Департамент образования города Москвы
Государственное бюджетное общеобразовательное учреждение города Москвы
«Гимназия №1505»»
«Московская городская педагогическая гимназия-лаборатория»»

РЕФЕРАТ
"Сравнение объектного и процедурного стилей програмирования"

Выполнил:
Никулин Иван Андреевич
Руководитель:
Ветюков Дмитрий Алексеевич
_____ (подпись руководителя)
Рецензент:
ФИО рецензента
_____ (подпись рецензента)

Москва
2017/2018 уч.г.

Оглавление

1 Принципы и примеры. 1

1.1 Процедурное програмирование. . . . 1

1.1.1 Когда удобно использовать 1

1.2 Объектно ориентированное програмирование. . . . 2

1.2.1 Подробнее об объектах 2

1.2.2 Три слона ООП 2

1.2.3 Инкапсуляция 3

1.2.4 Полиморфизм 3

1.2.5 Наследование 3

1.2.6 Прмер 3

1.2.7 Черный ящик 3

2 Сравнение. 5

2.1 Читабельность. . . . 5

2.2 Логчность. . . . 5

2.3 Скорость написания. . . . 5

2.4 Легкость обучения. . . . 6

2.5 Популярность. . . . 6

2.6 Сведение баллов. . . . 6

3 Выведение алгоритма, заключение. 7

3.1 Скриптовые проекты. . . . 7

3.2 Полниценные, большие проекты. . . . 7

3.3 Командные проекты. . . . 8

3.4 Заключение. . . . 8

4 Приложения. 9

4.1 Интерфейс. . . . 9

4.2 Фирма. . . . 10

4.3 Черный ящик. . . . 13

4.4 Читабельност (ООП). . . . 13

4.5 Читабельност (процедурный стиль). . . . 15

Введение.

Человеку очень часто нужно совершать разнообразные вычисления. И если в древности было достаточно счёта на пальцах, то сейчас потребности в вычислительных мощностях стали гораздо больше. Мы сталкиваемся с задачами от подсчёта чека в магазине до расчета моста. И для многих из этих задач просто мозга недостаточно, как следствие люди пытаются помочь себе в вычисления. Этому служат разнообразные вычислительные машины.

Первая автоматическая машина для вычисление была изобретена в 1623 году немцем Вильгельмом Шиккардом, он называл её считающие часы. Но эти часы мало походили на современные инструменты для вычислений. Приобретать более привычную нам схему работы вычислительные машины стали уже в XX веке. Тогда же появился и первая ЭВМ(электронная вычислительная машина). Это была ЭВМ разработанная и построенная в 1941 году гарвардским математиком Говардом Эйксоном при сотрудничестве четырёх инженеров компании IBM. Эту машину современный человек уже мог бы назвать компьютером.

На данный момент компьютеры получили колоссальное развитие, но их всех объединяет одно: они не понимают человеческой речи. Для объяснения компьютеру его задание было придумано огромное количество разнообразных языков программирования: от assembler до java и python. Но ведь код должен быть понятен не только компьютеру, но и человек, а еще желательно и быть удобным в написание. Поэтому программисты постоянно экспериментировали и экспериментируют со стилями написания кода.

На данный момент выделились два основных стиля: процедурный и объектно ориентированный(ооп). И не всегда понятно какой из них выбрать, пора бы разобраться. В своем реферате я хочу найти информацию о них и сравнить эти два стиля. В качестве метрики я взял следующие параметры:

1. Читабельность - простота понимания кода при чтение.
2. Логичность - логичность построения программы.
3. Скорость написания (одного и того же алгоритма).
4. Легкость обучения.
5. Популярность.

Популярность я включил в метрику т.к. если стиль(язык) популярный, то по нему больше вспомогательных материалов, как следствие его проще использовать.

Как конечный результат я хочу если не узнать лучший из этих двух стилей, то выявить какой-то алгоритм выбора.

Глава 1

Принципы и примеры.

