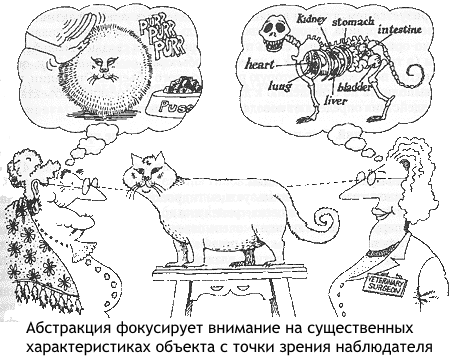
1. Составные части объектного подхода: абстрагирование, инкапсуляция,

модульность, иерархия.

Объектная модель имеет четыре главных элемента [1]:

• абстрагирование - это механизм, который выделяет существенные характеристики некоторого объекта, отличающие его от других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя ИЛИ это процесс "необращения внимания" (отбрасывания лишнего) на несущественные свойства изучаемого предмета, на данном уровне абстракции, и выделения нужных.



• инкапсуляция - это механизм языка, позволяющий ограничить прямой доступ к полям и методам класса извне (другим классам), с целью запретить бесконтрольную модификацию состояния объекта или вызов методов, которые также могут изменить его состояние.

• модульность - суть модульного программирования состоит в разбиении сложной задачи на некоторое число более простых подзадач и составлении программ для решения достаточно независимо друг от друга.

• иерархия - это упорядочение абстракций, расположение их по уровням это упорядочение абстракций. Это позволяет представлять отношения "родитель-потомок" между элементами.

2. Составные части объектного подхода: типизация, сохраняемость. Параллелизм.

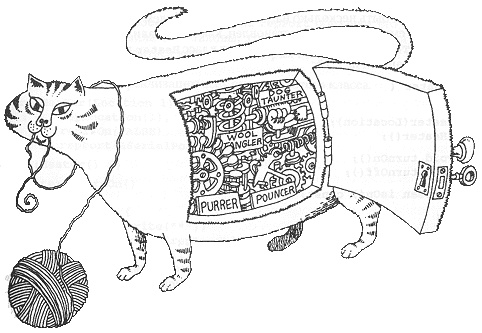
* Типизация - это то, как языки программирования работают с данными разного типа: строками, числами, объектами и другими.
* Сохраняемость - способность объекта существовать во времени независимо от породившего его процесса и/или в пространстве, возможно перемещаясь из своего первоначального адресного пространства
* Параллелизм - относится к концепции выполнения нескольких задач или операций одновременно. Это позволяет увеличить производительность программы за счет эффективного использования ресурсов процессора.

3. Абстракция и инкапсуляция как базовые понятия объектной модели. Реализация

объектного типа: правила защиты, инициализация, свойства класса.

* абстракция - по своей сути это процесс выделения общих характеристик и функциональности объектов или системы, игнорируя детали реализации.

• инкапсуляция - это механизм языка, позволяющий ограничить прямой доступ к полям и методам класса извне (другим классам), с целью запретить бесконтрольную модификацию состояния объекта или вызов методов, которые также могут изменить его состояние.



**Инициализация** - это присвоение начального значения объекту данных или переменной.

**Метки прав доступа;**

1. Private – метод или свойство доступны только в том классе, в котором они присутствуют.
2. protected – полям имеют доступ наследники класса.
3. Public - полям имеют доступ все.

**Свойства класса** - характеристики объекта и его состояние

4. Модульность как базовое понятие объектной модели. Принципы модульного

стиля.

**Модульное программирование** – это не просто способ упростить разработку программ, это фундаментальный принцип, который делает код более понятным, надежным и легко модифицируемым.

Концепцию модульного программирования можно сформулировать в виде нескольких положений:

* Большие задачи разделять на подзадачи
* Подзадачи должны быть сформирования так, чтобы можно было составить из них не только одну изначальную задачу, а несколько вариаций для использования в других частях программы, те подзадачи должны быть самоценны
* Код модуля должен быть понятен
* Ошибки, которые есть в модуле, не должны влиять на другие модули

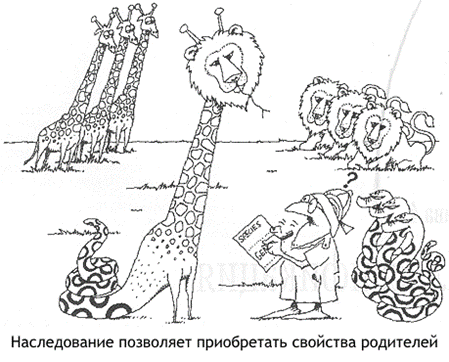
В ООП модули часто представлены в виде классов – шаблонов для создания объектов, которые инкапсулируют данные и функции, работающие с этими данными.

