В нашем проекте уже 5 классов Timer, Task, Mgr, Client. С их помощью можно создавать таски, запускать их, останавливать. Однако, не хватает хранилища существующих задач. Наша задача на сегодня - написать такое хранилище.

Class Storage

```
class Storage {
public:
   virtual  ~Storage() = default;
   virtual bool create_t(Task *task) = 0;
   virtual void update_t(Task *task) = 0;
   virtual bool delete_t(Task *task) = 0;
   virtual Task* get_t(string name) = 0;
   virtual vector<Task*> get_all_tasks_t() = 0;
};
```

Обратите внимание, все методы и поля класса равны 0 и начинаются со служебного слова virtual. Такой класс, без методов, не имеющий данных и состоящий из виртуальных функций называется интерфейсным классом. Этот подход позволяет полностью отделить реализацию от интерфейса — клиент (в нашем случае это Mgr) использует интерфейсный класс, — а в другом месте создается производный класс, в котором переопределяются чисто виртуальные функции и определяется функция, которая и будет этот класс создавать(функция-фабрика).

Наш проект поддерживает 2 возможных хранилища - map и базу данных. Обращаться к ним мы собираемся одинаково, и сначала хотим реализовать map хранилище. Значит сначал реализуем абстрактый класс с перечислением всех необходимых функций, затем определим функцию (функция-фабрика) которая будет звать конструктор map storage, а потом реализуем все виртуальные функции для map storage. Эта конструкция работает благодаря наследованию.
`class Storage (`абстрактный класс - есть только определение методов) ,
`class StorageMap: public Storage` - наследник с переопределенными методами класса Storage. И последнее - функция фабрика, которая зовет нужный конструктор - в нашем случае конструктор `StorageMap()`.

```
class Storage {
public:
   virtual  ~Storage() = default;
   virtual bool create_t(Task *task) = 0; создает таску в хранилище
   virtual void update_t(Task *task) = 0; обновляет данные для заданной таски
   virtual bool delete_t(Task *task) = 0; удаляет таску из хранилища
   virtual Task* get_t(string name) = 0; - получение таски по имени
   virtual vector<Task*> get_all_tasks_t() = 0; - возвращает вектор всех тасок
};
```

Обращаю ваше внимание на то, что всегда нужно проверять - нашлась ли таска с заданным именем в хранилище.

Первое хранилище `class StorageMap:`

```
class StorageMap: public Storage {
   map<string, Task*> storage;  в качестве хранилища используем map
public:
   StorageMap();
   bool create_t(Task *task); - описание смотрите выше
   void update_t(Task *task);
   bool delete_t(Task *task);
   Task* get_t(string name);
   vector<Task*> get_all_tasks_t();
   ~StorageMap();
};
```

Вызов нужного конструктора:
Есть абстрактный класс

```
class Creator{
public:
   virtual ~Creator() {};
   virtual Storage *FactoryMethod() const = 0; - вызов нужного конструктора
};
```

Наследуемся от него и вписываем вызов нужного конструктора.

```
class StorageMapCreator : public Creator {
public:
   Storage* FactoryMethod() const override {
       return new StorageMap();
   }
};
```

Таким образом:

```
int main(int argc, char *argv[]) {
   /*Create storage according to flags*/
   /*Storage*/
   Storage *storage = new StorageMap(); - создали нужное хранилище.
}
```

Теперь добавляем хранилище в Mgr:

```
class Mgr {
   Task* curr = NULL;
   Mgr();
   Storage* storage;
};
```

https://habr.com/ru/post/445948/
http://cpp-reference.ru/patterns/creational-patterns/factory-method/
https://refactoring.guru/ru/design-patterns/factory-method/cpp/example

Вдогонку про хэширование и хэш функции
https://www.youtube.com/watch?v=qD9t9ML4XnY&ab_channel=%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D1%81%D1%82%D1%80%D0%B8%D0%BCMail.RuGroup

https://habr.com/ru/post/509220/ ← читать!
https://habr.com/ru/post/112069/
https://yourbasic.org/algorithms/hash-tables-explained/
https://www.geeksforgeeks.org/hashing-set-3-open-addressing/