**РТиППО**

**1. Система, информационная система и понимание этого термина в различных нормативных документах.**

**Система** - комплекс элементов, находящихся во взаимодействии и единстве (Л. Берталанфи).

**Информационная система:**

1) Информационной системой совокупность содержащейся в базах данных информации и обеспечивающих её обработку информационных технологий и технических средств (149-ФЗ)

2) Автоматизированную систему, результатом функционирования которой является представление выходной информации для последующего использования ГОСТ РВ 51987

3) Информационно-вычислительная система для обозначения совокупности данных (или баз данных), систем управления базами данных и прикладных программ, функционирующих на вычислительных средствах как единое целое для решения определённых задач ГОСТ Р 53622-2009

4) Система, предназначенная для хранения, поиска и обработки информации, и советующие организационные ресурсы (человеческие, технические, финансовые и т.д.), которые обеспечивают и распространяют информацию ISO/IEC 2382:2015

5) Система, предназначенная для обработки, хранения, поиска, распространения, передачи и предоставления информации ГОСТ 7.0-99

**2. Состав информационной системы.**

Состав информационной системы:

* **Оборудование.** Аппаратное обеспечение (hardware).
* **Программное обеспечение.** Компьютерные программы и документация к ним.
* **Данные (т.е. базы данных).** Данные – это факты, которые используются программами для получения полезной информации.
* **Процедуры.** Процедуры – это политики, управляющие работой компьютерной системы. (Процедуры для людей, как ПО для оборудования)
* **Люди (можно назвать стейкхолдеры).** Влияют на успех или неудачу.
* **Обратная связь.** Обеспечивают модификацию системы.

**3. Стандарты, лучшие практики и фреймворки в области информационных систем.**

Стандарты, лучшие практики и фреймворки дают опыт, который накоплен другими инженерами и исследователями. Если этот опыт не применяется, то в лучшем случае разработчик изобретает то, что уже давно изобретено, а в худшем случае разработчик использует не оптимальные методы.

**ГоСТ** - стандарт России

**Стандарты в области программной и системной инженерии(международные стандарты):**

ISO - международная организация по стандартизации

IEC - международная электротехническая комиссия

IEEE - институт инженеров электротехники и электроники

**Национальные стандарты:**

ANSI - Американский национальный институт стандартов

**Лучшие практики и ведомственные стандарты:**

Университет Карнеги-Меллон

Институт управления проектами

Военно-воздушные силы США

Ассоциации по информационным системам

Ассоциация вычислительной техники

**4. Проект (Project).**

**Проект:** комплекс взаимосвязанных мероприятий, направленный на создание уникального продукта или услуги в условиях временных и ресурсных ограничений.

Проект состоит из уникального набора процессов. Процессы состоят из координируемых и контролируемых работ с датами начала и окончания, которые выполняется для достижения целей проекта. Достижение целей проекта требует получения определенных результатов, отвечающих конкретным требованиям.

**5. Стандарты в области управления проектами.**

ГОСТ Р ИСО 10006–2005. Системы менеджмента качества. Руководство по менеджменту качества при проектировании

РМВОК Guide/ A/Guide to the Project Management Body of Knowledge. Руководство к своду знаний по УП, PMI

PRINCE2. стандарт уп

ГОСТ Р ИСО 21500–2014. Руководство по проектному менеджменту

ГОСТ Р 54869–2011 Проектный менеджмент. Требования к УП

**6. Проектирование (Design) в инженерных дисциплинах.**

**Программирование** — это крупномасштабное упражнение по прикладному абстрагированию, требующее как математических способностей, так и знаний компонентного инженера (Edsger Wybe Dijkstra)

**Инженеры** стараются использовать известные абстракции и механизмы, как основу, на которой строятся новые системы.

**Проектирование (в инженерной дисциплине)**

Строгий (формализированный) подход, с помощью которого находится пути решения определенной проблемы, обеспечивая, таким образом, переход от требований к их исполнению.

**Цель проектирования** - создать систему, которая удовлетворяет требованиям:

* удовлетворяет данным (возможно, неформальным) функциональным требованиям
* имеет приемлемую цену
* удовлетворяет явным и неявным требованиям по эксплуатационным качествам и ресурсопотреблению
* удовлетворяет явным и неявным критериям дизайна
* удовлетворяет требованиям к самому процессу разработки, таким, например, как его стоимость и продолжительность, а также не требует привлечения дополнительных инструментальных средств.

Цель проектирования - создать ясную и относительно простую внутреннюю структуру, иногда называемую архитектурой (Bjarne Stroustrup)

**7. Методологии разработки программного обеспечения.**

<https://habr.com/ru/companies/edison/articles/269789/>

***Waterfall или метод «водопад».*** Все этапы проходят друг за другом. Сначала выполняется один этап и только потом следующий.

***V-образный метод***. Этот метод чем-то похож на предыдущий, но отличается тем, что требования и описания к этапам разработки и тестированию прописываются одновременно.

***Инкрементная модель.*** Это модель, при которой продукт разрабатывается не комплексно, а отдельно по рабочим частям. Каждая отдельная часть разработки — это самостоятельный рабочий продукт. То есть делается часть продукта — выкатывается на рынок. Если все «ок», то разрабатывается другая часть и т. д.

***Итеративная модель.*** Эта модель предполагает, что заказчик продукта может даже не понимать, что получится в итоге, и, соответственно, не способен прописать подробное техническое задание. При такой модели делается часть продукта — согласовывается с заказчиком, потом следующая часть продукта — опять все согласовывается. И так, пока продукт разработки не будет закончен.

***Спиральная модель.***Данная модель делает упор на анализ рисков продукта. Сам же продукт разрабатывается поэтапно. Но в конце каждого этапа проводится анализ и принимается решение, будет ли продолжаться разработка или нет.

***Гибкая модель.***Данная модель подразумевает ведение разработки под постоянным контролем заказчика, когда он может часто лицезреть результат, чтобы понимать, устраивает его продукт или нет. Также эта модель может характеризоваться ежедневными короткими спринт-встречами для постановки задач разработчикам.

***RAD-модель.***Данная модель может характеризоваться параллельной разработкой частей одного проекта разными командами. По сути получается, что один большой проект разбивается на несколько маленьких, которые выполняют разные специалисты. Потом все части объединяются в одну, и получается готовый продукт. Такая модель свойственна ограниченной по времени разработке, когда проект нужно окончить в экстремально сжатые сроки.

**8. Процессный подход при автоматизации и организации проекта.**

**Процесс** – совокупность взаимосвязанных или взаимодействующих видов деятельности, преобразующих входы в выходы.

* Входами к процессу обычно являются выходы других процессов.
* Продукция (product): Результат процесса
  + услуги;
  + программные средства;
  + технические средства;
  + перерабатываемые материалы

В модели жизненного цикла важным **является процесс**. Процессный подход определен Total Quality Management (последствия которого явился ГОСТ ISO 9000). Требования на вход определяет потребитель и качество на выходе определяет потребитель. Оценка качества — это удовлетворенность потребителя.

