Объектно-ориентированное программирование

Методическое пособие по программированию

для технических вузов

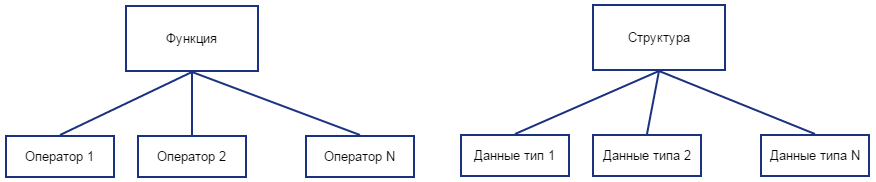
Охотников Д.А.

2018 г. Москва

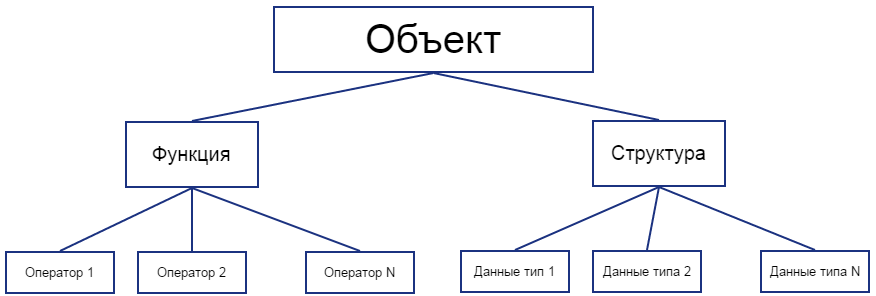
Введение

Со временем размер программ становился все больше, и писать такие программы становилось все труднее. Вследствие чего необходимо увеличение уровней абстракции для выполнения задач декомпозиции.

Ввиду увеличения объема программ процедурное программирование предложило два весомых средства для формирования нового уровня абстракции – это функции и структуры.



Объединение различных сущностей под единым смыслом позволяет сформировать новый уровень абстракции.



1. **Классы. TCar.**

Программа генерирует объект типа TCar и случайным образом заполняет поля объекта: model, price, year, color. После чего проверяет полученный объект на соответствие полей объекта требованиям покупателя (model=«Lada», price<1000, и т.д.). В случае если какой-то из параметров не соответствует требованию, программа выводит информационное сообщение об этом, уничтожает объект, создает новый и начинает проверку заново. Если все поля объекта соответствуют требованиям, то программа проверяет работоспособность автомобиля, вызывая у него методы: bool start(); bool ternLeft(); bool ternRight(); bool stop();

Методы с заданной вероятностью возвращают true, что свидетельствуют о прохождении испытания. В случае если какое-то испытание не пройдено, программа выводит об этом сообщение, уничтожает объект, создает новый и начинает проверку с самого начала. Если все испытания пройдены успешно, программа выводит сообщение о покупке автомобиля.

1. **Инкапсуляция. TBike.**

Создать класс TBike, с приватными полями: размер бензобака (fuelTank) и количество бензина (fuel). Класс имеет методы для считывания значений каждого поля и метод заправки бака бензином (refuel(value)). Написать программу демонстрирующую работу класса.

1. **Наследование. TCalc.**

Написать класс TCalc реализующий работу калькулятора с методами сложения и вычитания: add(x,y) и sub(x,y).

Написать класс TBuhCalc, наследующий класс TCalc с новым методом взятия процентов: procent(x).

Написать класс TIngeneerCalc, наследующий класс TBuhCalc с новым методом вычисления функции синус: sinus(x).

Написать программу, демонстрирующую работу все трех классов.

1. **Массив объектов. TStudent.**

Создать класс TStudent с полями: фамилия, возраст, группа. Добавить несколько объектов класса TStudent в массив. Написать программу позволяющую отображать список всех студентов, сортировать список по фамилии, возрасту и группе.

1. **Полиморфизм. TShape.**

Создать классы TCircle, TTriangle, TRectangle, реализующие виртуальный метод drow(), унаследованный от их общего предка TShape, который отображает круг, треугольник и прямоугольник соответственно. Поместить по несколько экземпляров каждого класса в массив и в цикле вывести все фигуры, вызывая метод drow().

1. **Статические поля и методы**

Создать класс, со статическим методом getCount, который возвращает количество экземпляров этого класса.

1. **Шаблон проектирования «Одиночка»**
2. **ООП проектирование «Бабушка и лампочка»**

Бабушка каждый вечер включает лампочку. В случае если лампочка перегорела, бабушка вызывает электрика. Электрик может починить лампочку только в том случае, если ему достаточно роста, чтобы дотянуться до лампочки. В противном случае бабушка дает электрику табуретку, после чего тот, встав на табуретку, чинит перегоревшую лампочку.

1. **Делегирование**

Грузовик может перемещать груз по двум координатам X и У. Кран может перемещать груз по координате Z. Создать класс Грузовик с краном, способный перемещать груз по всем трем координатам, перенаправляя вызовы о перемещении груза объектам грузовик и кран, которые являются его полями.