В этой главе опишу логику этих принцыпов, а так же приведу примеры программ реализующих разнообразные алгоритмы в этих стилях. Все примеры будут написаны на языке програмирования JAVA. Не смотря на то что это объектно ориентированный язык, я использую именно его т.к. писать в том или ином стиле можно на любом языке, вопрос лишь в логике построения программы

1.1 Процедурное програмирование.

Любой из пока-что существующтх стилей програмирования требует деление кода на некие логические блоки. В случае процедурного програмирования они следующие:

1. Основное тело программы.
2. Процедура и функция.

Основное тело программы. Основное тело программы - это кусок кода вызывающий процедры и функции в определенном порядке. Часто является циклом.

Процедуры и функции. Процедуры и функции - это части программ, выделенные в логический блок, для дальнейшего множественного вызова в основном теле программы . Их очень удобно использовать в тех случаях когда нужно часто повторять один и тот-же алгоритм. Функция отличается от процедуры тем, что она возвращает какое-то значание(результат своей работы).

1.1.1 Когда удобно использовать

Как я уже сказал в процедуры и функции выносятся часто повторяющиеся блоки программы. Это очень удобно использовать в задачах с однообразными, повторяющимеся алгоритиами. Например, такой стиль очень органично смотрится в программе из приложения 4.1.

Эта программа реализует простейший алгоритм:

1. Выведи приветствие.
2. Спроси имя пользователя.
3. Запусти цикл.
4. Вывведи инструкцию.
5. Сроси команду пользователя.
6. Ответ равен "exit"?

   Да.

   Спроси подтверждение выхода.

   Если ответ равен "yes"выйди из цикла

   Нет.

   Выведи случайное число.
7. Начни заново.

Для его реализации была использованна следующая структура кода:

1. main - основное тело программы
2. ask - функция получающая вопрос, выводящая его на экран и возврощающая ответ пользователя
3. sayHi - процедура выводящая приветствие программы
4. sayInstructions - аналог sayHi выводящий инструкцию по использованию программы
5. writeRandom - процедура выводящая случайное число от 1 до числы to (int to)

Как видно функция ask используется в main несколько раз - в этом и состоит главное удобство процедурного стиля(в простейшем случае).

1.2 Объектно ориентированное програмирование.

В объектно ориентированном програмирование(далее ООП) можно выделить несколько больше блоков:

1. Класс
2. Метод
3. Поле

Класс Класс - это самая большая логическая еденица в ООП. Он представляет собой некий шаблон, для создание объектов(о них я расскажу подробнее потом). Внутри себя он содержит методы и поля(а иногда и другие классы).

Метод Медот - это аналог процедур и функций в мире ООП. Они описываются для работы с объектами класса внутри которого они находятся.

Поле Поле - это ничто иное как простая переменная, только привязанная к конкретному объекту.

1.2.1 Подробнее об объектах

Если классы - это шаблоны, то объекты это их отпечатки. Возможно, это сравнение немного не понятно поэтому рассмотрим объекты на более реальном примере.

Представте себе компанию. Обычную компанию с директором и работниками. Но ведь работники - не просто безликие серые шаблоны. У работников есть своя должность, зарплата, имя и т.д. И если проводить аналогию с ООП, то в этой компании есть класс работник, и когда нового человека принемают в фирму, создается новый объект этого класса, а его полям(должность, имя, зарплата) присваиваются определенные значения. Теперь эти значения привязанны конкретному работнику-объекту. Но ведь директору надо как-то управлять своими подчиненными, для этого у класса работник определены специальные методы для постановки задачи работнику. И чтобы поставить работнику Васе задачу написать отчет, директор должен вызвать для него метод поставить задачу. Это будет выглядеть примерно так:

Вася.поставить_задачу(Напиши_отчет.)

Кстати, в этом примере мозг деректора можно сравнить с главным классом, в котором работает алгоритм постановки задач для каждого работник.

1.2.2 Три слона ООП

ООП в отличие от процедурного програмирование (1.1) дикларирует ни только логику оформления кода, но и принципы построения классов:

1. Инкапсуляция
2. Полиморфизм
3. Наследование

1.2.3 Инкапсуляция

Своими словами инкапсуляция - это запирание полей и методов внутри классов. Если демонстрировать инкапсуляцию в рамках примера 1.2.1, то инкапсулированные(спрятанные) методы в классе работник - это , на пример, метод постановки задачи на выходные, ведь эту задачу он определяет сам, и доступа к этому методу начальник иметь не должен, он будет объявлен приватным(недоступным для других классов).

