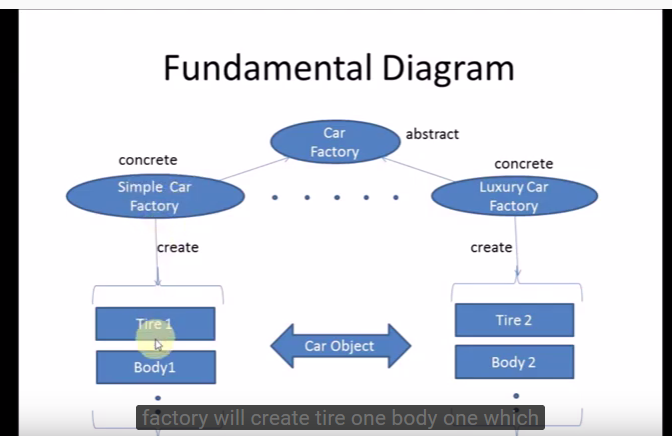
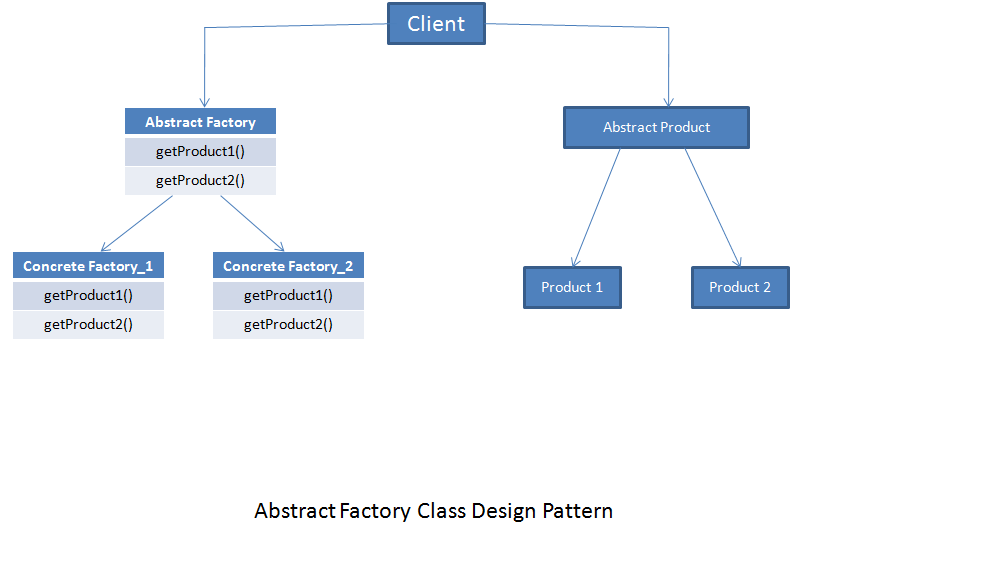
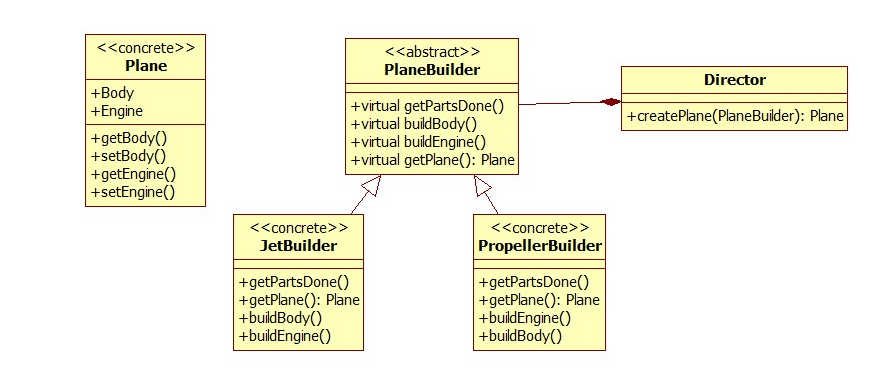
**Abstract Factory Design Pattern**

