LAB 4: Working with Java

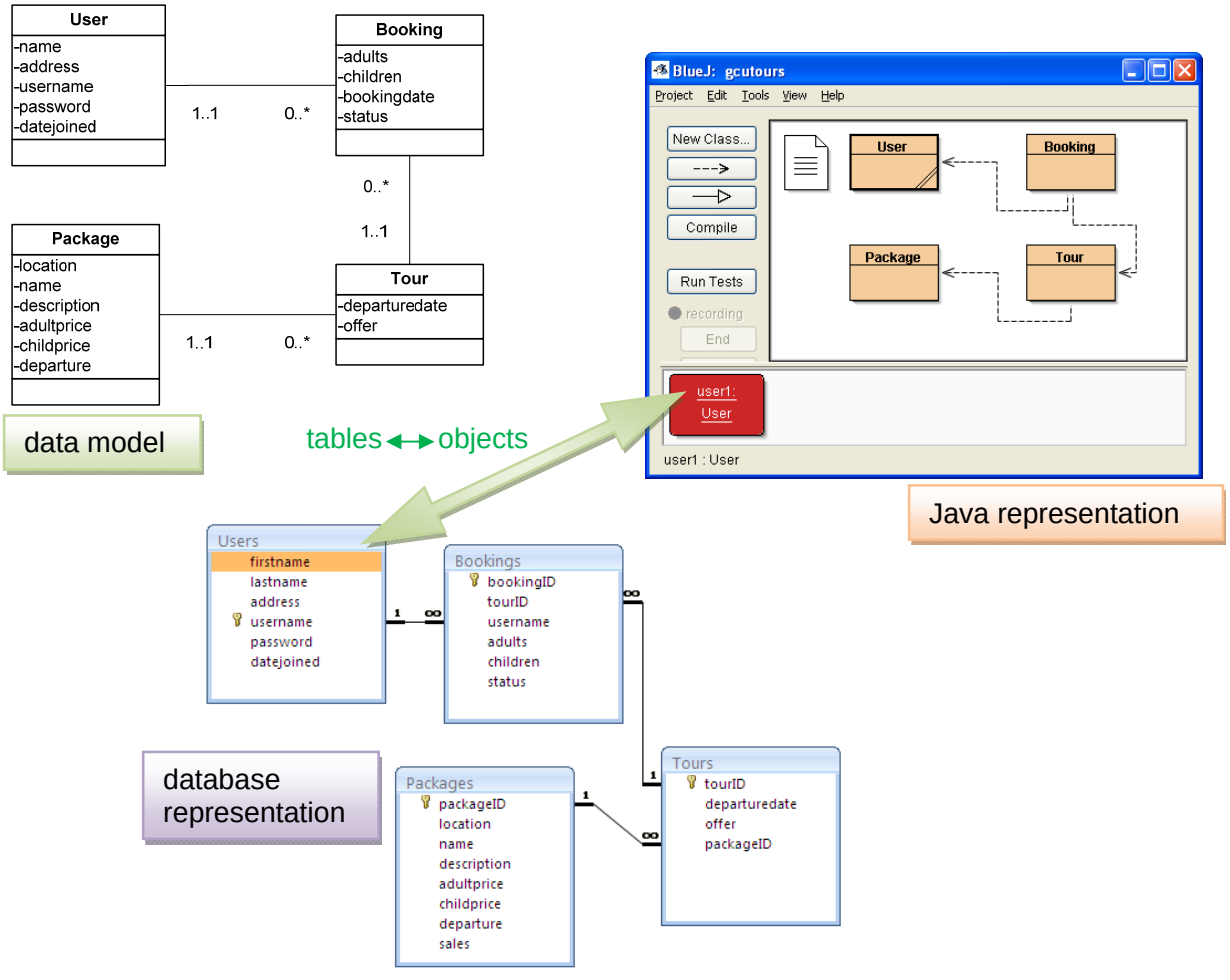
Introduction

Wait a minute! This is a **database** module, isn't it?

Well, yes, but databases are often vital components of applications written in popular programming languages such as Java. In this lab you will look at a Java project, in BlueJ, which uses the GCUTours database. The Java code will be provided for you.

The data model

In this system, the **data model** (or **domain model**) is represented as Java classes **and** as database tables. The **business logic** of the system is carried out by **Java objects**, while the **database** provides permanent **storage** for those objects. Java objects are stored in the database and retrieved later when they are needed.