1. У процессов есть владелец процесса, т.е. ответственный. Его задача - постоянно повышать качество с помощью цикла Дёминга. **Цикл Дёминга (Deming Cycle, круг качества)** – это постоянный круг регулирования усовершенствования продукта и производственных процессов, оптимизации отдельных единиц и объектов. Цикл Дёминга состоит из 4 шагов:
   1. Планирование (менеджмент ресурсов) - планируем как будут устроены не только ресурсы, но и как будут происходить процессы жизненного цикла
   2. Выполнение процессов
   3. Измерение процессов - определяем достигли ли тех планов, которые запланировали на первом шаге
   4. Анализ и изменение планов

**Бесконечный цикл** приводит к автоматическому улучшению качества. В процессном подходе организация процессов решает всё.

1. У процесса всегда есть входы и выходы, соответственно есть источники входа и получатели выхода.
2. Деятельность — это последовательность действий, которые преобразуют входы в выходы.
3. Управление, задачей которой является повышение качества этой деятельность.

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

**9. Процессы жизненного цикла систем и программного обеспечения**

Стандарты, которые определяют процессы, но они не определяют последовательность процессов. Стандарты программного обеспечения и разработки систем: ИСО 15288 (процессы жизненного цикла ИС) и ИСО 12207 (процессы жизненного цикла ПО)

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

ГОСТ Р ИСО/МЭК 15288–2005(Процессы жизненного цикла систем)

**Процессы предприятия:**

* Управление средой предприятия
* Управление инвестициями
* Управление процессами жизненного цикла
* Управление ресурсами
* Управление качеством

**Процессы соглашения:**

* Приобретение
* Поставка

**Процессы проекта:**

* Планирование проекта
* Оценка проекта
* Контроль проекта
* Принятие решений
* Управление рисками
* Управление конфигурацией
* Управление информаций

**Технические процессы**

* Определение требований правообладателей
* Анализ требований
* Проектирование архитектуры
* Реализация
* Комплексирование
* Верификация
* Передача
* Валидация
* Функционирование
* Обслуживание
* Изъятие и списание

ГОСТ Р ИСО/МЭК 12207–2010(Процессы жизненного цикла программного обеспечения)

* Процессы реализации программных средств
* Процессы поддержки программных средств
* Процессы повторного применения программных средств

**10. Признаки сложной системы.**

Пять признаков сложной системы

1. Сложность часто представляется в виде иерархии. (композиция, декомпозиция)
2. Выбор низшего уровня абстракции достаточно произволен и в большой степени определяется наблюдателем
3. Внутриэлементные связи обычно сильнее межэлементных связей. Поэтому высокочастотные взаимодействия внутри структуры оказываются естественным образом отделены от низкочастотных взаимодействий между структурами
4. Иерархические системы обычно состоят из нескольких подсистем разного типа, реализованных в различном порядке и в разнообразных комбинациях
5. Работающая сложная система неизбежно оказывается результатом развития работающей простой системы.

**11. Методы управления сложными системами: абстракция, декомпозиция, иерархия, проектирование сложных систем.**

* **Абстракция** - существенные характеристики некоторого объекта, которые отличают его от всех других видов объектов и, таким образом, четко определяют особенности данного объекта с точки зрения дальнейшего рассмотрения и анализа
* **Декомпозиция** -
* **Алгоритмическая декомпозиция** - структурное проектирование, т.е. связь между функциональными элементами
* **объектно-ориентированная декомпозиция** - список автономных действующих лиц, которые взаимодействуют друг с другом, чтобы обеспечить поведение системы, соответствующее более высокому уровню (скорее всего это нужно)
* **Иерархия** -
  + В науке, как метод классификации, соответствует общим и частным признакам
  + При проектировании и эксплуатации технических объектов соответствует “деталировке”: разбиению крупных объектов на более мелкие
  + В программировании как метод порождения от общего предка объектов, обладающих всё более детализированными признаками(наследование) (скорее всего это нужно)
* **Проектирование сложных систем -** Цель проекта - создать систему, которая удовлетворяет требованиям

1. функциональные требования
2. приемлемая цена
3. эксплуатационным качествам и ресурсопотреблению
4. критерий дизайна
5. процесс разработки

**12. Методы проектирования и их сравнение**

**Методы проектирования**

* **Метод структурного проектирования сверху вниз**
  + Yordan, Wirth, Dijkrsta, Hoara, Mils
  + Проблемы: абстрагирование данных, сокрытие информации, масштабирование, средства для совместной работы
* **Информационное программирование**
  + Ориентированный на данные дизайн смещает перспективу программирования с объектов на сами данные: тип данных, то, и как они будут считываться и обрабатываться в приложении
  + Языки Perl, AWK
* **Объектно-ориентированное проектирование**

**Сравнение**

Структурное проектирование:

* Атрибуты и поведения обычно разделяются
* Данные являются глобальными, благодаря чему их легко модифицировать вне области видимости вашего кода. Доступ к данным неконтролируемый и непредсказуемый.
* Поскольку у вас нет контроля над тем, кто сможет получить доступ к данным, тестирование и откладка намного усложняется

Объектно-ориентированное проектирование:

* Атрибуты и поведения размещаются в рамках одного объекта (управление доступом к членам объектов)
* Данные, содержащиеся в объекте представляют его состояние
* Поведения объектов содержатся в методах, а вызов метода осуществляется путём отправки ему сообщения

**13. Объектно-ориентированные программирование (OOP).**

**Объектно-ориентированное программирование (OOP)** - методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определённого класса, а классы образуют иерархию. на принципах наследуемости

**14. Объектно-ориентированное проектирование (OOD).**

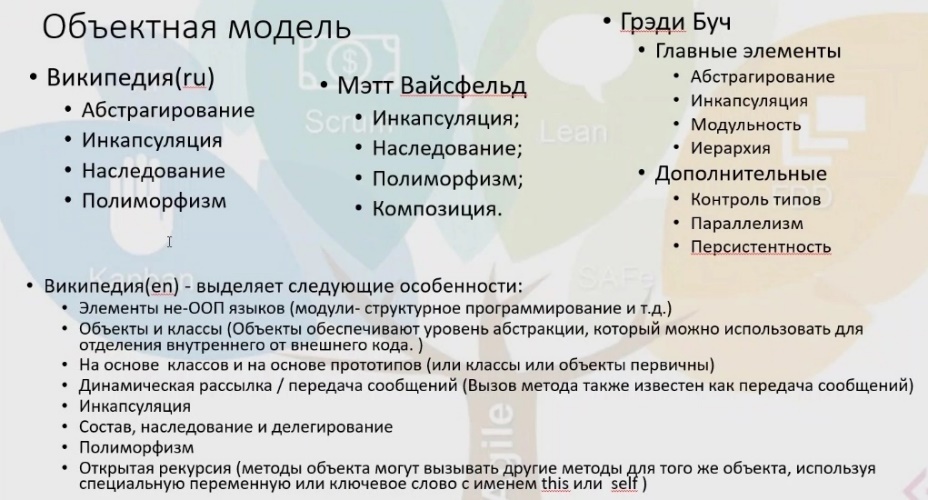
**Объектно-ориентированное проектирование (OOD)** - методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления как логической и физической, так статической и динамической моделей проектируемой системы