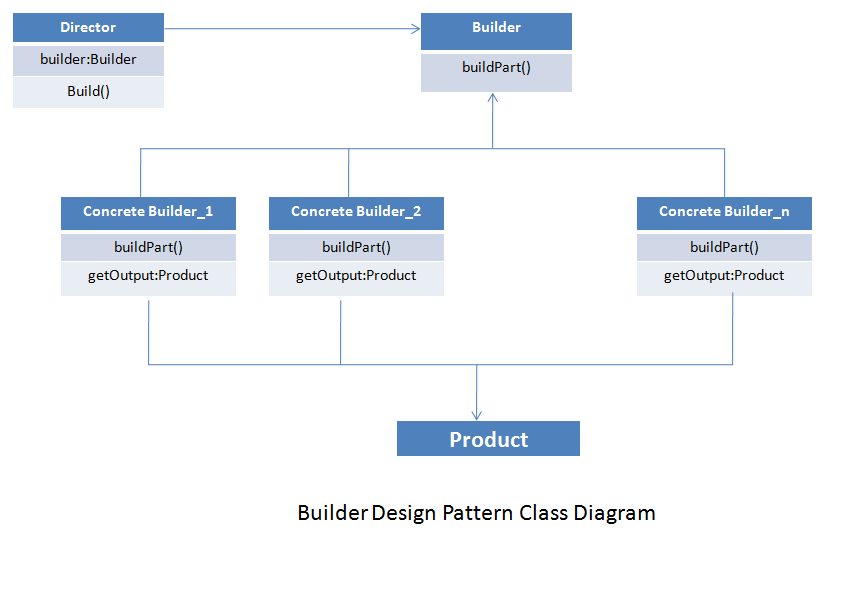
Define an abstract class creating families of related object but without specifying their concrete sub class

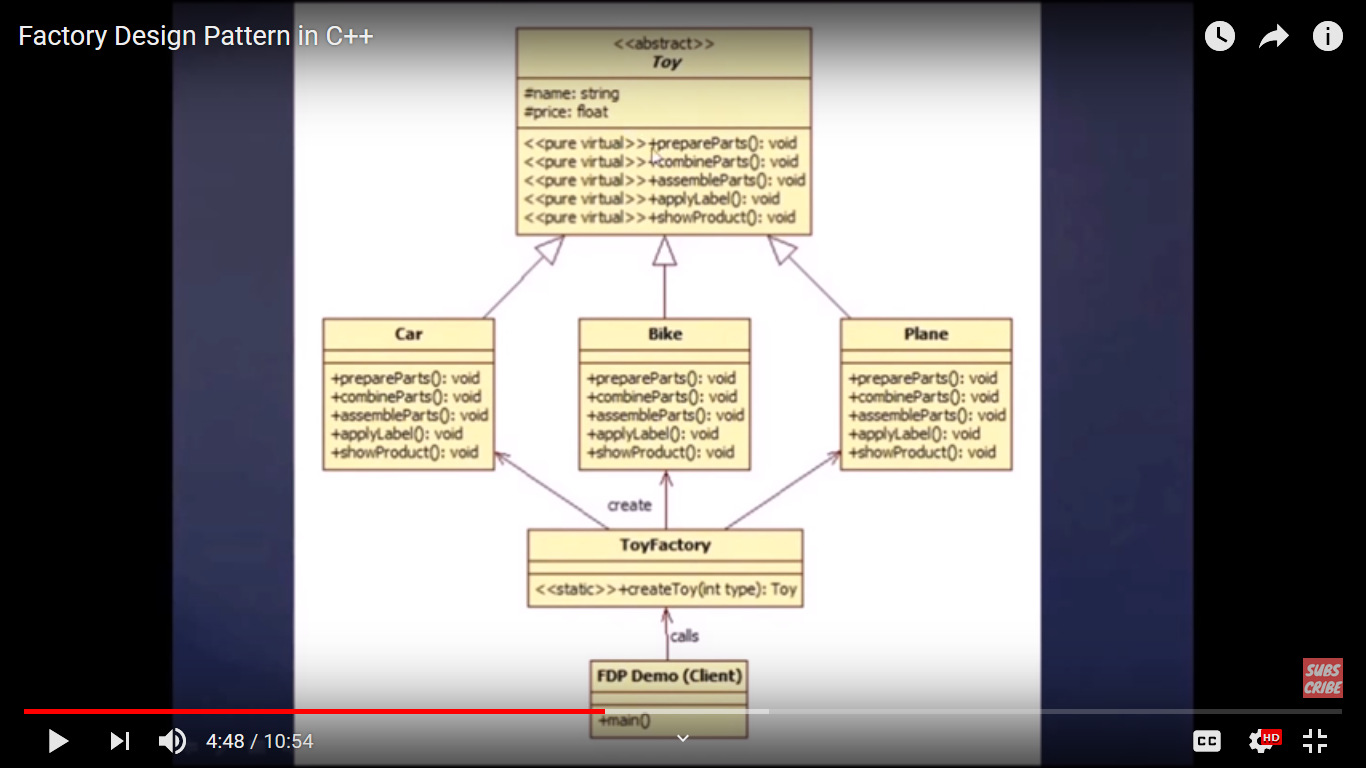
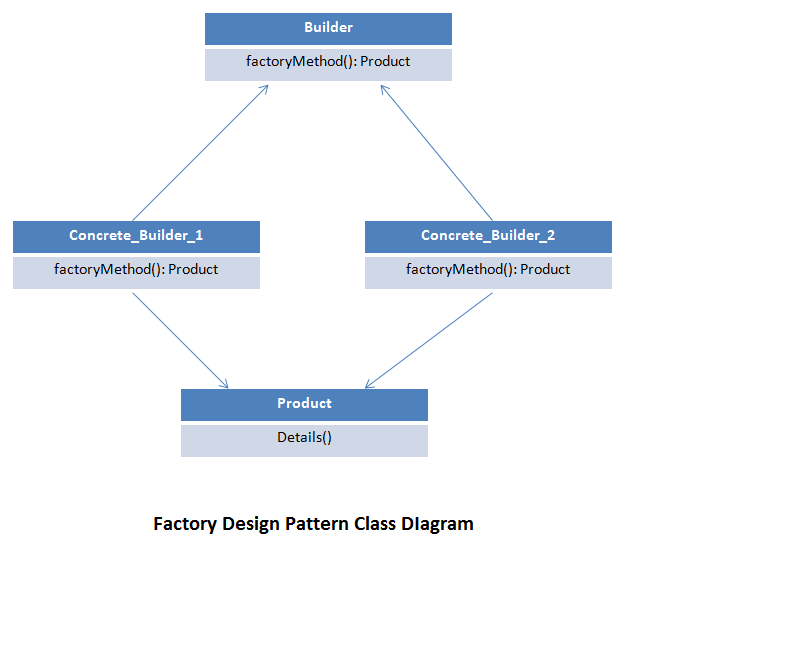