Task 1

In this task you will create a new Java object and investigate what happens to that object when the Java project is closed down and then opened again.

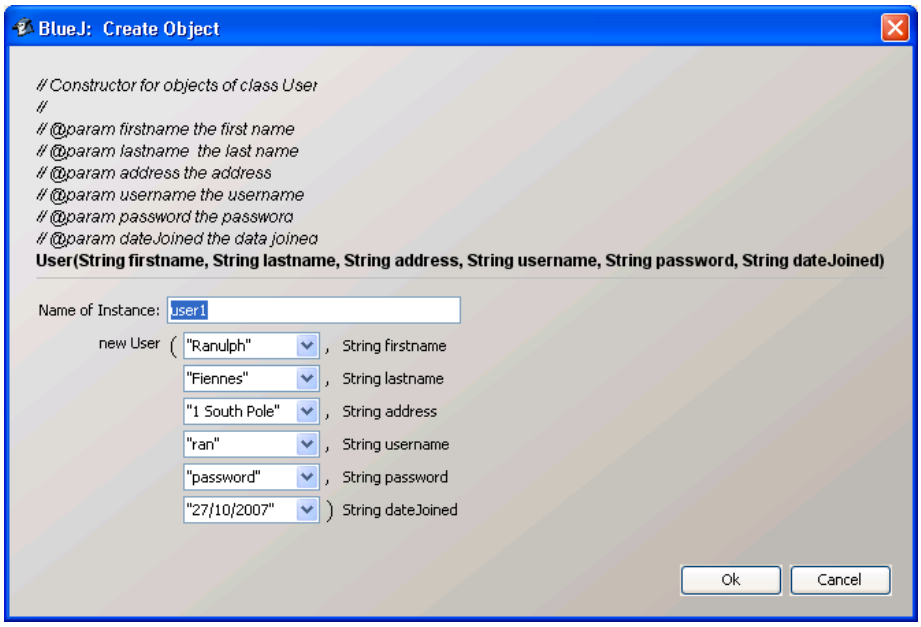
1. Start the **IDE Apps** virtual machine in VMWare player. Create a new folder called *databases* inside *Documents*.

2. Download the database file **gcutours_app.mdb** from your course web site and save it in the folder *Documents\databases*. Note that this is a Microsoft Access database which contains the same data as the GCUTours database you have used before.

3. Download all the Java files from your course web site and save them in *Documents*. There should be eight Java files altogether.

4. Start BlueJ and create a new project called *gcutours* in *Documents*. Select the **Edit > Add Class from File** menu option. Select **User.java** from the files you downloaded and click **Add**. A new class *User* should appear in the project. Compile the *User* class.

5. Create a new *User* object using the values shown in the figure:



6. Inspect the *User* object, *user1*, on the Object Bench to make sure that the attributes have been set correctly.

7. Now close down BlueJ. Start BlueJ up again and open the *gcutours* project (you may not need to open it – BlueJ usually automatically opens the last project you were working on).

Is the user1 object still in the Object Bench? Why or why not?

How could you get the user that you created back onto the Object Bench again?

Task 2

In this task you will **store** a new Java object in a database and investigate whether the object can be **retrieved** after the Java project has been closed and opened again.

1. Add a new class called *UserDAO* to the project using the file **UserDAO.java**, and compile this class.

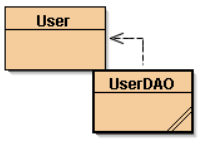
2. This new class is an example of a Data Access Object (DAO) class, whose job is simply to get objects in and out of a database. The job of *UserDAO* is to get *User* objects in and out of the GCUTours database.

3. *UserDAO* needs to know where the database file is. Open *UserDAO* in the BlueJ editor, and look for the following lines:

```
Connection c = DriverManager.getConnection(
        "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};" +
        "DBQ=c:/Users/student/My Documents
        /databases/gcutours_app.mdb"
)
```

   Make sure the path following "DBQ" matches the location of your database file.

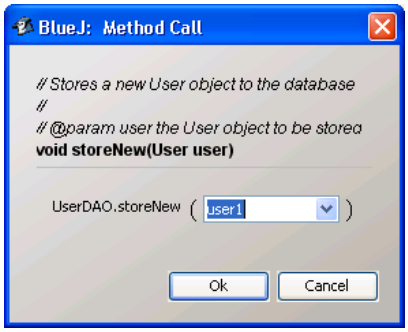
   Your project now has two classes. They are closely related, so it's a good idea to move them close together in the BlueJ class diagram:



4. Recreate the *User* object, *user1*, which you first created in Task 1.

5. Right-click the *UserDAO* class and select `void storeNew(User user)`.

6. To store *user1*, type *user1* in the box in the **Method Call** dialog, or click first in the box then on the *user1* object in the Object Bench, and click OK.