Классы позволяют разработчикам создавать сложные системы, разделенные на понятные, управляемые части.

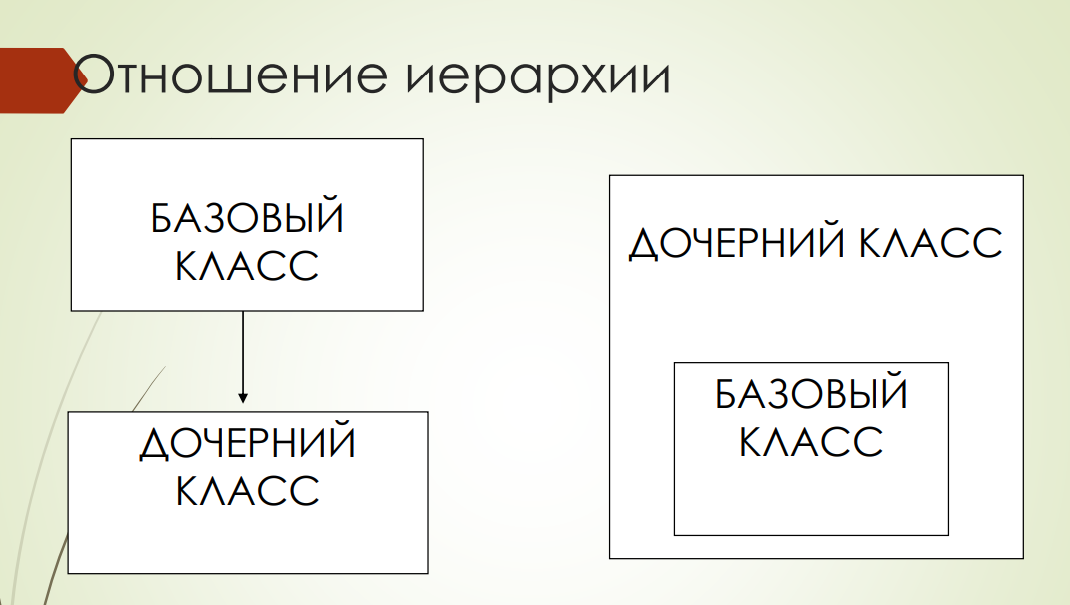
5. Иерархия как базовое понятие объектной модели. Иерархия «is а» и отношение «part of». Реализация механизма наследования: правила защиты, особенности конструкторов производных классов.

**иерархия** - это упорядочение абстракций, расположение их по уровням это упорядочение абстракций. Это позволяет представлять отношения "родитель-потомок" между элементами.

Наследование, один из основополагающих принципов ООП, это могучий механизм, который позволяет представить логику прикладной задачи в виде совокупности объектов, связанных друг с другом отношениями.



1. Первый из видов наследования моделирует иерархию, или тип отношения **«является» или «is a»** между двумя объектами. Иерархии классифицируют объекты по признакам «от общего к частному», или представляют прогрессию с течением времени. Иерархия подразумевает создание дочерних объектов путём сохранения свойств и поведения базовых объектов, и расширения их в дочерних. Позволяет структурировать и повторно использовать код

