Learning from strategy design pattern

**Brief explanation:**



(Ref: <http://upload.wikimedia.org/wikipedia/commons/3/39/Strategy_Pattern_in_UML.png>)

**Strategy pattern is a behavioral design pattern.**

* Strategy design pattern is typically used in places where you want to change behavior of algorithm at run-time.
* For example—
  + Suppose you have various compression algorithm such as Huffman, LZW etc. and you would like to use these algorithms based on some situations, choosing STRATEGY will be a good choice
  + In resource-allocation system, various strategies such as minimizing fragmentation, disk-access etc. are required based on what user want to optimize.
  + In web-services, you may ask data in json/xml/text format. Typically the back-end will have ‘**serialize ()’** method and based on format asked, it will convert to required format using strategy pattern.
  + Suppose you have many authentication schemes such as RSA-encryption (Custom), Facebook authentication or Google authentication etc. you may choose any of the mentioned strategies in your application using strategy design pattern. [Passport.js](http://passportjs.org/) uses strategy pattern to encapsulate various kind of authentication providers.
  + Data-packet sending protocol—You may want to use TCP strategy for reliable packet sending whereas UDP for functionalities such as downloading etc.
  + Java’s Collection.sort() uses comparators is an example of strategy design pattern
  + *Try finding more examples yourself*
* **How it is related to open-closed principle?** 
  + Open-closed principle states that a class should be open for extension but closed for modification. Just think how you would entertain change in strategy without following this pattern and using inheritance. Consider example of compression technique, you create various classes that extends basic encryption class. If any change is required to be made in the class, you will made modification to both this class and where it is used.   
    *Strategy pattern uses composition instead of inheritance which in-turn allows for between the behavior and class that uses this behavior. Behavior can be changed both at compile time and at run-time.*
* How it is different from state design pattern?
  + Covered in state design pattern doc. //TODO
* Look for the code covered in example folder to understand from implementation point of view.