**15. Объектно-ориентированный анализ (ООА).**

**Объектно-ориентированный анализ (OOA)** направлен на создание моделей, более близких к реальности, с использованием объектно-ориентированного подхода; это методология, при которой требования формируются на основе понятий классов и объектов, составляющих словарь предметной области

**16. Основные составляющие объектной модели.**

Главные элементы: абстрагирование; инкапсуляция; модульность; иерархия

Дополнительные: контроль типов; параллелизм; персистентность

**17. Абстрагирование и выделение абстракций.**

Абстракция - существенные характеристики некоторого объекта, которые отличают его от всех других видов объектов и, таким образом, четко определяют особенности данного объекта с точки зрения дальнейшего рассмотрения и анализа

|  |  |
| --- | --- |
| Абстракции | Характеристика |
| Сущности | Объект представляет собой модель существенных сторон предметной области |
| Поведения | Объект состоит из обобщённого множества операций, каждая из которых выполняет определённую функцию |
| Виртуальные машины | Объект объединяет группы операций виртуальной машины, которые используются либо для управления объектов, либо соответствуют функциям нижнего уровня |
| Произвольные | Объект включает в себя набор независимых по отношению друг к другу операций |

**18. Основные абстракции ООП, примеры абстракций информационной системы.**

Абстракция является основой объектно-ориентированного программирования и позволяет работать с объектами, не вдаваясь в особенности их реализации

Контрактная модель:

* Клиент – любой объект использующий ресурсы другого объекта (сервера)
* Поведение объекта характеризуется услугами (контрактом), которые он оказывает другим объектам, и операциями, которые он может выполнить над другими объектами
* Внешнее поведение объекта определяется контрактами с другими объектами, выполняемых внутренней структурой
* Контракт определяет ответственность объекта



**19. Инкапсуляция, реализация инкапсуляции в C#**

Инкапсуляция – это процесс разделения абстракций, определяющих её структуру и поведение, инкапсуляция предназначена для изоляции контрактных обязательств абстракции от их реализации.

Для того, чтобы абстракция работала, её реализация должна быть инкапсулирована

* механизм языка, позволяющий ограничить доступ одних компонентов программы к другим
* языковая конструкция, позволяющая связать данные с методами, предназначенными для обработки этих данных

Инкапсуляция (моё определение) – это механизм языка, позволяющий объединить данные и методы, работающие с этими данными в единый объект, и скрыть детали реализации от пользователя

**20. Пререквизиты public, private, static. Поля классов. Модификатор internal.**

* [public](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/public) (+): Доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.
* [private](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/private) (-): доступ к типу или члену возможен только из кода в том же объекте class или struct.
* static (\_) — отличается от нестатического тем, что нельзя создавать экземпляры статического класса, нельзя использовать оператор [new](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/new-operator) для создания переменной типа класса. Доступ к членам статического класса осуществляется с использованием самого имени класса.
* internal: Доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.
* [protected](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/protected) (#): доступ к типу или члену возможен только из кода в том же объекте class либо в class, производном от этого class.
* *Поле* ( В UML — “атрибут” ) является переменной любого типа, которая объявлена непосредственно в [классе](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/class) или [структуре](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/struct). Поля являются *членами* содержащих их типов.

**21. Конструкторы. Порядок инициализации.**

Конструктор — это метод, имя которого совпадает с именем его типа. Сигнатура метода содержит только необязательный Модификатор доступа, имя метода и список параметров; Он не включает тип возвращаемого значения. Каждый раз, когда создается класс или структура, вызывается конструктор. Класс или структура может иметь несколько конструкторов, принимающих различные аргументы.

Порядок инициализации:

1. Статические поля
2. Статический конструктор
3. Нестатические поля
4. Конструктор, вызванный другим конструктором
5. Явна вызванный конструктор
6. Метод

https://ulearn.me/course/cs2/Poryadok\_initsializatsii\_a530d860-f05b-485d-b20b-7733573faea8

**22. Перегруженные методы и параметры по умолчанию.**

Перегруженные методы – это методы которые имеют одинаковое название, но разное количество параметров (или же одинаковое количество параметров, но разный тип, порядок или разные модификаторы параметров.).

Параметры по умолчанию задаются в методе как:

public static void Print(string text, int paramDefault = 1)

{

…

}

Параметру paramDefault задано значение по умолчанию, это значит что при вызове метода, нам необязательно указывать значения для всех параметров, а только для тех аргументов для которых не задано значение по умолчанию: Print(“Пример”)

**23. Маскировка ошибок vs решение проблем.**

Маскировка ошибок – это плохая практика и лучше не маскировать ошибки. Если кусок кода с замаскированной ошибкой применить в другой ситуации, то есть вероятность, что ошибку на уровне алгоритма не заметят, хотя код будет по-прежнему работать, но выдавать неверные значения, и найти такую ошибку будет затруднительно

Лучше решать проблемы сразу и подбирать универсальное решение, которое будет учитывать все возможные ситуации.

**24. Основные характеристики объекта: состояние, поведение индивидуальность.**

**Состояние объекта** - характеризуется перечнем всех возможных (обычно статических) свойств данного объекта и текущими значениями (обычно динамическими) каждого из этих свойств.

Все свойства объекта характеризуется значениями их параметров:

* количественная характеристика;
* объект;

State Machine – конечные автоматы (абстрактный автомат, число возможных внутренних состояний которого конечно). Активные (собственный поток управления) и пассивные объекты (изменяют состояние под воздействием)

**Символ** -  любой атомарный  блок данных.

**Слово** – срока символов.

**Алфавит** – конечный набор различных символов (множество символов).

**Язык** – множество слов, формируемых символами данного алфавита (K = {Q, A, q0, g, F} ДКА (Детерминированный конечный автомат)

Q = {q0, q1, q2, …, qm} – множество состояний автомата;

A = {a0, a1, …, an} – входной алфавит;

q – начальным состоянием;

g – функция переходов QxA→Q(g(qi, aj)=qk);

F – множество конечных состояний F ⊆ Q.)