1. **Интерфейсы**

Создать следующие классы: фонарик, керосиновая лампа, электроутюг. Классы фонарик и керосиновая лампа реализуют интерфейсы ILitable, классы фонарик и электроутюг реализуют интерфейс IChargable. Класс электро-розетка позволяет заряжать фонарик, и включать электроутюг вызовом метода charge(IChargable\* chargable). Класс гирлянда позволяет собрать составной осветительный прибор из фонарей и керосиновых ламп, добавляя очередной элемент вызовом метода addLitable(ILitable\* litable).

1. **Игра Танк**

Программа отображает в консоли изображение танка:

---

|0| ---

---

Вводя символы «w», «a», «d», «s», пользователь может изменять положение и направление танка на один символ вверх, влево, вправо и вниз соответственно.

1. **MVC. TNewspaper**

Объявление: дата, дата закрытия, текст.

В программе есть возможность управлять информационными блоками (отобразить все, добавить блок, удалить блок).

Газета состоит из разных информационных блоков: новость, статья и объявление. Блоки имеют следующее наполнение:

Новость: дата, текст;

Статья: автор, текст, дата;

Объявление: дата, дата закрытия, текст.

В программе есть возможность управлять информационными блоками (отобразить все, добавить блок, удалить блок).

1. **ООП проектирование «Сказка про Красную шапочку».** Реализовать взаимодействие объектов, соответствующих персонажам из сказки. Рассмотреть несколько нестандартных вариантов событий: Красная шапочка не встретила в лесу волка, Красная шапочка не сказала волку, где живет бабушка. Охотники встретили волка до того, как тот съел бабушку и пр.
2. **Шаблон проектирования «Обозреватель». Задача «Ферма»**

У фермера на ферме живут разные животные: курочка, лошадь и верблюд. Каждое утро фермер ведет своих животных на водопой. Однако иногда фермер оказывается не в состоянии позаботиться о животных и они оказываются без воды. Каждое животное может жить определенное время без воды: курочка – 2 дня, лошадь – 4 дня, верблюд – 7 дней.

1. **Exceptions**

Для изготовления деталей на фирму закупили станок. При попытке изготовить на станке деталь она может быть изготовлена, может оказаться, что заготовка для изготовления имеет дефект, такжже возможет полный выход станка из строя. Перечень состояний, который может принимать станок, описывается следующим перечислением:

enum **State**{OK, DEFECT, BROKEN};

Станок реализует интерфейс IExecutable и его запуск осуществляется посредством запуска переопределенного метода State execute(), который возвращает состояние станка на текущий момент.

Решение вопросов связанных с бракованной заготовкой решается благодаря реализации интерфейса IFixable и его метода void fixDefect(). Починка же станка происходит посредствам реализации метода void repair() интерфейся IRepairable. Диаграмма классов приведена на рисунке except1.

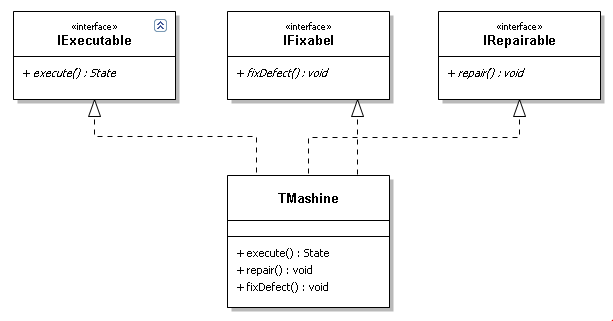


Рис. except1 - Диаграмма классов станка, закупленного на фирме

В фирме есть директор, который каждый день дает указание менеджеру работать. Начальник дает указание работать рабочему, который обрабатывает заготовку, производя из нее деталь. Заготовка может оказаться бракованной, в этом случае, рабочий сообщает начальнику об этом. Начальник решает проблему рабочего, и рабочий продолжает свою работу. Также у рабочего может сломаться станок, в этом случае он также сообщает об этом менеджеру. Но начальник не может решить проблему покупки нового станка, поэтому пробрасывает это сообщение директору. Директор решает вопрос со станком, начальник и рабочий продолжают свою работу.

Идея механизмов исключений в том, чтобы отделить код бизнес логики от кода обработки ошибок. Поэтому в блок **catch** не должен попадать код, реализующий бизнес логику, только код обработки ошибок.

В задаче в блоке catch должны быть вопросы, связанные с решением проблемы бракованной детали (у начальника) и проблемы починки станка (у директора). Запуск менеджера и рабочего в работу в этом блоке не должны присутствовать.

Ошибки времени выполнения

Ошибки времени выполнения ловить бессмысленно, поэтому и не нужно. К таким исключительным ситуациям относятся:

- переполнение памяти

Готовые классы исключений в STL

Помечаем методы, бросающие и не бросающие исключения

1. **Шаблоны типов templates**
2. **Свой вектор**

Реализовать вектор на базе динамического массива, изначальный размер вектора 4 элемента, при добавлении 5ого элемента вектор увеличивает свой размер в два раза.