1.2.4 Полиморфизм

Полиморфизм - это способность метода принемать в качестве аргумента переменные рзных типов, и работать с ними по разному. Попробуем добавить полиморфизм в наш пример 1.2.1. Допустим, у класса работник есть метод установить_зарплату, он принемает как аргумент - размер зарплаты в тысячах. Но ведь в зарплате может быть целое число тысяч, тогда метод будем принемать целую переменную типа integer, но ведь зарплата может быть и не целой, тогда он будет принемать дробное число double. Такой метод будет называться полиморфным.

1.2.5 Наследование

Наследование - это порадигма позволяющая получать в наследство от класса предка какие-то черты классу ребенку. В рамках примера 1.2.1 это бдет выглядеть так. У нас уже есть класс работник, но в фирме есть разные вакансии (инженер, системный администратор, бухалтер) можно конечно просто присваивать определенным полям класса работник, значения соответствующие его вакансии, но куда правильние наследовать от класса работник классы пинженр, бухалтер и т.д. При таком способе они получат все поля и методя класса работник, но к ним можно будет добавить и свои методы и поля. Фактически когда мы наследуем класс A от класса B мы говорим, что класс A похож на класс B, но имеет какие-то свои дополнения.

1.2.6 Прмер

Как пример можно привести программу из приложения 4.2. Этот код реализует структуру кода из примера 1.2.1 и дополнений к нему (1.2.3, 1.2.4, 1.2.5).

Рассмотрим структуру программы.

Класс ”firm”. Это класс который содержит в себе точку входа и простейший линейный алгоритм, который просто демонстрирует возможности данной программы.

Класс ”worker”. Этот класс релизует работника, точнее свойственные для него поля и методы. Особый интерес прдставляют методы "setSalary"и "setHolidayTask".

Как видно из кода метод "setSalary"объявляется два раза ("public void setSalary(int salary)"и "public void setSalary(double salary)"), единственное что их различает- это тип переменной-аргумента, в одном случае целое, в другом дробное. Это и есть проявление полиморфизма (1.2.4) в этой программе.

Метод "setHolidayTask"имеет префикс protecte, этот префкс реализует инкапсуляцию в JAVA. Методы объявленные с таким префиксом доступны только внутри этого класса, классов - наследников и пакета(папки) в которой находится класс, то есть из класса firm он недоступен.

Класс ”engineer” и ”seller”. Эти классы демонстрируют принцип наследования. Они унаследованны от класса ”worker”. И как мы видим получили от него некоторые методы, но так же добавили и свои, свойственные именно для них.

1.2.7 Черный ящик

Так в чем же принципиальное отличие ооп от процедурного програмирования, ведь по сути классы - это просто функции, способные сохронять свои промежуточные данные, какие преимущества дает ооп, взамен на сложную структуру? На самом деле ооп, дает возможность легко реализовать очень удобную парадигму, она называется "Принцип черного ящика."

На самом деле, черный ящик - очень простой, понятный и удобный принцип, сильно упрощающий работу програмиста. Пример черного ящика можно найти в приложении 4.3.

Алгоритм в этом примере очень простой, в классе BlackBox содержится некий код, написанный программистом №1, и не известный программисту №2, написавшему класс FrontEnd. Но не смотря на это программист №2 использует алгоритм из класса BlackBox, чрез метод getResult. То есть программист

№2 может просто попросить программиста №1 написать класс который делает то-то и то-то, а потом пользоваться им, не понимает внутренних процессов, получая готовый результат, а инкапсуляция (1.2.3) поможет ему ничего не испортит, "опасный"метод. Такой подход очень полезен в больших корпоротивных проектах, когда над программой работают сотни програмистов.

С таким подходом программа становится модульной. Поддерживать такой проект так же удобно, как обновлять компьютер, например, в компьютере утарела видеокарта, ее просто вытасквают, и ставят новую, но процессор, материнская плата и все остальные части сборки об этом даже не подозревают,ведь интерфейс работы с системой у новой карты совподает со старой.

