Design: introduction
Part of this design was taken from the previous Java RMI lab. CarRentalCompany, CarType, Car, Reservation, Quote, ReservationException and ReservationConstraints remain largely the same, meaning all mentioned classes except Car are Serializable (because they need to be communicated to the customer) and CarRentalCompany is annotated Remote (because it needs to be remotely accessible).

Central lookup & session management
The class CarRentalAgency followed very naturally from this structure, since we needed an instance to keep track of all CarRentalCompanies and (un)register them if necessary.
At first glance, it would be easy to give CarRentalAgency the (additional) responsibility of managing communication with the client. However, this would give CarRentalAgency 2 distinct responsibilities (communication management AND company lookup/management). We instead created a new class SessionManager to deal with communication related issues and life cycle management of sessions (essentially implementing part of the functionality of Session Beans) and in doing so avoided a cluttered CarRentalAgency class. As a consequence, SessionManager is annotated Remote (CarRentalAgency isn't!).

Sessions
Next up are the sessions themselves. Since managers and clients need to perform different activities, we created two distinct classes, ManagerSession and RentalSession. Shared functionality (e.g. identification, the SessionManager they belong to etc.) are implemented in a superclass Session, following basic OO design principles. Naturally any session needs to be remotely accessible because it provides a mechanism to query information from CarRentalAgency.

What runs where?
In this design we made the assumption that CarRentalAgency and SessionManager run on the same machine. Different instances of CarRentalCompany and Session can run on different machines (not the same machine as the one for CarRentalAgency and SessionManager). These are natural assumptions because companies and clients normally use different machines and a central server for multiple companies can also be in a separate location.

RMI registry
Instances of RentalSession, ManagerSession and RentalCompany are registered via the built-in RMI registry to perform lookups. These lookups are executed by the CarRentalAgency. No other objects need to be registered via the RMI registry because they all depend on either a Session or a RentalCompany for lookups.

Life cycle management of session
The SessionManager is in charge of creating and deleting sessions. A SessionManager is always linked to one CarRentalAgency and thus it can provide the CarRentalAgency easily to any session it creates. The SessionManager also registers sessions via the RMI registry.

Synchronization
When two or more session run concurrently, there needs to be a mechanism to ensure thread-safety. This means that methods in CarRentalAgency and CarRentalCompany (classes

that all sessions can query) need to be synchronized. Both CarRentalAgency and CarRentalCompany can become bottlenecks because of this. (Suppose 100 clients want to make a reservation with the same company. All these reservation would need to be executed sequentially, which is very slow.)

Annotations: recap

Classes that are annotated <<Remote>>: CarRentalCompany, SessionManager, RentalSession, ManagerSession

Classes that are annotated <<Serializable>>: CarType, Reservation, Quote, ReservationException, ReservationConstraints