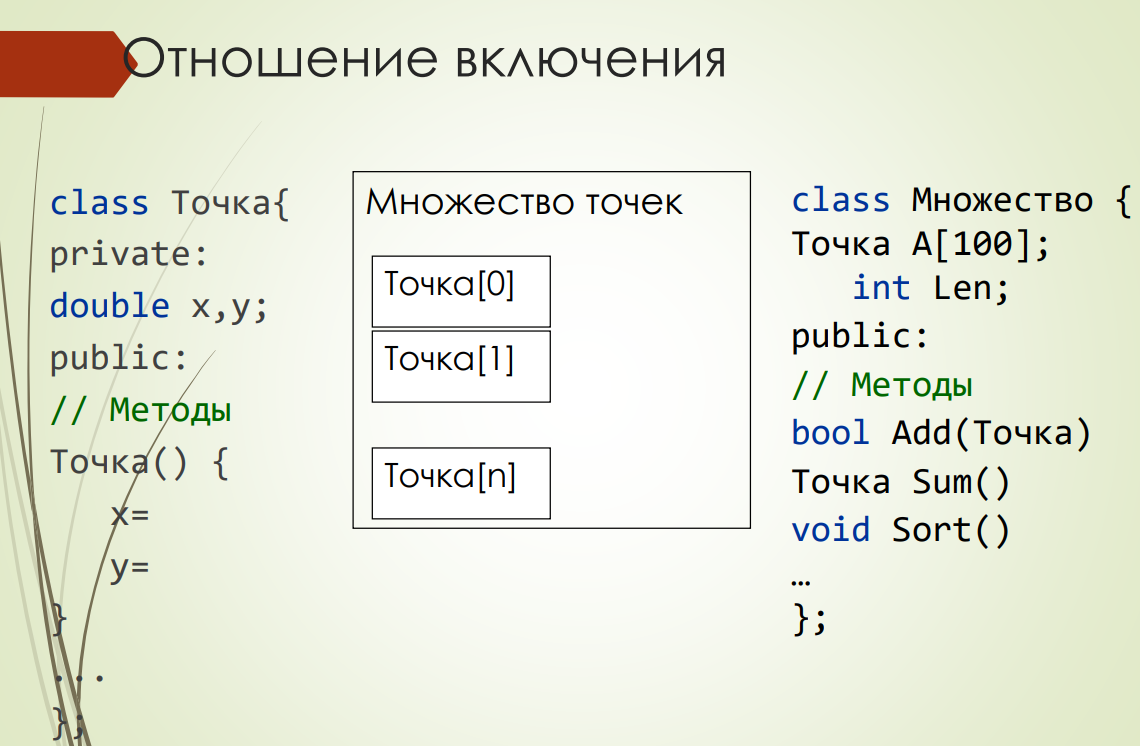


1. Второй вид наследования моделирует композицию, или тип отношения **«имеет» или «part of»** между двумя объектами, внутренние данные одного класса (контейнера) включают в себя объекты другого класса, или объект содержится в другом объекте.

Пример.

«Ящик с игрушками» содержит множество

объектов «Игрушка».



**Метки прав доступа;**

1. Private – метод или свойство доступны только в том классе, в котором они присутствуют.
2. protected – полям имеют доступ наследники класса.
3. Public - полям имеют доступ все.

Конструктор производного класса в первую очередь всегда должен вызывать конструктор базового класса. Если это действие не выполняется явно, то по умолчанию вызывается конструктор без параметров (если он есть, если его нет, будет ошибка). Если класс имеет несколько базовых, то конструкторы базовых классов должны вызываться в порядке перечисления этих классов в списке базовых.

Конструкторы классов позволяют правильно инициализировать объекты иерархической модели в соответствии с основным принципом наследования

6. Иерархия как базовое понятие объектной модели. Отношение «part of». Особенности контейнерных классов. Коллекции как пример объектной реализации абстрактных типов данных.

Второй вид наследования моделирует композицию, или тип отношения **«имеет» или «part of»** внутренние данные одного класса (контейнера) включают в себя объекты другого класса, или объект содержится в другом объекте.

