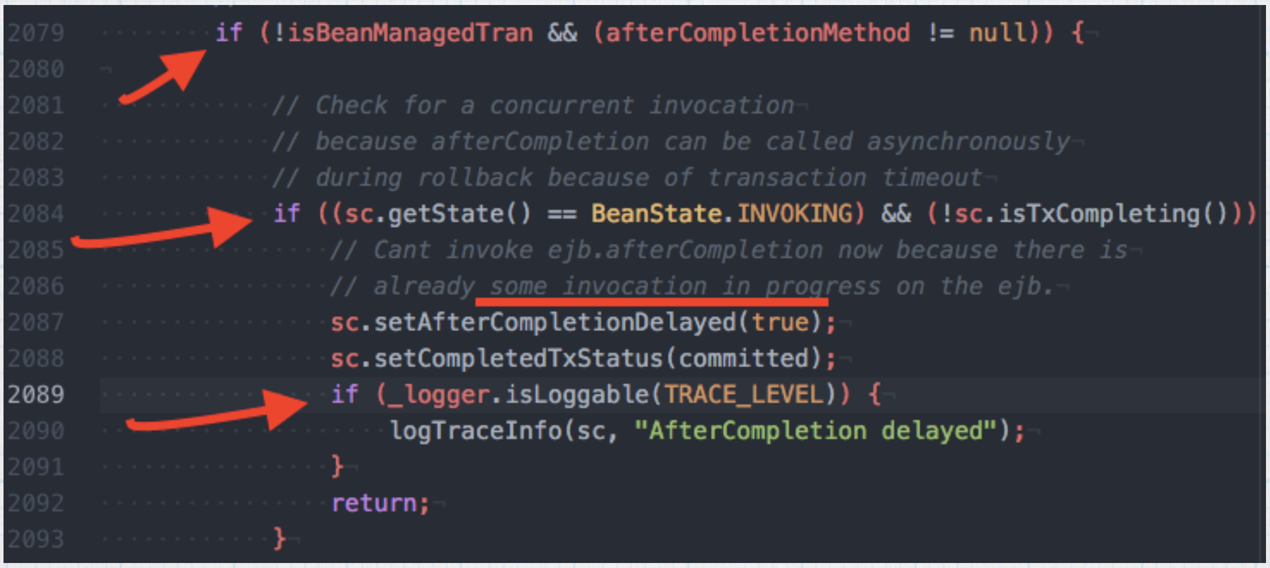
### 1. Naming Conventions

### Checklist (1-7)

We have checked all criteria for Naming Convention and all of them are correct written

### 2. Indention
### Checklist (8-9)

8. in our case developer have used 4 spaces of indentation and we found some mistakes. you can see it in below screenshot