**Поведение** – это действия и реакция объекта, выраженные через **изменения состояния** объекта и **передачу сообщения**.

Поведение – видимая извне деятельность;

* Сообщение – это операция, которую один объект выполняет над другим;
* Сообщение (message) – односторонняя коммуникация между двумя объектами в виде передачи потока управления от объекта-отправителя объекту-получателю с передачей информации в виде параметров (UML)
* Сообщение ≡ операция объекта получателя, которую вызывает объект-отправитель

*Состояние объекта представляет собой суммарный результат его поведений; Поведение зависит не только от сообщения, но и от состояния.*

Операция – это услуга, которую объект (или класс) оказывает клиентам.

**Индивидуальность** - это свойство объекта, отличающее его от других объектов;

Имя объекта ≠ индивидуальность;

Структурное разделение (возможность псевдонима объекта) → порождает;

* Висячие ссылки;
* Утечки памяти

**25. Роли и обязанности объекта.**

Протокол- совокупность всех методов, связанных с объектом

* Для сложных объектов протокол делят на роли
* Состояние + поведение = роль

Обязанность объекта выражают смысл его предназначения и место в системе

* Обязательства – это совокупность всех услуг, предусмотренных во всех контрактах
* Роль обеспечивает выполнение обязанностей

**26. Проектирование на основе обязанностей (Responsibility-Driven Design — RDD).**

Обязанности – к действиям объекта.

* Выполнение некоторых действий самим объектом, например создание экземпляра или выполнение вычислений (создание нового Photo).
* Инициирование действий других объектов.
* Управление действиями других объектов и их координирование

Обязанности – к знаниям объекта.

* Наличие информации о закрытых инкапсулированных данных (информация о Photo)
* Наличие информации о связанных объектах
* Наличие информации о следствиях или вычисляемых величинах

**27. Соотношение понятия индивидуальности объекта и имени объекта.**

**Индивидуальность** - это свойство объекта, отличающее его от других объектов;

Имя объекта ≠ индивидуальность (“экземпляры могут иметь разные имена, но если они указывают на один и тот же участок памяти, программа не видит в них разницы”);

**28. Псевдонимы, как причины появления утечки памяти.**

**Утечка памяти** — это ситуация, когда в куче есть объекты, которые больше не используются, но сборщик мусора не может удалить их, что приводит к нерациональному расходованию памяти.

**Изображение выглядит как диаграмма, зарисовка, линия, белый

Автоматически созданное описание**

Несмотря на то что объект **item1** и объект, на который ссылается указатель **item2**, имеют одинаковое состояние, они остаются разными объектами. Кроме того, состояние объекта, на который ссылается указатель **item3**, было изменено с помощью его нового косвенного имени **item4**. Эта ситуация называется **структурным разделением** и означает, что объект может именоваться по-разному. Иначе говоря, объект может иметь псевдонимы. Структурное разделение порождает много проблем в объектно-ориентированном программировании. Трудность распознавания побочных эффектов при обращении к объекту через псевдонимы часто приводит к "утечкам памяти", нарушению правил доступа к памяти. Например, если уничтожить объект, на который ссылается указатель **item3**, то значение указателя **item4** окажется бессмысленным. Такая ситуация называется **висячей ссылкой**.

Изображение выглядит как диаграмма, зарисовка, линия, белый

Автоматически созданное описание

Рассмотрим рисунок, иллюстрирующий результат изменения значения указателя **item2** так, чтобы он ссылался на объект **item1**. Теперь указатель **item2** ссылается на объект **item1**. К сожалению, при этом произошла утечка памяти — объект, на который первоначально ссылался указатель **item2**, больше не именуется ни прямо, ни косвенно, и его индивидуальность потеряна. В таких языках программирования, как **Smalltalk**, **Java и C#**, память, выделенная для подобных объектов, уничтожается механизмом "сборки мусора". Такие утечки памяти могут вызвать и простое неудобство, и крах, особенно если программа должна непрерывно работать длительное время.

**29. Понятие класса и классификатора.**

**Класс (Структура + поведение)**

* Это множество объектов, обладающих общей структурой, поведением и семантикой (Г. Буч)
* Класс описывает структуру и поведение, общую для всех родственных объектов
* Класс — это конкретная реализация структурных и поведенческих классификаторов, с целью указать классификацию объектов и указать особенности, которые характеризуют структуру и поведение объектов (Спецификация UML)

**Класс (Контракт)**

* Класс это контракт связывающий абстракцию с её клиентом
* → внешнее и внутреннее представление класса
* Внешнее представление класса – интерфейс (только public операции)

**Классификатор**

* Классификатор представляет собой классификацию экземпляров в соответствии с их характеристиками
* Классификатор имеет набор Особенностей (Feature), некоторые из которых являются свойствами (Property)
* Свойства – это структурные признаки, которые представляют атрибуты классификаторов, окончание ассоциации и часть структурированного классификатора.

**30. Отношения (Relationships) между классами с связи (link) между объектами.**

Поведение системы определяется отношением между объектами

**Связь (link)** - физическое или концептуальное соединение между объектами

**Отношение** между классами:

* **Наследование** каждый дочерний класс может использоваться вместо родительского, но не наоборот
* **Зависимость** – отношение сервера (поставщика) / клиента между элементами модели, модификатор поставщика может повлиять на элементы модели клиента
* **Ассоциация** – экземпляры классификаторов могут быть либо связаны друг с другом, либо объединены логически или физически
* **Агрегация(has-a)** форма ассоциации, описывающая отношение часть-целое
* **Композиция(part-of)** форма агрегации, части принадлежат целому.
* **Реализация(instance-of)** отношение между спецификацией и её программной реализацией, поведение наследуется без структуры.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

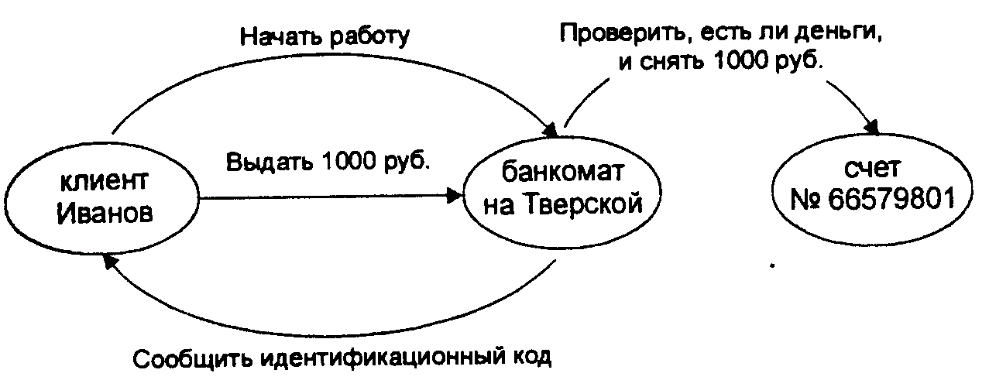
Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

**31. Модель взаимодействия объектов**

**Модель взаимодействия объектов (МВО)** - графическое представление взаимодействия между моделями состояний и внешними сущностями

МВО рисуется в овале, внешняя сущность – прямоугольник (именуемый терминатором). События, которые порождаются одной моделью для другой, рисуются стрелкой – так же могут приходить события от внешних сущностей. События могут быть направлены к терминаторам. МВО формируется иерархически – объекты, наиболее осведомленные о всей системе (активные) располагаются вверху диаграммы.



