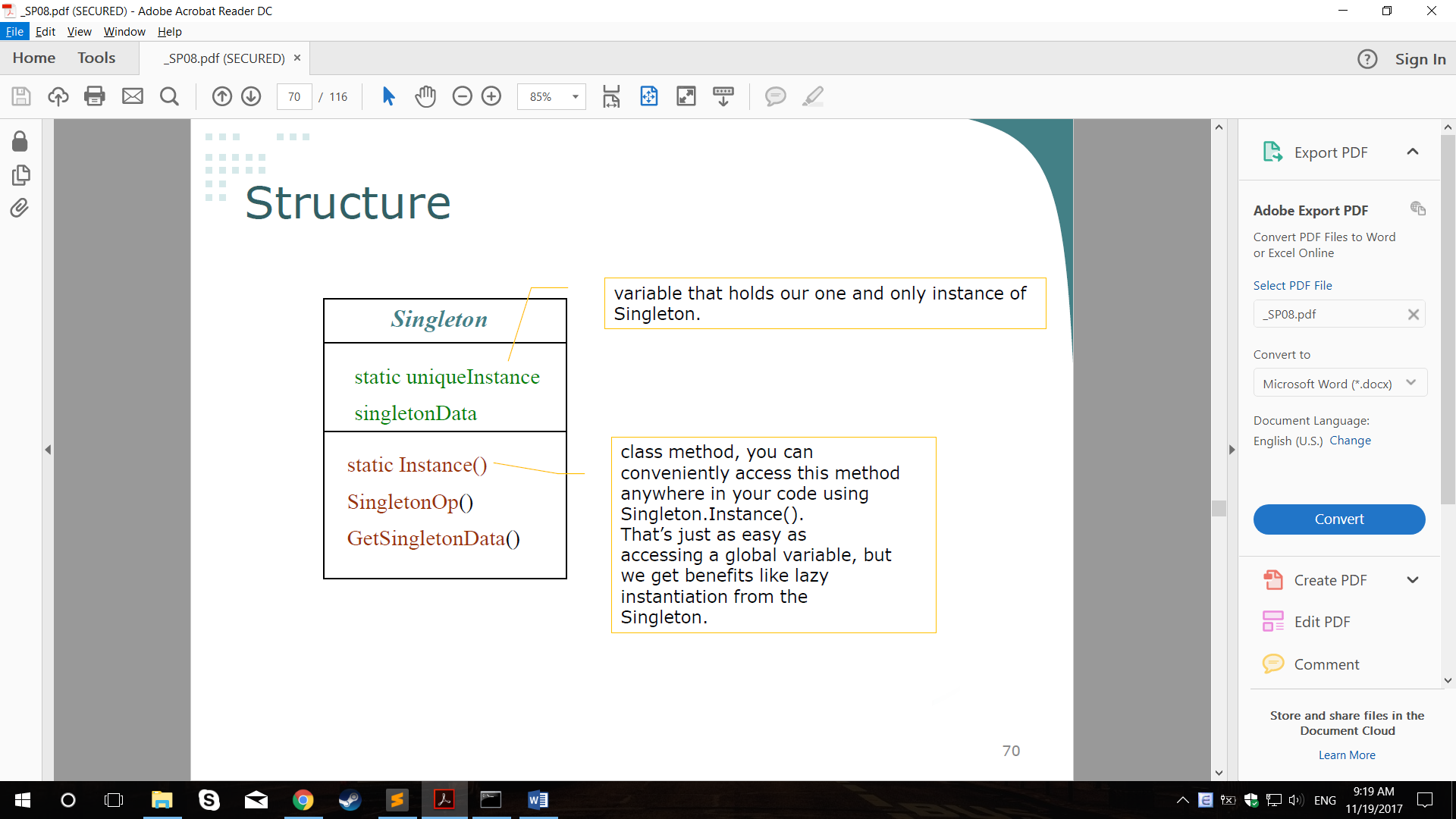
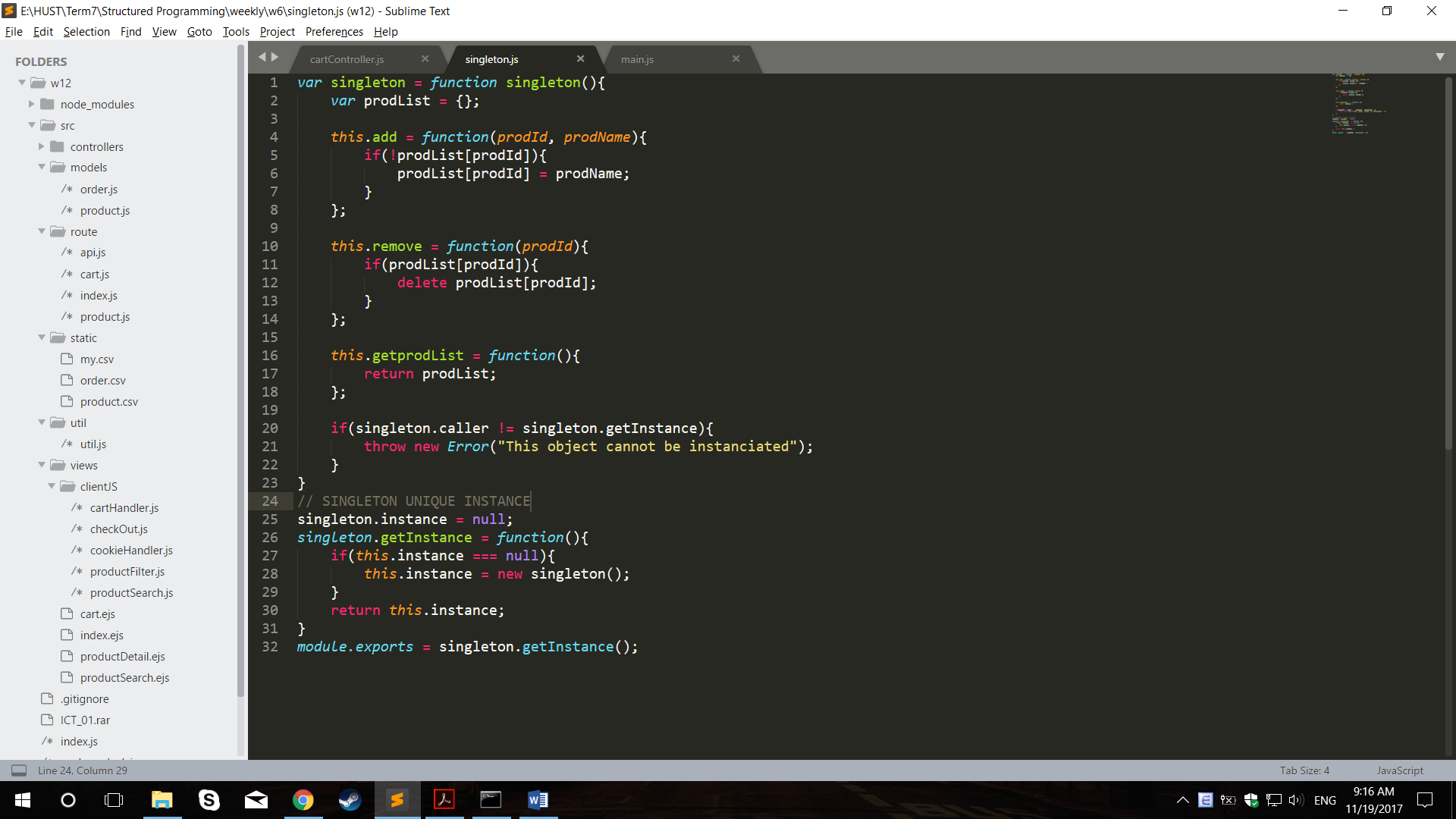
