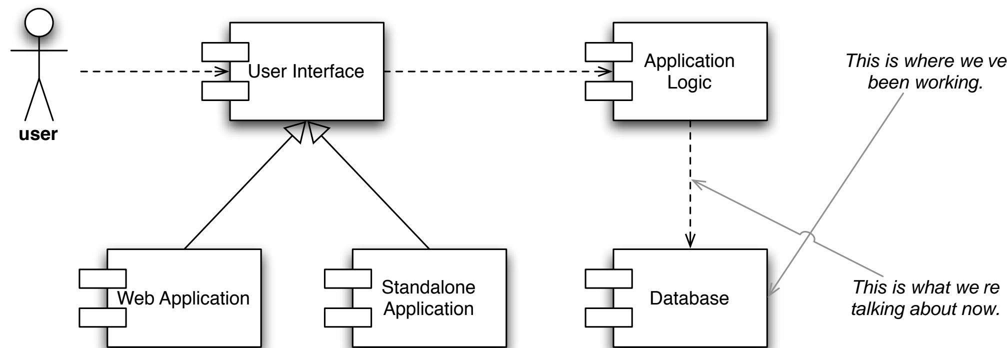
Database Programming

- At this point, we've learned how to:
  - Specify a database (E-R, UML, relational)
  - Express queries and other operations in the relational database model (relational algebra)
  - Define and manipulate a real instance of that database (SQL, PostgreSQL)
- Of course, your end-users won't know SQL; they will interact with your database the way they interact with all other software — through an application



- We develop an application using the same tools for any other application — nothing different here
- But, since our application needs to hit the database, part of our code needs some "gateway" into that world
- In our case, we need programmatic access to a relational database and SQL

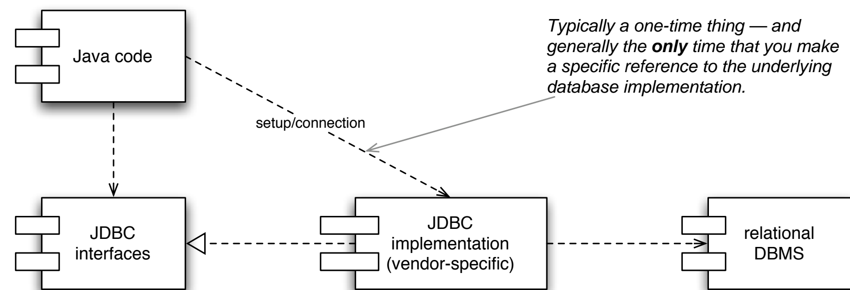
Database Programming Interfaces

- *Embedded SQL:* an SQL layer "on top of" an existing language; a preprocessor snags the embedded SQL statements and converts them into lower-level programming calls
  - A "cursor" object represents a current retrieved result, which the rest of the code can access
  - Queries directly expressed ("canned") in the code, which leads us to…
- *Dynamic SQL:* allows us to form queries on the fly

- A lot of these things have been developed over the years:
  - Embedded SQL has been implemented for C, Cobol, Pascal, Java (SQLJ), PL/I, Fortran, etc.
  - Two major dynamic SQL implementations:
    - ODBC ("Open Database Connectivity") for the C family of languages, as well as C# and Visual Basic
    - JDBC (formerly known as "Java Database Connectivity") for Java
- In this class, we focus on:
  - Database applications written in Java (Swing, Web, plain text, whatever)
  - JDBC, using the "driver" for PostgreSQL

How JDBC Works

- JDBC is a set of Java interfaces that determine how Java software can interact with a relational database
- Standard-issue with Java ≥ 1.4; optional extension in older versions
- To see the full set of interfaces, look up the *java.sql* and *javax.sql* packages
- Database vendors (PostgreSQL, MySQL, Oracle, Sybase) provide implementations of these interfaces (sometimes called "drivers")



For PostgreSQL specifically:

- Download the driver from http://jdbc.postgresql.org
- Make sure it's in your classpath
- For questions that are *specific* to the PostgreSQL JDBC implementation, see the Documentation section
- Otherwise, any generic JDBC resources on the Web will suffice

Key Concepts

We'll generally let the code and API do the talking here, but here are some broad strokes:

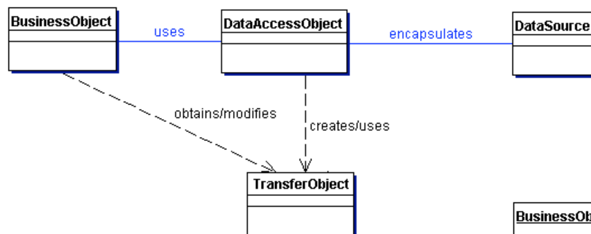
- You reach the database through an explicit *Connection*, acquired either through the JDBC *DriverManager* or a JDBC *DataSource* (recommended, JDBC 2 and later)
- Queries are prepared as straightforward strings; facilities are available for parameterizing them
- Query results are returned as *ResultSet* objects, over which your code would iterate in order to get to the underlying tuples

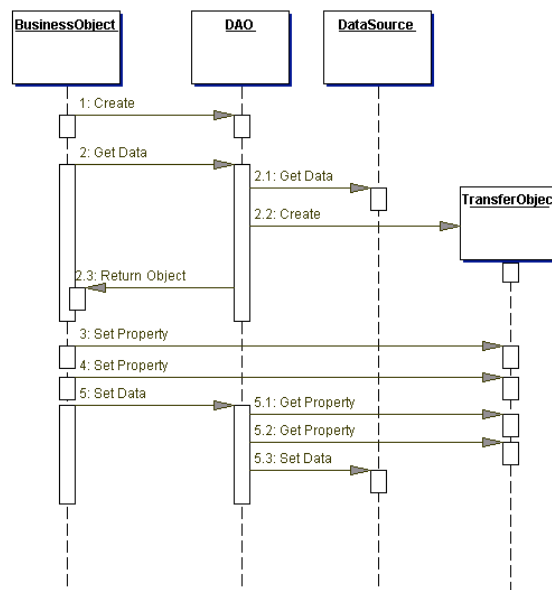
Implementation Issues

- The underlying database is a *finite* resource — your code needs to clean up after itself (i.e., close connections after you're done with them)
  - *Connection pools* help with this; available if you use *DataSource*
- Unless you really really have to, you shouldn't be married to a specific database implementation — as long as it's relational, your code should work
- From application to application, it is really the *queries* that differ; otherwise, a lot of the code is functionally identical — this observation leads us to…

The Data Access Object (DAO) Design Pattern

- Ultimately, JDBC is a programming interface — but there is a right way and a wrong way to use it
- The Data Access Object design pattern is a design approach that hides the underlying database implementation from the rest of the code
- The application should worry about *what* data it needs and when, but not *how* the data is retrieved or stored
- Official Web site: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html





DataSource: the underlying database implementation

DataAccessObject (DAO): abstracts your application's data needs for a specific category of data

[*Data*]*TransferObject (DTO)*: the object representation of this data

BusinessObject: the rest of the application