Глава 2

Сравнение.

Начнем сравнение стилей по метрике описанной во введении:

1. Читабельность - простота понимания кода при чтение.
2. Логичность - логичность построения программы.
3. Скорость написания (одного и того же алгоритма).
4. Легкость обучения.
5. Популярность.

2.1 Читабельность.

В этой категории баллы будут выставляться по результатам опроса.В рамках этого опроса, я покажу опрашиваемым две программы(4.5, 4.4) в разных стилях, и попрашу объяснить их смысл.

Точнее определили алгоритм описанный процедурно	33.3%
Точнее определили алгоритм описанный объектно ориентированно	66.6%

Таблица 2.1: Результаты опроса

Опрос я выбрал в качестве метода оценивания т.к. читабельность - это весьма субъективный пункт оценки

2.2 Логчность.

Под логичностью я подразумеваю приближенность текстового описания алгоритма к его сути. Что я под этим подразумеваю, я покажу на примере.

В приложении 4.4 объекты класса Robot хранятся в одном массиве, а в той же программе оформленной в процедурном стиле (4.5) есть аж 3 массива, где хранятся x, y кординаты и назначения роботов. В первом случае хранения данных о роботах более приближено к реальности, т.е. мы прямом смысле храним самих роботов в массиве, а не просто не понятные кусочки данныз, как во втором случае. Именно поэтому балл снова достается объектному стилю.

2.3 Скорость написания.

Здесь определить победителя очень просто, без всяких рассуждений. На описание этого алгоритма в процедурном стиле у меня ушло примерно в 4 раза меньше времени, это заметно даже по по тому, что строк в этой программе гораздо меньше. В этом пункте очевидно побеждает процедурный стиль.

2.4 Легкость обучения.

После прочтения некоторых статей и форумов стало понятно, что новички, начинающие изучать програмирование с ООП, очень быстро путаются и возникают недопонятые моменты. Это в принципе логичтно т.к. ООП вклчает в себя гораздо больше логических блоков (1.2), новичку в их структуре разобраться гораздо сложнее. Из этого можно сделать вывод, что проще для изучении процедурный стиль, бал достается ему.

2.5 Популярность.

Популярная соцсеть для разработчиков "github"(*https://github.com*) регулярно подводит топ популярных языков програмирования. На рисунке 2.1 приведены первые десять позиций этого топа, и все языки веделены цветами по признаку объектно/процедурно ориентированности.

- зеленая рамка - объектно ориентированный язык
- желтая рамка - в языке ориентирован на смешенный стиль
- красная рамка - процедурно ориентированный язык

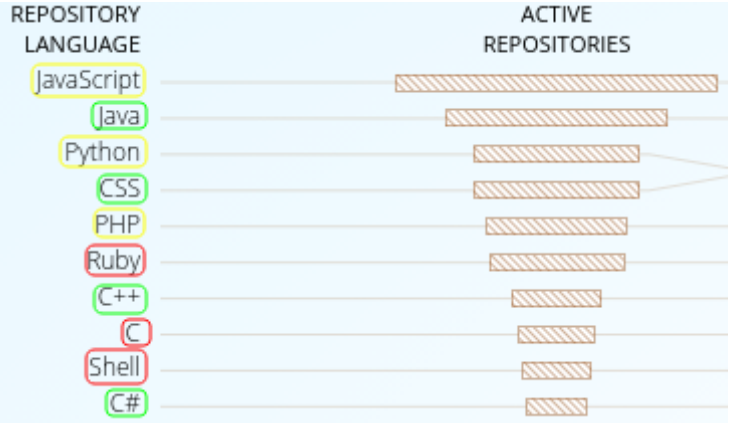

Рис. 2.1: Топ языков по популярности(github)

Не сложно заметить, что объектно ориентированных языков в топе гораздо больше, и их средняя позиция выше. Из этого можно сделать вывод, что на данный момент объектно ориентированный подход популярнее процедурного.