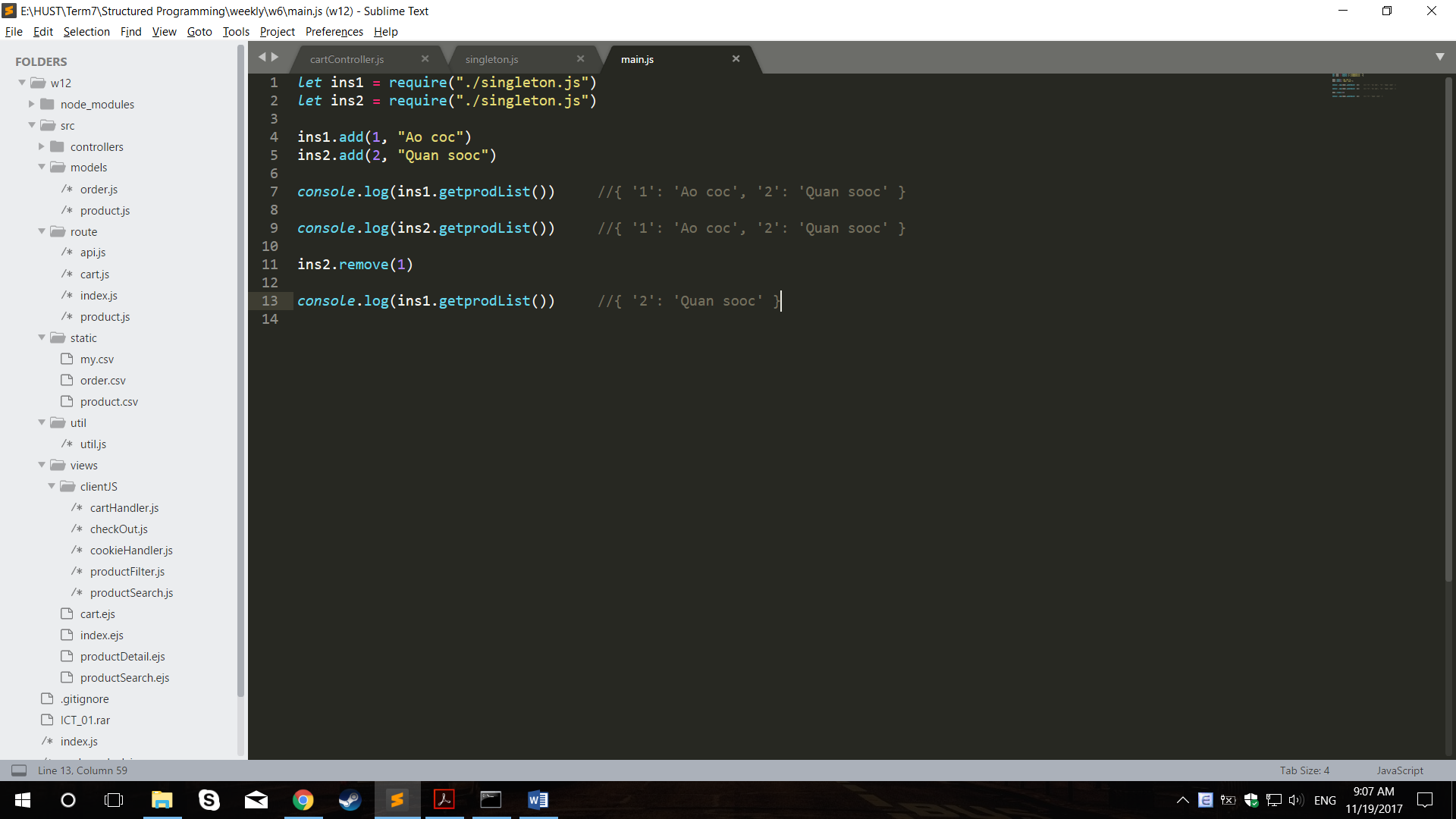
