User Project Data Store 1.0 Component Specification

1. Design

1.1 Overview

The User/Project Data Store component provides data structure and read-only access to a database of component designers, developers, and actual projects. This is for access by an "external" system, hence the name "external" in the interface names.

There are interfaces to represent the data structures and data access objects (DAOs), as defined in the Requirements Specification. Concrete implementation classes are provided for the data structure interfaces and for the DAOs. The implementation of the DAO interfaces provide the required retrieval methods, and some new retrieval methods were added, such as "find by name" and "find by handle, case insensitively" in anticipation of more complex uses of the component.

Also, information about a user's rating is encapsulated into a new data structure, and the "rating type" (i.e. design or development) is represented as a type-safe enumeration.

1.2 Design Patterns

Template method – The `retrieveObjects` method of the abstract class `BaseDBRetrieval` uses the template method. It relies on an abstract method (`createObject`), implemented by a concrete subclass, to "fill in" the template.

Type-safe enumeration – The Typesafe Enum Component was utilized to distinguish between design and development ratings, representing the type-safe enumeration pattern.

1.3 Industry Standards

JDBC

1.4 Required Algorithms

1.4.1 Retrieve objects

There is a generic method, `BaseDBRetrieval.retrieveObjects(ps)` which iterates through a prepared statement's result set and creates a map of `ExternalObjects`. The pseudocode is:

```
Map objects = new HashMap();
ResultSet rs = ps.executeQuery();
while rs.next
      // call the abstract method which creates an object of the
      // appropriate type from the columns in the RS
      ExternalObject o = createObject(rs)
      objects.put(new Long(o.getId()), o)
rs.close()
return objects
```

1.4.2 Retrieve users

All of the `retrieveUsers` methods (which return more than one record[1]) in the `DBUserRetrieval` class implement their functionality based on this pseudocode. The type of parameter to set in the prepared statement depends on the query and will be either a long or a String.

[1] All methods except for `DBUserRetrieval.retrieveUser(long)` and `DBUserRetrieval.retrieveUser(String)`

```
conn = super.getConnection()
//construct the query string (see Section 1.4.2)
query = “select * from . . .”
// prepare the statement
ps = conn.prepareStatement(query)
// set parameters (note that there may not be an array, e.g., for
// retrieveUsersByName)
for i = 0 to number of parameters
      // will be either setLong or setString
      ps.setXXX(i, parameter[i]);

// call super method which calls this.createObject
Map objects = super.retrieveObjects(ps);

// Prepare the email query (Section 1.4.3.1)
emailQuery = “select * from . . .”
ps2 = conn.prepareStatement(emailQuery)
// set parameters (see above)
for i = 0 to number of parameters
      // will be either setLong or setString
      ps2.setXXX(i, parameter[i]);
// call private method
updateEmails(map, ps2)

// Prepare the ratings query (Section 1.4.3.2)
ratingsQuery = “select * from . . .”
ps3 = conn.prepareStatement(ratingsQuery)
// set parameters (see above)
for i = 0 to number of parameters
      // will be either setLong or setString
      Ps3.setXXX(i, parameter[i]);
// call private method
updateRatings(map, ps3)

// close the Prepared Statements and connection
ps2.close();
ps3.close();
super.close(ps, conn);

// convert the map to an array.
ExternalUser [] users = (ExternalUser [])
      new LinkedList(map.values()).toArray(new ExternalUser[0]);
return users;
```

1.4.2.1 Retrieve users by handles

This functionality is used by `DBUserRetrieval.retrieveUsers(String[] handle)`. The query to construct as needed in Section 1.4.1 is:

```
select u.user_id id, first_name, last_name, handle, address
from user u, email
where u.user_id = email.user_id
  and email.primary_ind = 1
  and handle in (?,?,?,. . . ?)
```

Note that if a primary email address is not found, the user is considered not found.

