CollabraSuite BEA Edition®

Integration Guide

Version 5.1

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

Getting Started

Using the API

Samples

CHAPTER 1

Getting Started

An effective collaborative environment not only brings people together, it provides access to important data by tightly integrating with other business critical systems. CollabraSuite BEA Edition provides an Application Programming Interface (API) to facilitate this integration. This API allows developers to seamlessly bring existing data and systems into their collaborative environment, tailoring it to their specific requirements.

For example, documents can be created in a room or user's briefcase using real-time data such as an RSS feed. The document's subscriber list can then be modified to automatically notify users of the new information. Another example might involve dynamically creating rooms or sessions to deal with a situation in real-time, such as an intrusion detection system. When a pre-defined event occurs, the API could be used to create a new collaborative session and bring a set of online users into that new session.

Compiling

In order to begin compiling code using the Integration API, the following CollabraSuite BEA Edition JAR is required: `csuite-i9n-client.jar`. This JAR is located under the CollabraSuite BEA Edition installation in the lib directory. Additionally, the standard Java 2 Enterprise Edition (J2EE) classes are required. These can usually be found bundled with your J2EE application server. For example, WebLogic includes these classes in `weblogic.jar`. Below is an example of compiling a client using the Integration API:

```
% javac -classpath
csuite-i9n-client.jar:${WL_HOME}/server/lib/weblogic.jar:.
CSuiteIntegrationClient.java
```

Running

The Integration API uses Log4j for logging, so running the client code requires all of the JARs mentioned above plus log4j.jar. The following command illustrates running a stand-alone client that connects to a WebLogic application server:

```
% java –classpath csuite-i9n-client.jar:log4j.jar:
${WL_HOME}/server/lib/weblogic.jar:. CSuiteIntegrationClient
```

Running client code inside the web tier of an application server requires access to the same list of JAR files. These can be made available by placing them in the `WEB-INF/lib` directory of a WAR. Additionally, the following changes must be made to `web.xml` and the application specific deployment descriptor such as `weblogic.xml`. Note that all of the necessary modifications are automatically performed when installing CollabraSuite BEA Edition into an existing web application via WebLogic Workshop. See the **Installation Guide** for additional details.

Figure 1: Additions to web.xml

```
<ejb-ref>
   <ejb-ref-name>ejb/CSuiteAdmin</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <home>
com.collabraspace.csuite.server.i9n.interfaces.CSuiteAdminRemoteHome
   </home>
   <remote>
com.collabraspace.csuite.server.i9n.interfaces.CSuiteAdminRemote
   </remote>
</ejb-ref>
<ejb-ref>
   <ejb-ref-name>ejb/CSuiteCollaboration</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <home>
com.collabraspace.csuite.server.i9n.interfaces.CSuiteCollaborationRem
oteHome
   </home>
   <remote>
com.collabraspace.csuite.server.i9n.interfaces.CSuiteCollaborationRem
ote
   </remote>
</ejb-ref>
```

Figure 2: Additions to weblogic.xml

```
<ejb-reference-description>
    <ejb-ref-name>ejb/CSuiteAdmin</ejb-ref-name>
    <jndi-name>ejb/CSuiteAdmin</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/CSuiteCollaboration</ejb-ref-name>
    <jndi-name>ejb/CSuiteCollaboration</jndi-name>
</ejb-reference-description>
```

Connecting

The Integration API is accessed via Stateless Session Enterprise Java Beans (EJBs) provided by the CollabraSuite BEA Edition application. The API can be access both locally and remotely. Local clients run inside the application server while remote clients run stand-alone outside of the application server. The only difference between the two methods is how the code finds and connects to the server using JNDI. When running inside the web tier of an application server, no extra information is required to lookup one of the Stateless Session EJBs:

```
CSuiteAdminRemote csAdmin =
        CSuiteFactory.getCSuiteAdminRemoteInstance();
```

