BasicPoolingRepository

org.hippoecm.hst.core.jcr.pool.BasicPoolingRepository

Coverage | Dependencies | Duplications | LCOM4 | Sources | **Violations**

0 | 3 | 49 | 8 | 0 | Filter: [redacted] | Expand:

```
43  public class BasicPoolingRepository implements PoolingRepository, MultipleRepositoryAware {
44
45      private static Logger log = LoggerFactory.getLogger(BasicPoolingRepository.class);
46
47      protected Repository repository;
```
Visibility Modifier : Variable 'repository' must be private and have accessor methods.

```
48      protected Credentials defaultCredentials;           // credentials provided by the configuration or the user
```
Visibility Modifier : Variable 'defaultCredentials' must be private and have accessor methods.

```
49      protected Credentials internalDefaultCredentials;   // credentials used for real JCR API invocations.
```
Visibility Modifier : Variable 'internalDefaultCredentials' must be private and have accessor methods.

```
50      protected boolean isSimpleDefaultCredentials;
```
Visibility Modifier : Variable 'isSimpleDefaultCredentials' must be private and have accessor methods.

```
51      protected SessionDecorator sessionDecorator;
```
Visibility Modifier : Variable 'sessionDecorator' must be private and have accessor methods.

```
52
53      protected String repositoryProviderClassName = "org.hippoecm.hst.core.jcr.pool.JcrHippoRepositoryProvider";
```
Visibility Modifier : Variable 'repositoryProviderClassName' must be private and have accessor methods.

```
54      protected JcrRepositoryProvider jcrRepositoryProvider;
```
Visibility Modifier : Variable 'jcrRepositoryProvider' must be private and have accessor methods.

```
55      protected String repositoryAddress;
```
Visibility Modifier : Variable 'repositoryAddress' must be private and have accessor methods.

```
56      protected String defaultCredentialsUserID;
```
Visibility Modifier : Variable 'defaultCredentialsUserID' must be private and have accessor methods.

```
57      protected String defaultCredentialsUserIDSeparator = "@";
```
Visibility Modifier : Variable 'defaultCredentialsUserIDSeparator' must be private and have accessor methods.

```
58      protected char [] defaultCredentailsPassword;
```
Visibility Modifier : Variable 'defaultCredentailsPassword' must be private and have accessor methods.

```
59      protected String defaultWorkspaceName;
```
Visibility Modifier : Variable 'defaultWorkspaceName' must be private and have accessor methods.

```
60
61      protected boolean refreshOnPassivate = true;
```
Visibility Modifier : Variable 'refreshOnPassivate' must be private and have accessor methods.

```
62      protected long maxRefreshIntervalOnPassivate;
```
Visibility Modifier : Variable 'maxRefreshIntervalOnPassivate' must be private and have accessor methods.

```
63      protected boolean keepChangesOnRefresh = false;
```
Visibility Modifier : Variable 'keepChangesOnRefresh' must be private and have accessor methods.

```
64      protected long sessionsRefreshPendingTimeMillis;
```
Visibility Modifier : Variable 'sessionsRefreshPendingTimeMillis' must be private and have accessor methods.

```
65      protected ResourceLifecycleManagement pooledSessionLifecycleManagement;
```
Visibility Modifier : Variable 'pooledSessionLifecycleManagement' must be private and have accessor methods.

```
66      protected MultipleRepository multipleRepository;
```
Visibility Modifier : Variable 'multipleRepository' must be private and have accessor methods.

```
67
68      public void setRepositoryProviderClassName(String repositoryProviderClassName) {
69          this.repositoryProviderClassName = repositoryProviderClassName;
70      }
71
```

```
90      }
91
92      public void setDefaultCredentials(Credentials defaultCredentials) {
93          isSimpleDefaultCredentials = (defaultCredentials != null && (defaultCredentials instanceof SimpleCredentials));
94          internalDefaultCredentials = this.defaultCredentials = defaultCredentials;
```
Inner Assignment : Inner assignments should be avoided.

```
95
96          if (isSimpleDefaultCredentials) {
97              String userID = ((SimpleCredentials) defaultCredentials).getUserID();
98              String userIDOnly = StringUtils.substringBefore(userID, defaultCredentialsUserIDSeparator);
99
```

```
122     public String getDefaultCredentialsUserIDSeparator() {
123         return this.defaultCredentialsUserIDSeparator;
