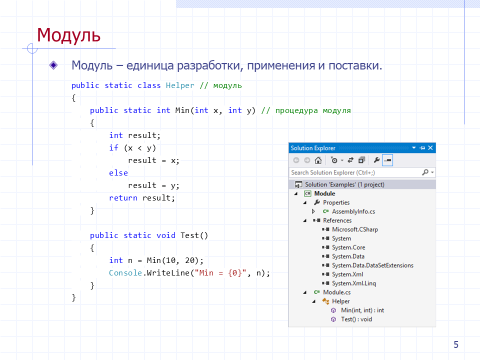
**Примеры вопросов к экзамену по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования», часть 1**

**1. Объект как динамический модуль. Понятие модуля. Принципы модульного программирования. Разграничение доступа к элементам модуля. Термин «инкапсуляция». Понятие объекта как динамического модуля. Понятие класса как описания динамического модуля.**

Понятие модуля:



Модуль в программировании представляет собой функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом, предназначенный для использования в других программах. Модули позволяют разбивать сложные задачи на более мелкие в соответствии с принципом модульности.

Принципы модульного программирования:

Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

Основные концепции модульного программирования:

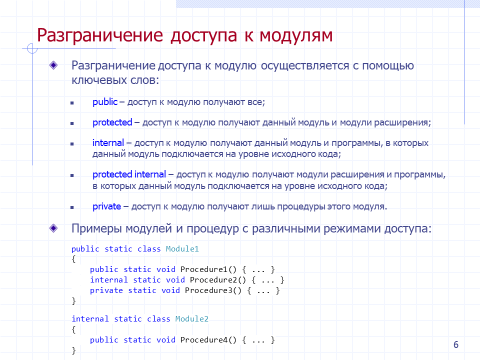
-каждый модуль имеет единственную точку входа и выхода;

-размер модуля по возможности должен быть минимизирован;

-вся система построена из модулей;

-каждый модуль не зависит от того, как реализованы другие модули.

Разграничение доступа к элементам модуля:



Термин «инкапсуляция»:

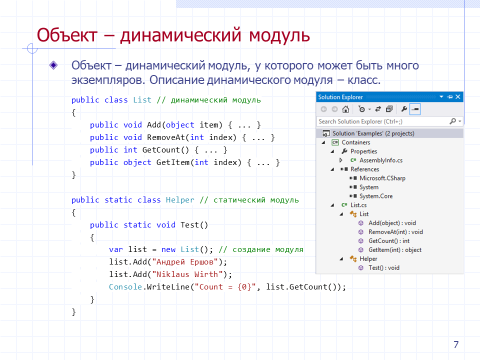
Инкапсуля́ция — свойство языка программирования, позволяющее объединить и защитить данные и код в объектe и скрыть реализацию объекта от пользователя (прикладного программиста). При этом пользователю предоставляется только спецификация (интерфейс) объекта. Пользователь может взаимодействовать с объектом только через этот интерфейс. Сокрытие реализации целесообразно применять в следующих случаях:

-предельная локализация изменений при необходимости таких изменений;

-прогнозируемость изменений (какие изменения в коде надо сделать для заданного изменения функциональности) и прогнозируемость последствий изменений.

Понятие объекта как динамического модуля,

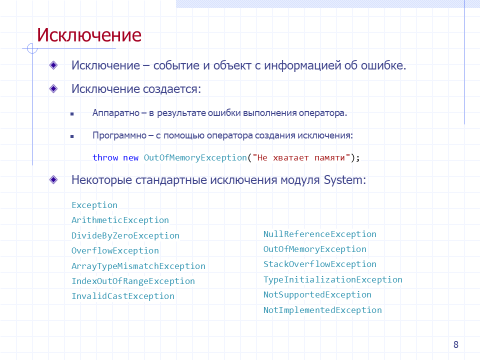
Понятие класса как описания динамического модуля:

****

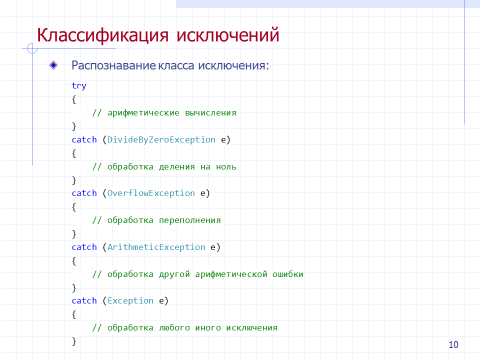
**2. Исключение. Понятия ошибки и исключения. Создание исключений. Классы исключений. Обработка исключений. Защита ресурсов от утечки в случае исключений. Приемы надежного программирования.**

Понятия ошибки и исключения,

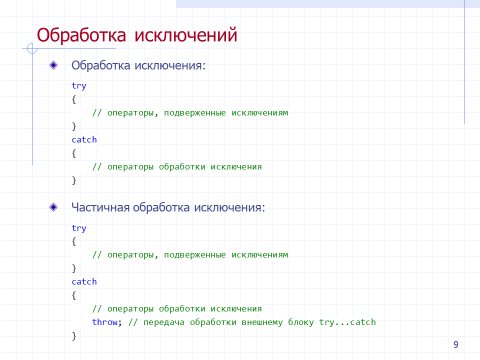
Создание исключений:



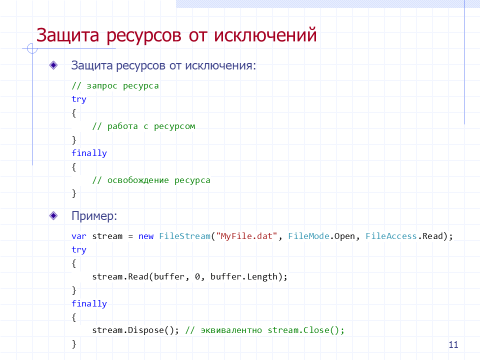
**Классы исключений:**

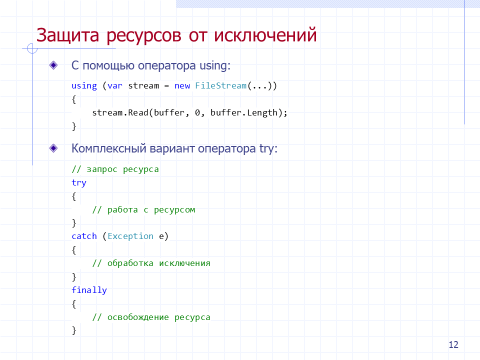


**Обработка исключений:**



**Защита ресурсов от утечки в случае исключений:**



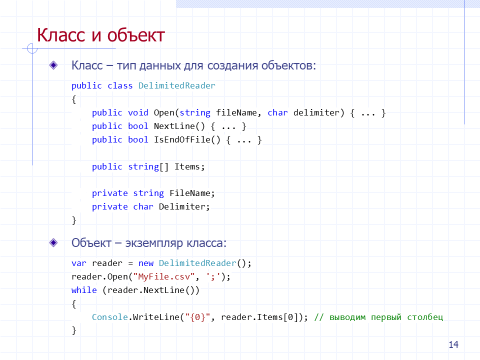


**3. Класс и объект. Понятие класса. Создание объекта. Понятие метода. Представление метода в виде обычной процедуры. Понятие конструктора и деструктора.**

**Класс и объект,**

**Понятие класса,**

**Создание объекта:**

****

Каждый объект всегда принадлежит некоторому классу объектов. Класс объектов — это

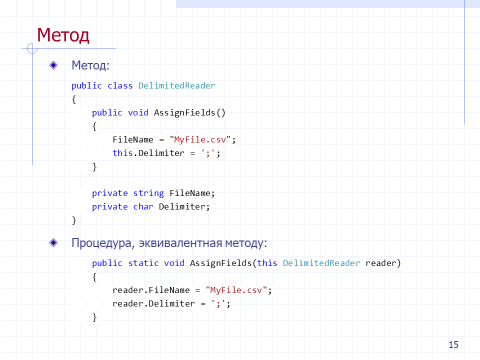
обобщенное (абстрактное) описание множества однотипных объектов. Объекты являются

конкретными представителями своего класса, их принято называть экземплярами класса.

