Коллоквиум по ОС от 26.10.2023

Вариант 4

Нулевая группа вопросов

1. Win API, необходимое для решения Лабораторной работы номер 2:

CreateProcess()/CreateProcessA(),

WaitForSingleObject(),

CloseHandle() и др.

2.

Процесс - это программа или команда, выполняемая на компьютере. С помощью команд вы сообщаете операционной системе, какую задачу ей следует выполнить. Введенные команды расшифровываются интерпретатором команд (называемым оболочкой), после чего задача выполняется.

3. **Критическая секция** (англ. *critical section*) — объект синхронизации потоков, позволяющий предотвратить одновременное выполнение некоторого набора операций (обычно связанных с доступом к данным) несколькими потоками. Критическая секция выполняет те же задачи, что и мьютекс.

4. **Семафо́р** (англ. *semaphore*) — примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счётчика является блокирующейся. Служит для построения более сложных механизмов синхронизации и используется для синхронизации параллельно работающих задач, для защиты передачи данных через разделяемую память, для защиты критических секций, а также для управления доступом к аппаратному обеспечению

5. Сравнительный анализ стандарта C++ 98 с и без применения библиотеки boost

В начале 2000-х новая библиотека набирает популярность — Boost.

Boost — это набор C++ библиотек, которые заполняют различные пробелы не только в Standard Library, но и в самом языке, двигая его вперёд. C++ вышел в 1983 году, после 4 лет разработки, и базировался на C, который вышел в 1972.

Что касается продвинутых вещей, вроде лямбда-функций(которые так же нативно поддерживаются многими языками) — библиотеки типа Boost будут полезны.

Библиотека Boost в конечном счёте служит нескольким целям:

Во первых, она предоставляет программистам как продвинутые вещи, такие как функциональное программирование, так и базовые, вроде смарт-указателей.

Во-вторых, она является своего рода инкубатором для новых возможностей языка, которые могут стать стандартными.

Прямо на главной странице Boost замечено, что 10 библиотек из её состава было включено в стандарт C++ 11. Да, и она вышла в 2003 году — 10 лет назад, задолго до принятия стандарта.
Итак, давайте сравним Boost со Стандартом.

Стандарт C++ 11 включает несколько изменений в самом языке(например,

приведён в порядок механизм компиляции шаблонов, что позволяет использовать «внешние шаблоны» («extern template»);

так же проще инициализировать объекты-контейнеры;

и даже мехнизм определения типов(type inference mechanism)).

Но так же стандарт содержит множество улучшений в Standard Library.

Так же, одна вещь, которую нужно знать о Boost — это что она на самом деле состоит из нескольких библиотек, и её дизайнеры намеренно разделили их. Это значит, что вы можете использовать только те библиотеки из Boost, которые вам нужны. Там есть несколько взаимозависимостей, но они хорошо документированы. Таким образом, вы можете комбинировать Boost со Standard Library, если необходимо. Например, старые добрые iostream являются частью Standard Library со времён первого стандарта 1998 года, и Boost легко работает с ними(но так же включает свои собственные iostream, если вам так удобнее).

Но Boost, на самом деле, гораздо больше, чем Standard Library.

Она содержит около 175 классов, и текущая версия на SourceForge(1.53.0) занимает 93 Мб (включая документацию, а распакованные .hpp файлы в количестве 9000 штук занимают 90 Мб). Там есть классы, которые лично я нахожу невероятно полезными в моей работе, например те, что относятся к категории «эмуляция особенностей языка» («Language Features Emulation»).

Например — конструкция foreach, спасибо механизму шаблонов в C++, позволяет писать циклы, которые выходят за рамки стандартных циклов C++.

Boost так же предоставляет огромный ассортимент могучих, но узко специализированных классов. Например, в Boost есть библиотека для работы с графами. Нужен многомерный массив? Как насчёт контейнера, основанного на стеке? Boost всё это имеет. Но что пользуется наибольшей популярностью — это целый набор хороших шаблонов программирования и полезная библиотека для работы с регулярными выражениями.

Первая группа вопросов - см. Github-репозиторий.

Вторая группа вопросов

1. *Процедурной декомпозицией* называют представление разрабатываемой программы в виде совокупности вызывающих друг друга подпрограмм. Каждая подпрограмма в этом случае выполняет некоторую операцию, а вся совокупность подпрограмм решает поставленную задачу.
2. При динамическом полиморфизме вызвать переопределеный метод можно при выполнении программы. Скопировать метод родительского класса и обозначить его в дочернем можно с помощью ссылочной переменной родительского класса, при этом вызванный метод будет определяться по объекту, на который она ссылается. Такую операцию еще называют Upcasting.

3. Инкапсуляция — это принцип, согласно которому внутреннее устройство сущностей нужно объединять в специальной «оболочке» и скрывать от вмешательств извне. Доступ к объектам возможен через специальные открытые методы, а напрямую обратиться к их содержимому нельзя.