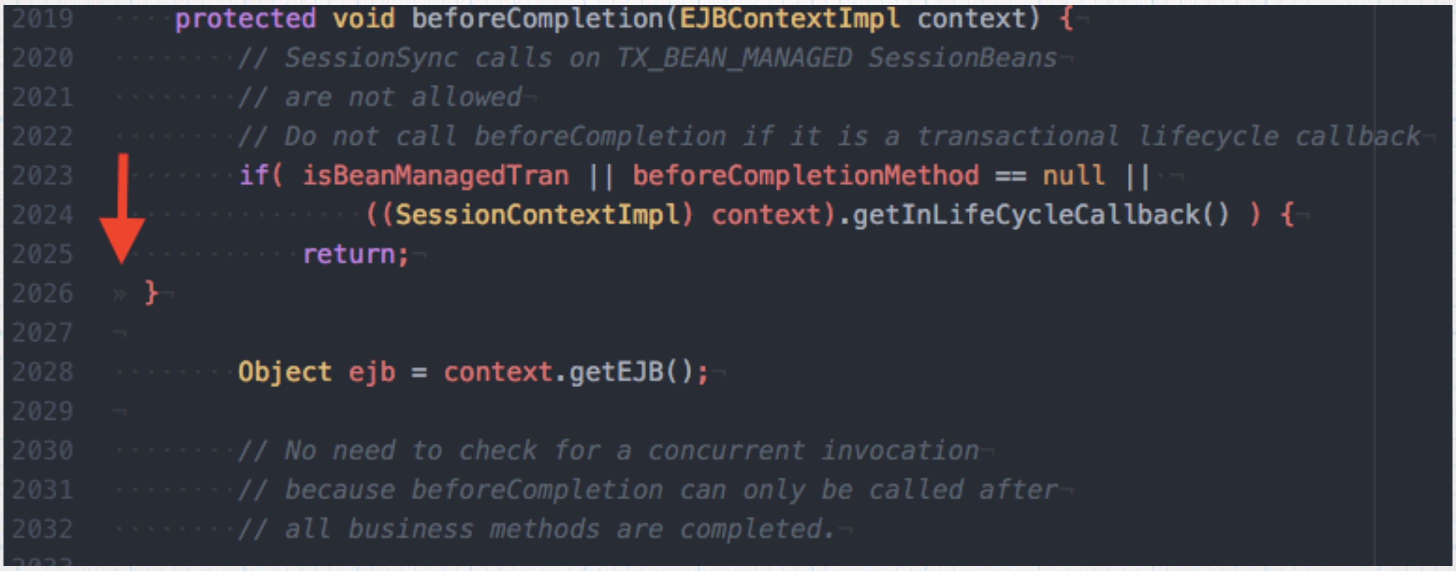


```
        if (!isBeanManagedTran && (afterCompletionMethod != null)) {

            // Check for a concurrent invocation
            // because afterCompletion can be called asynchronously
            // during rollback because of transaction timeout
            if ((sc.getState() == BeanState.INVOKING) && (!sc.isTxCompleting()))
                // Cant invoke ejb.afterCompletion now because there is
                // already some invocation in progress on the ejb.
                sc.setAfterCompletionDelayed(true);
                sc.setCompletedTxStatus(committed);
                if (_logger.isLoggable(TRACE_LEVEL)) {
                    logTraceInfo(sc, "AfterCompletion delayed");
                }
                return;
            }
```

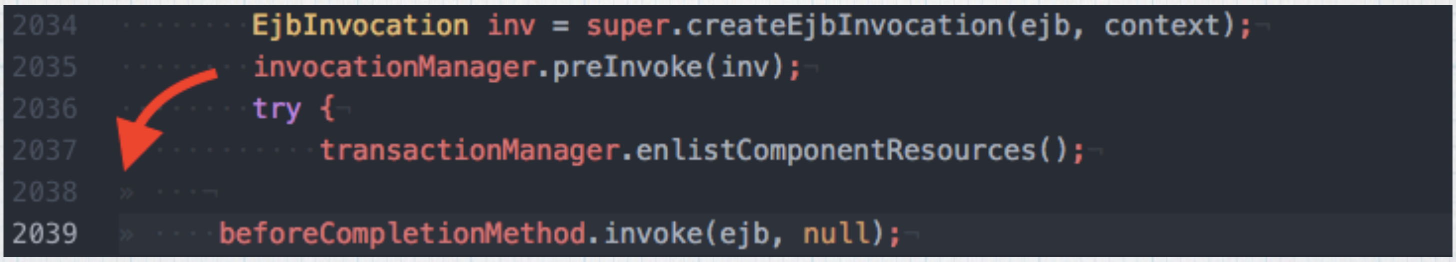
# 2. List of issues found by applying the checklist

***2. Indention***
***Checklist (8-9)***

9. Developer did mistake on line 2026 and 2038 writing tab instead of dot.



```
2019     protected void beforeCompletion(EJBContextImpl context) {
2020         // SessionSync calls on TX_BEAN_MANAGED SessionBeans
2021         // are not allowed
2022         // Do not call beforeCompletion if it is a transactional lifecycle callback
2023         if( isBeanManagedTran || beforeCompletionMethod == null ||
2024                 ((SessionContextImpl) context).getInLifeCycleCallback() ) {
2025             return;
2026 »   }
2027
2028         Object ejb = context.getEJB();
2029
2030         // No need to check for a concurrent invocation
2031         // because beforeCompletion can only be called after
2032         // all business methods are completed.
```



```
2034         EjbInvocation inv = super.createEjbInvocation(ejb, context);
2035         invocationManager.preInvoke(inv);
2036         try {
2037             transactionManager.enlistComponentResources();
2038 »   
2039 »   beforeCompletionMethod.invoke(ejb, null);
```