Connecting from a remote client requires more information such as the JNDI initial context factory, provider URL, username and password. Additionally, in WebLogic there is an extra JNDI parameter to pass in order to execute calls as a specific user instead of as the *anonymous* user. Setting `weblogic.jndi.enableDefaultUser` to `true` will allow the call to execute as the user specified in the `SECURITY_PRINCIPAL` parameter. The following is an example of connecting remotely using WebLogic:

```
Hashtable h = new Hashtable();
h.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
      "weblogic.jndi.WLInitialContextFactory");
h.put(javax.naming.Context.PROVIDER_URL, "t3://localhost:7001");
h.put(javax.naming.Context.SECURITY_PRINCIPAL, "username");
h.put(javax.naming.Context.SECURITY_CREDENTIALS, "password");
h.put("weblogic.jndi.enableDefaultUser", "true");
CSuiteAdminRemote csAdmin = CSuiteFactory.getCSuiteAdminRemoteInstance(h);
```

CHAPTER 2

Using the API

The API consists of classes defined in two packages:

- `com.collabraspace.csuite.server.i9n.util`
- `com.collabraspace.csuite.server.i9n.ejb`

After installation, the CollabraSuite JavaDoc API documentation can be found at `http://host:port/csuite/docs`.

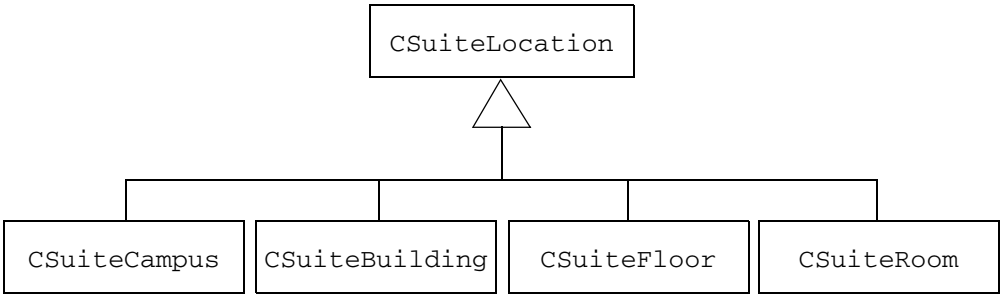
Utility Classes

The `com.collabraspace.csuite.server.i9n.util` package consists of a selection of utility classes and application exceptions, which facilitate the use of the EJBs that define the main API functionality.

The `com.collabraspace.csuite.server.i9n.util` package contains a set of utility classes which are used as arguments and return types of the main EJB methods that form the backbone of the API. These utility classes can be separated into three main types; those which represent a CollabraSuite BEA Edition location; those which represent a CollabraSuite BEA Edition folder or document; those which crrepresenteate a CollabraSuite BEA Edition group or user. Each utility class accepts a string as a descriptor to construct the related object. There are static methods in each class that can be used to construct these descriptor strings from the basic building blocks. The user is free to construct the strings manually, as well. Supplying a descriptor string with an incorrect format (i.e., accidentally using a `CSuiteUser` descriptor String in a `CSuiteRoom` constructor) will result in a `MalformedDescriptorException`. (see below)

The classes responsible for representing a CollabraSuite BEA Edition location, as well as, extending the base class `CSuiteLocation` are represented below:

Figure 3: Location Class Hierarchy



The constructors for the `CSuiteLocation` classes accept a string argument that describes the fully qualified location within a `CSuiteCampus`. A descriptor of "SampleCampus/SampleBuilding/SampleFloor/SampleRoom", for example, defines a room, "SampleRoom", located within a floor, "SampleFloor", contained within a building, "SampleBuilding", all within the campus "SampleCampus". Each level of this descriptor string can be represented by its own `CSuiteLocation` object, which is a subset of the "SampleRoom" example, above.

In addition to providing descriptor strings, alternate constructors, defined above, allow one to build a location relative to an existing `CSuiteLocation` object. Thus, given a `CSuiteCampus` object, a `CSuiteBuilding` within that campus can be created by using the constructor of the form `CSuiteBuilding (CSuiteCampus, String)`, where the `String` argument is the name of the building. A similar process can be used to create a `CSuiteFloor` and `CSuiteRoom`.

