Оценка калорийности.

В этой задаче будет удобно использовать паттерн Composite.
Создадим базовый абстрактный класс Dish, в котором будет определяться виртуальная функция для подсчета количества калорий в продукте. От него будут наследоваться классы с единичными ингредиентами и составные блюда, состоящие из разных ингредиентов и блюд. В каждом ингредиенте переопределяется виртуальная функция подсчета калорий, возможно стоит добавить поля массы и каллоража на единицу массы для функционирования метода подсчета калорий. В составном блюде метод подсчета калорий переопределяется как сумма результатов вызова этого метода для всех составляющих. В класс составного блюда следует добавить поле с массивом ссылок на его составляющие и методы добавления состаляющих в блюдо.

Назначение стипендии.

Для этой задачи можно использовать паттерн Decorator.
Мы создаем абстрактный класс IScholarship, в котором будет интерфейс для подсчета стипендии. Далее создаем класс наследник - BasicScholarship. В этом классе будет метод расчета базовой стипендии. Создаем другой класс-наследник ScholarshipDecorator. В нем должно быть поле wrapped, хранящее адрес обертываемого объекта класса IScholarship и переопределенный виртуальный метод подсчета стипендии. Наследниками класса SchorarchipDecorato будут уже определенные декораторы для конкретных надбавок стипендии.

Генерация индексов.

Для реализации такого алгоритма удобно использовать паттерн Шаблонный метод.
Мы создаем класс базовый почтовый класс Index, в котором определяем метод genIndex(), в котором описываем генерацию международного кода, общего для всех стран, и к нему прибавляем код, получившийся в результате функции localCode(). Функцию localCode() делаем чисто виртуальной.
Классы индексов конктретных стран наследуются от Index и переопределяют метод localCode().

Стоимость доставки.

В этой задаче удобно применить паттерн Strategy.
Создадим абстрактный класс CountingDeliveryCost, в котором определим виртуальную функцию getCost(). Далее создадим классы конкретных стратегий для подсчета стоимости доставки по уде точному алгоритму для определенного вида транспорта, наследуя класс CountingDeliveryCost.
Далее создаем класс DeliveryCost с приватным полем, в котором хранится указатель на объект класса CountingDeliveryCost. В этом классе определяем метод getCost(), возвращающий результат выполнения метода getCost() для объекта класса CountingDeliveryCost.

Огневая поддержка.

При решении этой задачи следует применять паттерн Observer.
Создаем абстрактный класс Observer, в котором определяем метод update(), который выполняет обновление информации у наблюдателей. Создаем классы наблюдателей, тактических артиллерийских групп и других войсковых подразделений на участке, наследуем их от Observer и переопределяем в метод update().
Далее создаем класс Subject, наследуемый от Observer. В Subject создаем поле с массивом наблюдателей и поля с его значениями (типа местоположения техники). Переопределяем метод update(), чтобы при вызове этого метода вызывался метод update() для каждого из наблюдателей в массиве. Этот метод вызывается при изменении какого-то параметра в Subject.