**Command Design Pattern**

Command design pattern is used to encapsulate the request as an object.

**Intent**

* Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

The intent is to encapsulate the request as an object, parameterize clients with different requests, provide a queue, logging of requests and supporting the undo operation.

**Problem**

* It is required to issue the requests to objects. The requested operation and the receiver is not known.

There are scenarios where the requested operation and receiver is not known and only receiver knows what operation has to be done.

**Solution**

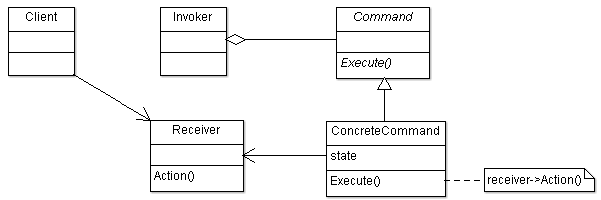
* Come up with an object for requested operation, so that it can be passed as any other object.

So the request can be encapsulated as an object, will be invoked and the object will have receiver information which will carry out the operation.

**Where it is applicable?**

* The actions have to be performed but receiver and operation are not known.
* Undo operation is required to be supported.
* Logging of actions are required to recover the system from crash using logs.
* There are different types of actions that have to be invoked in a similar manner.

**Structure**



**Participant classes**

* **Command** class provides interface Execute() to invoker.
* **ConcreteCommand** class implements the method Execute() which uses the receiver to carry out the operation. It has instance of class Receiver.
* **Invoker** class has the instance of class Command and invokes the command interface to carry out the operation.
* **Receiver** class has methods to do the operation for a request.

**How they work together?**

* Client creates the object of class ConcreteCommand and sets its receiver. Invoker has the instance of this object and it invokes the request using the interface Execute(). The Execute() method of ConcreteObject uses the receiver to do the operation.
* The command pattern is decoupling the invoker and receiver. This provides flexibility to add new commands and modify the existing ones.
* There can be easy facility of Undo operation with added operation Unexecute or Undo in Command object. This may require storing of states and having the history list for execution of commands, so redo operation can be provided as well.
* There can be MacroCommand which manages the sequence of commands. This will have list of commands and will execute them one by one with simple traversal.