Третья группа вопросов

1. **Одиночка** — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Одиночка решает сразу две проблемы, нарушая *принцип единственной ответственности* класса.

1) Гарантирует наличие единственного экземпляра класса. Чаще всего это полезно для доступа к какому-то общему ресурсу, например, базе данных.

Представьте, что вы создали объект, а через некоторое время пробуете создать ещё один. В этом случае хотелось бы получить старый объект, вместо создания нового.
Такое поведение невозможно реализовать с помощью обычного конструктора, так как конструктор класса всегда возвращает новый объект.

2) Предоставляет глобальную точку доступа. Это не просто глобальная переменная, через которую можно достучаться к определённому объекту. Глобальные переменные не защищены от записи, поэтому любой код может подменять их значения без вашего ведома.

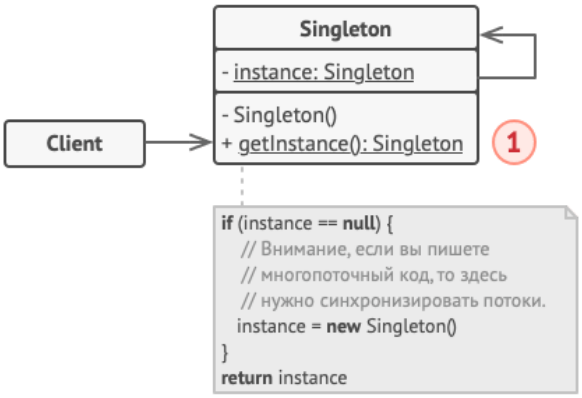
Но есть и другой нюанс. Неплохо бы хранить в одном месте и код, который решает проблему №1, а также иметь к нему простой и доступный интерфейс.

Решение

Все реализации одиночки сводятся к тому, чтобы скрыть конструктор по умолчанию и создать публичный статический метод, который и будет контролировать жизненный цикл объекта-одиночки.

Если у вас есть доступ к классу одиночки, значит, будет доступ и к этому статическому методу. Из какой точки кода вы бы его ни вызвали, он всегда будет отдавать один и тот же объект.

Структура



Одиночка определяет статический метод getInstance, который возвращает единственный экземпляр своего класса.
Конструктор одиночки должен быть скрыт от клиентов. Вызов метода getInstance должен стать единственным способом получить объект этого класса.

2. **Состояние** — это поведенческий паттерн проектирования, который позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.

Основная идея в том, что программа может находиться в одном из нескольких состояний, которые всё время сменяют друг друга. Набор этих состояний, а также переходов между ними, предопределён и *конечен*. Находясь в разных состояниях, программа может по-разному реагировать на одни и те же события, которые происходят с ней.

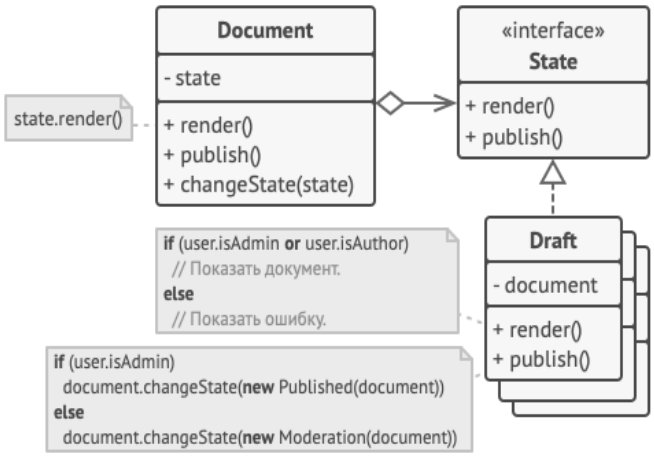
Такой подход можно применить и к отдельным объектам. Например, объект Документ может принимать три состояния: Черновик, Модерация или Опубликован. В каждом из этих состоянии метод опубликоватьбудет работать по-разному:

- Из черновика он отправит документ на модерацию.
- Из модерации — в публикацию, но при условии, что это сделал администратор.
- В опубликованном состоянии метод не будет делать ничего.

Решение

Паттерн Состояние предлагает создать отдельные классы для каждого состояния, в котором может пребывать объект, а затем вынести туда поведения, соответствующие этим состояниям.

Вместо того, чтобы хранить код всех состояний, первоначальный объект, называемый *контекстом*, будет содержать ссылку на один из объектов-состояний и делегировать ему работу, зависящую от состояния.



Документ делегирует работу своему активному объекту-состоянию.

Благодаря тому, что объекты состояний будут иметь общий интерфейс, контекст сможет делегировать работу состоянию, не привязываясь к его

классу. Поведение контекста можно будет изменить в любой момент, подключив к нему другой объект-состояние.