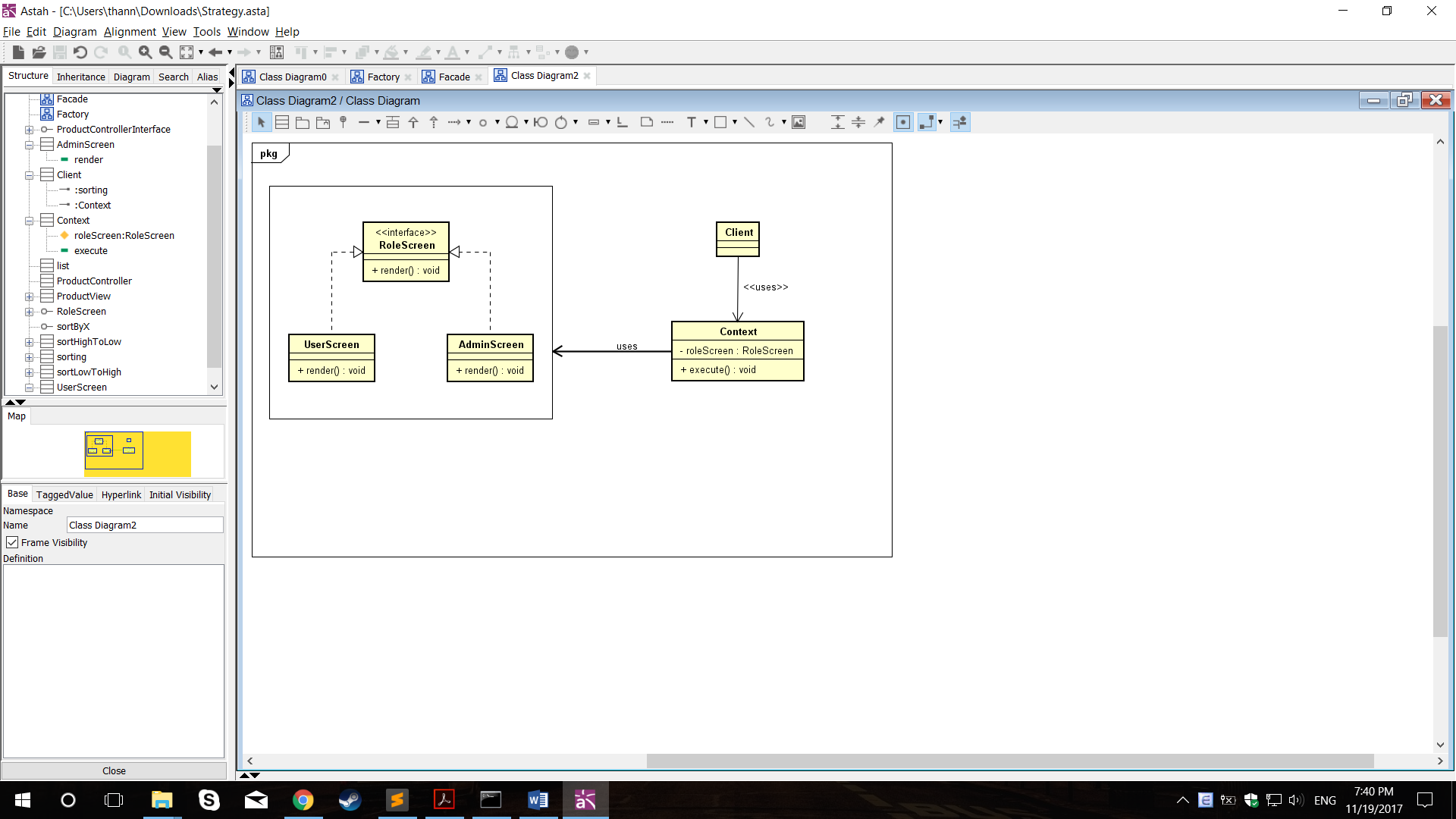
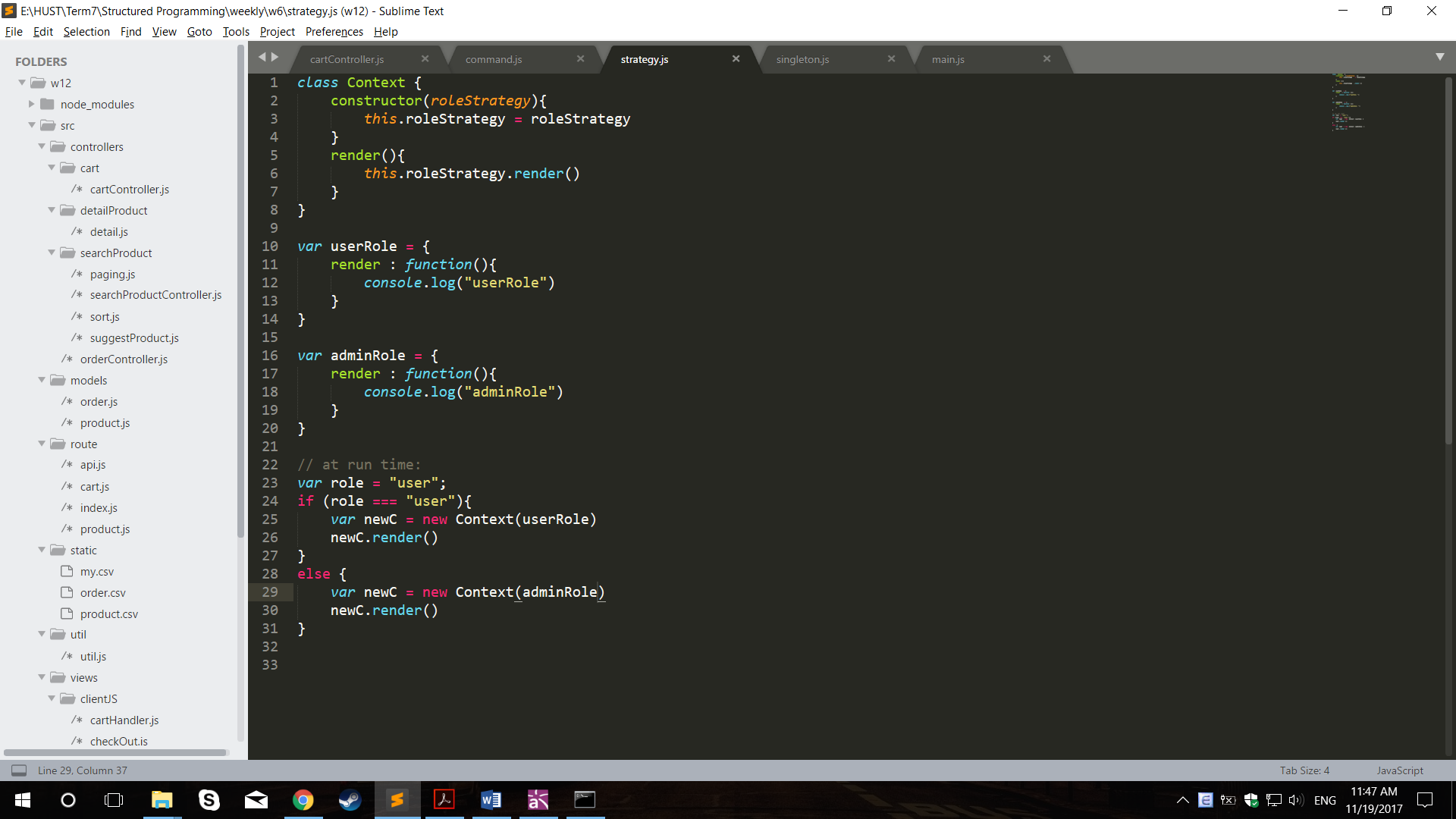
1. **Singleton Pattern:**
   1. Singleton can be applied to cart handler:
      * + Restrict instantiation of a cart class to a single object.
