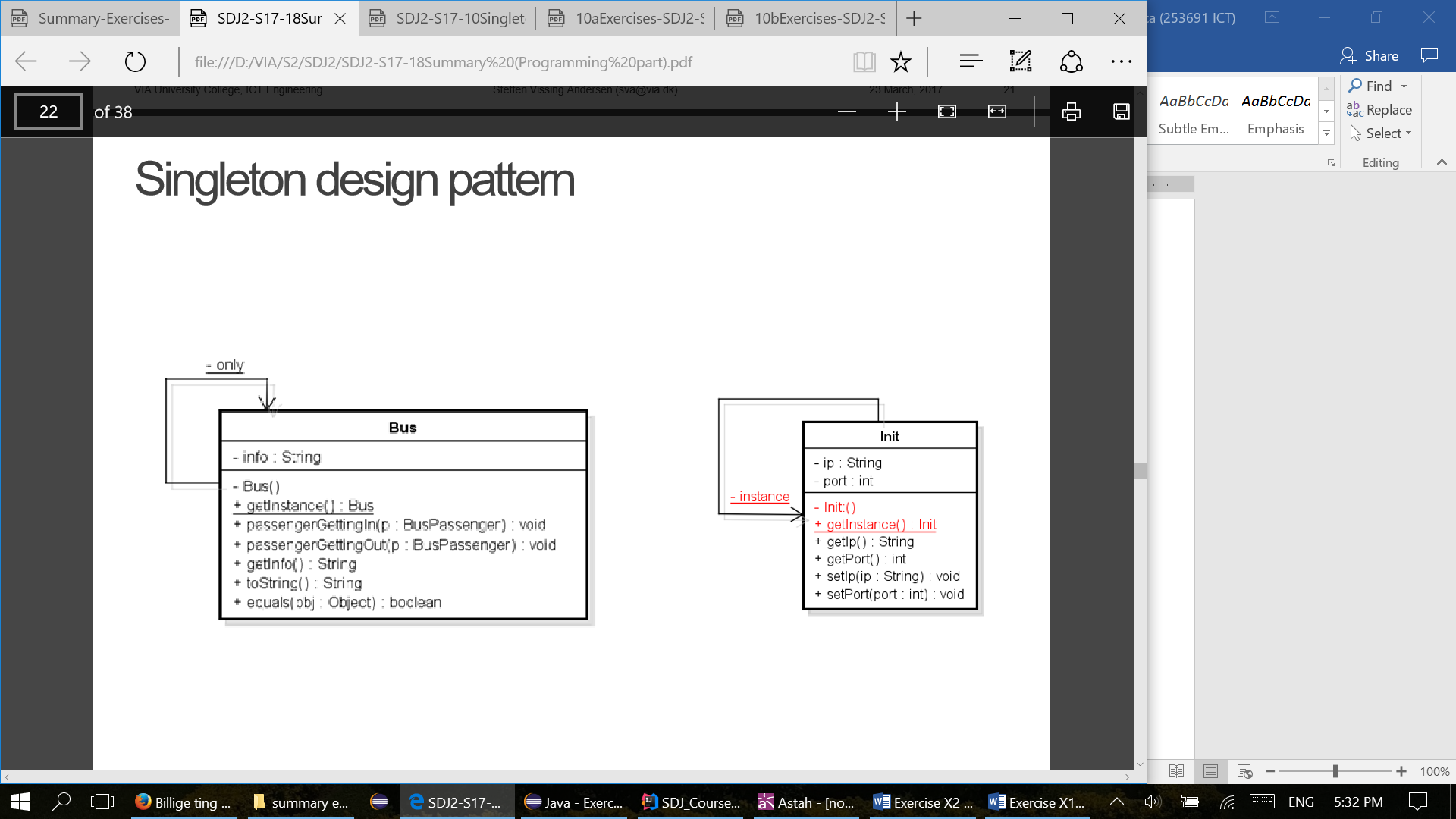
**Singleton design pattern**

**General UML class diagram for singleton design pattern:**



**The overall purpose for the singleton design pattern:**

* Singleton involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

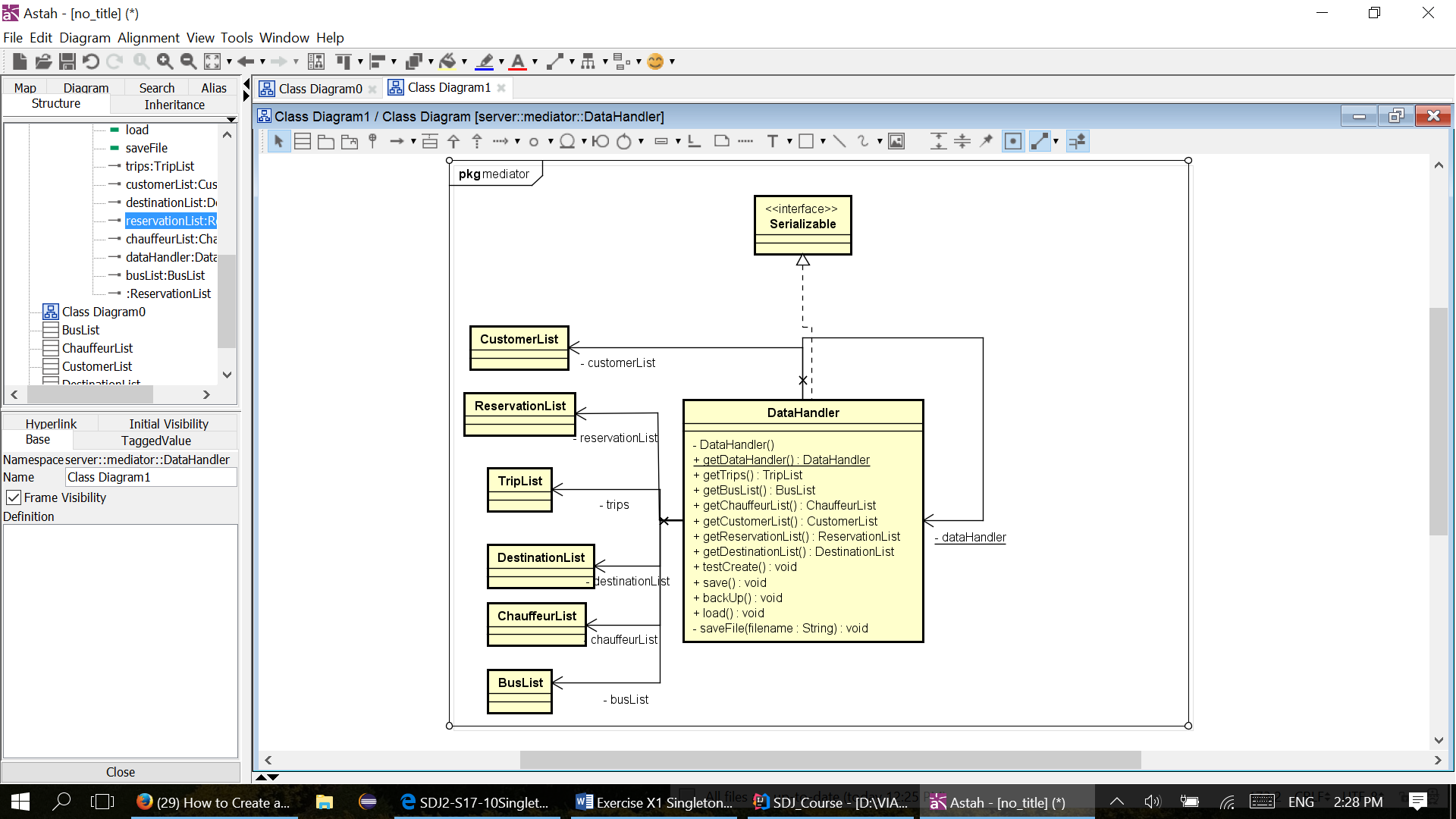
**Criteria for singleton:**

* private static Singleton instance - Private static class variable of the same type as the class that is the only instance of the class.
* private Singleton() – Private constructor to restrict instantiation of the class from other classes.
* public static Singleton getInstance() – Public static class method returning an instance of the class, this method has to ensure that only one instance is create. This is the global access point for outer world to get the instance of the singleton class.

**Usage of singleton design pattern:**

* Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.
* To use the singleton class, we need to have static member of class, private constructor and static factory method.
  + **Static member:** It gets memory only once because of static, it contains the instance of the Singleton class.
  + **Private constructor:** It will prevent to instantiate the Singleton class from outside the class.
  + **Static method:** This provides the global point of access to the Singleton object and returns the instance to the caller.