**32. Наследование в ООП Наследование реализации (implementation inheritance)**

**Наследование** – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом.

**Наследование реализации (implementation inheritance)** означает, что тип происходит от базового типа, получая от него все поля и функции базового класса.

**33. Наследование интерфейса (interface inheritance)- класс реализует интерфейс.**

**Наследование интерфейса (interface inheritance)** означает, что тип наследует только сигнатуру функций, но не наследует никакой реализации. Этот тип наследования полезен, когда нужно специфицировать, что тип обеспечивает доступ к определенным средствам.

**34. Чем опасно наследование?**

**НАСЛЕДОВАНИЕ НАРУШАЕТ ИНКАПСУЛЯЦИЮ**

Правильное функционирование подкласса зависит от деталей реализации его суперкласса

* Реализация суперкласса может меняться, следовательно подкласс может не работать
* В новых версиях суперкласс может обзавестись новыми методами, следовательно новая функциональность появится в подклассе

**35. Иерархия и модульность.**

**Иерархия** — это ранжирование или упорядочение абстракций

Наиболее важными абстракциями

* наследование
* агрегация

**Модульность** — это разделение программы на фрагменты, которые компилируются по отдельности, но связаны между собой

**36. Полиморфизм: Ad hoc (специальный)полиморфизм, Параметрический полиморфизм, Полиморфизм подтипов**

**Полиморфизм** - предоставление единого интерфейса к объектам различных типов; использование одного символа для представления нескольких различный типов

**Ad hoc (специальный)полиморфизм:** определяет общий интерфейс для произвольного набора индивидуально указанных типов

* Перегрузка методов
* Перегрузка операторов

**Параметрический полиморфизм:** когда один или несколько типов указываются не по имени, а по абстрактным символам, которые могут представлять любой тип.

* Generics (универсальные шаблоны) методы, классы, интерфейсы, делегаты

**Полиморфизм подтипов:** в исходном коде можно вызвать метод базового класса, а при выполнении к объекту дочернего класса будет приводится метод, определённый в его классе(т.е вызываться метод дочернего класса, к которому принадлежит объект) – virtual, override

**37. Касты к базовым классам. Касты к интерфейсам**

Мы можем конверсию делать только между классами наследниками. Конверсия к базовому классу – Upcast. Конверсия к производящему классу – Downcast.

**Каст к интерфейсу возможен всегда (ошибки компиляции не будет, но будет Exception),** потому что может быть в переменной лежит объект класса наследника, который стал реализовывать тот интерфейс, который не реализует класс в переменной. Если имеется два одинаковых с названием методы интерфейса наследуемых, следует указывать к какому интерфейсу мы кастим, а потом только вызывать метод. **As** не вызывает исключения

**38. Чистые и грязные методы.**

Чистые функции – не меняют состояние объекта и никак не контактируют с этим объектом. Чистые функции принимают необходимые аргументы и возвращает определённый объект (облегчает тестирование). Также можем сделать эти методы статическими при желании

Грязные методы – взаимодействие осуществляется через поля классов. Не можем сделать статическим, так как нет доступа к полям объекта. При тестировании нужно смотреть какие поля есть у изменяемого объекта и как инициализировать эти поля, чтобы функция работала. Также меняют состояние полей и это состояние полей влияет на следующие методы, которые находятся вне иерархии вызова. Имеют непредсказуемый эффект: неотлаживаемое поведение, не тестируемое. Ведут к неработоспособности программы

**39. Generic-классы. Generic-методы.**

**Обобщенные классы (Generic classes)** — это классы, обладающие параметрическим полиморфизмом (т. е. классы, которые изменяют свое поведение в зависимости от приписываемого им типа. Этот тип указывается в угловых скобках <T> сразу после имени класса).

**Generic-методы** — это методы, которые содержатся в обычных, не дженерик-класса, но при этом имеют дженерик-параметры. Необязательно указывать тип generic метода, по сравнению с классом, это делает компилятор сам, но только в том случае если параметрах также присутствует generic тип.

**40. Ковариация и контравариация**

Иерархия наследования дженерик-аргумента не переносится на дженерик-классы! — Это инвариантность. По умолчанию, дженерик-интерфейсы также инвариантны

Имеется три возможных варианта поведения:

* **Ковариантность (ключевое слово out)**: позволяет использовать более конкретный тип, чем заданный изначально
* **Контравариантность (ключевое слово in)**: позволяет использовать более универсальный тип, чем заданный изначально
* **Инвариантность**: позволяет использовать только заданный тип

Ключевое слово **out** означает, что интерфейс ковариантен по соответствующему параметру. Ковариация: если из комнаты выходят только сотрудники, то верно, что выходят только люди. Неверно, что выходят только менеджеры.

Ключевое слово **in** означает, что интерфейс контравариантен по соответствующему параметру. Контравариация: если все сотрудники могут войти в комнату, то верно, что все менеджеры могут войти. Неверно, что все люди могут.

**41. Делегирование без делегатов**

Это вынос функционала(кода) больших делегатов в отдельный класс, с целью упрощения процесса тестирования.

**Делегирование** (альтернатива наследованию) – объект поручает свое поведение другому

Делегирование без делегатов приводит к реализации наследования

Альтернативное определение:

Это применение паттерна делегирования без использования делегатов, с целью упрощения процесса тестирования. То есть, фактически, класс в котором был применен данный паттерн, не реализует определенный функционал сам, а делегирует его какому-то другому классу, описанному для реализации этого функционала. Паттерн делегирования не заменяет наследование делегатами. Использование самих делегатов уместно только в том случае, если функционал достаточно простой, в противном случае, такой функционал будет проще вынести в отдельный класс.

**42. Рефлексия типов. Профилирование рефлексии.**

**Рефлексия** процесс выявления типов во время выполнения приложения.

**Рефлексия** — это доступ к объектной модели исходного кода программы, во время работы программы.

**Профилирование** - сбор характеристик работы программы, таких как время выполнения отдельных фрагментов, число верно предсказанных условных переходов, число кэш-промахов и т. д.

