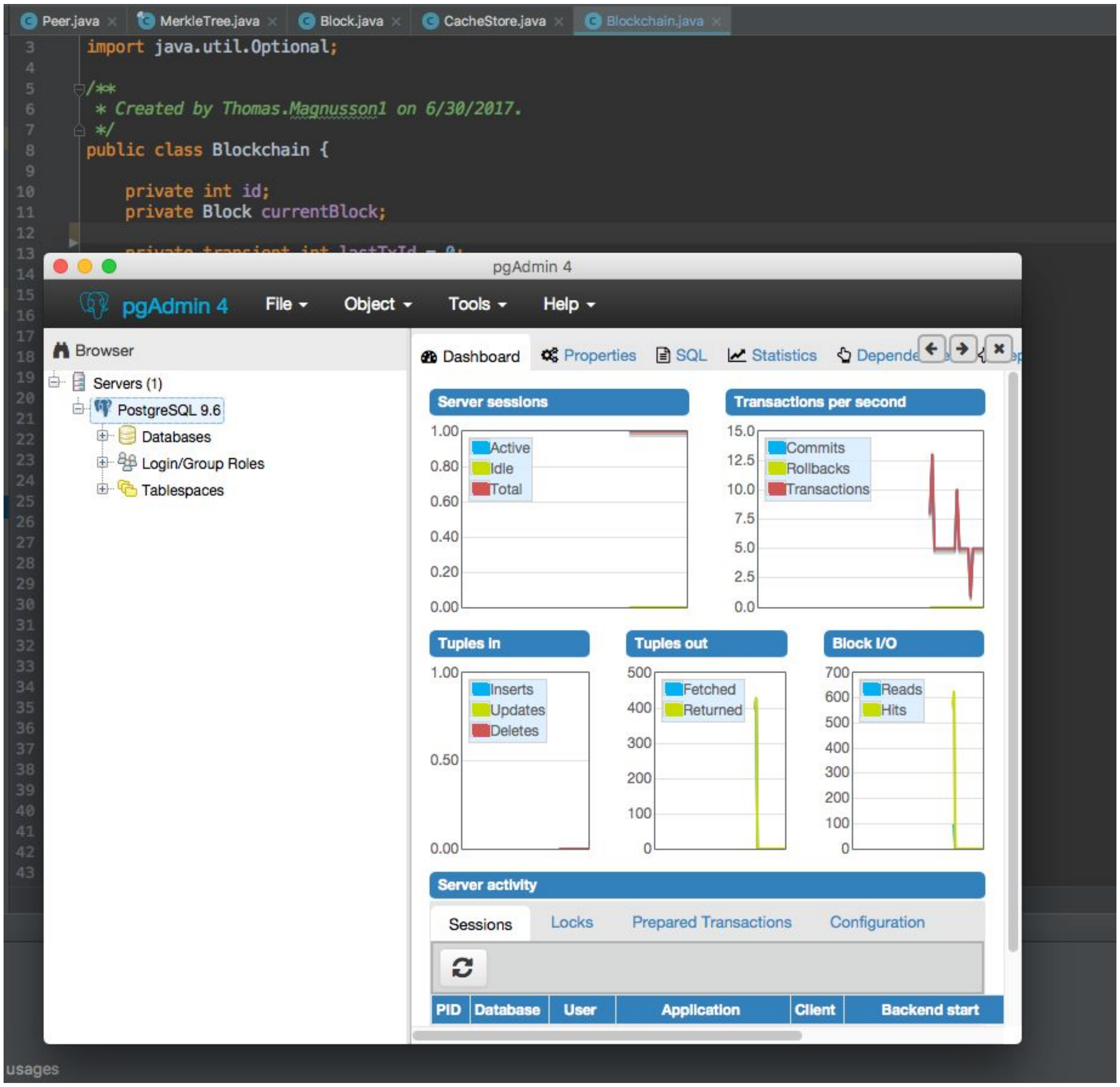
1. Screenshot (who else would have a blockchain class in IntelliJ?):



```java
import java.util.Optional;

/**
 * Created by Thomas.Magnusson1 on 6/30/2017.
 */
public class Blockchain {

    private int id;
    private Block currentBlock;

    private transient int lastTxId = 0;
```