Github - это соцсеть основаная на технологии git. Git - создан для упращения разработки кода командой разработчиков, она позволяет коллективно изменят проект, и всегда будет понятно кто какие изменения внес(подробнее об этом можно почитать здесь: *https://ru.wikipedia.org/wiki/Git*). Из-за этого github имеет доступ к огромной базе проектов, и может подводить топ языков по популярности.

2.6 Сведение баллов.

—	Читабельность	Логичность	Скорость нап.	Легкость обуч.	Популярность
Объектныйй					
Процедурный					

В следующей главе на основе этой таблицы будут подведены итоги сравнения и выведен алгоритм выбора стиля написания кода.

Глава 3

Выведение алгоритма, заключение.

Что бы вывести алгоритм выбора стиля написания кода, сначала мы должны определить список возможных сценариев. Для этого я проанализирую все, когда либо написаные мной, программы и программы моих знакомых.

Вот такие основные типы проектов я выделил:

- Скриптовые проекты
- Полноценные, большие проекты
- Командные проекты

Как видите типов проектов не очен много, поэтому каждый из них я разберу в отдельном подпункте.

3.1 Скриптовые проекты.

Под скриптовыми проектами я подразумеваю какие-то небольшие программы, помещающиеся в один файл, созданные либо для облегчения труда програмисто, либо для каких-то исследований/эксперементов, либо промежуточных задач большой программы. Например есть программа оюрабатывающая некоторую информацию о пользователях социальной сети "вконтакте написанная на языке golang, но у этого языка не реализована удобная библиотека для работы с api этой социальной сети. Поэтому бял написан скрипт на языке "python скачивающий сведения о пользователях.

> API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface, API [эй-пи-ай][1]) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах. Используется программистами при написании всевозможных приложений.
> (*https://wikipedia.org*)

Такие программы (проекты) гораздо проще писать в процедурном стиле, так как разбирать этот код или пользоваться им с большой вероятностью будете только вы, а вот скорость написания в данном случае принципиальна. Поэтому если перед вами стоит задача написания небольшой вспомогательной программы, или вы собираетесь разработать и протестировать какой-то небольшой алгоритм, то процедурный стиль - это ваш выбор.

3.2 Полниценные, большие проекты.

Под полноценными проектами я подразумеваю проекты, которые представляют из себя законченый продукт (бухгалтерская программа, игра и т.п.). Такие проекты зачастую обладают достаточно сложной структурой, из этого можно сделать вывод, что в таких проектах отлично подойдет объектно ориентированный подход. Но т.к. в таких проектах алгоритмы могут быть не столь сложными, хотя и объемными, поэтому в некоторых случаях использовать процедурный подход для написания подобных програм так же может быть уместно. Так же если вы новичок, то так же используйте процедурный стиль, ООП вас только запутает.

Подитог: если вы новичок или ваш проект не содержит в себе очень сложных алгоритмов или большого количества абстракций, то используйте процедурный стиль. В остальных случаях вам идеально подойдет объектный стиль.

Уровень абстракции — один из способов сокрытия деталей реализации определенного набора функциональных возможностей. Применяется для управления сложностью проектируемой системы при декомпозиции, когда система представляется в виде иерархии уровней абстракции.
(*https://wikipedia.org*)

3.3 Командные проекты.

Эти проекты почти ничем не отличаются от выше упомянутых, но над ними работает целая команда програмистов, часто совершенно разных специализаций (разработчики серверного ПО, разработчики пользовательского интерфейса, специальсты по машинному обучению и т.д.). И из этого очевидно следует то, что объектно ориентированный стиль для проектов такого рода подойдет отлично.

Даже если алгоритм, реализуймый в проекте, вам не кажется достаточно сложным, что бы реализовывать его в объектном стиле, то используйте ООП. Ведь идеология черного ящика(1.2.7) очень поможет вам в командной работе. Например специалист по машинному обучению написал нейронную сеть, методы работы с которой он упаковал в отдельный класс, теперь второму разработчику достаточно создать объект принадлежащий этому классу и вызывать нужные методы, не задумываясь о принципах и спицифики работы самой нейронной сети. Так же за счет модификаторов доступа (инкапсуляции (1.2.3)) он не сможет ничего в ней сломать.