**Builder Design Pattern** – Creating Object with composing complex object and director will handle that







**Factory Design Pattern:**

It create object intialized object , It kind of virtual constructor

**Singleton Design Pattern**

It will create only one instance of object

- Private Constructor

- Static Member variable

- static member function

**Object Pool**

Object pooling can offer a significant performance boost; it is most effective in situations where the cost of initializing a class instance is high, the rate of instantiation of a class is high, and the number of instantiations in use at any one time is low.

class ObjectPool

{

private:

std::list<Resource\*> resources;

static ObjectPool\* instance;

ObjectPool() {}

public:

/\*\*

\* Static method for accessing class instance.

\* Part of Singleton design pattern.

\*

\* @return ObjectPool instance.

\*/

static ObjectPool\* getInstance()

{

if (instance == 0)

{

instance = new ObjectPool;

}

return instance;

}

/\*\*

\* Returns instance of Resource.

\*

\* New resource will be created if all the resources

\* were used at the time of the request.

\*

\* @return Resource instance.

\*/

Resource\* getResource()

{

if (resources.empty())

{

std::cout << "Creating new." << std::endl;

return new Resource;

}

else

{

std::cout << "Reusing existing." << std::endl;

Resource\* resource = resources.front();

resources.pop\_front();

return resource;

}

}

/\*\*

\* Return resource back to the pool.

\*

\* The resource must be initialized back to

\* the default settings before someone else

\* attempts to use it.

\*

\* @param object Resource instance.

\* @return void

\*/

void returnResource(Resource\* object)

{

object->reset();

resources.push\_back(object);

}

};

ObjectPool\* ObjectPool::instance = 0;

int main()

{

ObjectPool\* pool = ObjectPool::getInstance();

Resource\* one;

Resource\* two;

/\* Resources will be created. \*/

one = pool->getResource();

one->setValue(10);

std::cout << "one = " << one->getValue() << " [" << one << "]" << std::endl;

two = pool->getResource();

two->setValue(20);

std::cout << "two = " << two->getValue() << " [" << two << "]" << std::endl;

pool->returnResource(one);

pool->returnResource(two);

/\* Resources will be reused.