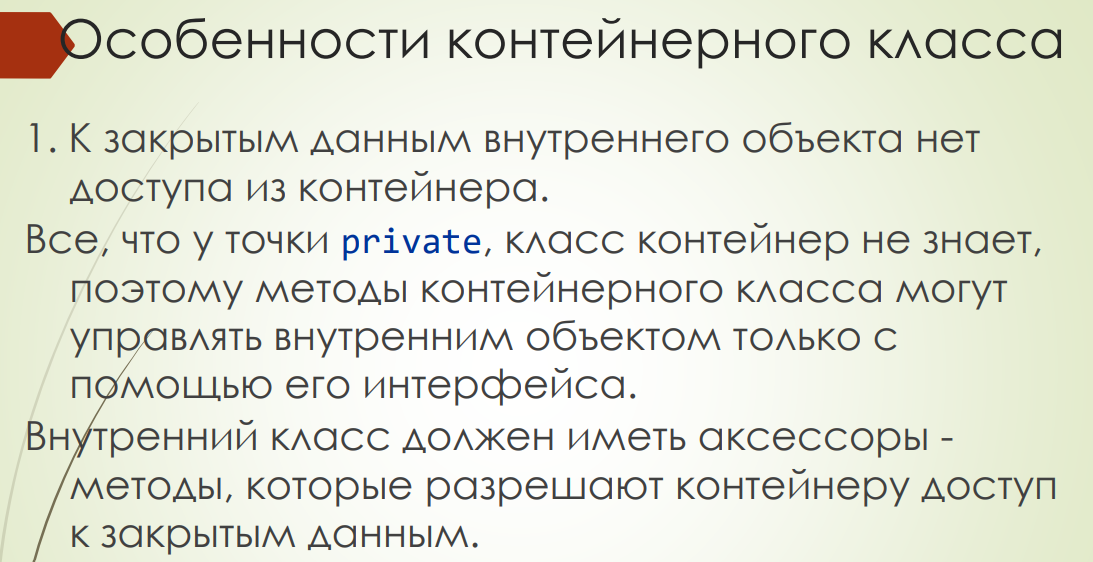
Примеры.

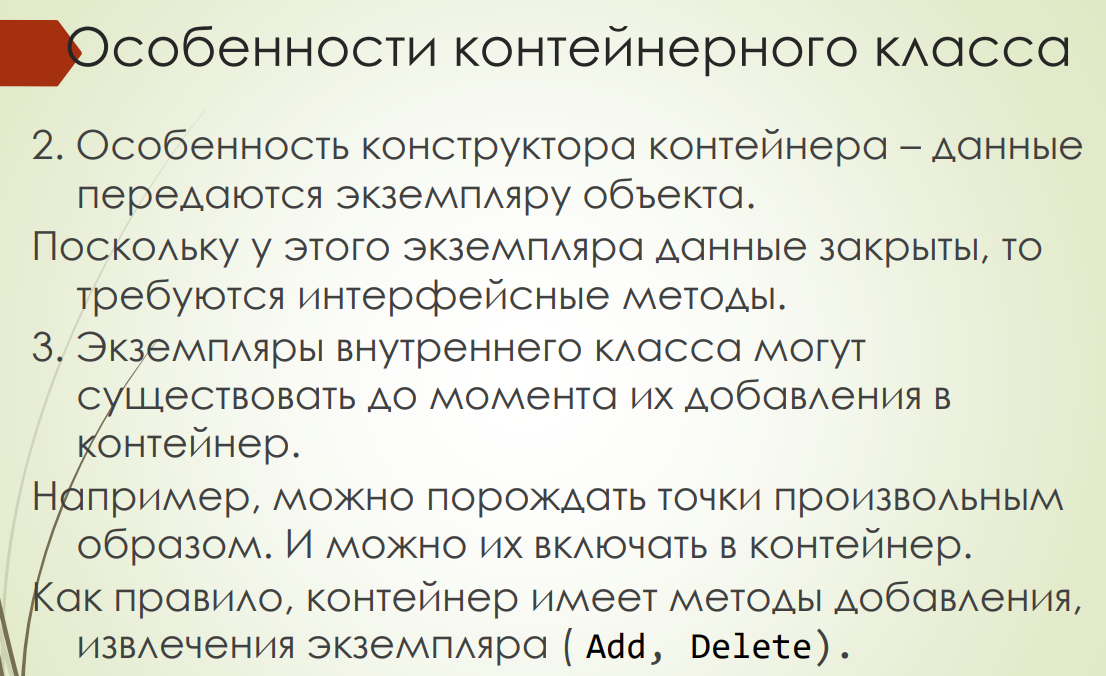
«Ящик с игрушками» содержит множество объектов «Игрушка»