**43. Для чего требуется классификация? Идентификация классов и объектов.**

Идентификация классов и объектов одна из самых сложных задач ОО анализа и проектирования. Ее решение содержит в себе элементы открытия и изобретения

* открытие позволяет распознать основные абстракции и механизмы, образующие словарь предметной области
* изобретение дает возможность сконструировать обощенные абстракции и новые механизмы, определяющие основные способы взаимодействия объектов

Классификация в ООП позволяет определить:

1. Иерархии классов (обобщенную, специализированную и собирательную)
2. Общие формы взаимодействия объектов
3. Механизм, который может стать основой реализации проекта.
4. Модульную структуру (Расположение в одном или разных модулях зависит от степени общности объектов; Взаимосвязь и влияние — всего лишь меры этой общности.)

**44. Подходы категоризации : 1) классический подход; 2) подход на основе семейного сходства; 3) прототипический; 4) логический; 5) полевой; 6) оппозиционный; 7) когнитивный.**

https://cyberleninka.ru/article/n/kategorizatsiya-i-ee-osnovnye-printsipy/viewer

**Классическое подход** — это наличие четко определенных единиц категории, которые определялись общими, достаточными и необходимыми свойствами объекта.

**Подход на основе семейного сходства** – выделение категорий производится не по наличию у основного объекта существенных признаков, а на основе тех или иных характеристик у основного члена группы и у остальных, входящих в данную категорию

**Прототипический** – это семантика прототипа определяет значение наименования, исходя из представления о типичных условиях его употребления.

**Логический подход** - заключается в целенаправленном переходе от видового понятия к родовому понятию, путем отбрасывания видообразующих признаков

**Полевой подход -** выделяет категории на основе принципа семантической инвариантности / вариантности.

**Оппозиционный подход** - к выделению категорий образован на основе противопоставления членов одной категории к другой.

**Когнитивный подход -** также выделяет категории на основе прототипов. Но в отличие от определения прототипа с психологической точки зрения, связанного с выделением категорий на основе экспериментов по категоризации, когнитивный подход определяет прототип как когнитивный ориентир.

**45. Примеры категоризации в ООА и ООП.**

Примеры категоризации в ООА и ООП: анализ предметной области, CRC карточки, анализ прецедентов, структурный анализ.

**CRC карточки** (Class-Responsibility-Collaboration – Класс-Взаимодействия-Сотрудничества) – удобный способ для определения классов и их взаимодействия. Главным предназначением CRC карточек является концентрирование разработчики на главных абстракциях задачи. Вверху карточки пишется название класса, в левой половине – за что он отвечает, в правой – с кем сотрудничает.

**Структурный анализ.** Много аналитиков применяют этот подход и имеется большое число программных CASE-средств, поддерживающих автоматизацию этих методов**.** После проведения структурного анализа мы уже имеем модель системы, описанную диаграммами потоков данных и другими продуктами структурного анализа. Эти диаграммы дают нам формальную модель проблемы. Исходя из модели, мы можем приступить к определению осмысленных классов и объектов.

Первоначальное исследование внутренних механизмов системы **называется анализом прецедентов** (use case analysis). При анализе прецедентов в начальной, обобщенной степени определяется, как должны взаимодействовать внутренние элементы, чтобы выполнить функциональные требования к системе, и как они относятся друг к другу статически.

Классификатор представляет собой классификацию экземпляров в соответствии с их характеристиками  
Классификатор имеет набор Особенностей (Feature), некоторые из которых являются свойствами (Property)

**46. Критерии чистого кода. SRP Модульность.**

Критерии чистого кода:

* **простота и понятность** (чтобы в будущем инженер смог быстро разобраться и доработать компонент под изменившиеся требования)
* **корректность** (чтобы в будущем инженер своими правками случайно не сломал работоспособность системы)
* **расширяемость** (чтобы в будущем инженеру проще было вносить доработки под новые требования)
* **универсальность** (чтобы в будущем инженеру было проще использовать этот код в контексте другой задачи или проекта)

**SRP** **(Single Responsibility Principle)** - У каждого модуля (класса / метода) должна быть лишь одна реалистичная причина для изменения

Модульность:

* **Декомпозиция** (разбитие большой задачи на подзадачи);
* Компонуемость (подзадачи должны быть теоретически более вероятны для повторного использования);
* **Читаемость** (должен быть понятен код модуля);
* **Защищенность** (всё плохое, что случилось внутри модуля, должно остаться внутри модуля. Ошибки, которые совершаются внутри модуля не должны влиять существенно на работу других модулей)

**47. История UML. Понятие модели. Таксономия диаграмм. Инструменты проектирования.**

https://ru.wikipedia.org/wiki/UML

**UML** - Unified Modeling Language - универсальный язык моделирования

С начала было три компании, которые разрабатывали универсальный язык моделирования для объектно-ориентированного программирования

* Booch
* OOSE
* OMT

После компания OMG решает всё это объединить и так появляется UML в 1996г.

**Модель UML** – это совокупность конечного множества конструкций языка, главные из которых – это сущности и отношения между ними.

**Таксономия диаграмм**

Диаграммы UML разделяются на две группы.

* структурные диаграммы
* диаграммы поведения

**Инструменты проектирования**

* Star UML
* UML дизайнерский инструмент
* Умбрелло
* И т.д.

**48. Структурные диаграммы и и диаграммы поведения. Диаграммы взаимодействия.**

**Структурные диаграммы:**

1. Диаграммы пакетов (package diagram)
2. Диаграммы классов (class diagram)
3. Диаграммы компонентов (component diagram)
4. Диаграммы развертывания (deployment diagram)
5. Диаграммы объектов (object diagram)
6. Диаграммы композитных структур (composite structure diagram)
7. Диаграмма профилей (UML2.2)

**Диаграммы поведения:**

1. Диаграмма прецедентов (use case diagram) (диаграмма вариантов использования)
2. Диаграмма деятельности (activity diagram)
3. Диаграмма состояний (state machine diagram)
4. Диаграммы взаимодействий (interaction diagram)
   1. Диаграмма последовательностей (sequence diagram)
   2. Диаграмма коммуникации (communication diagram)
   3. Диаграмма обзора взаимодействий (interaction overview diagram)
   4. Диаграмма синхронизации (timing diagram)(UML2.0)