\* Notice that the value of both resources were reset back to zero.

\*/

one = pool->getResource();

std::cout << "one = " << one->getValue() << " [" << one << "]" << std::endl;

two = pool->getResource();

std::cout << "two = " << two->getValue() << " [" << two << "]" << std::endl;

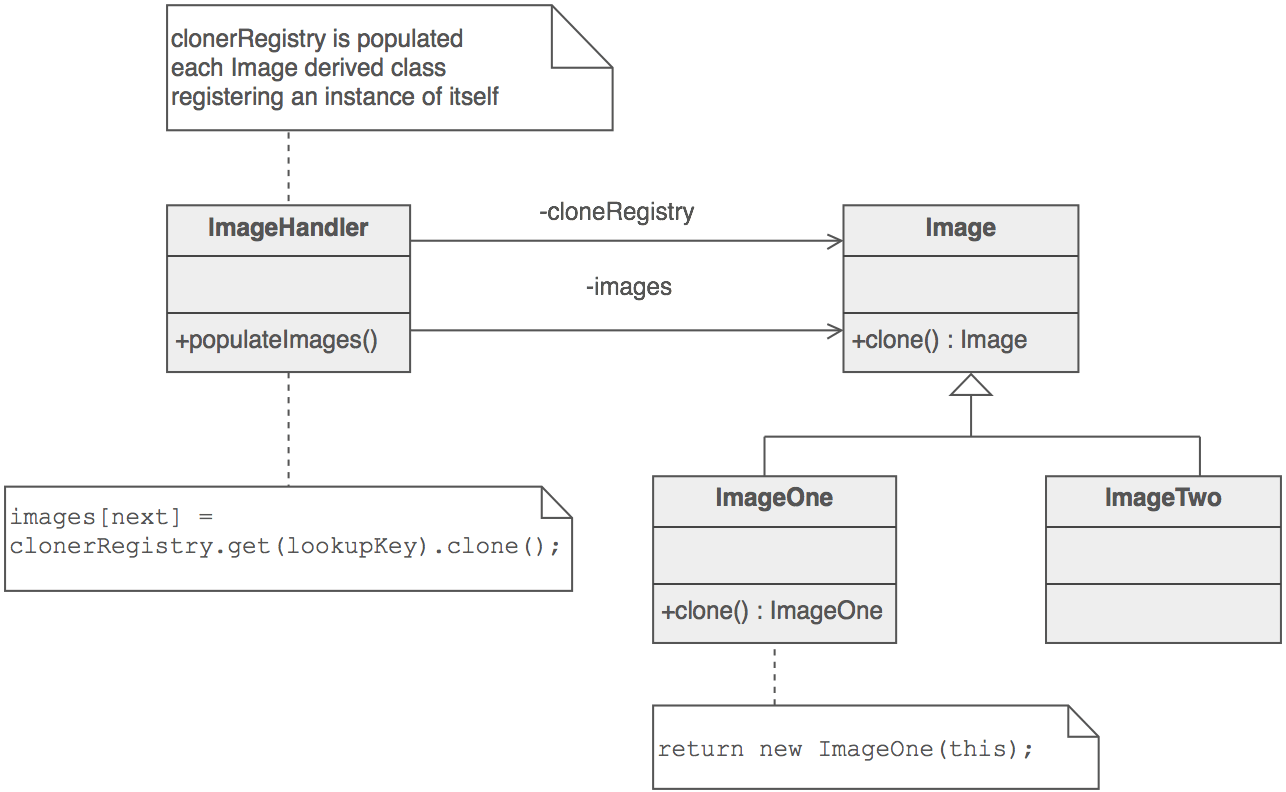
return 0;