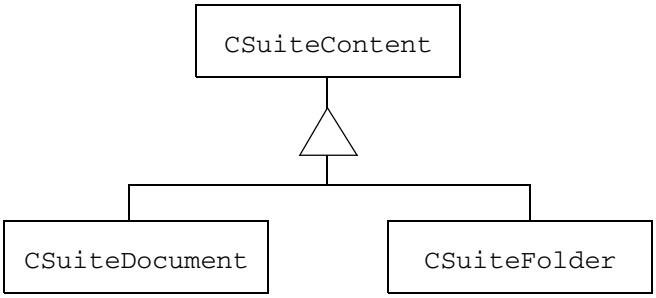
Finally, a third constructor form allows for individual strings, corresponding to each of the elements of the descriptor string. Thus, for the SampleRoom example above, the object can also be created using a constructor of the form CSuiteRoom ("SampleCampus", "SampleBuilding", "SampleFloor", "SampleRoom").

The `CSuiteLocationInfo` object acts as a wrapper around a `CSuiteLocation` and contains additional information about a CollabraSuite location.

Where a `CSuiteLocation` object contains information only about a specific location (e.g. building), the `CSuiteLocationInfo` object contains information about the location's description and icon. In the case of a room it also contains its lockable status.

The classes responsible for creating a folder or document are depicted below, and all extend the base class `CSuiteContent`:

Figure 4: Content Class Hierarchy



The constructors for the `CSuiteContent` classes take a string argument that describes the file structure within CollabraSuite BEA Edition of the desired content. For example, a document named “MyDocument” within a folder named “Folder1” would be constructed with a descriptor `String` of “/Folder1/MyDocument”. A forward slash (“/”) is always used as the file separator. As with `CSuiteLocation`, nested folders must be created one at a time. (although this is not true for deleting content)

It should also be noted that the `CSuiteContent` classes are used for both File Cabinet and Briefcase operations. The distinction is made by the passing either a user (for Briefcase) or room (for File Cabinet) into the specific API methods. The descriptor string “/” always denotes the root of the location (either the File Cabinet or the Briefcase).

Once content is created, its metadata can be manipulated via `CSuiteDocumentInfo` and `CSuiteFolderInfo` objects. Content information can be read by using the `getFolderContentInfo()` methods, while it is set through the use of the `setContentInfo()` methods.

The classes responsible for representing a CollabraSuite BEA Edition group and user include:

- `CSuiteGroup`
- `CSuiteUser`

The constructors for these objects consist of a campus name and either a user or group name, as appropriate. For example, to create a `CSuiteUser` object for a user in campus "SampleCampus" whose login is "testUser1", the descriptor string would be: "SampleCampus:testUser1".

The `CSuiteACL` class defines an Access Control List comprised of a set of users and/or groups. These users and/or groups can be either granted or denied access to a location or resource.

The `CSuiteDocumentType` class represents a Document Type in CollabraSuite BEA Edition and consists of a mime-type, a file extension, a name, a description and an image.

Where the `CSuiteUser` class represents a CollabraSuite BEA Edition user, the `CSuiteUserInfo` class contains information about a user such as home room and contact information.

The `CSuitePriority` class is used to define a priority within CollabraSuite BEA Edition that can be used as an argument to a page. Unlike the other utilities, this class does not take a descriptor String in its constructor. Rather, it takes the name of the priority (i.e., "High") and a numeric sort order used to compare one priority against another. An icon should also be supplied.

Finally, the API also defines the `CSuiteFactory` utility class to perform JNDI lookups associated with obtaining handles to the two session EJBs:

Exceptions

The `com.collabraspace.csuite.server.i9n.util` package contains two varieties of exceptions that are described in the API. The two varieties are the standard Java/EJB Exceptions and CollabraSuite BEA Edition API specific Integration Application Exceptions. These Integration Application Exceptions are defined in the document as:

- `InvalidResourceException` – denotes an invalid resource or location in CollabraSuite BEA Edition that is being passed as a descriptor. An example is an incorrect path to a campus location.
- `MalformedDescriptorException` – denotes a syntactical error in the string being passed as a descriptor or supplying a descriptor string with an incorrect format (i.e., accidentally using a `CSuiteUser` descriptor String in a `CSuiteRoom` constructor)

It must be noted that there are several other CollabraSuite BEA Edition exception types that may be thrown by the API method. One example of this exception type is the `com.collabraspace.cserver.interfaces.ServiceException`. These exceptions are generated from within the CollabraSuite BEA Edition server code that sits behind the API methods themselves. These are not Integration-specific CollabraSuite BEA Edition exceptions and are not detailed in this API.