124     }
125
126     public void setDefaultCredentialsPassword(char [] defaultCredentailsPassword) {
```
Security - Array is stored directly : The user-supplied array 'defaultCredentailsPassword' is stored directly.

```
127         this.defaultCredentailsPassword = defaultCredentailsPassword;
```
Malicious code vulnerability - May expose internal representation by incorporating reference to mutable object : org.hippoecm.hst.core.jcr.pool.BasicPoolingRepository.setDefaultCredentialsPassword(char[]) may expose internal representation by storing an externally mutable object into BasicPoolingRepository.defaultCredentailsPassword
```
128     }
129
130     public char [] getDefaultCredentialsPassword() {
131         return this.defaultCredentailsPassword;
```
Malicious code vulnerability - May expose internal representation by returning reference to mutable object : org.hippoecm.hst.core.jcr.pool.BasicPoolingRepository.getDefaultCredentialsPassword() may expose internal representation by returning BasicPoolingRepository.defaultCredentailsPassword
```
132     }
133
134     public void setDefaultWorkspaceName(String defaultWorkspaceName) {
135         this.defaultWorkspaceName = defaultWorkspaceName;
136     }
```

```
186     public void setResourceLifecycleManagement(ResourceLifecycleManagement pooledSessionLifecycleManagement) {
187         this.pooledSessionLifecycleManagement = pooledSessionLifecycleManagement;
188
189         if (this.pooledSessionLifecycleManagement != null && this.pooledSessionLifecycleManagement instanceof PoolingRepositoryAware) {
190             ((PoolingRepositoryAware) this.pooledSessionLifecycleManagement).setPoolingRepository(this);
```
Dodgy - Unchecked/unconfirmed cast : Unchecked/unconfirmed cast from org.hippoecm.hst.core.ResourceLifecycleManagement to org.hippoecm.hst.core.jcr.pool.PoolingRepositoryAware in org.hippoecm.hst.core.jcr.pool.BasicPoolingRepository.setResourceLifecycleManagement(ResourceLifecycleManagement)
```
191         }
192     }
193
194     public ResourceLifecycleManagement getResourceLifecycleManagement() {
195         return this.pooledSessionLifecycleManagement;
```

```
223      * @throws LoginException
224      * @throws RepositoryException
225      * @return a read-only session
226      */
227     public Session login() throws LoginException, RepositoryException {
```
Redundant Throws : Redundant throws: 'LoginException' is subclass of 'RepositoryException'.
```
228         Session session = null;
229
230         try {
231             session = (Session) this.sessionPool.borrowObject();
```
Multithreaded correctness - Inconsistent synchronization : Inconsistent synchronization of org.hippoecm.hst.core.jcr.pool.BasicPoolingRepository.sessionPool; locked 85% of time
```
232
233             // If client retrieves a session, then register it as a disposable.
234             if (pooledSessionLifecycleManagement != null && pooledSessionLifecycleManagement.isActive()) {
235                 pooledSessionLifecycleManagement.registerResource(session);
236             }
237         } catch (NoSuchElementException e) {
238             throw new NoAvailableSessionException("No session is available now. Probably the session pool was exhasuted.");
```
Preserve Stack Trace : Caught exception is rethrown, original stack trace may be lost
```
239         } catch (Exception e) {
240             throw new LoginException("Failed to borrow session from the pool.", e);
241         }
242
243         return session;
```

```
249      * @throws LoginException
250      * @throws RepositoryException
251      * @return a writable session
252      */
253     public Session login(Credentials credentials) throws LoginException, RepositoryException {
```
Redundant Throws : Redundant throws: 'LoginException' is subclass of 'RepositoryException'.
```
254         if (equalsCredentials(credentials)) {
255             return login();
256         } else {
257             throw new LoginException("login by credentials other than the default is not supported.");
258         }
```

```
264      * @throws LoginException
265      * @throws RepositoryException
266      * @return a read-only session
267      */
268     public Session login(String workspaceName) throws LoginException, NoSuchWorkspaceException, RepositoryException {
```
Redundant Throws : Redundant throws: 'LoginException' is subclass of 'RepositoryException'.