The query for retrieve user by lower case (case insensitive) is similar, except that the `handle_lower` column is used instead. The input is first converted to lower case so it will actually match the `handle_lower` column.

1.4.2.2 Retrieve users by ids

This functionality is used by `DBUserRetrieval.retrieveUsers(long[] ids)`. The query to construct as needed in Section 1.4.1 is:

```
select u.user_id id, first_name, last_name, handle, address
from user u, email
where u.user_id = email.user_id
  and email.primary_ind = 1
  and u.user_id in (?,?,?,. . . ?)
```

Note that if a primary email address is not found, the user is considered not found.

1.4.2.3 Retrieve users by name

This functionality is used by `DBUserRetrieval.retrieveUsersByName(String firstName, Stirng lastName)`. Before populating the two query parameters, append a percent ("%") character, so that it acts as a "starts with" query. The query to construct as needed in Section 1.4.1 is:

```
select u.user_id id, first_name, last_name, handle, address
from user u, email
where u.user_id = email.user_id
  and email.primary_ind = 1
  and u.first_name like ?
  and u.last_name like ?
```

Note that if a primary email address is not found, the user is considered not found.

1.4.3 Secondary user queries

Each of the `DBUserRetrieval.findUsers*` methods actually performs three queries: one set was described in Section 1.4.2. Those methods must prepare two more queries. These retrieve the alternative email address and the rating information. *Each of these queries repeats the same `where` clause as the above three sections, except it **also** joins on the `email` or `user_rating` table.* Depending on which query was performed in the first step (Section 1.4.2) the query performed in these secondary queries should be constructed accordingly. See the next two sections for details.

1.4.3.1 Retrieve alternative email addresses

To retrieve the alternative email addresses, the query is:

```
select u.user_id id, address
from user u, email
where u.user_id = email.user_id
  and email.primary_ind = 0
  and <see Table>
```

Query type	**Where clause**
Users by ids	`u.user_id in (?,?,? . . .)`
Users by handles	`u.handle in (?,?,?,. . . ?)`
Users by name	`u.first_name like ? and u.last_name like ?`

Note that it is acceptable for a user to have no alternative addresses.

The `DBUserRetrieval.updateEmails` method performs this pseudocode to add each alternative email to the users already retrieved in the map.

```
rs = ps.executeQuery()
while rs.next() {
      long id = rs.getLong("id")
      ExternalUserImpl user = map.get(new Long(id))
      If (user != null)
```

```
                user.addAlternativeEmail(rs.getString(“address”))
        }
```

1.4.3.2 Retrieve rating information

To retrieve the rating information, the query is:

```
        select u.user_id id,
               r.rating,
               r.phase_id,
               vol volatility,
               num_ratings,
               ur.rating reliability
        from user u, user_rating r, OUTER user_reliability ur
        where u.user_id = r.user_id
          and u.user_id = ur.user_id
          and r.phase_id = ur.phase_id
          and <see Table>
```

Query type	**Where clause**
Users by ids	`u.id in (?,?,? . . .)`
Users by handles	`u.handle in (?,?,?,. . . ?)`
Users by name	`u.first_name like ? and u.last_name like ?`

Note this is an inner join for rating, since it is acceptable for a user to have no rating, but an outer join for reliability, since theoretically a user could have a rating but no reliability.

The `updateRatings` method performs this pseudocode to add each alternative email to the users already retrieved in the map.

```
        rs = ps.executeQuery()
        while rs.next() {
                long id = rs.getLong(“id”)
                ExternalUserImpl user = map.get(new Long(id))
                if (user != null) {
                        RatingType rt;
                        rt = RatingType.getRatingType(rs.getLong(“phase_id”))
                        RatingInfo ri;
                        double reliability = rs.getDouble(“reliability”)
                        if (rs.wasNull())
                                // no reliability - use 4-arg ctor
                                ri = new RatingInfo(rt,
                                                    rs.getInt(“rating”),
                                                    rs.getInt(“num_ratings”),
                                                    rs.getInt(“volatility”));
                        else
                                // reliability existed - use 5-arg ctor
                                ri = new RatingInfo(rt,
                                                    rs.getInt(“rating”),
                                                    rs.getInt(“num_ratings”),
                                                    rs.getInt(“volatility”),
                                                    reliability);
                        user.addRatingInfo(ri)
                }
        }
```