   2. Illustrate by UML:

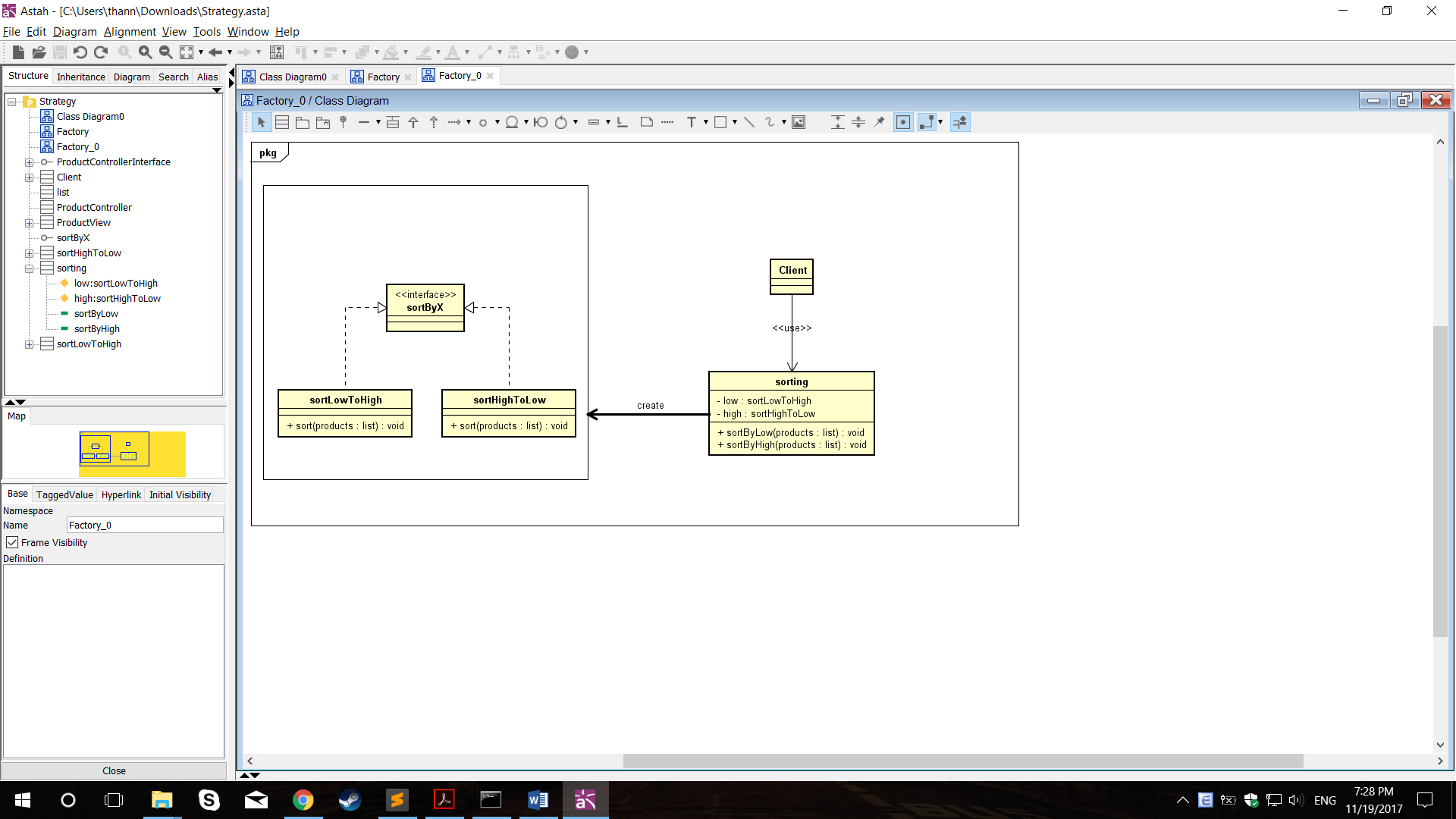


* 1. Justify by code:
     + - Class Singleton
       - Ins1 and Ins2 share same object singleton