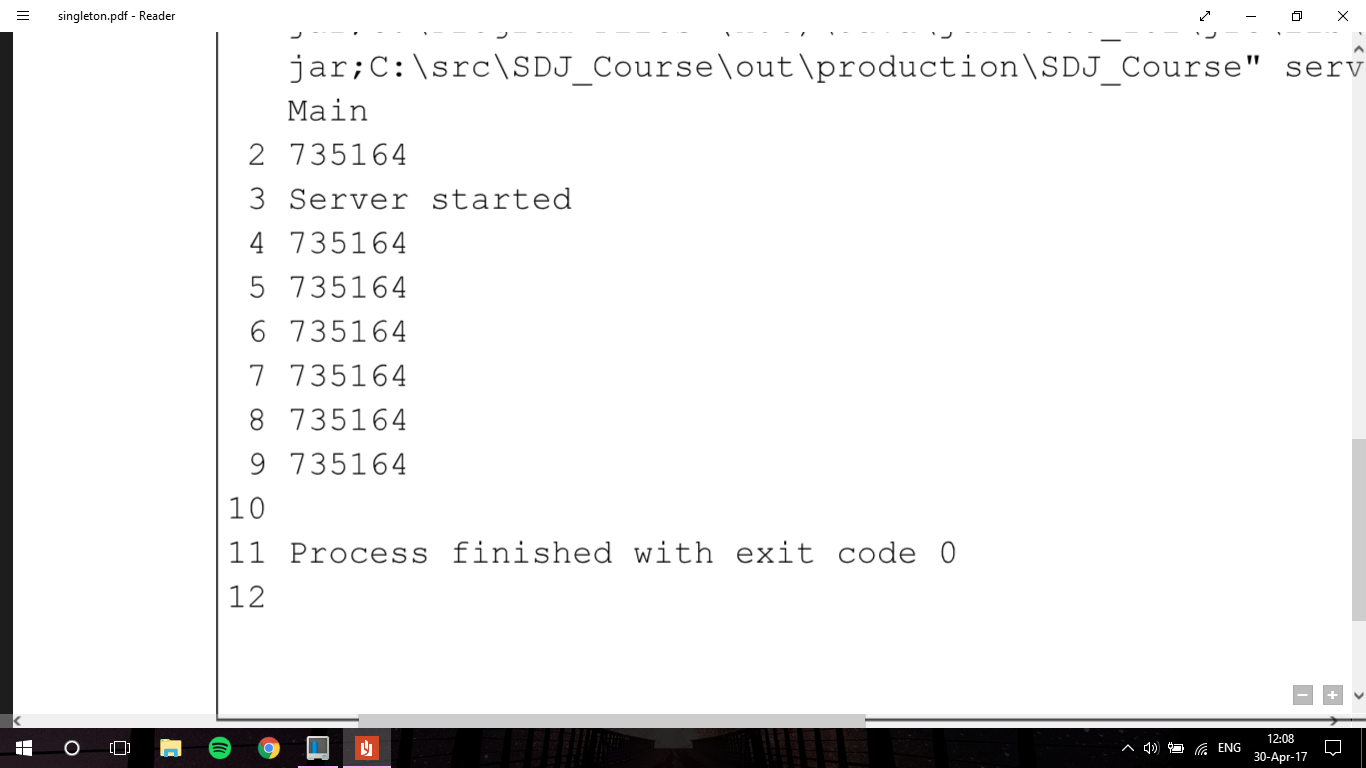
**Our implementation:**



**Code example:**

public class DataHandler implements Serializable {  
  
 public static final char *CLASSIC* = 'c';  
 public static final char *LUXURY* = 'l';  
 public static final char *PARTY* = 'p';  
 public static final char *MINI* = 'm';  
 private static DataHandler *dataHandler*;  
 private ProxyTripList trips;  
 private BusList busList;  
 private ChauffeurList chauffeurList;  
 private CustomerList customerList;  
 private ReservationList reservationList;  
 private DestinationList destinationList;  
  
  
 private DataHandler() {  
  
 }  
  
 public static DataHandler getInstance() {  
 if (*dataHandler* == null) {  
 *dataHandler* = new DataHandler();  
 }  
 return *dataHandler*;  
 }

* We have inserted printout statement in the getInstance() method in which we are printing hashcode of datahandler object to the console. By doing this we can easily see if we are using only one instance of datahandler in the whole project.
* After running project and trying several parts of program, we got following output in the console:



* Number 735164 is the hashcode of the datahandler instance. Since the number is the same in all printouts, we consider test as success, we conclude that we have implemented singleton design pattern correctly.