POLITECNICO
MILANO 1863

Politecnico di Milano
2016-2017 Software Engineering 2 Project:
PowerEnJoy
Code Inspection

Version 1.0

Alessandro Polenghi 879111

Alessandro Terragni 879112

1 Assigned Class

1.1 Class path

../apache-ofbiz-16.11.01/specialpurpose/passport/src/main/java/org/apache/ofbiz/passport/user/**GitHubAuthenticator.java**

1.2 Functional role of the class

This class is used to manage the registration, the login and the logout into Apache OFBi service through a GitHub Account.
It implements the Interface Authenticator, that is implemented also by LinkedInAuthenticator, TestFailAuthenticator, TestPassAuthenticator.

Methods:

- initiate(LocalDispatcher dispatcher)

  After being created with the default constructor, inititiate() must be called to initialize a dispatcher and a delegator.

- authenticate(String userLoginId, String password, boolean isServiceAuth) throws AuthenticatorException

  This method is used to authenticate a user trough its GitHubAccount.
  It checks if the userLoginId passed as parameter has an authId that has a valid accessToken in GitHubUser entity.
  If the condition is satisfied, it saves the accessToken and retrieves the JsonMap userInfo, calling the method getUserInfo.
  At the end it returns true (false) if the user is authenticated (or not), but in the meanwhile it can throw this exception:
  org.apache.ofbiz.common.authentication.api.AuthenticatorException,
  if a fatal error occurs during authentication

- logOut(String username) throws AuthenticatorException

  The method is not implemented but probably it should let the user log out from the system

- public void syncUser(String userLoginId) throws AuthenticatorException

  It reads the user information and syncs it to OFBiz (i.e. UserLogin, Person, etc)

  It retrieves the Jason userMap calling the method getGitHubUserInfo(), then, after suspending the current transaction, it begins a new transaction.
  If the user doesn't exist, it creates a new one calling the method createUser(), if the user already exists, it just updates it calling the method updateUser().

  At the end it always tries to commit the new transaction and then it resumes the suspended transaction.

  If the user synchronization fails, it throws this exception: org.apache.ofbiz.common.authentication.api.AuthenticatorExcept ion

- private Map<String, Object> getGitHubUserinfo(String userLoginId) throws AuthenticatorException

  This method performs exactly the same operations of the method authenticate.
  The only difference is that this method return the user, not just true or false.
  This is a repetition that will be formally reported in the code Inspection Checklist (paragraph 2.9.3).

- public String createUser(Map<String, Object> userMap) throws AuthenticatorException

    It just calls createUser(userMap, system), throwing this exception:
    AuthenticatorException(e.getMessage(), e)
    If the delegators fails calling findOne().

- private String createUser(Map<String, Object> userMap, GenericValue system) throws AuthenticatorException

    Firstly it generates a userLoginID, then it assigns (if present) the associated gitHubName and login as lastName and externalAuthId of the OFBiz's profile it is creating.
    Then it assigns the UserLoginId and the password passed as parameters.
    After creating the actual profile, it assigns the role of Customer and it associates a new email and a security group Map to it.
    At the end it returns the userID but in the meanwhile it can call several exceptions if something goes wrong.

- private void updateUser(Map<String, Object> userMap, GenericValue system, GenericValue userLogin) throws AuthenticatorException

    This method has not been implemented yet.

- public void updatePassword(String username, String password, String newPassword) throws AuthenticatorException

    This method has not been implemented yet.

- public float getWeight()

    This method has not been implemented yet, up to now it always returns 1

- public boolean isUserSynchronized()

  This method has not been implemented yet, up to now it always returns true.

- public boolean isSingleAuthenticator()

  This method has not been implemented yet, up to now it always returns false.

- public boolean isEnabled()

  It returns:
  "true".equalsIgnoreCase(UtilProperties.getPropertyValue(props, "github.authenticator.enabled", “true”));

  That simply check if the the method getPropertyValue() returns true, ignoring the upper and lower case distinction.