# 2. List of issues found by applying the checklist

***3. Braces***

***Checklist (10-11)***

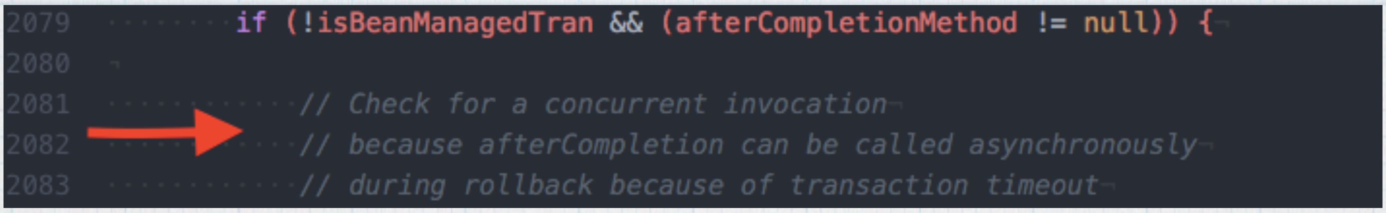
10. In our case consistent bracing style is preferred as «Kernigan and Ritchie» and we didn't find any mistake

11. All if, while, do-while, try-catch and for statements that have only one statement to execute are surrounded by curly braces.

***4. File organization***

***Checklist (12-14)***

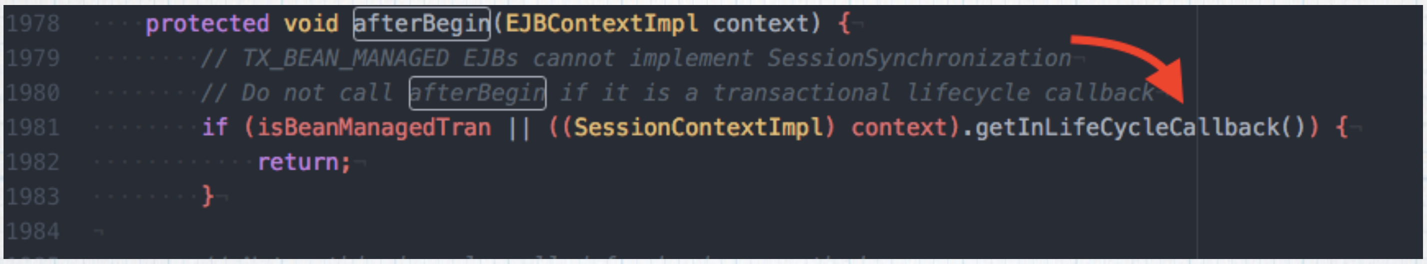
12.In generally comments written very well but we found out that, Author did use only // even with large comments instead of /..../



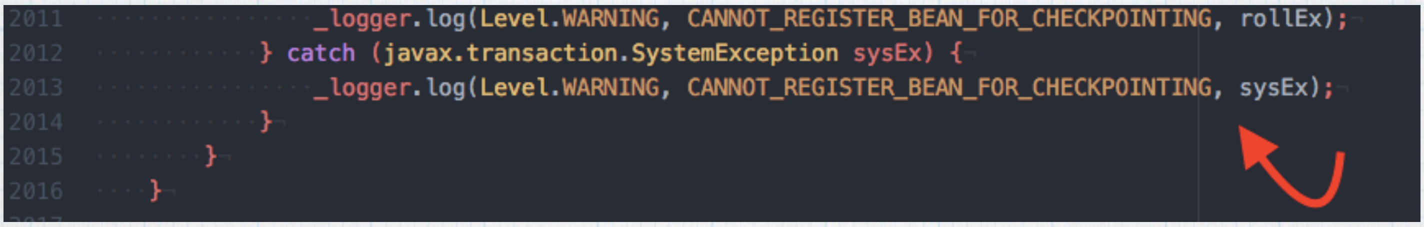
```
        if (!isBeanManagedTran && (afterCompletionMethod != null)) {

            // Check for a concurrent invocation
            // because afterCompletion can be called asynchronously
            // during rollback because of transaction timeout
```