Redundant Throws : Redundant throws: 'NoSuchWorkspaceException' is subclass of 'RepositoryException'.
```
269         return login();
```

```
270     }
271
272     /**
273      * <strong>BasicPoolingRepository does not support workspaceName parameter. So it returns a writable session.</strong>
```

```
275      * @throws LoginException
276      * @throws RepositoryException
277      * @return a writable session
278      */
279     public Session login(Credentials credentials, String workspaceName) throws LoginException,
```

Redundant Throws : Redundant throws: 'LoginException' is subclass of 'RepositoryException'.

```
280             NoSuchWorkspaceException, RepositoryException {
```

Redundant Throws : Redundant throws: 'NoSuchWorkspaceException' is subclass of 'RepositoryException'.

```
281         return login(credentials);
282     }
283
284     public void returnSession(Session session) {
285         try {
```

```
300     public void setMultipleRepository(MultipleRepository multipleRepository) {
301         this.multipleRepository = multipleRepository;
302     }
303
304     public Session impersonate(Credentials credentials) throws LoginException, RepositoryException {
```

Redundant Throws : Redundant throws: 'LoginException' is subclass of 'RepositoryException'.

```
305         Session session = null;
306
307         if (this.multipleRepository != null && this.multipleRepository.containsRepositoryByCredentials(credentials)) {
308             session = this.multipleRepository.login(credentials);
309         }
```

```
315
316     /**
317      * The object pool that internally manages our sessions.
318      */
319     protected GenericObjectPool sessionPool;
```

Visibility Modifier : Variable 'sessionPool' must be private and have accessor methods.

```
320     /**
321      * The maximum number of active sessions that can be allocated from
322      * this pool at the same time, or negative for no limit.
323      */
324     protected int maxActive = GenericObjectPool.DEFAULT_MAX_ACTIVE;
```

Visibility Modifier : Variable 'maxActive' must be private and have accessor methods.

```
325     /**
326      * The maximum number of sessions that can remain idle in the
327      * pool, without extra ones being released, or negative for no limit.
328      */
329     protected int maxIdle = GenericObjectPool.DEFAULT_MAX_IDLE;
```

Visibility Modifier : Variable 'maxIdle' must be private and have accessor methods.

```
330     /**
331      * The minimum number of active sessions that can remain idle in the
332      * pool, without extra ones being created, or 0 to create none.
333      */
334     protected int minIdle = GenericObjectPool.DEFAULT_MIN_IDLE;
```

Visibility Modifier : Variable 'minIdle' must be private and have accessor methods.

```
335     /**
336      * The behavior of borrowing a session when the pool is exhausted.
337      * <ul>
338      *   <li>
339      *     When {@link #whenExhaustedAction} is {@link PoolingRepository#WHEN_EXHAUSTED_FAIL},
```

```
353      *     If {@link #maxWait} is non-positive, borrowing will block indefinitely.
354      *   </li>
355      * </ul>
356      */
357     protected String whenExhaustedAction = WHEN_EXHAUSTED_BLOCK;
```

Visibility Modifier : Variable 'whenExhaustedAction' must be private and have accessor methods.

```
358     /**
359      * The initial number of sessions that are created when the pool
360      * is started.
361      */
362     protected int initialSize = 0;
```

Visibility Modifier : Variable 'initialSize' must be private and have accessor methods.

```
363     /**
364      * The maximum number of milliseconds that the pool will wait (when there
365      * are no available sessions) for a session to be returned before
366      * throwing an exception, or <= 0 to wait indefinitely.
367      */
368     protected long maxWait = GenericObjectPool.DEFAULT_MAX_WAIT;
```

Visibility Modifier : Variable 'maxWait' must be private and have accessor methods.

```
369     /**
370      * The indication of whether objects will be validated before being
371      * borrowed from the pool.  If the object fails to validate, it will be
372      * dropped from the pool, and we will attempt to borrow another.
373      */
374     protected boolean testOnBorrow = true;
```

Visibility Modifier : Variable 'testOnBorrow' must be private and have accessor methods.

```
375     /**
376      * The indication of whether objects will be validated before being
377      * returned to the pool.
378      */
379     protected boolean testOnReturn = false;
```

Visibility Modifier : Variable 'testOnReturn' must be private and have accessor methods.

```
380     /**
381      * The number of milliseconds to sleep between runs of the idle object
382      * evictor thread.  When non-positive, no idle object evictor thread will
383      * be run.
384      */
385     protected long timeBetweenEvictionRunsMillis = GenericObjectPool.DEFAULT_TIME_BETWEEN_EVICTION_RUNS_MILLIS;
```

Visibility Modifier : Variable 'timeBetweenEvictionRunsMillis' must be private and have accessor methods.

```
386     /**
387      * The number of objects to examine during each run of the idle object
388      * evictor thread (if any).
389      */
390     protected int numTestsPerEvictionRun = GenericObjectPool.DEFAULT_NUM_TESTS_PER_EVICTION_RUN;
```

Visibility Modifier : Variable 'numTestsPerEvictionRun' must be private and have accessor methods.

```
391     /**
392      * The minimum amount of time an object may sit idle in the pool before it
393      * is eligable for eviction by the idle object evictor (if any).
394      */
395     protected long minEvictableIdleTimeMillis = GenericObjectPool.DEFAULT_MIN_EVICTABLE_IDLE_TIME_MILLIS;
```

Visibility Modifier : Variable 'minEvictableIdleTimeMillis' must be private and have accessor methods.

```
396     /**
397      * The indication of whether objects will be validated by the idle object
398      * evictor (if any).  If an object fails to validate, it will be dropped
399      * from the pool.
400      */
401     protected boolean testWhileIdle = false;
```