The Terminal window should open with a message, which will report what has happened. This should consist of an SQL query which has been performed, and a report that 1 row has been inserted. If there is an error message, you should check first that your database is available in the correct folder on your computer.

Write down the SQL query, and describe what this query does

---

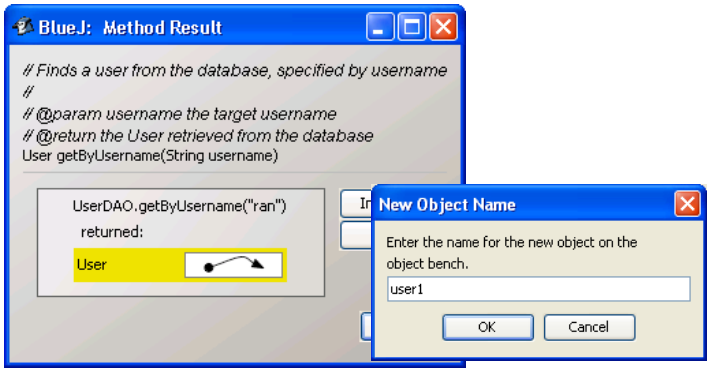
7. Now close down BlueJ. Start BlueJ up again and open the *gcutours* project if it is not open.

   Is the user1 object still in the Object Bench? Why or why not?

---

8. Right-click the *UserDAO* class and select `User getByUsername(String username)`.

9. Enter *"ran"* in the box in the Method Call dialog and click OK. The method result is a *User* object. Click **Get** in the **Method Result** dialog. Enter *user1* for **New Object Name** and click OK. Click **Close** in the Method Result dialog. A new object should appear on the Object Bench.



Inspect the new user1 object. What are its properties?

Write down the SQL query, and describe what this query does.

What role has each of the following played in this task: User. UserDAO, database

---

NOTE

It looks like you have **stored an object** in the database. Actually, a database doesn't really store Java objects. What happens is that a **row in a database table** contains the data needed by the Java application to **reconstruct** the original Java object.

Task 3

In this task you will retrieve **many objects** of the same type from the database.

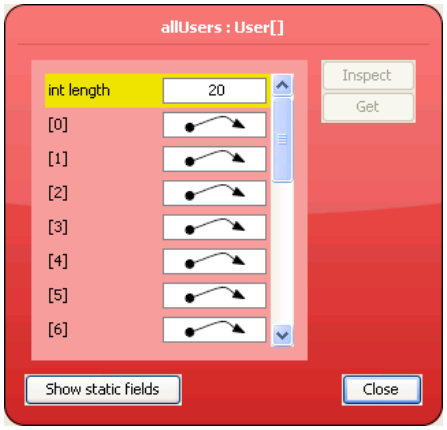
1. Right-click on *UserDAO* and select `User[] getAll()`.

2. Click **Get** in the Method Result dialog and choose *allUsers* for the New Object Name. Close the Method Result dialog. You should see a new object on the Object Bench with an unusual icon:

3. This looks like a pile of cards, and shows that the object is in fact many objects of the same type. In Java, this is called an **array** – *allUsers* is an **array of *User* objects**.

   Write down the SQL query, and describe what this query does.

4. Inspect *allUsers*. You will see an array of results, numbered 0 to 19, each of which is an object reference. Highlight any one in yellow by clicking on it, and click **Inspect**.



What are the properties of the object? What kind of object is it? Can you find the corresponding data in the database?

NOTE

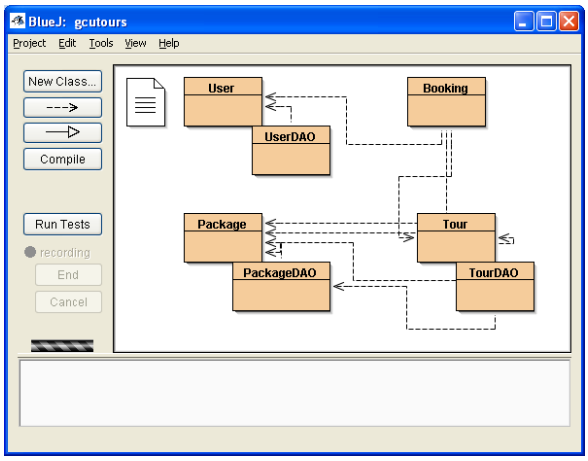
The array used here is limited to 20 results for simplicity. In Java, the objects in an array are counted starting with 0, not 1, so they are numbered 0 to 19. There are better ways of dealing with an unknown, and possibly large, number of results returned by a query.