# 2. List of issues found by applying the checklist

13. Practical line length exceed 80 characters. in below snaps (line 1981 and 2013)



```
1978    protected void afterBegin(EJBContextImpl context) {
1979        // TX_BEAN_MANAGED EJBs cannot implement SessionSynchronization
1980        // Do not call afterBegin if it is a transactional lifecycle callback
1981        if (isBeanManagedTran || ((SessionContextImpl) context).getInLifeCycleCallback()) {
1982            return;
1983        }
1984
```



```
2011                _logger.log(Level.WARNING, CANNOT_REGISTER_BEAN_FOR_CHECKPOINTING, rollEx);
2012            } catch (javax.transaction.SystemException sysEx) {
2013                _logger.log(Level.WARNING, CANNOT_REGISTER_BEAN_FOR_CHECKPOINTING, sysEx);
2014            }
2015        }
2016    }
```

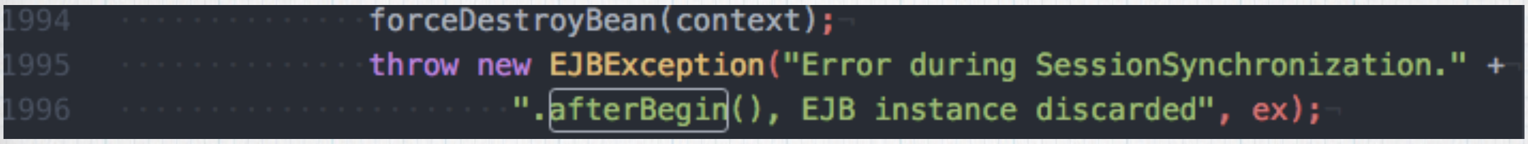
14.In our class we did not find where line exceed 120 characters

# 2. List of issues found by applying the checklist

***5. Wrapping Lines***

***Checklist  (15-17)***

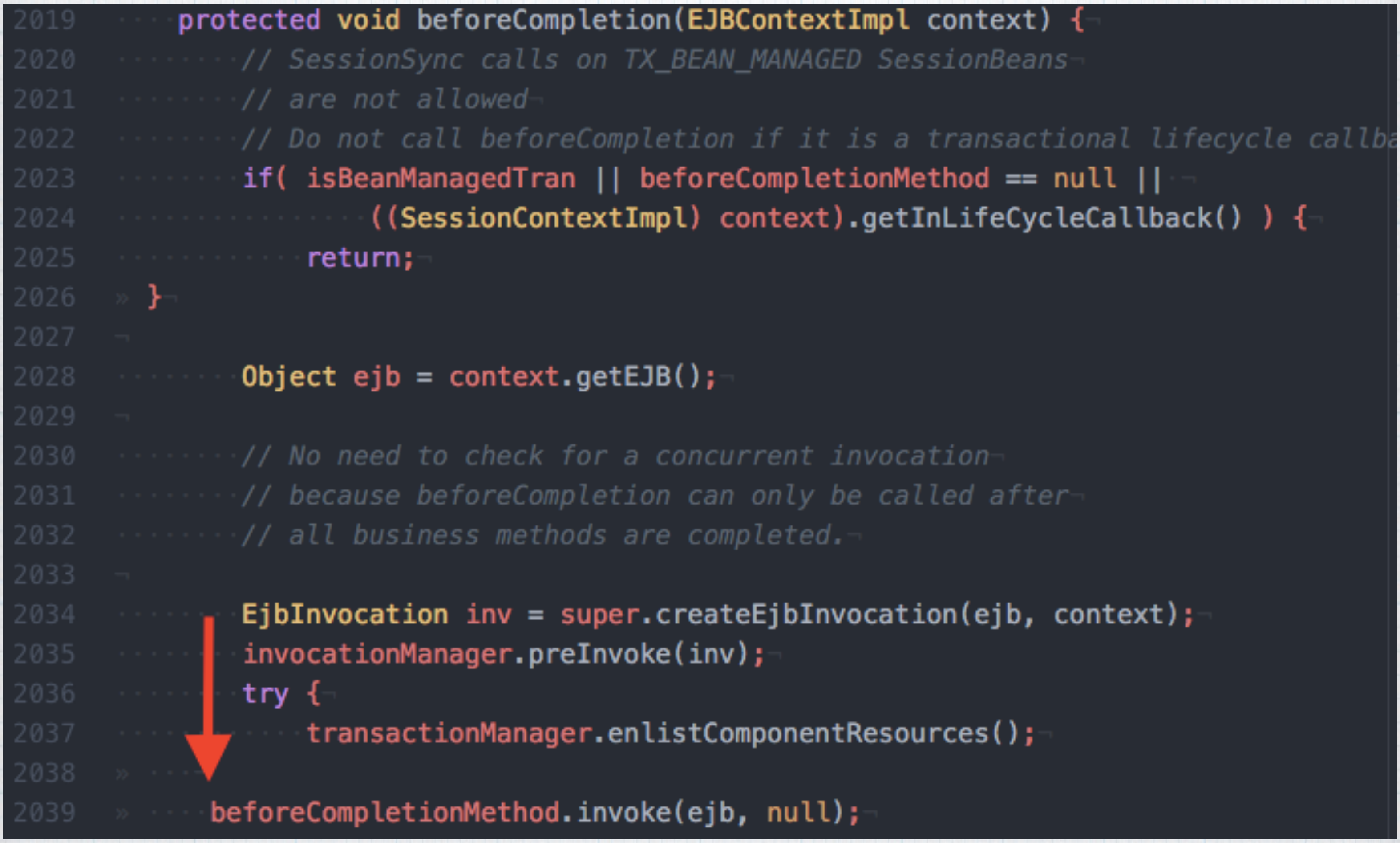
15. We found mistake on  line 1995. According  breaking an arithmetic expression. The first is preferred, since he break occurs outside the parenthesised expression, which is at a higher level



```
1994                forceDestroyBean(context);
1995                throw new EJBException("Error during SessionSynchronization." +
1996                        ".afterBegin(), EJB instance discarded", ex);
```