- public static Map<String, Object> getUserInfo(HttpGet httpGet, String accessToken, String tokenType, Locale locale) throws AuthenticatorException

  This method is the one that actually connects to GitHub and it retrieves and returns the JsonMap of the user.
  It can launch several exceptions due to connection problems with GItHub.

All the analysis has been driven by the JavaDoc of the class and the official documentation of the Apache OFBi.

2 Code Inspection checklist

Notation: [ ] is used to refer to the row in the original class file

2.1 Naming Conventions

2.1.1
All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.

These variables should have more meaningful names:

[148] GenericValue system;
[163] Transaction parentTx;
[232] GenericValue system;

2.1.2
If one-character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.

Satisfied

2.2.3
Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: class Raster; class ImageSprite;

Satisfied

2.2.4
Interface names should be capitalized like classes.

Satisfied

2.2.5
Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().

[206] private Map<String, Object> getGitHubUser**info**(String userLoginId)

info should start with a capital I

2.2.6
Class variables, also called attributes, are mixed case, but might begin with an underscore (‘ _’) followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized. Examples: _windowHeight, timeSeriesData.

Satisfied

2.2.7
Constants are declared using all uppercase with words separated by an underscore. Examples: MIN WIDTH; MAX HEIGHT.

These constants are all lowercase:

[65] private static final String **module** =
GitHubAuthenticator.class.getName();

[67] public static final String **props** = "gitHubAuth.properties";

[69] public static final String **resource** = "PassportUiLabels";

2.2 Indention

2.2.1
Three or four spaces are used for indentation and done so consistently.

Tabs indentation is used in the whole class

2.2.2
No tabs are used to indent.

Tabs indentation is used in the whole class

2.3 Braces

2.3.1
Consistent bracing style is used, either the preferred “Allman” style (first brace goes underneath the opening block) or the “Kernighan and Ritchie” style (first brace is on the same line of the instruction that opens the new block).

Satisfied: “Kernighan and Ritchie” style is used in this class.

2.3.2
All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.

Satisfied

2.4 File Organization

2.4.1
Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).

Satisfied

2.4.2
Where practical, line length does not exceed 80 characters.

NOT Satisfied in:
Line 100 length: 119
Line 104 length: 107
Line 106 length: 125
Line 112 length: 95
Line 150 length: 87
Line 157 length: 106
Line 190 length: 84
Line 200 length: 91
Line 206 length: 101
Line 218 length: 95
Line 229 length: 89
Line 239 length: 111
Line 241 length: 82
Line 257 length: 87
Line 262 length: 82
Line 267 length: 120
Line 284 length: 82
Line 289 length: 81
Line 295 length: 123
Line 299 length: 94
Line 314 length: 88
Line 319 length: 82
Line 326 length: 129
Line 339 length: 112
Line 376 length: 111
Line 379 length: 147
Line 383 length: 84
Line 393 length: 130

2.4.3
When line length must exceed 80 characters, it does NOT exceed 120 characters.

> **NOT** Satisfied in:
> Line 106 length: 125
> Line 295 length: 123
> Line 326 length: 129
> Line 379 length: 147
> Line 393 length: 130

2.5 Wrapping Lines

2.5.1
Line break occurs after a comma or an operator.

> Satisfied

2.5.2
Higher-level breaks are used.

> Satisfied

2.5.3
A new statement is aligned with the beginning of the expression at the same level as the previous line.

> Satisfied

2.6 Comments

2.6.1
Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

> Very general and there are a lot of blocks not explained.
> [136] no comment about empty method bodies

2.6.2
Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.

Satisfied

2.7 Java Source Files

2.7.1
Each Java source file contains a single public class or interface.

Satisfied

2.7.2
The public class is the first class or interface in the file.

Satisfied

2.7.2
Check that the external program interfaces are implemented consistently with what is described in the javadoc.

Satisfied

2.7.3
Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

```
[60]
/**
 * GitHub OFBiz Authenticator
 */
public class GitHubAuthenticator implements Authenticator
```

This javaDoc should be more precise, explaining what is the main purpose of the class.