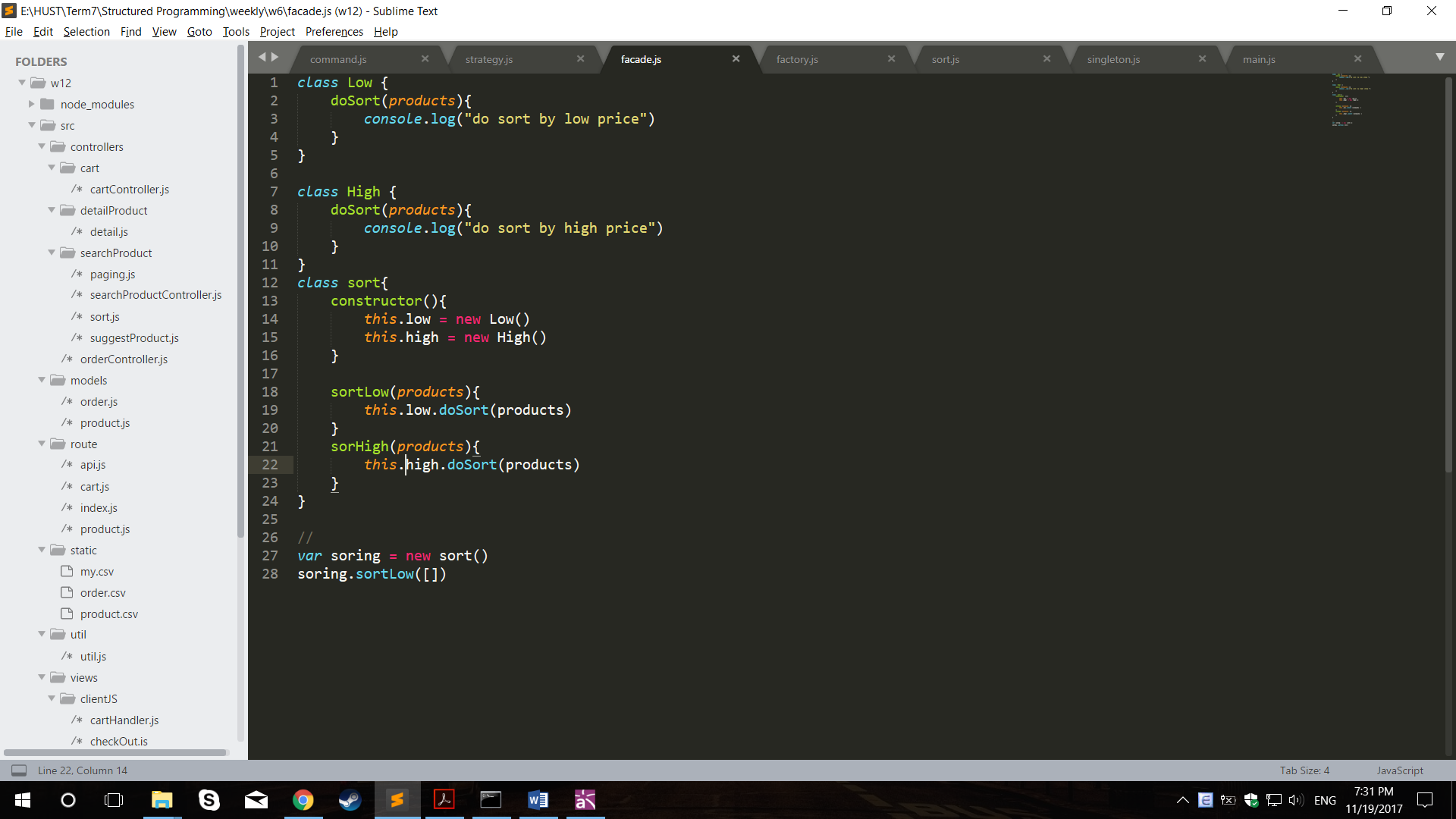
1. **Strategy Pattern**
   1. Applied to view:
      * + Behavior changes at run time.