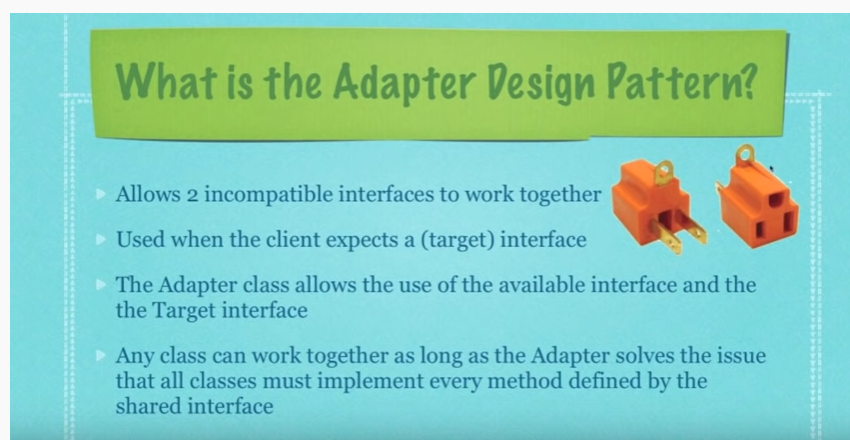
}

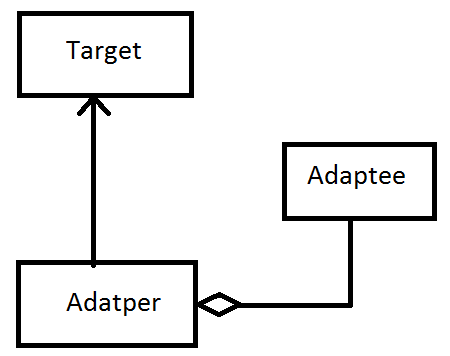
**Prototype Design Pattern**

Creating Object is expensive operation as compare to copying the object

return always cloned object

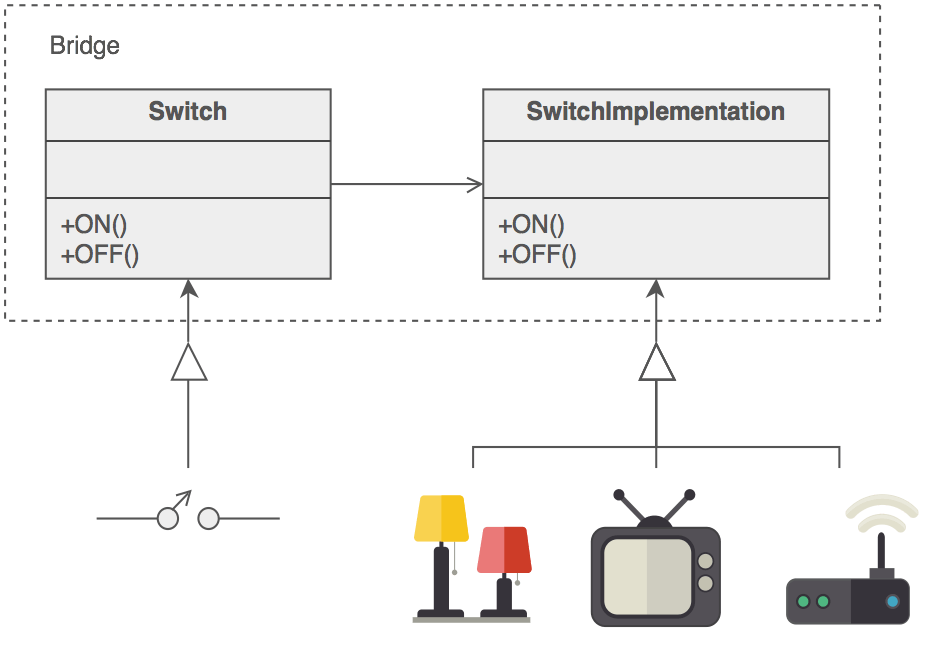


