

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Advanced Java ProgrammingCAR RENTAL COURSEWORK | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | ALEXANDROS **RANTOS**  **CHARISOPOULOS**  Student Number  **190587101** |  | | | |
|  | | | | |  | | --- | | NEWCASTLE UNIVERSITY |  |  | | --- | |  |   DATE  23/10/2019 |  | | | |
|  | | |  | | |  | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | DESIGN EXPLANATION | | |  | |
|  | |  |  |  |
|  | HIGH LEVEL EXPLANATION The purpose of creating the AbstractRentalCompany was to provide as generic implementation as possible. For that reason two HashMaps were added as class members. The key for both of these Maps are the RegistrationNumber’s hashcode of each company’s Car. So in carsFleet all company’s cars are saved based on their hashcode (which is overridden to be the registrationNumber hashcode since that object is unique) and in contracts the persons who have rented a car are being saved again based on the car’s hashcode.  Instantiating a CarRentalCompany object a car fleet of 30 Cars with random RegistrationNumbers are automatically created as instructed. | | | | |  |
|  | INTERFACES Interfaces have been used for this application’s design in order to provide an easier development & expansion in the future. Also, in this way multiple inheritance is allowed and sets a good base and set of rules for this program.  These are the three interfaces to accomplish the above goal :   1. Car 2. Person 3. RentalCompany   Each of the above interfaces includes methods that are general and every class who implements them must have. | | | | |  |
|  | ABSTRACT CLASSES Abstract classes have been used to implement each of the above interfaces hence only three exists :   1. AbstractCar 2. AbstractPerson 3. AbstractRentalCompany   Such design gives the opportunity for easier class extension since the basic methods of the interfaces have been implemented in the abstract classes and if an added functionality is desired to be added that can be achieved by inheriting the corresponding Abstract class and override a given method or adding a new.  For example, it was mandatory to use it in order to create LargeCars and SmallCars. Rest of abstractClasses were created in order to be able to add other kinds of Persons(like a manager) or RentalCompany(a RentalCompany who rents limuzines) in the application’s ecosystem. | | | | |  |
|  | FACTORIES The requirements of the design demanded unique instances different objects thus using factories was mandatory to avoid duplicates.  Specifically, these factories have been created :   * DrivingLicenseFactory   + It was required to have unique License Numbers for every Person object. * RegNoFactory   + Similarly, one instance of a Car’s RegistrationNumber was allowed for this application’s requirements. | | | | |  |
|  | IMMUTABILITY The only immutable object of this application is the RegistrationNumber. This object once created cannot be modified. Also, since its members were only String variables there was no need to return a new String for the getter Methods since String Object is immutable.  However, immutability was used for the following class members:   * AbstractPerson.dateOfBirth * AbstractCar.tankCapacity * AbstractCar.regNo * DrivingLicence.drivingLicenceID * DrivingLicence.serialNumber   All the above members cannot be changed within the program and all the necessary precautions (e.g. defensive copying, declaring them final, absence of SetterMethods) have been taken in order to remove such functionality | | | | |  |
|  | OBJECT METHODS OVERRIDING There were situation that, in order to guarantee a safe execution of the program, object methods such as equals() & hashCode() should be overridden.  For this application, unique instance of objects(RegistrationLicence & RegistrationNumber) were an absolute requirement so overriding equals method to check equality between class members was a must. However, if equals method is overridden so should hashCode in order to provide consistent results with the equals method.  Furthermore, toString() method has been overridden either to provide a more understandable and clear output of a class or due to the fact that the instructions said so. | | | | |  |
|  | UML Class Diagram  *(photo is also provided in the .zip file)* | | | | |  |