1.4.4 Retrieve projects

All of the “retrieve” methods in `DBProjectRetrieval` (which return more than one record[2]) implement code based on this pseudocode. The type of parameter to set depends on the query and will be either long or String.

```
        conn = super.getConnection()
```

[2] All methods except for `DBProjectRetrieval.retrieveProject(long)`.

```
//construct the query string (see Section 1.4.2 and 1.4.4)
query = “select * from . . .”
// prepare the statement
ps = conn.prepareStatement(query)
// set parameters
for i = 0 to number of parameters
      // will be either setLong or setString
      ps.setXXX(i, parameter[i]);

// call template method
Map objects = super.retrieveObjects(ps);

// close the Prepared Statement and connection
super.close(ps, conn);

// convert the map to an array.
ExternalProject [] projects = (ExternalProject[])
      new LinkedList(map.values()).toArray(new ExternalProject[0]);
return projects;
```

1.4.4.1 Retrieve projects by ids

This functionality is used by `DBProjectRetrieval.retrieveProjects(long[] ids)`. The query to construct as needed in Section 1.4.1 is:

```
select cv.comp_vers_id,
      cv.component_id,
      version,
      version_text,
      comments,
      component_name,
      description,
      cc.root_category_id category_id,
      forum_id
from comp_versions cv, comp_catalog cc, OUTER comp_forum_xref f
where cv.component_id = cc.component_id
and cv.comp_vers_id = f.comp_vers_id
and f.forum_type = ?
and cv.comp_vers_id in (?,?,?,. . . ?)
```

Note the outer join syntax with the `comp_forum_xref` table (not every component will have a forum). The `forum_type` is retrieved from the `forumType` property in the given Config Manager namespace, or defaults to 2 if it is not set.

1.4.4.2 Retrieve projects by name and version

This functionality is used by `DBProjectRetrieval.retrieveProjects(String[] names, String[] versions)`. The query to construct as needed in Section 1.4.1 is:

```
select cv.comp_vers_id,
      cv.component_id,
      version,
      version_text,
      comments,
      component_name,
      description,
      cc.root_category_id category_id,
      forum_id
from comp_versions cv, comp_catalog cc, OUTER comp_forum_xref f
where cv.component_id = cc.component_id
and cv.comp_vers_id = f.comp_vers_id
and f.forum_type = ?
and cc.component_name || cv.version_text in (?,?,?,. . .?)
```

In order to properly find both the name and version simultaneously, we must use a concatenation of the two strings. The query parameters must be accordingly concatenated before calling `ps.setString()`.

Note also that the join to `comp_catalog` is an inner join, but the join with `comp_forum_xref` is still outer. The `forum_type` is retrieved from the `forumType` property in the given Config Manager namespace, or defaults to 2.

1.4.5 RatingType configuration

The `RatingType` type-safe enum uses the Config Manager component to determine which "phase number" (phase_id column in certain tables) corresponds to which component phase, design or development. The private `RatingType.configure()` method was added to the class to determine this mapping. As described in Section 3.2, there are two optional parameters, "Design" and "Development" whose values map to the phase numbers.

When the public static enumerated type values `DESIGN` or `DEVELOPMENT` are accessed for the first time, or any time one of the `getRatingType` methods is called, the configure() method is called, to make sure the Config Manager is referenced to determine the mapping.