****

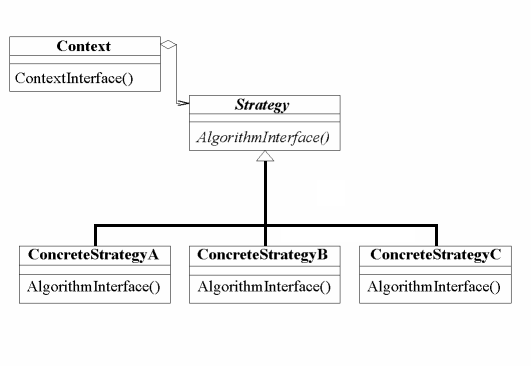
****

**adapterPattern.txt**

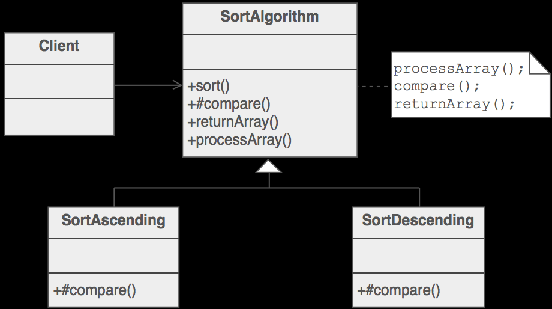
**Bridge Pattern (Structural Pattern)**

****

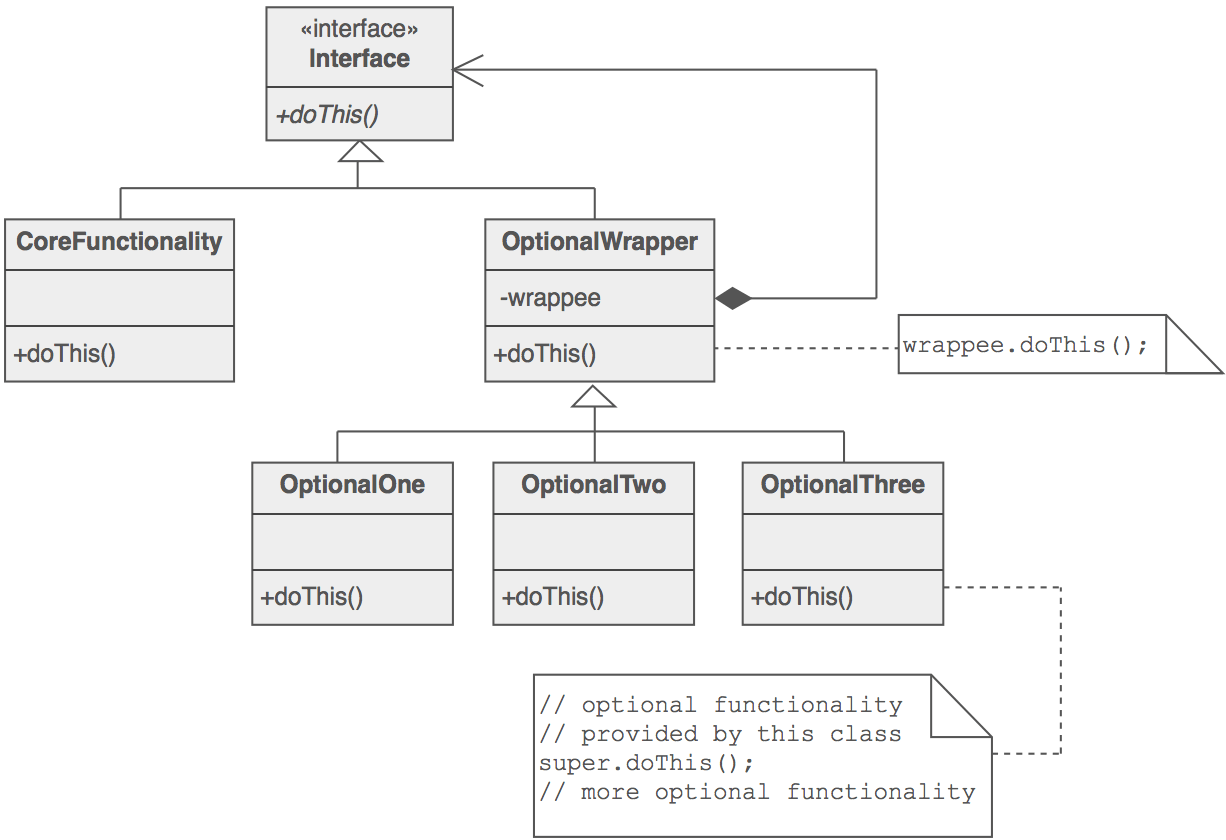
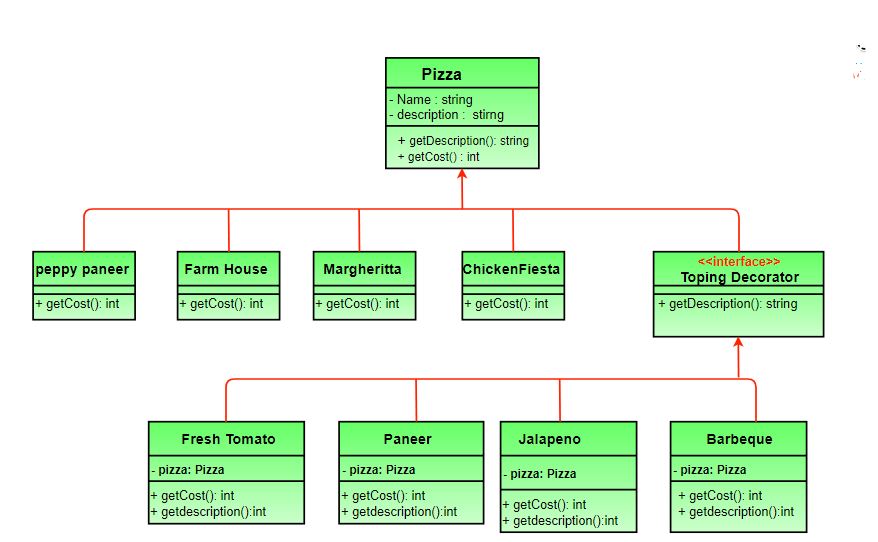
**Startegy Pattern (Behavioral Pattern )**