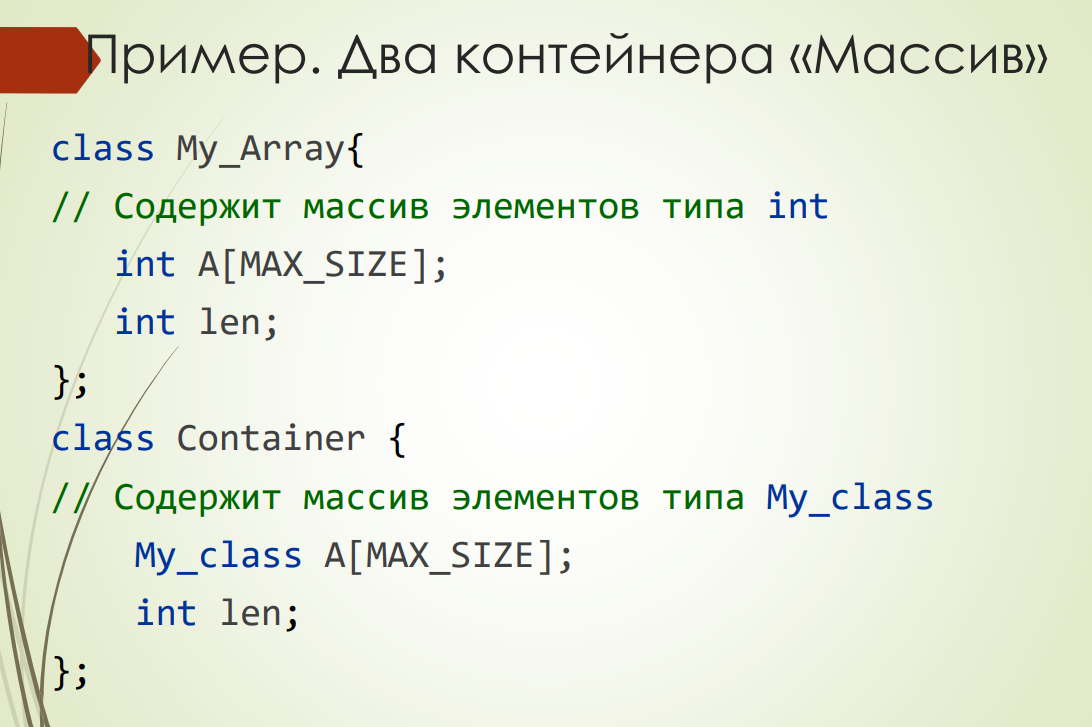
**иерархия** - это упорядочение абстракций, расположение их по уровням это упорядочение абстракций, расположение их по уровням

**Наследование**, один из основополагающих принципов ООП, это могучий механизм, который позволяет представить логику прикладной задачи в виде совокупности объектов, связанных друг с другом отношениями.

**Контейнерные классы** — это классы, которые могут хранить в себе элементы различных типов данных.







**Коллекции** - Коллекции в программировании представляют собой структуры данных, которые используются для хранения и управления группой элементов. Они позволяют программистам организовывать и работать с большими объемами данных более эффективно. Коллекции могут быть различных типов, включая списки, массивы, множества, карты и другие.

7. Типизация, полиморфизм как базовые понятия объектной модели. Абстрактные

классы. Раннее и позднее связывание.

* Типизация - это то, как языки программирования работают с данными разного типа: строками, числами, объектами и другими.

**Полиморфизм** — концепция, позволяющая создавать более гибкие, расширяемые и понимаемые программы.

**Полиморфизм** даёт возможность использовать одни и те же методы для объектов разных классов. Неважно, как эти объекты устроены, — в ООП можно сказать самолёту и квадрокоптеру: «Лети», и они будут делать это как умеют: квадрокоптер закрутит лопастями, а самолёт начнёт разгон по взлётно-посадочной полосе.

**Абстрактные классы в объектно-ориентированном программировании** — это базовые классы, которые можно наследовать, но нельзя реализовывать. То есть на их основе нельзя создать о Пример такого класса из жизни — животное. Рыбы, пауки и насекомые являются животными, но каждое из них также принадлежит своему подклассу с набором специфических свойств. При этом, просто «животных» в природе не существует объект. Для его объявления используют ключевое слово abstract

Имеются два термина, часто используемых, когда речь заходит об объектно-ориентированных языках программирования: раннее и позднее связывание. По отношению к С++ эти термины соответствуют событиям, которые возникают на этапе компиляции и на этапе исполнения программы соответственно.