In order to make this configuration transparent to the user, the `getRatingType` methods and the `configure()` method silently ignore any Config manager exceptions that might occur, and instead just default to Design mapping to phase id 112 and Development to phase id 113. As a result, however, the namespace that the class uses is fixed (see Section 3.2).

1.5 Component Class Overview

interface ExternalObject extends Serializable

This simple interface merely captures common information (the id) for the other two entities, namely `ExternalUser` and `ExternalProject`. It also implements `Serializable` so that when subclasses are used in a web application, they can be serialized for session replication.

interface ExternalUser extends ExternalObject

This interface represents a user in this component. It stores the user's first and last name, handle, email address(es) and design and development rating information. The unique id (user id) is described by the super interface.

interface ExternalProject extends ExternalObject

This interface describes a project within the User Project Data Store component. The component id, version id, forum id and catalog id are all included, as well as textual descriptions of the project and the component itself. The unique id (project id) is described by the super interface.

interface UserRetrieval

This is the interface which describes the various ways that users can be retrieved from persistent storage. There are two methods to retrieve individual users (by id or handle), and additional methods that retrieve sets of users in bulk (e.g., by name or by a set of ids or handles).

interface ProjectRetrieval

This is the interface for the User Project Data Store component which describes the various ways that projects can be retrieved from persistent storage. There is a method to retrieve individual projects by their id and additional methods that retrieve sets of projects in bulk, by a set of ids or by name and version.

class RatingInfo implements Serializable

This class encapsulates information about ratings for a particular rating type for a user. It includes the actual rating, the reliability, number of ratings and volatility, for a specific rating type (design or development.) This class is immutable and thread-safe.

class RatingType extends Enum implements Serializable

This typesafe enum represents the various ratings types (phases) that the User Project Data Store component supports, namely, <u>design</u> and <u>development</u>. The Config Manager is used to retrieve the id number of each phase (See Section 1.4.5.) The required namespace is defined in this class, along with a method to retrieve the appropriate id number for the rating type. This class is immutable and thread-safe.

class ExternalObjectImpl implements ExternalObject

Basic implementation of the `ExternalObject` interface; merely maintains the id field as a final long. This class is thread-safe, but subclasses are not guaranteed to be.

class ExternalUserImpl extends ExternalObjectImpl implements ExternalUser

Basic implementation of the `ExternalUser` interface. Most of the fields of this class are immutable, aiding in thread-safety; the rest are only modifiable within this package. The outside world should never create instances of or modify objects of this class; only methods within this package should and so the constructor and mutator methods are all package-scope.

class ExternalProjectImpl extends ExternalObjectImpl implements ExternalProject

Basic implementation of the `ExternalProject` interface. Some of the fields of this class are immutable, aiding in thread-safety; the rest are only modifiable within this package. The “outside world” should never create instances or modify objects of this class; only methods within this package should and so the constructor and mutator methods are all package-scope. If any field is already set and the calling program attempts to set them again, an exception will be thrown.

abstract class BaseDBRetrieval

Abstract base class for implementing the database (JDBC) versions of the "retrieval" interfaces. Utility methods are provided to get and close a connection, and to bulk create objects from a result set (via a `PreparedStatement`.) This class is immutable and therefore thread-safe.

class DBUserRetrieval extends BaseDBRetrieval implements UserRetrieval

This is the database (JDBC) implementation of the `UserRetrieval` interface. All the methods (except `retrieveUser(long)` and `retrieveUser(String)`) call `super.getConnection` to get a connection from the DBConnectionFactory, and then call to `super.retrieveObjects`, which calls `this.createObject`. Then, they call `updateEmails` and `updateRatings`. Afterwards, the prepared statement and connections are all closed using `super.close()`. All `SQLExceptions` in all methods should be wrapped in a `RetrievalException`. This class is immutable and therefore thread-safe.

class DBProjectRetrieval extends BaseDBRetrieval implements ProjectRetrieval

This is the database (JDBC) implementation of the `ProjectRetrieval` interface. All the methods (except `retrieveProject(long)`) call `super.getConnection` to get a connection from the `DBConnectionFactory`, and then call to `super.retrieveObjects`, which calls `this.createObject`. Afterwards, the prepared statement, result set and connections are all closed using `super.close()`. All `SQLExceptions` in all methods should be wrapped in `RetrievalException`. This class is immutable and therefore thread-safe.

class UserProjectDataStoreHelper

This is a final class which just contains some helper method for the component. Like, some parameter validation methods, some pre- processing methods and so on.