3.4 Заключение.

Главный вопрос всего реферата я поставил так: какой стиль лучше выбрать для конкретной ситуации. И вот ответ в краткой форме.

Если ваш проект разробатывается вами в одиночку, имеет достаточно простую структуру или вообще является эксперементом или промежуточным этапом разработки нового алгоритма, то процедурный стиль отлично вам подойдет. Если же структура вашего проекта обладает множеством абстракций, или его пишет несколько человек, то объектный подход сильно упростит разработку.

Глава 4

Приложения.

В этой главе приведены программы, написанные мной для упрощения понимания реферата.

4.1 Интерфейс.

Простая программа в процедурном стиле, которая общается с пользователем и выводит случайные числа

code/PP/ui.java

```
package TestField7;

import java.util.Random;
import java.util.Scanner;

public class ui {

    static Scanner reader = new Scanner(System.in);

    //enter point
    public static void main(String[] args) {
        //endless cycle
        while (true){
            sayHi();
            //ask user's name
            String name = ask("What's your name ");
            //flag to start new session
            Boolean cycle = true;
            while (cycle){
                sayInstructions();
                //wait to new command entering
                String answer = ask(name + ", enter your command: ");
                //handle the command
                switch (answer){
                    case "exit":{
                        String confirm = ask("Do you want to exit???(yes/no)");
                        cycle = !confirm.equals("yes");
                        break;
                    }
                    default:{
                        writeRandom(Integer.parseInt(answer));
                        break;
                    }
                }
            }
        }
```

```
    }

    //ask user to some info
    static String ask(String answer){
        System.out.print(answer);
        return reader.nextLine();
    }

    static void sayHi(){
        System.out.println();
        System.out.println("Hi, I'm random generator.");
        System.out.println();
    }

    //print some help info
    static void sayInstructions(){
        System.out.println();
        System.out.println("To generate one more random from 1 to your number?"
      + " just enter it.");
        System.out.println("To exit print exit");
        System.out.println();
    }

    //priny a random number
    static void writeRandom(int to){
        Random rnd = new Random();
        System.out.println(rnd.nextInt(to) + 1);
    }

}
```

4.2 Фирма.

Программа, состоящая из четырех классов, реализующая пример абстрактной фирмы и демонстрирующая основные принципы ООП.

code/OOP/firm.java

```
package TestField7;

import TestField7.people.engineer;
import TestField7.people.seller;

public class firm {
    public static void main(String[] args) {
        //create two new workers
        seller firstWorker = new seller("Ivan", 10);
        engineer secondWorker = new engineer("NotIvan", 20);
        //setting selary with two methods
        firstWorker.setSalary(15);
        secondWorker.setSalary(20);
        //-------------------------
        firstWorker.setSalary(15.5);
        secondWorker.setSalary(20.5);
        //set workers tasks
        firstWorker.setTask("some task");
        secondWorker.setTask("other task");
        //calling extra methods
```

```
        firstWorker.sell("ball");
        secondWorker.engendering("bridge");

    }
}
```

code/OOP/worker.java

```
package TestField7.people;

public class worker {
    //worker values
    private String task;
    private String name;
    private String holidayTask;
    private double salary;

    //constructor
    public worker(String name, double salary){
        this.salary = salary;
        this.name = name;
    }            sss

    //methods to change the values
    public void setTask(String task){
        this.task = task;
    }
    public void setSalary(int salary){
        this.salary = salary;
    }
    public void setSalary(double salary){
        this.salary = salary;
    }
    protected void setHolidayTask(String holidayTask){
        this.holidayTask = holidayTask;
    }
}
```

code/OOP/engineer.java

```
package TestField7.people;

public class engineer extends worker {

    //constructor of class engineer
    public engineer(String name, double salary) {
        super(name, salary);
    }

    //methods of super class
    //overriding start
    @Override
    public void setTask(String task) {
        super.setTask(task);
    }

    //first polimorph method
    @Override
    public void setSalary(int salary) {
        super.setSalary(salary);
    }
```

```java

    //second polimorph method
    @Override
    public void setSalary(double salary) {
        super.setSalary(salary);
    }

    @Override
    protected void setHolidayTask(String holidayTask) {
        super.setHolidayTask(holidayTask);
    }
    //methods of super class
    //overriding stop

    //extra method
    public void engendering(String item){
        System.out.println("I'm engendering " + item + ".");
    }

}
```