В терминах объектно-ориентированного программирования **раннее связывание означает**, что объект и вызов функции связываются между собой на этапе компиляции. Это означает, что вся необходимая информация для того, чтобы определить, какая именно функция будет вызвана, известна на этапе компиляции программы. В качестве примеров раннего связывания можно указать стандартные вызовы функций, вызовы перегруженных функций и перегруженных операторов.

Позднее связывание означает, что объект связывается с вызовом функции только во время исполнения программы, а не раньше. Позднее связывание достигается в С++ с помощью использования виртуальных функций и производных классов.

Какое именно связывание должна использовать программа, зависит от предназначения программы.

8. Интерфейсы как инструмент: реализация, преимущества перед полиморфными

классами.

**Интерфейс** — это объявление, схожее с классом, но в нем нет реализаций методов,

т. е. все методы его абстрактны и не содержат реализации. С их помощью вы можете описать, какими функциями должен обладать класс и что он должен уметь.

Объявление интерфейса начинается со слова interface.

Вы также не можете указать модификаторы доступа для описываемых методов — все они считаются открытыми. В этом весь смысл интерфейса — описать действия, через которые мы потом сможем взаимодействовать с классами, которые реализуют интерфейс.

**Полиморфизм** - это создание дочерних классов, каждый из которых является разновидностью базового. Реализация интерфейса всего лишь означает, что класс умеет давать ответы на некоторый набор вопросов. При этом являться он может быть чем угодно.

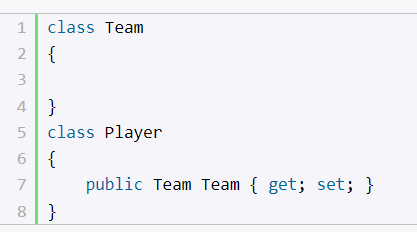
Например, класс офисной мебели может иметь подклассы - стулья и столы, каждый из которых является мебелью. При этом любой предмет в офисе должен реализовать интерфейс инвентаризации - выдавать свой инвентарный номер, чем бы он ни являлся.

9. Отношения между объектами. Типы отношений: ассоциация, наследование,

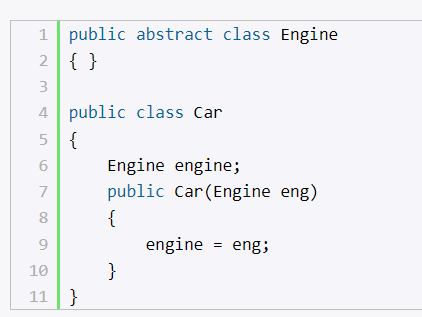
агрегация, использование.

Классы должны знать друг о друге, для того чтобы взаимодействовать между собой и сообща выполнить поставленную задачу.

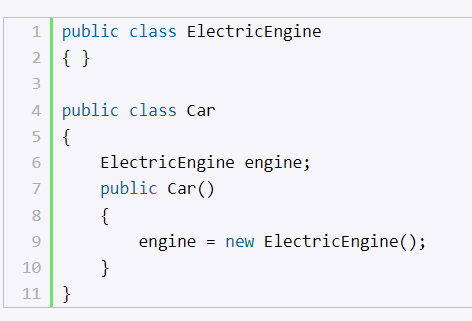
* **Наследование** является базовым принципом ООП и позволяет одному классу (наследнику) унаследовать функционал другого класса (родительского). Нередко отношения наследования еще называют генерализацией или обобщением. Наследование определяет отношение IS A, то есть "является".
* **Ассоциация** - это отношение, при котором объекты одного типа неким образом связаны с объектами другого типа. Например, объект одного типа содержит или использует объект другого типа. Например, игрок играет в определенной команде:



* Агрегация предполагает отношение HAS A. При агрегации реализуется слабая связь, то есть в данном случае объекты Car и Engine будут равноправны. В конструктор Car передается ссылка на уже имеющийся объект Engine. И, как правило, определяется ссылка не на конкретный класс, а на абстрактный класс или интерфейс, что увеличивает гибкость программы.



