1. **Есть класс CodeGenerator, который умеет генерить код на разных языках.**

**Предложите рефакторинг с учетом, что количество языков будет расширяться**

class CodeGenerator {

public:

enum Lang {JAVA, C\_PLUS\_PLUS, PHP};

CodeGenerator(Lang language) { \_

Language = language;

}

std::string generateCode(){

switch(\_language){

case JAVA: //return generated java code

case C\_PLUS\_PLUS: //return generated C++ code

case PHP: //return generated PHP code

}

throw new std::logic\_error("Bad language");

}

std::string someCodeRelatedThing(){ // used in generateCode()

switch(\_language) {

case JAVA: //return generated java-related stuff

case C\_PLUS\_PLUS: //return generated C++-related stuff

case PHP: //return generated PHP-related stuff

}

throw new std::logic\_error("Bad language");

}

private:

Lang \_language;

}

Проблема данного кода в том, что он трудно поддается расширению и модификации, например при добавлении нового языка потребуется дополнять switch-case, добавлять название языка в перечисление, трудно будет добавлять новый функционал.

Хочется иметь структуру данных, которая будет позволять добавлять язык, проверять его наличие и передавать заодно функцию генерации кода с произвольной сигнатурой.

Для более быстрой скорости доступа будем использовать std::unordered\_map<std::string, std::function<void()>> table\_;, где ключом будет являться строка с названием языка, а значением обобщённый контейнер function для любого вызываемого без аргументов и без возвращаемого значения.

Добавим метод для добавления языка:

bool add(const std::string& lang, std::function<void()> action){

if (!action){

throw std::invalid\_argument("Empty action for language: " + lang);

}

auto [it, inserted] = table\_.try\_emplace(lang, std::move(action)); // pair

return inserted;

}

Если действия нет, передали nullptr или что то еще, то выбросим invalid\_argument.

Потом через emplace пытаемся вставить элемент если ключ отсутствует. Создается std::pair с итератором и флагом.

Добавим метод для вызова функции генерации:

void call(const std::string& lang) const {

const auto it = table\_.find(lang);

if (it == table\_.end()) {

throw std::out\_of\_range("Language not found: " + lang);

}

if (!it->second) {

throw std::bad\_function\_call();

}

std::invoke(it->second);

}

проверяется есть ли язык в мапе, если его нет, то ошибка, если есть, то вызываем через

std::invoke(it->second); функцию генерации.  
  
Примеры добавления языков приведены с комментариями в самом коде.