Logging

The Integration API uses Log4j as its logging implementation. For details on configuring Log4j, see http://logging.apache.org/log4j/docs/manual.html. When running inside the WebLogic application server (or when weblogic.jar is on the CLASSPATH), log messages are integrated with the WebLogic log. In both cases, the `cs.log.debug` Java system property can be used as a convenience to enable debugging on a package or class basis. The following example starts a client with debugging turned on:

```
% java -classpath
csuite-i9n-client.jar:log4j.jar:${WL_HOME}/server/lib/weblogic.jar:.
-Dcs.log.debug=com.collabraspace CSuiteIntegrationClient
```

Transactions

It is often desirable to perform multiple Integration API method calls such that they are committed or rolled back as a group. This is accomplished by executing the multiple calls in the context of a single transaction.

If the caller does not have a current transaction, one will be started at the beginning of the Integration API method call and committed when the call sucessfully returns. The transaction will be rolled back if an exception is thrown. When the caller already has an active transaction, the Integration API methods will execute within the context of that transaction.

When invoking the Integration API from the Enterprise JavaBean tier of a J2EE application server, this is easily accomplished with container managed transactions. However, the transactions must be managed manually from the web tier or from a standalone client. The `CSuiteFactory` provides two convenience methods for manually managing transactions: `beginTransaction()` and `commitTransaction()`.

For a full discussion on the topic of transaction management please refer to the Enterprise JavaBeans Specification.

Package com.collabraspace.csuite.server.i9n.ejb

The `com.collabraspace.csuit.server.i9n.ejb` package contains two EJBs, one handling basic administrative functions and the other handling collaborative functions. These are described in more detail below.

Administrative Functions

Functions necessary for the administration of a CollabraSuite BEA Edition campus are provided in this API. They can be grouped into three categories: information retrieval/modification, resource creation/deletion, and permission modification.

Information Retrieval and Modification

The information retrieval functions provide users with information relating to the structure of the CollabraSuite BEA Edition campus, such as the definition of the buildings, floors, and rooms contained within the campus. They also provide information on the users in the campus; such as, who are the active users in a campus? Where are they located? Which are currently on-line? What are their skills? These methods typically return `java.util.Collections` containing utility types described in the above Utility Classes section. For example, `getUsers` returns a Collection of `CSuiteUser` objects, a utility class.

The information modification functions allow for the creation and deletion of skills which are assigned to users. The list of available skills is customizable and allow for greater knowledge sharing and problem solving because users can seek out other users with a required skill set in order to tackle an issue or problem. The functions, `createSkill` and `deleteSkill`, accept `CSuiteCampus` and a `String`, which is the skill to be created or deleted, and complete the action within the campus specified. The information modification functions also allow for the creation of priorities within a campus. These are used to prioritize pages and secure chat sessions. Default priorities are “Low”, “Medium” and “High”, but other priorities can be added to a campus. Using the function, `createPriority`, a `CSuiteCampus` class and the `CSuitePriority` class are used to assign a new, non-default priority to a CollabraSuite BEA Edition campus.

Specifically, the information retrieval methods are:

- `doesUserExist()`
- `getAssociates()`
- `getBuildings()`
- `getCampuses()`
- `getDocumentTypesByExtension()`
- `getFloors()`
- `getGroups()`
- `getLocationAccess()`
- `getOnlineUsers()`

- `getPriorities()`
- `getRooms()`
- `getSkills()`
- `getSkillsForUser()`
- `getUserInfo()`
- `getUserLocations()`
- `getUsers()`

The corresponding modification methods are:

- `createPriority()`
- `createSkill()`
- `deleteSkill()`
- `createGroup()`
- `deleteGroup()`
- `setGroupMembers()`
- `createUser()`
- `deleteUser()`
- `setUserInfo()`
- `modifyDocumentType()`

Location Administration

The location functions deal solely with the administration of locations within a campus. A location can be a room, a floor, a building, and even a campus. Also available is the verification of the existence of a location within a campus or of the campus, itself. This verification ensures that duplicate locations are not created or that a location can be identified before it is deleted or modified. The location functions are:

- `createLocation()`
- `moveFloor()`
- `MoveRoom()`
- `deleteLocation()`
- `doesLocationExist()`

- `getLocationInfo()`
- `setLocationInfo()`

To create new locations, create a `CSuiteLocation` object representing the new location, then build a new `CSuiteLocationInfo` object using the `CSuiteLocation` object and add the additional information. Then use the `createLocation()` method to create the location.

It is important to understand that when creating nested locations, it is necessary to create the locations in proper order. That is, one cannot create a `CSuiteRoom` object without first creating the `CSuiteFloor` object, or the `CSuiteFloor` before the `CSuiteBuilding`, etc. For example, in order to create the "SampleRoom" (described above), it would be necessary to create first the "SampleCampus", and then the "SampleBuilding" location, followed by the "SampleFloor", and lastly the "SampleRoom" location. Any attempt to create this location in a single call, that is, creating the campus, building, floor and room all at the same time, will result in an `InvalidResourceException`, defined in the **Exceptions** section, above. In the prior example, it is not necessary to create descriptor strings for each level. As indicated above in the **Utility Classes** section, `CSuiteLocation` objects can be used in the constructors for other `CSuiteLocations`, easing the developer's task in creating nested locations within a campus.

The moveFloor() and moveRoom() methods can be used to move floors and rooms to different locations within the same campus. They can also be used to rename a floor or room.simply

Permissions

Functions to modify permissions are also available in this API. These permissions take on a variety of forms. They can be access or administration privileges given to locations for specified groups and users. They can even be associates lists which are assigned to specific users. Regardless of form, these all, in some way, regulate or restrict access to locations and users within in the campus:

- `setAssociates()`
- `setLocationAccess()`
- `setLocationAdministrators()`
- `setSkillsForUser()`

In some of these cases, the arguments for the methods include arrays of `CSuiteUser` objects and arrays of `CSuiteGroup` objects. The general rule of thumb is that any user or group specified will be used in the given operation. If the users array is null, only the specified groups will be used. Conversely, if the groups array is null, only the specified users will be used. When both

the users and groups arrays are null, the operation will be applied to all users in the campus (i.e., the “Everyone” group).

In other instances, such as `setLocationAccess()`, a `CSuiteACL` object is used to specify the users, groups and the grant mode. The grant mode defines whether we intend to grant or deny access to the supplied users/groups. The constants used for the grant mode argument are defined in the `CSuiteACL` class.

Collaboration Functions

Functions required to establish and maintain a collaborative session are also included in this API. These functions relate to the creation, management, and deletion of documents, the sending of pages between users and/or groups and the initiation of sidebar sessions. These functions are provided via utility classes found in the `com.collabraspace.csuite.server.i9n.ejb` package. In a collaborative session, the creation of documents and folders, and the ability to manage and share them in that session, is essential. The following allow for the creation, deletion and examining of the contents of files and folders:

- `createDocument()`
- `createFolder()`
- `checkOutDocument()`
- `checkInDocument()`
- `deleteItem()`
- `getDocumentContents()`
- `getFolderContents()`
- `getFolderContentInfo()`
- `getItemInfo()`
- `setItemInfo()`

In order to modify a document the client must first check out the document from CollabraSuite. This can be accomplished by invoking either of the `checkOutDocument` methods (one method is for briefcase documents, the other for file cabinet documents). After the document is checked out, invoke the `getDocumentContents` method to retrieve the actual document to the local system. At this point the document may be modified either programmatically or via an external application (e.g. Word, Excel, etc). Once modifications are complete, the document must be checked back into CollabraSuite. This can be accomplished by using one of the

`checkInDocument` methods. In general, the call pattern to modify a document in CollabraSuite will be:

a. `checkOutDocument()`

b. `getDocumentContents()`

c. `checkInDocument()`

The `setItemInfo()` method takes a `CSuiteContentInfo` that contains a `CSuiteACL`, explained above in the **Administrative Functions** section. In this case, the `CSuiteACL` also contains an accessType that defines whether permissions for an item are being set to read, write or read-write.