# 2. List of issues found by applying the checklist

17.In the method beforeCompletion we found out 1 mistake. Line 2039



```
2019    protected void beforeCompletion(EJBContextImpl context) {
2020        // SessionSync calls on TX_BEAN_MANAGED SessionBeans
2021        // are not allowed
2022        // Do not call beforeCompletion if it is a transactional lifecycle callba
2023        if( isBeanManagedTran || beforeCompletionMethod == null ||
2024                ((SessionContextImpl) context).getInLifeCycleCallback() ) {
2025            return;
2026  }
2027
2028        Object ejb = context.getEJB();
2029
2030        // No need to check for a concurrent invocation
2031        // because beforeCompletion can only be called after
2032        // all business methods are completed.
2033
2034        EjbInvocation inv = super.createEjbInvocation(ejb, context);
2035        invocationManager.preInvoke(inv);
2036        try {
2037            transactionManager.enlistComponentResources();
2038
2039      beforeCompletionMethod.invoke(ejb, null);
```

# 2. List of issues found by applying the checklist

***6. Comments***

***Checklist (18-19)***

18.We have checked of all our comments and we state that  Our method contains sufficient comments

19. In our method we did not find out codes which is commented.

# 2. List of issues found by applying the checklist

***7. Java Source Files***

***Checklist (20-23)***

20. In our cases Java source file contains a single public class.

21. In our case the public class is the first class or interface in the file.

23. In our case Author did not use Javadoc

# 2. List of issues found by applying the checklist

### *8. Package and Import Statements*

### *Checklist 24*

24. In our case 1st line is package

### *9. Class and Interface Declarations*

### *Checklist (25-27)*

Question 25.

A. Documentation comments are in the top of our file.

B. We have classes not interfaces

C. We don’t have implementation comments

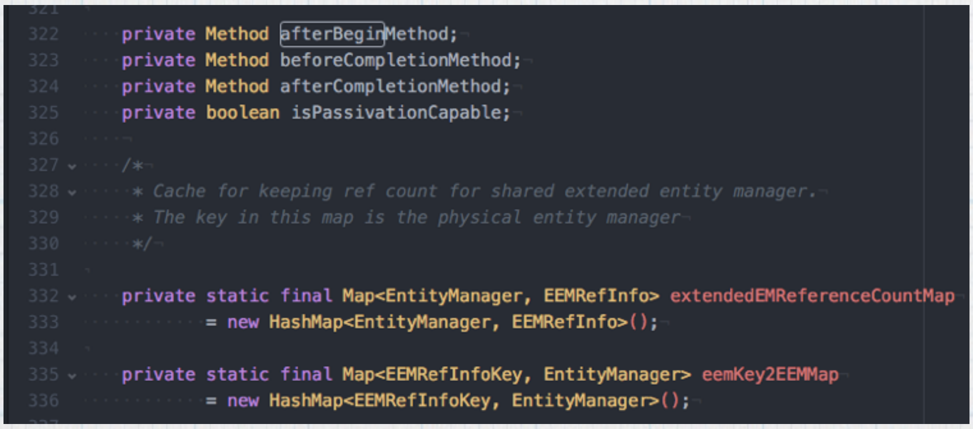
D. We have only static attributes

E. We found on line 332 and 335 static variables within order of instance variables and in order to be correct that variables need to be go up.