**49. Диаграммы пакетов, компонентов.**

**Диаграммой пакетов** является диаграмма, содержащая пакеты классов и зависимости между ни ми.

Изображение выглядит как Прямоугольник, линия, текст, белый

Автоматически созданное описание - пакет

Изображение выглядит как текст, Шрифт, белый, снимок экрана

Автоматически созданное описаниеназвание класса

Изображение выглядит как линия, антенна, зарисовка, гаечный ключ

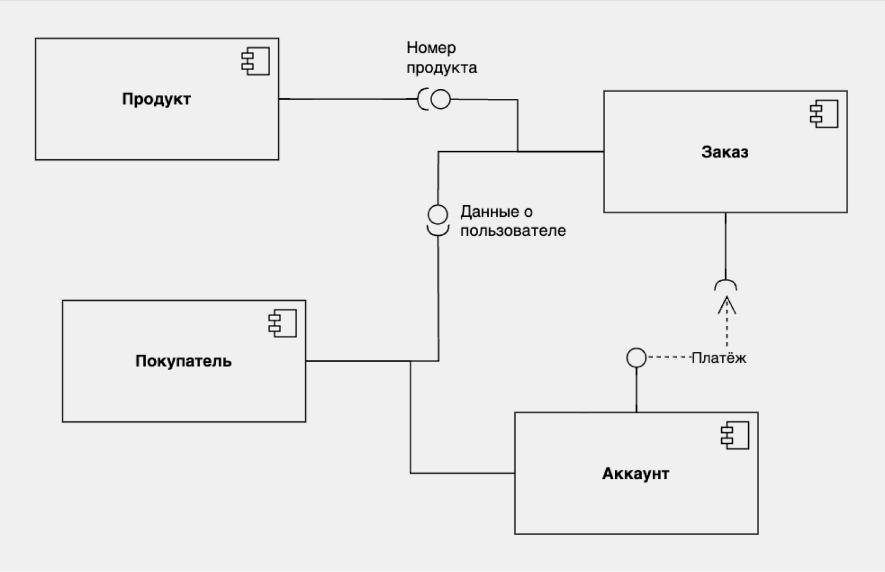
Автоматически созданное описание зависимости

 реализация пакетов

Изображение выглядит как текст, диаграмма, линия, Шрифт

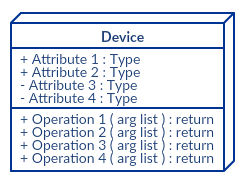
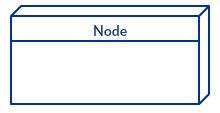
Автоматически созданное описание

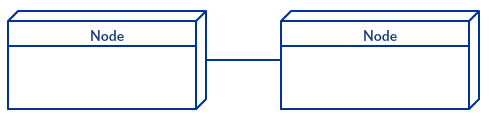
**Диаграмма компонентов**, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы.

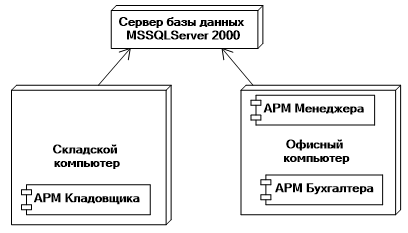


**50. Диаграмма развертывания и диаграмма прецедентов**

**Диаграмма развертывания** – это тип UML-диаграммы, которая показывает архитектуру исполнения системы.

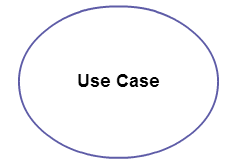
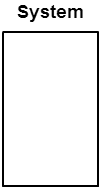


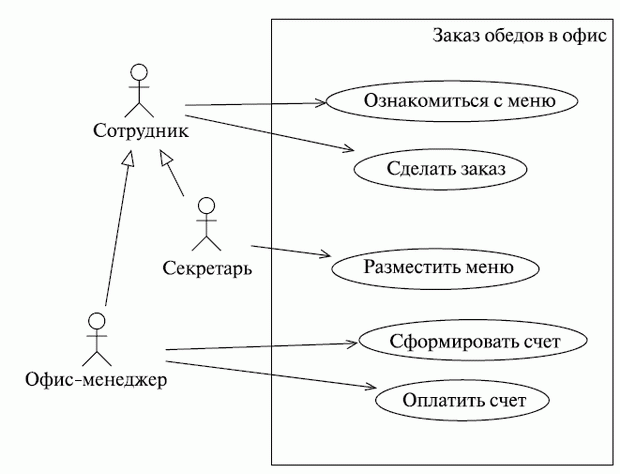
 коммуникационная ассоциация



**Диаграмма прецедентов (вариантов использования)** — это тип UML-диаграммы, отражающая отношения между акторами и прецедентами

**Прецедент** - последовательность взаимосвязанных операций, выполняющихся действующим субъектом (актор) в диалоге с системой для достижения некоторой цели.



**51. Классы.**

**Класс** - описание группы объектов со схожими ролями в системе.

**Класс** - структурный классификатор, который реализует классификаторы.

Замысел **Класса** стоит в том, чтобы определить классификацию объектов и определить особенности (Feature), которые характеризуют структуру и поведение этих объектов.

* **Operation (Операция)** - поведенческая характеристика интерфейса, типа данных или класса.
* **Property (Свойство)** - атрибут классификатора или полюс ассоциации
* **Reception (Прием)** — это декларация того, что классификатор готов отреагировать на получение сигнала

**52. Спецификация атрибута и операции. Отношения между классами: ассоциация, агрегация, композиция, обобщение, зависимость.**

**Атрибут (свойство)** – это именованное свойство класса, описывающее диапазон значений, которые может принимать экземпляр класса.

**Операция** - поведенческая особенность класса, которую можно напрямую вызывать в экземплярах этого класса.

* **Ассоциация** - взаимосвязь между классификаторами, которая используется, чтобы показать, что экземпляры классификаторов могут быть либо связаны друг с другом, либо объединены логически или физически в некоторую агрегацию. Самое слабое отношение между классами.
* **Агрегация** - форма ассоциации, описывающая отношение часть-целое.
* **Композиция** - форма агрегации, при которой части принадлежат целому, причем время жизни частей совладает со временем жизни композита. Часть может принадлежать только одному композиту
* **Обобщение** – это наследование. Каждый дочерний класс может использоваться вместо родительского, но не наоборот
* **Зависимость** - отношение сервера (поставщика) / клиента между элементами модели, где модификация поставщика может повлиять на элементы модели клиента

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

**53. Типизированные классы, квалификаторы. Ограничения.**

**Обобщенный (типизированный) класс** — это обычный класс C#, который дополнительно использует обобщенный тип T заключенный в угловые скобки <T>.

**Квалификаторы** - одно из зарезервированных слов **const**, **volatile** в языках программирования семейства Си.

**const -** объявляет объект как неизменяемый

**volatile** - сообщает компилятору, что значение переменной может быть изменено средствами, заданными в программе неявным образом

**Ограничения** сообщают компилятору о характеристиках, которые должен иметь аргумент типа.

**54. Классы ассоциаций.**

Класс- ассоциация - это ассоциация, которая в то же время является и классом. У класса-ассоциации присутствуют как свойства класса, так и свойства ассоциации. Экземплярами класса-ассоциации являются связи, у которых есть не только ссылки на объекты, но и значения атрибутов.