2. Data vs. Information: Weather.com's database for weather I think would be an interesting database full of lots of data, such as: 67, 68, 70, 74, 43, NNW, NW, W, 30.21. All of these data are without context; they are meaningless unless paired with more information about what they represent. The first sequence of integers are actually temperature measurements in degrees fahrenheit at certain points of the day in a certain location

(unspecified here). The three strings are directions of the wind, e.g. NNW stands for North North West. The floating point number at the end is the barometric pressure in inches of mercury. Most of this information allows people to understand what weather conditions might be for a particular time in the day at a certain location. This is beneficial for planning events, such as a picnic. If Weather.com shows the location and time at which the picnic is planned is going to be 41 degrees fahrenheit and raining, with winds blowing NW at 15 miles per hour, the picnickers might want to reschedule. This information is also useful for planning what to wear on a given day. The importance of units and context cannot be overstressed; the difference between fahrenheit and celsius is quite large, and dressing according to one and experiencing the other might give someone a shock.

3. Data models: The hierarchical model of databases relied on a tree structure to keep track of which pieces of data belonged to which other pieces of data. For example, a Player in a Game would have many Items. The relationship would show Game as the root, Player as a child of Game, and many Items as children of Player. A problem arises, however, when there exist unowned Items… every piece of data must fit on this graph. To whom do they belong? Is it appropriate to put the Items under Game? That would break the homogeneity of Game's children. Perhaps there must exist a ghost Player, who owns all the unowned Items. What happens when a naive programmer, not understanding the full schema of the hierarchy, asks the structure for how many Players there are in the game? The structure happily offers a number one greater than the actual amount of Players because of this ghost Player. Such Off-By-One-Bugs make basic hierarchy configurations

require lots of documentation. The hierarchical model also did not allow for a single instance of a particular piece of data to be owned by many parents. This required duplicating the given piece of data, which increased the mutable state of the database dramatically. Duplication destroys data integrity and consistency. The network model fixed this problem by allowing children to have multiple parents, although this did not fix the unowned Item problem from before. Both the network and hierarchy models were a huge step forward in database management because they allowed physical data independence. Database creators could focus on the abstract concepts of hierarchies and networks to model their code, rather than the underlying systems that implemented these models. Separating implementation from ideation allowed database creators to "write once, run anywhere" (anywhere there was an implementation of each of these models available).

a. The relational model takes a non-graph approach to databases. All entities are expressed through rows and columns, records and attributes respectively. To express a relationship between two entities, for example, a Player owns a particular Item, one creates another table to express that relation, consisting of that item's key and that player's key. A key is a piece of data that represents the identity of a given player. That is to say, no two players that are different have the same key, and if two players have the same key they are the same player. In this way, the aforementioned unowned Item problem disappears. The table containing Player ownership of Items simply doesn't contain a record referencing the unowned Items. The Items table, however, still contains a record of the unowned

Items. Entities are not duplicated thanks to the key references. Modifying one Item in the Item table actually modifies that singular Item. All other places containing a reference to that Item only contain its key.

b. eXtensible Markup Language (XML) is a human-readable data storage format. How it compares to other data models depends on how one uses it. There are two types of XML documents: well-formed, in which a serializer creates arbitrary HTML-like tags of elements without a strict format, and valid, in which a serializer create a Document Type Definition (DTD) document that dictates what possible elements are valid in a valid XML document. Valid XML is closer to relational databases in terms of strict types and definitions, meaning it is more difficult to create dynamic databases. Well-formed XML allows arbitrary structure for more fluid databases at the sacrifice of security and the "forced" documentation DTDs and Schemas afford. I don't think I would use XML as a data model, simply because it is verbose and clunky compared to its kin JavaScript Object Notation (JSON) and Graph Query Language (GraphQL). XML also offers too many arbitrary options for storing data in elements; should the year of a movie be a separate element within <Movie>, or should it be an attribute of that element, i.e. <Movie year="2007">? I also don't think XML would scale well considering its main use is for quick data-interchange over a network rather than persistent database storage on a disk.