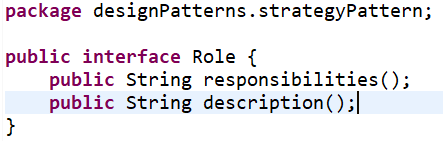
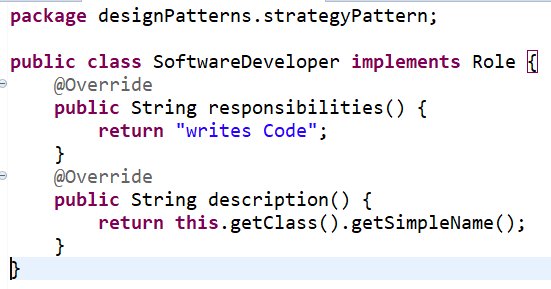
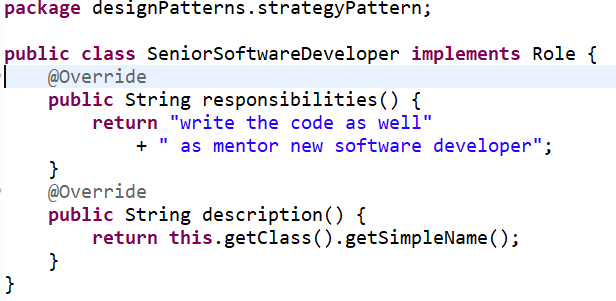
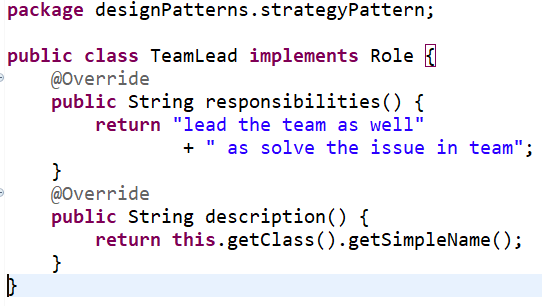
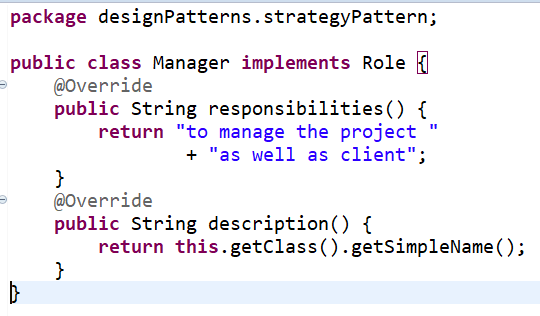
Strategy pattern is used when we have multiple algorithms for a specific task, and the client decides the actual implementation be used at runtime. A strategy pattern is also known as a policy pattern. We define multiple algorithms and let client applications pass the algorithm to be used as a parameter.

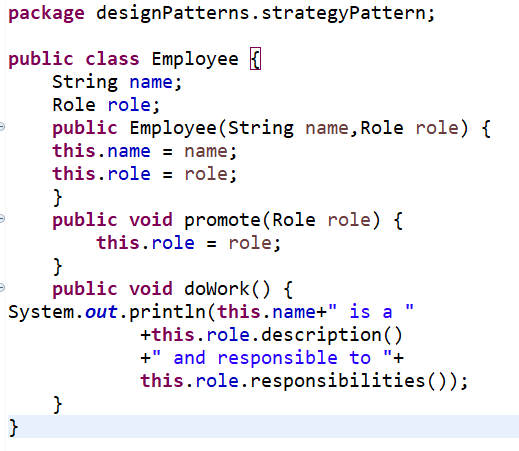


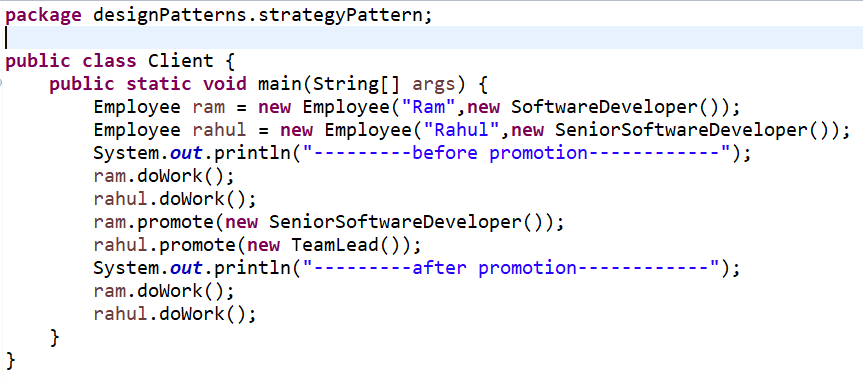












* **Use the Strategy pattern when you want to use different variants of an algorithm within an object and be able to switch from one algorithm to another during runtime.**
* The Strategy pattern lets you indirectly alter the object’s behavior at runtime by associating it with different sub-objects which can perform specific sub-tasks in different ways.
* **Use the Strategy when you have a lot of similar classes that only differ in the way they execute some behavior.**
* The Strategy pattern lets you extract the varying behavior into a separate class hierarchy and combine the original classes into one, thereby reducing duplicate code.
* **Use the pattern to isolate the business logic of a class from the implementation details of algorithms that may not be as important in the context of that logic.**
* The Strategy pattern lets you isolate the code, internal data, and dependencies of various algorithms from the rest of the code. Various clients get a simple interface to execute the algorithms and switch them at runtime.
* **Use the pattern when your class has a massive conditional operator that switches between different variants of the same algorithm.**
* The Strategy pattern lets you do away with such a conditional by extracting all algorithms into separate classes, all of which implement the same interface. The original object delegates execution to one of these objects, instead of implementing all variants of the algorithm.