Task 4

In this task you will retrieve an object which has an **association** with another object.

1. Add *Package*, *Tour* and *Booking* classes to your project using the files **Package.java**, **Tour.java** and **Booking.java**.

2. Add *PackageDAO* and *TourDAO* classes to your project using the files **PackageDAO.java** and **TourDAO.java.** For clarity, arrange your classes so that each DAO class is close to its data model class. Compile all the classes.



3. Right-click on *PackageDAO* and select `Package getByPackageID(int packageID)`. Choose a value of 2 for the *packageID*.

4. Get the *Package* object which is returned to the Object Bench as *package1*. Inspect *package1*. Check that the field values correspond to a row in the *Packages* table in the database.

   Write down the SQL query, and describe what this query does.

---

5. Right-click on *package1* and **Remove** it from the Object Bench.

6. Right-click on *TourDAO* and select `Tour getByTourID(int tourID)`. Choose a value of 6 for the *tourID*.

7. Get the *Tour* object which is returned and place in onto the Object Bench as *tour1*. Inspect *tour1*.

8. What field in the Java object corresponds to the database field *packageID*? What does the field in the Java object contain?

9. Highlight the *holidayPackage* field in the Object Inspector and click Inspect.

What is the packageID of the holidayPackage object?

Write down the SQL queries, and describe what these queries do. Why are there two queries? What does TourDAO do with the results of the query on the Packages table?

Task 5

In this task you will see some **business logic** being done by the Java objects, with the result stored in the database.

1. Add a *BookingDAO* class to your project using the file **BookingDAO.java**. Compile the class.

2. Use appropriate methods of *UserDAO* and *TourDAO* to retrieve the following objects from the database and place them on the Object Bench.

Type	Which object?	Name in Object Bench
User	*username*="vdagama"	*user1*
Tour	*tourID*=7	*tour1*

3. Inspect these objects and compare with the database to check that they contain the correct field values.

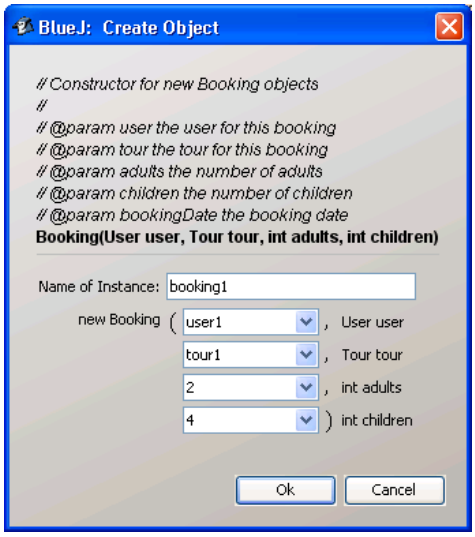
You saw in task 4 that when you retrieve a *Tour* object you also get the associated *Package* object - let's get that onto the Object Bench too.

4. Right-click on *tour1* and call `Package getHolidayPackage()`. Place the result as *package1* on the Object Bench. Inspect it to make sure *packageID* is **4**.

What is the value of sales for the package?

Now that you've found a user and a tour, you will **make a booking** for that user on that tour.

5. Create a new *Booking* object called *booking1*, using the values shown in the figure on the next page.



6. Inspect *package1* again.

   What is the value of sales now? How has the new value been calculated?

> **NOTE**
>
> This is an example of **business logic** done by the Java objects – when a booking is made then the number of sales for the package must be updated.

Now you need to **store the booking**.

7. Use the method `void storeNew(Booking booking)` of *BookingDAO* to store booking1.

   Write down the SQL query, and describe what this query should do.

Task 6

Additional things to try if you finish the other tasks:

1. The DAO classes contain methods to query the database in different ways. Experiment with these, and think about how they might be useful.

2. If you feel like writing some Java, note that *UserDAO* does not have a method to find users by last name.

   Write and test a new method in *UserDAO*, called `getByLastname`. How many results would you expect this to return? This will affect how you write the method.