# Design: introduction

Part of this design was taken from the previous Java RMI lab. `CarRentalCompany`, `CarType`, `Car`, `Reservation`, `Quote`, `ReservationException` and `ReservationConstraints` remain largely unchanged, 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 the system needs a way to keep track of all the `CarRentalComapanies` 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` two distinct responsibilities (session 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 Bean containers) and in doing so avoided a cluttered `CarRentalAgency` class. As a consequence, `SessionManager` has a remote interface (clients request new sessions from `SessionManager`) and the `CarRentalAgency` is a local class because it is only accessed directly by the local versions of the sessions and not by the clients.

# Sessions

Next up are the sessions themselves. Since managers and clients need to perform different activities, we created two distinct classes, `ManagerSession` and `RentalSession` which share some properties with stateless and stateful session beans respectively. Shared functionality (e.g. identification, the `SessionManager` they belong to etc.) are implemented in an abstract superclass `Session`, following basic OO design principles. Because only `ManagerSession` and `RentalSession` need to be remotely accessible and as a consequence they have a remote interface and `Session` does not.

# Distributed setup

In this design we made the assumption that `CarRentalAgency` and `SessionManager` run on the same machine. Different instances of `CarRentalCompany` and sessions 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`, `CarRentalCompany` and `SessionManager` are registered via the built-in RMI registry to perform lookups. On setup only the `SessionManager` is added to the registry and is used to create new sessions that are added to the registry. The `RentalCompany` instances are (possibly) located in other RMI registries. The `ManagerSession` is responsible for registering these remote

companies at the `CarRentalAgency`.

## Life cycle management of session

The `SessionManager` is in charge of creating and deleting sessions. A `SessionManager` is always associated with one CarRentalAgency and thus it can provide the `CarRentalAgency` easily to any session it creates. The `SessionManager` also registers sessions via the RMI registry and closes them when a `close()` call is performed on the session.

## 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`.