****

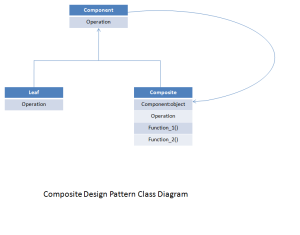
**Template Design Patter (Behavioral Pattern)**

****

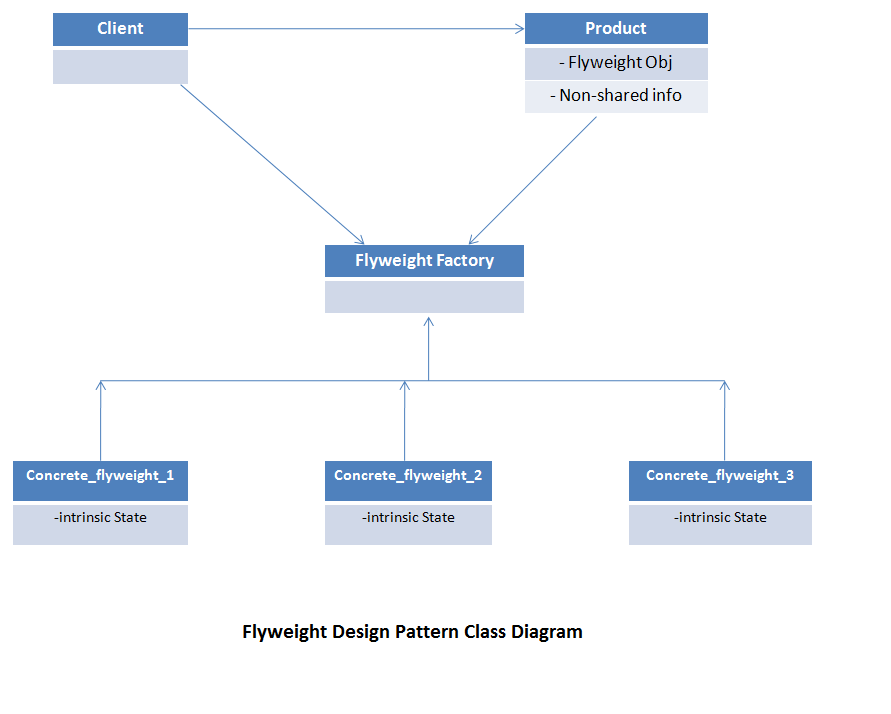
**Decorator Pattern (Structural Pattern)**