Sending pages and participating in sidebar sessions are also major aspects of collaboration and the methods, `sendPage()` and `createSidebar()`, allow for this functionality in a CollabraSuite campus.

Samples

Sample code is provided here to demonstrate typical uses of the Integration API.

Creating a document

```
CSuiteCollaborationRemote collaboration =
        CSuiteFactory.getCSuiteCollaborationRemoteInstance();
CSuiteAdminRemote administration =
  CSuiteFactory.getCSuiteAdminRemoteInstance();
// Build a campus descriptor
String campusName = "SampleCampus";
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);
// Owner of the document
String userName = "testUser";
String userDesc = CSuiteUser.buildUserDescriptor(campusName, userName);
CSuiteUser csUser = new CSuiteUser(userDesc);
// Get the file type for text documents
CSuiteDocumentType textDocType =
administration.getDocumentTypeByExtension(csCampus, "txt");
// Document's location inside of CollabraSuite
CSuiteDocumentInfo document =
       new CSuiteDocumentInfo(new CSuiteDocument("/", textDocType));
document.setDescription("File description");
// Path to the existing file to be imported into CollabraSuite
File file = new File("exampleFile.txt");
```

```
// Create the file in the user's briefcase
collaboration.createDocument(csUser, document, file);
```

Sending a Page to a User

```
CSuiteCollaborationRemote collaboration =
        CSuiteFactory.getCSuiteCollaborationRemoteInstance();
String campusName = "SampleCampus";
String userName = "testUser";
String userDesc = CSuiteUser.buildUserDescriptor(campusName, userName);
CSuiteUser csUser = new CSuiteUser(userDesc);
// Build a Set of recipients
Set toUsers = new HashSet();
userSet.add(csUser);
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);
// Send the page
collaboration.sendPage(csCampus, null, toUsers, "subject", "Page text",
                       PageConstants.NO_RESPONSE_REQUIRED_MODE);
```

Retrieving Online Users

```
CSuiteAdminRemote admin = CSuiteFactory.getCSuiteAdminRemoteInstance();
String campusName = "SampleCampus";
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);
// Get the online users and print them out
Collection onlineUsers = admin.getOnlineUsers(csCampus);
for (Iterator i = onlineUsers.iterator(); i.hasNext(); ) {
       CSuiteUser user = (CSuiteUser) i.next();
       System.out.println(user);
}
```

Retrieving Rooms

```
CSuiteAdminRemote admin = CSuiteFactory.getCSuiteAdminRemoteInstance();
String campusName = "SampleCampus";
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);
```

```
// Iterate over all Buildings, floors and rooms
Collection buildings = admin.getBuildings(csCampus);
for (Iterator i = buildings.iterator(); i.hasNext(); ) {
    Collection floors = admin.getFloors((CSuiteBuilding) i.next());
for (Iterator j = floors.iterator(); j.hasNext(); ) {
        Collection rooms = admin.getRooms((CSuiteFloor) j.next());
        for (Iterator k = rooms.iterator(); k.hasNext(); ){
            CSuiteRoom room = (CSuiteRoom) k.next();
            System.out.println(room);
        }
    }
}
```

Retrieving/Printing a User's Skills

```
CSuiteAdminRemote admin = CSuiteFactory.getCSuiteAdminRemoteInstance();
String campusName = "SampleCampus";
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);
String userName = "testUser";
String userDesc = CSuiteUser.buildUserDescriptor(campusName, userName);
CSuiteUser csUser = new CSuiteUser(userDesc);
// Get the user's skills and print them out
Collection userSkills = admin.getSkillsForUser(csUser);
for (Iterator i = userSkills.iterator(); i.hasNext(); ) {
    String skill = (String) i.next();
    System.out.println(skill);
}
```