code/OOP/seller.java

```java
package TestField7.people;

public class seller extends worker {

    //constructor
    public seller(String name, double salary) {
        super(name, salary);
    }

    //methods of super class
    //overriding start
    @Override
    public void setTask(String task) {
        super.setTask(task);
    }

    //first polimorph method
    @Override
    public void setSalary(int salary) {
        super.setSalary(salary);
    }

    //second polimorph method
    @Override
    public void setSalary(double salary) {
        super.setSalary(salary);
    }

    @Override
    protected void setHolidayTask(String holidayTask) {
        super.setHolidayTask(holidayTask);
    }
    //methods of super class
    //overriding stop

    //extra method
    public void sell(String item){
        System.out.println("I'm selling " + item + ".");
```

```
    }
}
```

4.3 Черный ящик.

Два класса, демонстрирующие принцип черного ящика на простейшем примере.

code/OOP/FrontEnd.java

```
  package   TestField7;

public class FrontEnd {

    public static void main(String[] args) {
        //creating black box object
        BlackBox blackBox = new BlackBox();
        //string to work on it with bb
        String s = "Sun - earth's satilite. Relly, sun doesn't light.";
        //print the string
        System.out.println(s);
        //print the "BlackBoxed" string
        System.out.println(blackBox.getResult(s));
    }

}
```

code/OOP/BlackBox.java

```
  package   TestField7;

public class BlackBox {

    //function getResult of BlackBox class,
    //it's including the "black box" algorithm
    public String getResult(String s){
        return s.replaceAll("Sun", "Moon").replaceAll("sun", "moon");
    }

}
```

4.4 Читабельност (ООП).

Программа в объектном стиле для проверки читабельности этого стиля.

code/OOP/Readable_oop.java

```
package TestField7;

public class Readable_oop {

    //robots' names
    public static String[] robotsNames = {"cleaner", "killer"};

    public static void main(String[] args) {
        //run the programm
```

```java
        new Readable_oop();
    }

    public Readable_oop(){
        //creating robots' manager
        RobotManager manager = new RobotManager();
        //call the fill method
        manager.generateRobots(10);
        //call two robots to coordinates
        manager.callTheNearestRobotToCordinates(robotsNames[0], 10, 10);
        manager.callTheNearestRobotToCordinates(robotsNames[1], 10, 10);
    }

}
```

code/OOP/RobotManager.java

```java
package TestField7;

import java.awt.*;
import java.awt.geom.Point2D;
import java.util.Random;

public class RobotManager {

    //array of robots
    private Robot[] robots;

    //constructor
    public RobotManager(){
        System.out.println("Robot manager started.");
    }

    //fill the array
    public void generateRobots(int i){
        System.out.println("New " + i + " robots created.");
        Random rnd = new Random();
        robots = new Robot[i];
        //run throw all the array's elements
        for (int j = 0; j < i; j++) {
            //creating a robot with random values
            robots[j] = new Robot(Readable_oop.robotsNames[rnd.nextInt(2)],
                    rnd.nextInt(100), rnd.nextInt(100));
        }
    }

    //call the nearest robot to coordinates
    public void callTheNearestRobotToCordinates(String name, int your_x, int
        your_y){
        //creating new robot variable
        Robot nearest = null;
        //save the target position
        Point your_pos = new Point(your_x, your_y);
        int index = 0;
        //if robot's name equals given name
        //and distance to current robot lesser
        //then distance to current nearest robot
        //or there is no it, save
        //current robot
        for (int i = 0; i < robots.length; i++) {
            if (robots[i].name.equals(name) &&
```

```
                    (nearest == null || (dist(your_pos, robots[i].pos)
            < dist(your_pos, nearest.pos)))){
                nearest = robots[i];
                index = i;
            }
        }
        //mave the nearest robot to te target point
        if (nearest == null) {
            System.out.println("There is no such robot.");
        } else {
            System.out.println("We sent robot-" + name + " under index " +
                index +
          " to x(" + your_x + ") y(" + your_y + ").");
            robots[index].moveTo(your_x, your_y);
        }
    }

    //count the distance
    private int dist(Point p1, Point p2){
        return (int) Math.sqrt(Math.pow(p1.x - p2.x, 2) + Math.pow(p1.y - p2.y,
            2));
    }

}
```