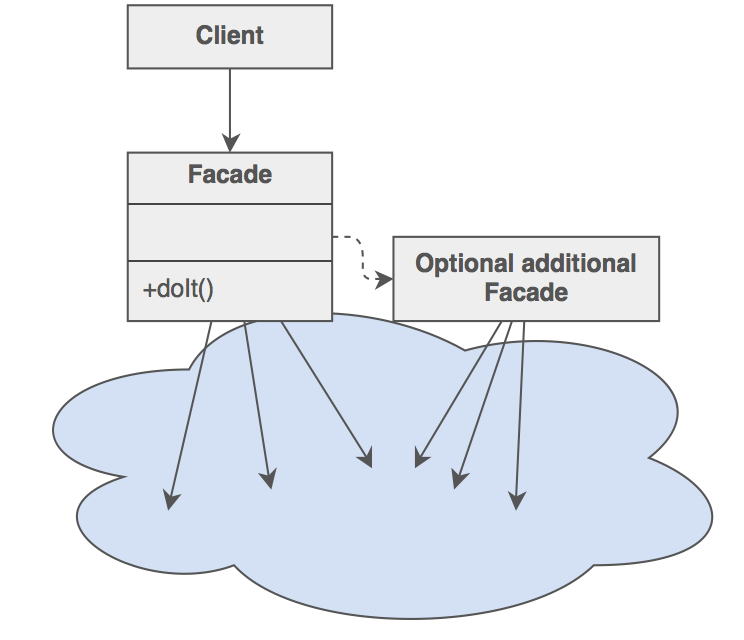
****

**Composite Pattern (SP)**

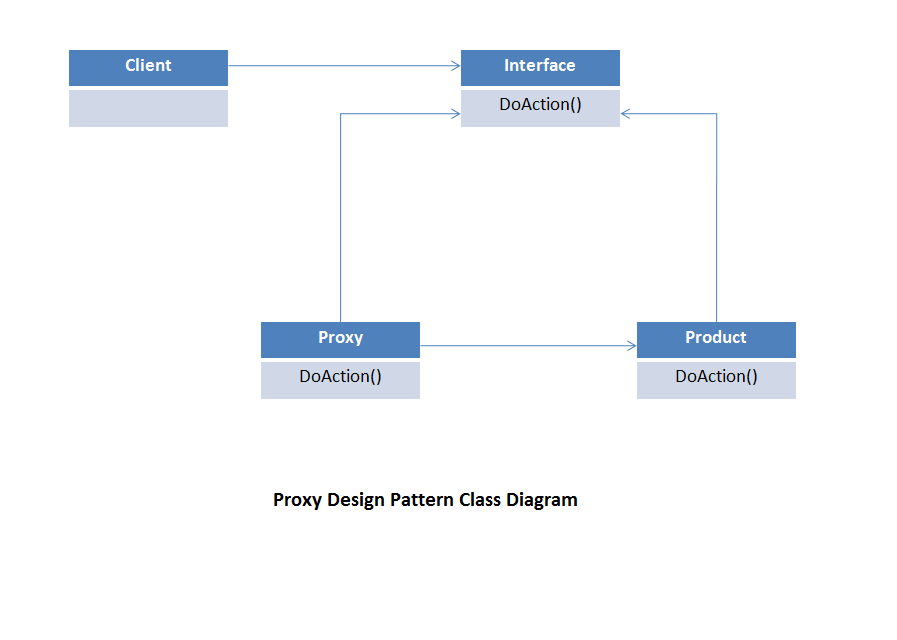
****

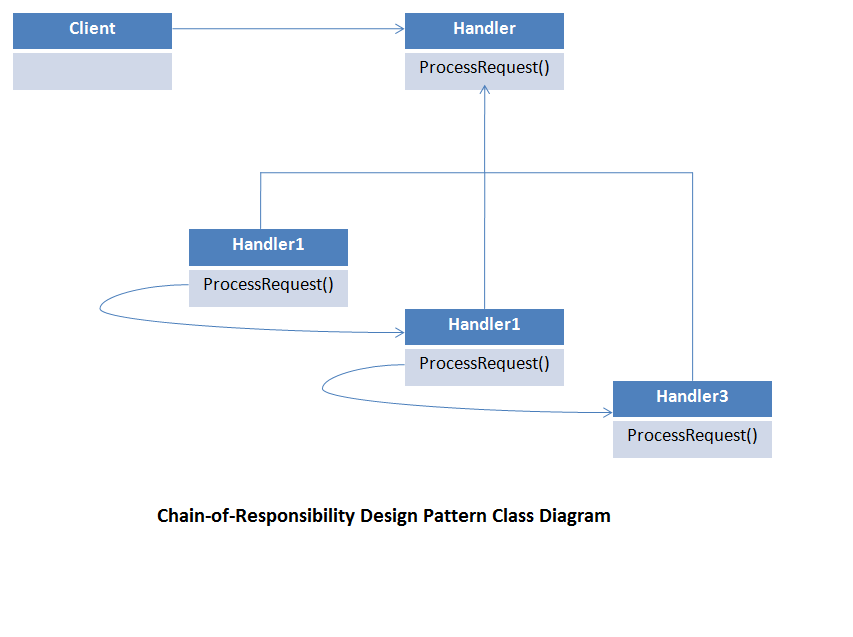
**Fly Weight Pattern (SP)**

**Facade Pattern (SP)**

****

**Proxy Pattern (SP)**

****

**Chain of Responsibility**