* Использование - Композиция определяет отношение HAS A, то есть отношение "имеет". Например, в класс автомобиля содержит объект класса электрического двигателя:



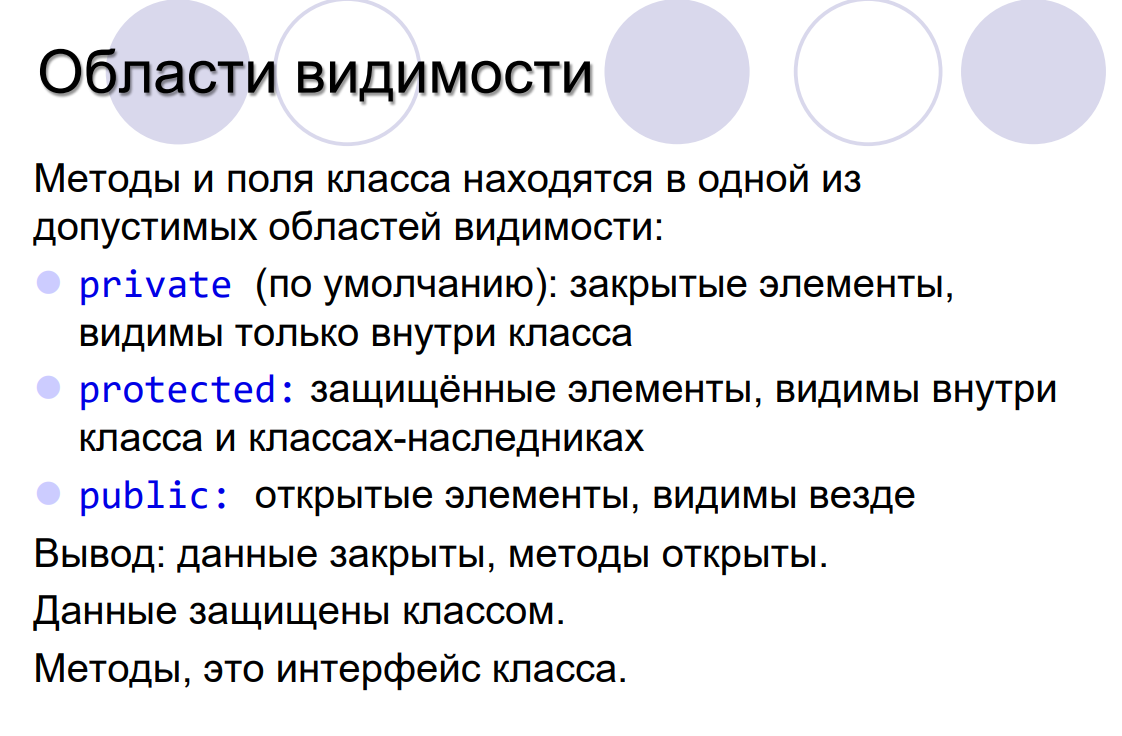
При этом класс автомобиля полностью управляет жизненным циклом объекта двигателя. При уничтожении объекта автомобиля в области памяти вместе с ним будет уничтожен и объект двигателя. И в этом плане объект автомобиля является главным, а объект двигателя - зависимой.

10. Использование объектных типов в составе проекта. Модульность в реализации

объектной модели. Область видимости имен.

Использование **объектных типов** в составе проекта программы является одним из основных принципов объектно-ориентированного программирования (ООП). Объекты в программировании представляют собой сущности, которые имеют свойства и методы, позволяющие эффективно управлять данными и взаимодействовать между собой. Они могут храниться в отдельном файле проекта и подключаться к основному файлу main

**модульность** - суть модульного программирования состоит в разбиении сложной задачи на некоторое число более простых подзадач и составлении программ для решения достаточно независимо друг от друга.



11. Роль классов и объектов в анализе и проектировании.

Роль классов и объектов в анализе и проектировании заключается в следующем:

1. На этапе анализа и ранних стадиях проектирования решаются две основные задачи:

* выявление классов и объектов, составляющих словарь предметной области;
* построение структур, обеспечивающих взаимодействие объектов, при котором выполняются требования задачи.

1. На ранних стадиях внимание проектировщика сосредоточивается на внешних проявлениях ключевых абстракций и механизмов. Такой подход создаёт логический каркас системы: структуры классов и объектов.
2. На последующих фазах проекта, включая реализацию, внимание переключается на внутреннее поведение ключевых абстракций и механизмов, а также их физическое представление.

Принимаемые в процессе проектирования решения задают архитектуру системы: и архитектуру процессов, и архитектуру модулей.