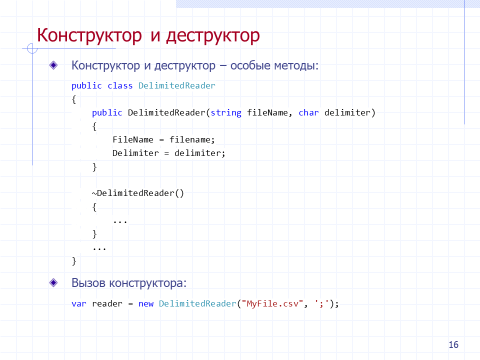
Например, класс СОБАКИ — понятие абстрактное, а экземпляр этого класса МОЙ ПЕС

БОБИК — понятие конкретное.

**Понятие метода. Представление метода в виде обычной процедуры:**

****

**Понятие конструктора и деструктора.**

****

Особой разновидностью методов являются конструкторы и деструкторы. Напомним, что

конструкторы создают, а деструкторы разрушают объекты. Создание объекта включает

выделение памяти под экземпляр и инициализацию его полей, а разрушение — очистку

полей и освобождение памяти. Действия по инициализации и очистке полей специфичны для каждого конкретного класса объектов. По этой причине язык Delphi позволяет

переопределить стандартный конструктор Create и стандартный деструктор Destroyдля

выполнения любых полезных действий.