Класс-ассоциация используется в том случае, когда у каждой связи должны быть свои собственные значения атрибутов, операции или ссылки на объекты.

Класс-ассоциация не может иметь ассоциацию сам с собой.

Класс-ассоциация изображается с помощью символа класса (прямоугольника), который соединяется пунктирной линией с маршрутом ассоциации

*Ассоциация* не может иметь атрибутов, но во многих случаях это крайне желательно. Например, если студент связан ассоциацией "*многие-ко-многим*" с курсом, то этой ассоциации целесообразно иметь *атрибут* под названием "оценка". Это достигается связыванием с ассоциацией специального класса - класса-ассоциации (*association class*), в котором и указываются все нужные атрибуты, как показано

Изображение выглядит как текст, диаграмма, линия, снимок экрана

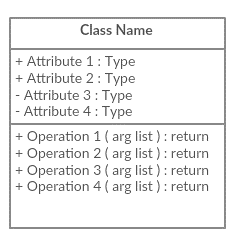
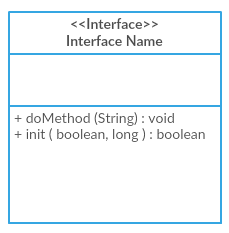
Автоматически созданное описание

**55. Диаграммы классов предметной области и программной системы. Обязанности классов.**

**Диаграмма классов**предназначена для представления внутренней структуры программы в виде классов и связей между ними.

**Класс** - описание группы объектов со схожими ролями в системе.

* Структурные особенности (атрибуты) определяют, что объекты класса знают (know)
  + Состояние объекта системы, описание структурных или статических особенностей класса
* Поведенческие особенности (операции) определяют, что могут делать (can do) объекты класса

 Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

**Обязанность** – это соглашение или обязательства класса. Когда вы создаете класс, выдвигается предположение, что все его объекты характеризуются одинаковым состоянием и одинаковым поведением.

**56. Диаграммы последовательности и коммуникаций.**

**Диаграммы коммуникации и последовательности** транзитивны, выражают взаимодействие, но показывают его различными способами и могут быть преобразованы одна в другую.

* **Диаграмма последовательности**: акцент на обмене сообщениями между несколькими линиями жизни (последовательность сообщений отображается индикатором последовательности)

Изображение выглядит как текст, диаграмма, линия, Шрифт

Автоматически созданное описание

* **Диаграммы коммуникации** (связи) - акцент на обмене сообщений между несколькими линиями жизни (порядок отображается номером сообщения)

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

**57. Жизненный цикл разработки информационной системы**

Жизненный цикл

* **жизненный цикл (life cycle)**: развитие системы, продукта, услуги, проекта или других изготовленных человеком объектов, начиная со стадии разработки концепции и заканчивая прекращением применения
* **стадии жизненного цикла (stage)**: период в пределах жизненного цикла некоторого объекта, который относится к состоянию его описания или реализации
  + стадии относятся к основному развитию и достижению контрольных точек этим объектом в течении его жизненного цикла
  + стадии могут быть взаимно перекрывающимися

Традиционные основные этапы ЖЦ ПО:

* анализ требований;
* проектирование;
* кодирование (программирование);
* тестирование и отладка;
* эксплуатация и сопровождение.

Жизненный цикл информационных систем регламентирует международный стандарт ISO/IEC 12207-2010.

**58. Итерации и модель жизненного цикла.**

**Итерации -** последовательность работ в рамках утвержденного плана, приводящая к созданию работоспособного варианта ИС или ПО

**Модель жизненного цикла (life cycle model)** - структура процессов и действий, связанных с жизненным циклом, организуемых в стадии, которые также служат в качестве общей ссылки для установления связей и взаимопонимания сторон

**59. Agile - гибкая методологий разработки.**

В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике.

Использовать:

* Когда потребности пользователей постоянно меняются в динамическом бизнесе.
* Изменения на Agile реализуются за меньшую цену из-за частых инкрементов.
* В отличие от модели водопада, в гибкой модели для старта проекта достаточно лишь небольшого планирования.

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Автоматически созданное описание

Экзаменационный билет предполагает создание диаграмм UML, таких как, диаграмма вариантов использования, диаграммы классов, диаграммы взаимодействия, для выбранной предметной области, например:

1. Ресторан быстрого питания. Предполагается что цена на товары будет изменяться динамически, в зависимости от количества остававшихся ингредиентов. Чем меньше осталось – тем дороже покупка.

2. Служба заказа такси. Цена заказа зависит от количества свободных водителей в указанном районе, и количества заказов.

3. Продуктовый магазин. Цена на товар в магазине дешевеет по мере того как к концу приближается срок годности.

4. Служба курьерской доставки. Заказы должны распределяться по курьерам таким образом, чтобы путь движения был минимальным, но при этом доставка до всех пунктов укладывалась в требуемые сроки.

5. Магазин. Система должна анализировать сезонный спрос, и на основе исторических данных корректировать цену на товар.

6. Интернет магазин. Система должна анализировать какие товары покупают совместно, и предлагать пользователю их покупку при выборе одного из товаров.

7. Сеть аптек. Система должна анализировать сезонный спрос на лекарства, и рассчитывать предполагаемый объём продаж.

8. Сеть аптек. Для препаратов безрецептурного отпуска, система предлагает покупателю аналоги.

9. Сеть аптек. Система просматривает список покупаемых лекарств, и выдаёт предупреждение, если действующие вещества несовместимы.

10. Регулировка пассажиропотока на общественном (автобусном) транспорте. Система должна, на основании количества купленных билетов и времени покупок, регулировать количество автотранспорта на маршруте, путём изменения интервала выезда транспорта на маршрут.

11. Столовая. Система должна анализировать какие блюда чаще всего покупают совместно, и предлагать покупателям соответствующие, комплексные обеды.

Допускается задавать дополнительные теоретические вопросы по теме курса, например:

<https://habr.com/ru/articles/92570/>

https://habr.com/ru/companies/otus/articles/505852/

1. Опишите шаблоны GRASP

2. В чём заключается шаблон Controller и какие проблемы он решает. Покажите пример использования данного шаблона

3. В чём заключается шаблон Low Coupling и какие проблемы он решает. Покажите пример использования данного шаблона

4. В чём заключается шаблон High Cohesion и какие проблемы он решает. Покажите пример использования данного шаблона

5. В чём заключается шаблон Creator и какие проблемы он решает. Покажите пример использования данного шаблона

6. В чём заключается шаблон Information Expert и какие проблемы он решает. Покажите пример использования данного шаблона

7. Напишите отличия доменной модели от программной модели.

8. Шаблон Layers

9. Логическая архитектура приложения