[139]

```
/**
 * Reads user information and syncs it to OFBiz (i.e. UserLogin, Person,
etc)
 *
 * @param userLoginId
 * @throws
org.apache.ofbiz.common.authentication.api.AuthenticatorException
 *          user synchronization fails
 */
   public void syncUser(String userLoginId) throws
AuthenticatorException
```

Even if it is self explanatory, the @param userLoginId should be explained

These methods don’t have any JavaDoc:

[206]

```
private Map<String, Object> getGitHubUserinfo(String userLoginId)
throws AuthenticatorException
```

[229]

```
public String createUser(Map<String, Object> userMap) throws
AuthenticatorException
```

[239]

```
private String createUser(Map<String, Object> userMap, GenericValue
system) throws AuthenticatorException
```

[326]

```
private void updateUser(Map<String, Object> userMap, GenericValue
system, GenericValue userLogin) throws AuthenticatorException
```

[379]

```
public static Map<String, Object> getUserInfo(HttpGet httpGet, String
accessToken, String tokenType, Locale locale) throws
AuthenticatorException
```

2.8 Package and Import Statements

2.8.1
If any package statements are needed, they should be the first non- comment statements. Import statements follow.

Satisfied

2.9 Class and Interface Declarations

2.9.1
The class or interface declarations shall be in the following order:

(a) class/interface documentation comment;
(b) class or interface statement;
(c) class/interface implementation comment, if necessary;
(d) class (static) variables;
 i. first public class variables;
 ii. next protected class variables;
 iii. next package level (no access modifier);
 iv. last private class variables.
(e) instance variables;
 i. first public instance variables;
 ii. next protected instance variables;
 iii. next package level (no access modifier);
 iv. last private instance variables.
(f) constructors;
(g) methods.

[63] public class GitHubAuthenticator implements Authenticator

the order of the class static variables is wrong: private variables are before protected ones

2.9.2
Methods are grouped by functionality rather than by scope or accessibility.

Methods getGitHubUserInfo [206] and getUserInfo[379] should be grouped close to each other

2.9.3
Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

The method createUser [239] is too long and should be reduced or split.

The method getGitHubUserinfo [206] is the exact repetition of the method authenticate [100], the only thing that differs is the return statement.
Moreover, in the method authenticate the parameters: password, isServiceAuth are never used, maybe because a part of the implementation is missing.

The method public String createUser(Map<String, Object> userMap) [229] just calls the method createUser(Map<String, Object> userMap, GenericValue system) [239], so it should be mixed with it.

The class is characterized by 'loose coupling', the class implements the Authenticator interface and it is quite modular and detached from the other classes.
At the same time cohesion is high, the single responsibility principle is applied: each method has a precise task.
Encapsulation is preserved: objects are protected and are not exposed to modifications by other classes, the majority of methods are private.

2.10 Initialization and Declarations

2.10.1
Check that variables and class members are of the correct type.
Check that they have the right visibility (public/private/protected).

Satisfied

2.10.2
Check that variables are declared in the proper scope.

Satisfied

2.10.3
Check that constructors are called when a new object is desired.

Satisfied

2.10.4
Check that all object references are initialized before use.

These objects are declared but not immediately initialized:

[148] GenericValue system;
[155] GenericValue userLogin;
[230] GenericValue system;
[255] Map<String, Object> createPersonResult;
[282] Map<String, Object> createEmailResult;
[312] Map<String, Object> createSecGrpResult;
[410] Map<String, Object> userMap;

2.10.5
Variables are initialized where they are declared, unless dependent upon a computation.

Satisfied

2.10.6
Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces '{' and '}'). The exception is a variable can be declared in a for loop.