1. **Класс и объект. Понятие класса. Создание объекта. Понятие метода. Представление метода в виде обычной процедуры. Понятие конструктора и деструктора.**
2. **Свойство. Понятие свойства как виртуального поля. Назначение свойства. Методы получения и установки значений свойства. Режимы доступа к свойству. Свойство-индексатор или свойство-массив (в зависимости от языка программирования).**
3. **Расширение класса. Расширение класса путем создания производного класса. Термин «наследование». Существование «прародителя» всех классов. Перекрытие элементов класса в производных классах. Совместимость объектов различных классов. Контроль и преобразование типов.**
4. **Виртуальный метод и виртуальное свойство. Понятие виртуального метода. Перекрытие виртуального метода в производном классе. Абстрактный виртуальный метод. Механизм вызова виртуального метода. Динамический виртуальный метод (в некоторых языках программирования). Метод обработки сообщений (в некоторых языках программирования). Термин «полиморфизм». Понятие виртуального свойства. Запрет на расширение класса.**
5. **Делегат. Понятие делегата (ссылки на метод объекта – в зависимости от языка программирования). Пример применения делегата. Понятие события как списка делегатов. Пример применения события. Методы регистрации события.**
6. **Метакласс. Понятие метакласса (в некоторых языках программирования). Методы, применяемые к классам. Виртуальные конструкторы (в некоторых языках).**
7. **Интерфейс. Понятие интерфейса. Описание интерфейса. Поддержка интерфейса классом. Механизм подсчета ссылок в интерфейсах. Расширение интерфейса. Глобально-уникальный идентификатор интерфейса. Совместимость интерфейсов и классов. Получение интерфейса через другой интерфейс. Представление интерфейса в памяти. Механизм вызова метода объекта через интерфейс.**
8. **Шаблон. Понятие шаблона как параметризованного класса. Создание переменной на базе шаблонного типа. Представление данных, заданных шаблоном, в памяти. Параметры шаблона. Установка ограничений на параметры шаблона. Шаблон делегата.**
9. **Атрибут. Информация о классах во время выполнения программы. Понятие «рефлексии» объектов. Понятие атрибута (аннотации – в другом языке программирования) как механизма рефлексии. Создание пользовательского атрибута. Проверка атрибута во время выполнения программы. Пример применения атрибутов.**
10. **Анонимная функция. Варианты анонимной функции. Шаблон делегата-функции. Представление анонимной функции в виде данных. Ограничения на выражения внутри анонимной функции.**
11. **Деление типов данных на типы-«значения» (value-types) и типы-«ссылки» (reference-types). Отличие структур (записей) от классов на платформе .NET. Автоматическое управление памятью ссылочных данных. Упаковка и разупаковка данных. Типы данных со значением null.**
12. **Объектно-ориентированное программирование в языке Java. Модуль, класс, объект, метод, атрибуты доступа к элементам объекта, интерфейс, шаблон, аннотация. Отличия средств объектно-ориентированного программирования в языках C# и Java.**
13. **Визуальное компонентное и сборочное программирование. Понятие компонента. Понятие визуального программирования. Инструментальные средства визуального компонентного программирования. Современные библиотеки компонентов. Понятие расширяемой программы и расширяемого программирования. Принципы расширяемого программирования. Применение методологии объектно-ориентированного программирования для построения расширяемых программ.**
14. **Принципы расширяемого программирования, известные как SOLID (Single responsibility, Open/closed, Liskov substitution, Interface segregation, Dependency inversion) и SOA (Service-Oriented Architecture).**
15. **Приемы объектно-ориентированного программирования. Итератор.**
16. **Приемы объектно-ориентированного программирования. Одиночка.**
17. **Приемы объектно-ориентированного программирования. Заместитель.**
18. **Приемы объектно-ориентированного программирования. Компоновщик.**
19. **Приемы объектно-ориентированного программирования. Мост.**
20. **Приемы объектно-ориентированного программирования. Наблюдатель.**
21. **Приемы объектно-ориентированного программирования. Посетитель.**
22. **Приемы объектно-ориентированного программирования. Фабричный метод. Фабрика классов.**
23. **Приемы объектно-ориентированного программирования. Пул объектов.**
24. **Имитация модульного программирования в языке C++. Имитация модульного программирования в языке C++. Пространство имен.**
25. **Классы и объекты в языке C++. Классы в языке C++. Наследование. Конструкторы и деструкторы. Стандартные конструкторы. Создание объектов по значению (на стеке) и по ссылке (в динамической памяти). Операторы new и delete. Размещающий оператор new. Порядок конструирования и разрушения объектов. Вложенные определения классов. «Друзья» класса. Статические члены класса.**
26. **Исключения в языке C++. Обработка исключений ситуаций в языке C++. Защита от утечки ресурсов. Имитация оператора try-finally. Понятие автоматического указателя (auto\_ptr). Использование автоматических указателей для защиты от утечки ресурсов.**
27. **Множественное наследование в языке C++. Множественное наследование. Проблема повторяющихся базовых классов. Замена множественного наследования наследованием от интерфейсов в других языках объектно-ориентированного программирования.**
28. **Виртуальные методы в языке C++. Константные методы. Виртуальные методы в языке C++. Недостатки синтаксиса определения и перекрытия виртуальных методов в языке C++. Понятие константного метода. Проблемы, порождаемые наличием константных методов.**
29. **Ссылки в языке C++. Рекомендации по работе со ссылками. Типичные ошибки при работе со ссылками.**
30. **Операторы приведения типа в языке C++: const\_cast, reinterpret\_cast, static\_cast, dynamic\_cast. Перегрузка операторов в языке C++. Перегрузка бинарных операторов. Перегрузка унарных операторов. Перегрузка операторов преобразования типа.**
31. **Шаблоны функций и классов в языке C++. Шаблоны функций в языке C++. Шаблоны классов в языке C++. Специализации шаблонов классов. Проблемы шаблонов в языке C++.**
32. **Сравнение языков объектно-ориентированного программирования: C#, Java, C++.**