code/OOP/Robot.java

```
package TestField7;

import java.awt.*;

//robot realisation
public class Robot {

    String name;
    Point pos;

    //constructor
    public Robot(String name, int x, int y){
        pos = new Point(x, y);
        this.name = name;
    }

    //method to move the robot
    public void moveTo(int x, int y){
        pos = new Point(x, y);
    }

}
```

Пример вывода

Robot manager started.
New 10 robots created.
We sent robot-cleaner under index 8 to x(10) y(10).
We sent robot-killer under index 1 to x(10) y(10).

4.5 Читабельност (процедурный стиль).

Программа в процедурном стиле для проверки читабельности этого стиля.

code/PP/Readable.java

```java
package TestField7;

import java.util.Random;

public class Readable {

    //robot's count
    final static int size = 10;
    //robot's names
    static String[] names = new String[size];
    //x - coordinates
    static int[] x_coordinates = new int[size];
    //y - coordinates
    static int[] y_coordinates = new int[size];
    //available robot's names
    static String[] robotsNames = {"cleaner", "killer"};
    //random numbers generator
    static Random rnd;

    public static void main(String[] args) {
        rnd = new Random();
        //fill robot's atributes
        fill(size);
        //call the nearest robots to (10, 10)
        callTheNearestRobotToCordinates(robotsNames[0], 10, 10);
        callTheNearestRobotToCordinates(robotsNames[1], 10, 10);
    }

    public static void fill(int size){
        //report about robots creating
        System.out.println("New " + size + " robots created.");
        //run throw all elements of all arrays
        for (int i = 0; i < size; i++) {
            //set up a random name
            names[i] = robotsNames[rnd.nextInt(2)];
            //set up a random x
            x_coordinates[i] = rnd.nextInt(100);
            //set up a random y
            y_coordinates[i] = rnd.nextInt(100);
        }
    }

    //call the nearest robot by name to coordinates
    public static void callTheNearestRobotToCordinates
            (String robot_name, int x, int y){
        //robot's index variable
        int index = -1;
        //run throw all elements of all arrays
        for (int i = 0; i < names.length; i++) {
            //if robot's name equals given name
            //and distance to current robot lesser
            //then distance to robot under index
            //or index equals -1, save the index of
            //current robot
            if (names[i].equals(robot_name) &&
                    (index == -1 || (
                    dist(x, y, x_coordinates[i], y_coordinates[i]) <
                    dist(x, y, x_coordinates[index], y_coordinates[index]))) ){
                index = i;
```

```java
            }
        }
        //print info about the nearest robot and move it
        if (index == -1) {
            System.out.println("There is no such robot.");
        } else {
            System.out.println("We sent robot-" + robot_name + " under index " +
                    index + " to x(" + x + ) y(" + y+ ").");
            x_coordinates[index] = x;
            y_coordinates[index] = y;
        }
    }

    //count the distance
    public static int dist(int x1, int y1, int x2, int y2){
        return (int) Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2));
    }

}
```

Пример вывода

Robot manager started.
New 10 robots created.
We sent robot-cleaner under index 4 to x(10) y(10).
We sent robot-killer under index 9 to x(10) y(10).

Источники

- http://artanovy.com/20robyl11/03/procedurnoe-protiv-obyektno-orientirovannogo/
- https://myrusakov.ru/oop-vs-procedure-style.html
- Обзор Объектно-Ориентированного Программирования (http://www.firststeps.ru/theory/oop/)
- Объектная парадигма провалилась Ричард П. Гэбриэл
- Процедурное (алгоритмическое) программирование (http://bourabai.kz/alg/classification01.htm)