1.6 Component Exception Definitions

UserProjectDataStoreException extends BaseException

This is the base exception for all exceptions in the User Project Data Store component. It can be used with or without a message, and with or without an underlying cause.

ConfigException extends UserProjectDataStoreException

This exception represents a problem with the configuration in the User Project Data Store component. It is thrown by the `UserRetrieval` and `ProjectRetrieval` interfaces (and their implementing and base classes.) It can be used with a message, and with or without an underlying cause. Usually it wraps a `ConfigManagerException` that has occurred.

RetrievalException extends UserProjectDataStoreException

This exception represents a problem with retrieving data from persistent storage in the User Project Data Store component. It is thrown by the `UserRetrieval` and `ProjectRetrieval` interfaces (and their implementing and base classes.) It can be used with a message, and with or without an underlying cause.

1.7 Thread Safety

All classes except for `ExternalUserImpl` and `ExternalProjectImpl` are designed to be completely thread-safe. These last two classes are not inherently thread-safe, because of their unsynchronized setters. However, they are only created or modified by the `DBUserRetrieval` and `DBProjectRetrieval` classes, which themselves are thread-safe. The `*Retrieval` classes return arrays of the corresponding interface, which do not have the setter methods that the implementations do. An outside user would have to explicitly cast a returned value to a (non-thread-safe) `*Impl` class to access the setter methods. It seems impractical to require thread-safety of the `*Impl` classes because they should only be created or modified by the implementation-specific `*retrieval` classes via a single calling thread anyway.

2. Environment Requirements

2.1 Environment

- At minimum, Java 1.4 is required for compilation and executing test cases.
- Informix 10.0

2.2 TopCoder Software Components

- Base Exception 1.0: The base class for all custom exceptions in this component.
- DB Connection Factory 1.0: This is used by the `BaseDBRetrieval` class to create a connection to the database.
- Typesafe enum 1.0: The base class for `RatingType`.
- Configuration Manager 2.1.4: Used by DB Connection Factory for configuration. Also, the `BaseDBRetrieval` class obtains the connection name from this component, and the `RatingType` class retrieves the id numbers of the Design and Development phases as well.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component *installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.*

2.3 Third Party Components

- Informix JDBC driver must be added to the runtime classpath.

3. Installation and Configuration

3.1 Package Name

com.cronos.onlinereview.external

com.cronos.onlinereview.external.impl

3.2 Configuration Parameters

A sample xml file, Sample.xml, is included in the conf directory, to demonstrate how to configure this component.

The following two optional parameters can be defined in the namespace as given to the constructor of one of the DB* classes.

Parameter	Description	Values
`connName`	The name of the connection from the DB connection factory to use. (Optional; defaults to using the default connection	User-defined
`forumType`	The id number of the forum type code to use when retrieving the forum for a component's project. (Optional; defaults to 2)	2

The following two parameters, if defined, must be in the `com.cronos.onlienreview.external.RatingType` namespace (it must be in this namespace). The parameters are *optional* and have default values. If any new phases are included in the `user_rating` or `user_reliability` tables, they can be added here to allow additional values of the `RatingType` typesafe enum to be defined.

Parameter	Description	Values
`Design`	The id number of the "Design" phase. (Optional; defaults to 112)	112
`Development`	The Id number of the "Development" phase. (Optional; defaults to 113)	113

3.3 Dependencies Configuration

The DBConnection Factory component must be configured, within the same namespace as given to the constructor of the DB*Retrieval classes. At least one connection provider must be defined.