These variables are not declared at the beginning of a block:

[155] GenericValue userLogin;
[163] Transaction parentTx = null;
[164] boolean beganTransaction = false;
[255] Map<String, Object> createPersonResult;
[264] String partyId = (String) createPersonResult.get("partyId");
[267] GenericValue partyRole = delegator.makeValue("PartyRole", UtilMisc.toMap("partyId", partyId, "roleTypeId", "CUSTOMER"));
[282] Map<String, Object> createEmailResult;
[294] Timestamp now = UtilDateTime.nowTimestamp();
[312] Map<String, Object> createSecGrpResult;
[385] CloseableHttpResponse getResponse = null;

```
[409] JSONToMap jsonMap = new JSONToMap();
[410] Map<String, Object> userMap;
```

2.11 Method Calls

2.11.1
Check that parameters are presented in the correct order.

Satisfied

2.11.2
Check that the correct method is being called, or should it be a different method with a similar name.

Satisfied

2.11.3
Check that method returned values are used properly.

Satisfied

2.12 Arrays

2.12.1
Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).

Satisfied

2.12.2
Check that all array (or other collection) indexes have been prevented from going out-of-bounds.

Satisfied

2.12.3
Check that constructors are called when a new array item is desired.

Satisfied

2.13 Object Comparison

2.13.1
Check that all objects (including Strings) are compared with equals and not with ==.

In line 176 and line 389, == are used for comparison.

2.14 Output Format

2.14.1
Check that displayed output is free of spelling and grammatical errors.

Satisfied

2.14.2
Check that error messages are comprehensive and provide guidance as to how to correct the problem.

[125] Use of abbreviation may threat the comprehension

2.14.3
Check that the output is formatted correctly in terms of line stepping and spacing.

Satisfied

2.15 Computation, Comparisons and Assignments

2.15.1
Check that the implementation avoids “brutish programming”: (see http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html).

[376]
return "true".equalsIgnoreCase(UtilProperties.getPropertyValue(props, "github.authenticator.enabled", “true"));

it should return only UtilProperties.getPropertyValue(props, "github.authenticator.enabled", “true”)

2.15.2
Check order of computation/evaluation operator precedence and parenthesizing.

Satisfied

2.15.3
Check the liberal use of parenthesis is used to avoid operator precedence problems.

Satisfied

2.15.4
Check that all denominators of a division are prevented from being zero.

Satisfied

2.15.5
Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.

Satisfied

2.15.6
Check that the comparison and Boolean operators are correct.

Satisfied

2.15.7
Check throw-catch expressions, and check that the error condition is actually legitimate.

Satisfied

2.15.8
Check that the code is free of any implicit type conversions.

In the class, casting is widely used.
It's better to avoid it to improve clarity, performance and maintainability of the code.

2.16 Exceptions

2.16.1
Check that the relevant exceptions are caught.

Satisfied

2.16.2
Check that the appropriate action are taken for each catch block.

[405] catch block is empty
Other exceptions are caught throwing new exceptions, we can't say nothing more about the appropriateness of the actions.

2.17 Flow of Control

2.17.1
In a switch statement, check that all cases are addressed by break or return.

Satisfied (switch statements are not present in the class)

2.17.2
Check that all switch statements have a default branch.

Satisfied (switch statements are not present in the class)

2.17.3
Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

Satisfied

2.18 Files

2.18.1
Check that all files are properly declared and opened.

Satisfied (files are not present in the class)

2.18.2
Check that all files are closed properly, even in the case of an error.

Satisfied (files are not present in the class)

2.18.3
Check that EOF conditions are detected and handled correctly.

Satisfied (files are not present in the class)

2.18.4
Check that all file exceptions are caught and dealt with accordingly.

Satisfied (files are not present in the class)

3 Appendix

3.1 Division Of Work

The code inspection checklist has been equally divided in two parts:

- Alessandro Polenghi :
  2.3 2.4 2.5 2.6 2.11 2.12 2.13 2.14 2.16 2.17 2.18

- Alessandro Terragni:
  2.1 2.2 2.7 2.8 2.9 2.10 2.15

3.2 Hours of work

- Alessandro Polenghi : 7h

- Alessandro Terragni: 8h