Visibility Modifier : Variable 'testWhileIdle' must be private and have accessor methods.

```
402     /**
403      * The query that will be used to validate sessions from this pool
404      * before returning them to the caller.  If specified, this query
405      * <strong>MUST</strong> be a valid statement.
406      */
407     protected String validationQuery = null;
```

Visibility Modifier : Variable 'validationQuery' must be private and have accessor methods.

```
408
409     public synchronized void initialize() throws Exception {
```

Signature Declare Throws Exception : A method/constructor shouldn't explicitly throw java.lang.Exception

```
410         doClose();
411         doInitialize();
412     }
413
414     private void doInitialize() throws Exception {
```

Signature Declare Throws Exception : A method/constructor shouldn't explicitly throw java.lang.Exception

```
415         if (getRepository() == null && getRepositoryProviderClassName() != null) {
416             try {
417                 this.jcrRepositoryProvider = (JcrRepositoryProvider) Class.forName(getRepositoryProviderClassName()).newInstance();
418             } catch (Exception e) {
```

Dodgy - Exception is caught when Exception is not thrown : Exception is caught when Exception is not thrown in org.hippoecm.hst.core.jcr.pool.BasicPoolingRepository.doInitialize()

```
419                 throw new RepositoryException("Cannot create an instance of JcrRepositoryProvider: " + getRepositoryProviderClassName());
```

Preserve Stack Trace : Caught exception is rethrown, original stack trace may be lost

```
420             }
421
422             Repository repository = this.jcrRepositoryProvider.getRepository(getRepositoryAddress());
```

Hidden Field : 'repository' hides a field.

```
423             setRepository(repository);
424         }
425
426         if (getDefaultCredentials() == null && getDefaultCredentialsUserID() != null) {
427             setDefaultCredentials(new SimpleCredentials(getDefaultCredentialsUserID(), getDefaultCredentialsPassword()));
```

```
487      * session remain open until closed.  Once the pooling repository has
488      * been closed, no more sessions can be obtained.
489      * @throws Exception
490      */
491     public synchronized void close() throws Exception {
```

Signature Declare Throws Exception : A method/constructor shouldn't explicitly throw java.lang.Exception

```
492         doClose();
493     }
494
495     private void doClose() throws Exception {
```
Signature Declare Throws Exception : A method/constructor shouldn't explicitly throw java.lang.Exception
```
496         if (this.sessionPool != null) {
497             try {
498                 this.sessionPool.close();
499             } catch (Exception e) {
500                 if (log.isDebugEnabled()) {
```

```
896     }
897
898     private class SessionFactory implements PoolableObjectFactory {
899
900         public void activateObject(Object object) throws Exception {
```
Signature Declare Throws Exception : A method/constructor shouldn't explicitly throw java.lang.Exception
```
901             // If sessionRefreshPendingAfter is set to specific time millis,
902             // each session should be refreshed if it is not refreshed after the time millis.
903             if (object instanceof PooledSession) {
904                 PooledSession session = (PooledSession) object;
905                 session.activate();
906
907                 if (sessionsRefreshPendingTimeMillis > 0L) {
908                     if (session.lastRefreshed() < sessionsRefreshPendingTimeMillis) {
```
Collapsible If Statements : These nested if statements could be combined
```
909                         session.refresh(keepChangesOnRefresh);
910                     }
911                 }
912             }
913         }
914
915         public void destroyObject(Object object) throws Exception {
```
Signature Declare Throws Exception : A method/constructor shouldn't explicitly throw java.lang.Exception
```
916             Session session = (Session) object;
917
918             try {
919                 if (session instanceof PooledSession) {
920                     PooledSession pooledSession = (PooledSession) session;
```

```
934                 }
935             }
936         }
937
938         public Object makeObject() throws Exception {
```
Cyclomatic Complexity : Cyclomatic Complexity is 11 (max allowed is 10).

Signature Declare Throws Exception : A method/constructor shouldn't explicitly throw java.lang.Exception
```
939             Session session = null;
940
941             if (internalDefaultCredentials == null && defaultWorkspaceName == null) {
942                 session = getRepository().login();
943             } else if (internalDefaultCredentials != null && defaultWorkspaceName == null) {
```

```
954
955             return session;
956         }
957
958         public void passivateObject(Object object) throws Exception {
```
Signature Declare Throws Exception : A method/constructor shouldn't explicitly throw java.lang.Exception
```
959             Session session = (Session) object;
960
961             if (session instanceof PooledSession) {
962                 PooledSession pooledSession = (PooledSession) object;
963
```

```
987             if (!validated) {
988                 return validated;
989             }
990
991             String validationQuery = getValidationQuery();
```
Hidden Field : 'validationQuery' hides a field.
```
992
993             if (validationQuery != null) {
994                 Node nodeFound = null;
995
996                 try {
```