F. It is also correct. Because Constructor is after variables on line 343.

G. It is also correct

27.We have used software namely «IntelliJIDEA» for finding any inconsistency and eventually we did not find any duplicate



```
322     private Method afterBeginMethod;
323     private Method beforeCompletionMethod;
324     private Method afterCompletionMethod;
325     private boolean isPassivationCapable;
326 
327     /*
328      * Cache for keeping ref count for shared extended entity manager.
329      * The key in this map is the physical entity manager
330      */
331 
332     private static final Map<EntityManager, EEMRefInfo> extendedEMReferenceCountMap
333             = new HashMap<EntityManager, EEMRefInfo>();
334 
335     private static final Map<EEMRefInfoKey, EntityManager> eemKey2EEMMap
336             = new HashMap<EEMRefInfoKey, EntityManager>();
```

## 2. List of issues found by applying the checklist

***9. Initialization and Declarations***

***Checklist (28-33)***

28.After using code inspection tool «IntelliJIDEA» we can tell that it is correct

29.Our variables declared properly

All parameters is correct for the Initialisation and Declarations parts

***10. Method Calls***

***Checklist (34-36)***

All parameters is correct for the Method Calls

***11. Arrays***

***Checklist (37-39)***

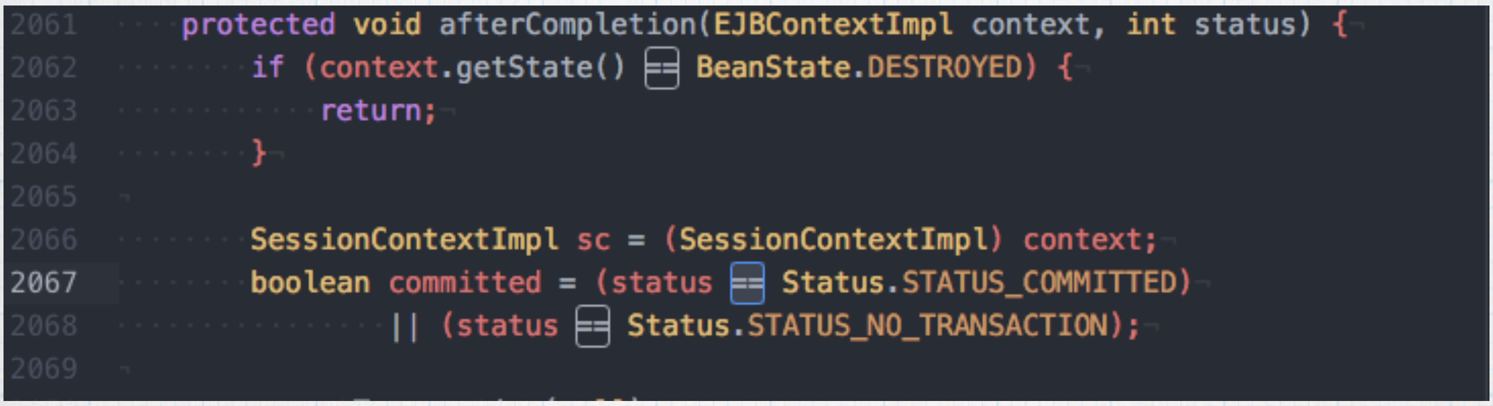
In our method we didn't find arrays

# 2. List of issues found by applying the checklist

***12. Object Comparison***

***Checklist 40***

Mainly it is correct but on line 2062 we found error. Because  it is comparison between 2 objects not with primitive type



```
2061     protected void afterCompletion(EJBContextImpl context, int status) {
2062         if (context.getState() == BeanState.DESTROYED) {
2063             return;
2064         }
2065
2066         SessionContextImpl sc = (SessionContextImpl) context;
2067         boolean committed = (status == Status.STATUS_COMMITTED)
2068                 || (status == Status.STATUS_NO_TRANSACTION);
2069
```

# 2. List of issues found by applying the checklist

***13. Output Format***

***Checklist 41-43***

After using code inspection tool «IntelliJIDEA» we did not find any spelling error