Configure and populate an Informix 10 database with the desired data to retrieve.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.

- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

- Configure the DB connection factory according to the documentation and your particular database configuration (e.g., IP address, database name, user name, password).
- Configure the ConfigManager component with the appropriate namespaces and config files.
- Instantiate one of the DB*Retrieval classes
- Use the API either as in the Requirements Specification or as augmented in this design.

4.3 Demo

Before executing the demo, the various ConfigManager namespaces must be configured as described in the previous sections.

Also, since this component is read-only, this demo assumes that the database is populated with certain user and project/component records; ***that step is not shown here***. The demo assumes that most data is retrieved (and indicates when there should be multiple records, or no records retrieved.)

4.3.1 Retrieve user(s) by handle

```
// Should be non-null.
ExternalUser user = defaultDBUserRetrieval.retrieveUser("Handle B");

// Should be 1002.
System.out.println(user.getId());
// Should all be non-null.
System.out.println(user.getHandle());
System.out.println(user.getFirstName());
System.out.println(user.getLastName());

// Should be N/A or zero.
System.out.println(user.getDesignRating());
System.out.println(user.getDesignReliability());
System.out.println(user.getDesignNumRatings());
System.out.println(user.getDesignVolatility());

// There are two alt-emails.
System.out.println(user.getAlternativeEmails().length);
System.out.println(user.getAlternativeEmails()[0]);
System.out.println(user.getAlternativeEmails()[1]);

// Should be not N/A and not zero.
System.out.println(user.getDevRating());
System.out.println(user.getDevVolatility());
System.out.println(user.getDevReliability());
System.out.println(user.getDevNumRatings());

// Should be null if user is not found.
ExternalUser shouldBeNull = defaultDBUserRetrieval.retrieveUser("Not Exist");
System.out.println(shouldBeNull);

// Find all users, case insensitively. This demo assumes there is only 1 user
// with this handle as a lower-case string.
ExternalUser[] userA = defaultDBUserRetrieval.retrieveUsersIgnoreCase
             (new String[] { "hAnDle A" });
// Should only have one.
System.out.println(userA.length);
ExternalUser sameAsUser = userA[0];
// Outputs his info.
System.out.println(sameAsUser.getHandle());
```

```
// Should have a maximum of 2 users, but the order is indeterminate
ExternalUser[] users = defaultDBUserRetrieval.retrieveUsers(
             new String[] {"Handle A", "Handle C", "Handle Z"});
System.out.println(users.length);
```

4.3.2 Retrieve user(s) by id

```
// The record exists.
ExternalUser user = defaultDBUserRetrieval.retrieveUser(1002);
System.out.println(user.getId());

// This record does not exist; should be null if user is not found.
ExternalUser shouldBeNull = defaultDBUserRetrieval
             .retrieveUser(Long.MAX_VALUE);
// Should be null
System.out.println(shouldBeNull);

// Should have a maximum of 2 users even though there were 3 values given.
ExternalUser[] users = defaultDBUserRetrieval.retrieveUsers
             (new long[] { 1001, 1002, Long.MAX_VALUE });
// The length should be 2.
System.out.println(users.length);
```