   2. Illustrate by UML: 
   3. Justified by code:
      * + Depend on user’s role, change the controller’s behavior to render corresponding page:
        + Default behavior can be achieved by adding default roleStrategy

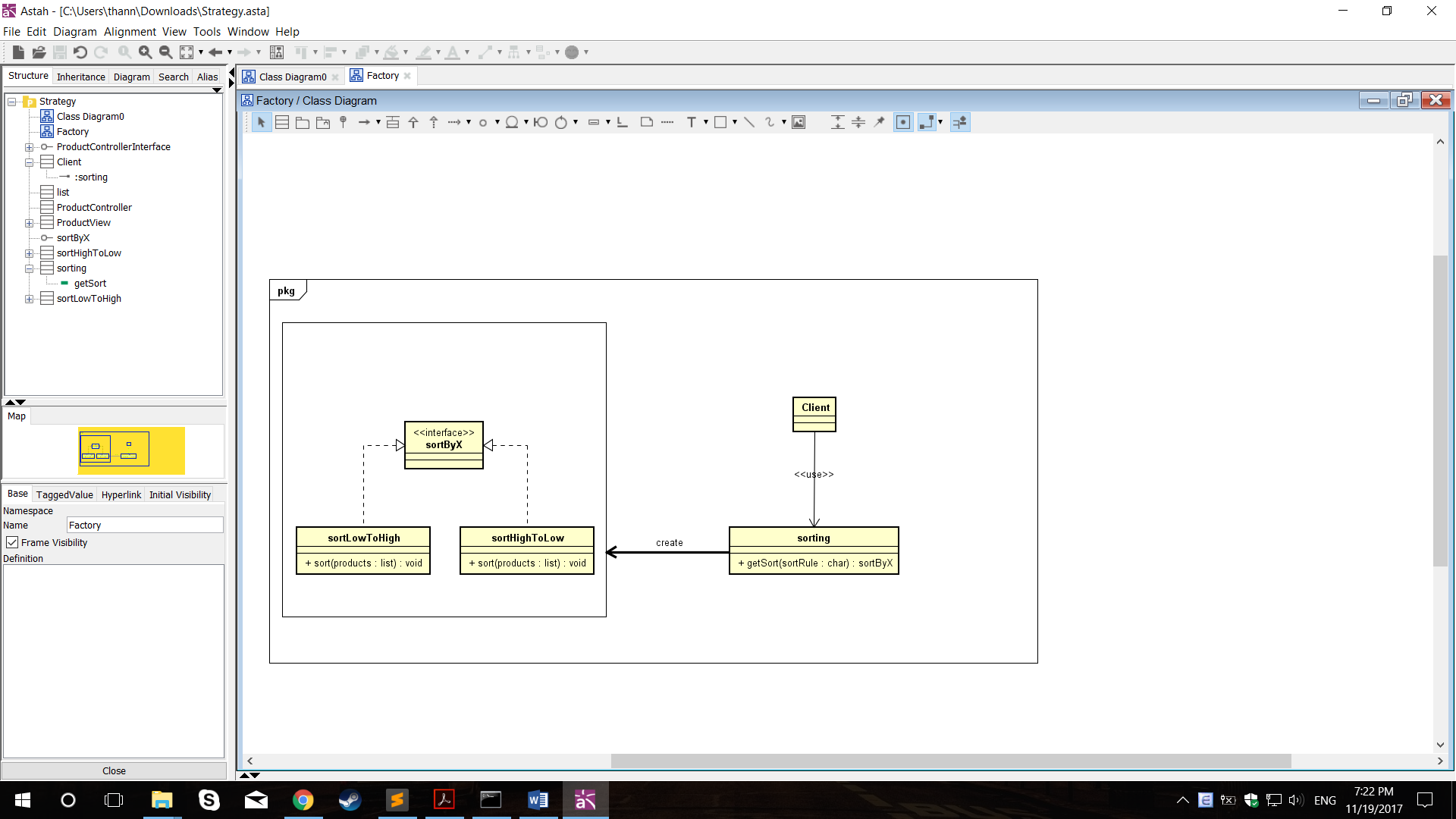


1. **Facade Pattern**
   1. Applied to functionality like sort:
      * + Hiding the complexities of system and provide an interface to the client.
   2. Illustrate by UML:



* 1. Justify by code:
     + - Low and high implement interface with doSort()



1. **Factory Pattern**
   1. Applied to function like sort:
      * + Create object without exposing the creation logic to the client and refer to newly created object using a common interface.
   2. Illustrate by UML: 
   3. Justify by code:
      * + Class sort used as to get a sorting method object needed