4.3.3 Retrieve users by last name, first name

```
// Should retrieve all users whose first name starts with 'First' and last
// name starts with 'Last'.
ExternalUser[] users = defaultDBUserRetrieval.retrieveUsersByName
             ("First A", "Last A");
// There should be 1 user got.
System.out.println(users.length);

ExternalUser user = users[0];
// There is only one alternative email.
System.out.println(user.getAlternativeEmails().length);
// The email address should be the same to User1@163.com.
System.out.println(user.getAlternativeEmails()[0]);
// The design number ratings should be the same as 10.
System.out.println(user.getDesignNumRatings());
// The design rating should be the same as 1563.
System.out.println(user.getDesignRating());
// The design reliability should be the same 1.00 %.
System.out.println(user.getDesignReliability());
// The design volatility should be the same as 431.
System.out.println(user.getDesignVolatility());
// There is no dev rating of the user.
System.out.println(user.getDevNumRatings());
// There is no dev rating of the user.
System.out.println(user.getDevRating());
// There is no dev rating of the user.
System.out.println(user.getDevReliability());
// There is no dev rating of the user.
System.out.println(user.getDevVolatility());
// The email address should be the same as User1@gmail.com.
System.out.println(user.getEmail());
// The first name should be the same as First A.
System.out.println(user.getFirstName());
// The last name should be the same as Last A.
System.out.println(user.getLastName());
```

```
 // The handle should be the same as Handle A.
 System.out.println(user.getHandle());
 // The user id should be the same as 1001.
 System.out.println(user.getId());

// Should retrieve all users whose last name starts with Last (ignores first name)
ExternalUser[] tcsUsers = defaultDBUserRetrieval.retrieveUsersByName
       ("", "Last");
// There should be 3 users got.
System.out.println(tcsUsers.length);

// must have at least first or last name be not empty.
try {
       defaultDBUserRetrieval.retrieveUsersByName("", "");
} catch (IllegalArgumentException e) {
}
```

4.3.4 Retrieve project(s) by id

```
// Retrieve a known project
ExternalProject project = defaultDBProjectRetrieval.retrieveProject(2);
System.out.println(project.getId());

// Outputs its info.
System.out.println(project.getComponentId());
System.out.println(project.getForumId());
System.out.println(project.getName());
System.out.println(project.getVersion());
System.out.println(project.getVersionId());
System.out.println(project.getDescription());
System.out.println(project.getComments());

// Not found – should be null which is acceptable
ExternalProject shouldBeNull = defaultDBProjectRetrieval
             .retrieveProject(Long.MAX_VALUE);
System.out.println(shouldBeNull);

// Should only have a maximum of 1 entry.
ExternalProject[] projects = defaultDBProjectRetrieval.retrieveProjects
             (new long[] { 1, 100 });
System.out.println(projects.length);
System.out.println(projects[0].getName());
```

4.3.5 Retrieve projects by name and version

```
// There might be more than one with this name and version (e.g., different catalog)
ExternalProject[] projects = defaultDBProjectRetrieval.retrieveProject
             ("Project A", "Version 1");

for (int i = 0; i < projects.length; ++i) {
       // Outputs the info of each project.
       ExternalProject project = projects[i];
       System.out.println(project.getId());
       System.out.println(project.getName());
       System.out.println(project.getVersion());
       System.out.println(project.getCatalogId());
       System.out.println(project.getForumId());
}

// Should only have a maximum of 2 entries but the order may vary.
ExternalProject[] projects2 = defaultDBProjectRetrieval.retrieveProjects(
                new String[] { "Project A", "Project C" },
                new String[] { "Version 1", "Version 2" });
// Should get only one.
System.out.println(projects2.length);
System.out.println(projects2[0].getName());
```

5. Future Enhancements

- Other implementations of the `UserRetrieval` interface e.g., XML (e.g., from the TopCoder XML user feed) or based on Hibernate.
- Other implementations of the `ProjectRetrieval` interface e.g., based on Hibernate.
- Create a factory to instantiate implementations of the `UserRetrieval` or `ProjectRetrieval` interfaces using the TopCoder Object Factory component.
- Create versions of the retrieval interfaces that cache their values (especially for the `UserRetrieval` interface) to avoid hitting the database for every query. It is unknown how frequently the user or project information is updated or how “fresh” the data is required to be, which is why this was not modeled in this version.
- Integrate with the TopCoder Search Builder component for even more complex searches.
- Creating completely thread-safe versions of the `*Impl` classes.