#### **Основные понятия ООП**

##### **Класс**

Представьте себе, что вы проектируете автомобиль. Вы знаете, что автомобиль должен содержать двигатель, подвеску, две передних фары, 4 колеса, и т.д. Ещё вы знаете, что ваш автомобиль должен иметь возможность набирать и сбавлять скорость, совершать поворот и двигаться задним ходом. И, что самое главное, вы точно знаете, как взаимодействует двигатель и колёса, согласно каким законам движется распредвал и коленвал, а также как устроены дифференциалы. Вы уверены в своих знаниях и начинаете проектирование.  
  
Вы описываете все запчасти, из которых состоит ваш автомобиль, а также то, каким образом эти запчасти взаимодействуют между собой. Кроме того, вы описываете, что должен сделать пользователь, чтобы машина затормозила, или включился дальний свет фар. Результатом вашей работы будет некоторый эскиз. Вы только что разработали то, что в ООП называется *класс*.  
  
**Класс** – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью (контракт).  
  
С точки зрения программирования класс можно рассматривать как набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов).  
  
С точки зрения структуры программы, класс является сложным типом данных.  
  
В нашем случае, класс будет отображать сущность – автомобиль. Атрибутами класса будут являться двигатель, подвеска, кузов, четыре колеса и т.д. Методами класса будет «открыть дверь», «нажать на педаль газа», а также «закачать порцию бензина из бензобака в двигатель». Первые два метода доступны для выполнения другим классам (в частности, классу «Водитель»). Последний описывает взаимодействия внутри класса и не доступен пользователю.  
  
*В дальнейшем, несмотря на то, что слово «пользователь» ассоциируется с пасьянсом «Косынка» и «Microsoft Word», мы будем называть пользователями тех программистов, которые используют ваш класс, включая вас самих. Человека, который является автором класса, мы будем называть разработчиком.*

##### **Объект**

Вы отлично потрудились и машины, разработанные по вашим чертежам, сходят с конвейера. Вот они, стоят ровными рядами на заводском дворе. Каждая из них точно повторяет ваши чертежи. Все системы взаимодействуют именно так, как вы спроектировали. Но каждая машина уникальна. Они все имеют номер кузова и двигателя, но все эти номера разные, автомобили различаются цветом, а некоторые даже имеют литьё вместо штампованных дисков. Эти автомобили, по сути, являются объектами вашего класса.  
  
**Объект (экземпляр)**– это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.  
  
Говоря простым языком, объект имеет конкретные значения атрибутов и методы, работающие с этими значениями на основе правил, заданных в классе. В данном примере, если класс – это некоторый абстрактный автомобиль из «мира идей», то объект – это конкретный автомобиль, стоящий у вас под окнами.

##### **Интерфейс**

Когда мы подходим к автомату с кофе или садимся за руль, мы начинаем взаимодействие с ними. Обычно, взаимодействие происходит с помощью некоторого набора элементов: щель для приёмки монеток, кнопка выбора напитка и отсек выдачи стакана в кофейном автомате; руль, педали, рычаг коробки переключения передач в автомобиле. Всегда существует некоторый ограниченный набор элементов управления, с которыми мы можем взаимодействовать.  
  
**Интерфейс** – это набор методов класса, доступных для использования другими классами.  
  
Очевидно, что интерфейсом класса будет являться набор всех его публичных методов в совокупности с набором публичных атрибутов. По сути, интерфейс специфицирует класс, чётко определяя все возможные действия над ним.  
Хорошим примером интерфейса может служить приборная панель автомобиля, которая позволяет вызвать такие методы, как увеличение скорости, торможение, поворот, переключение передач, включение фар, и т.п. То есть все действия, которые может осуществить другой класс (в нашем случае – водитель) при взаимодействии с автомобилем.  
  
При описании интерфейса класса очень важно соблюсти баланс между гибкостью и простотой. Класс с простым интерфейсом будет легко использовать, но будут существовать задачи, которые с помощью него решить будет не под силу. В то же время, если интерфейс будет гибким, то, скорее всего, он будет состоять из достаточно сложных методов с большим количеством параметров, которые будут позволять делать очень многое, но использование его будет сопряжено с большими сложностями и риском совершить ошибку, что-то перепутав.  
  
Примером простого интерфейса может служить машина с коробкой-автоматом. Освоить её управление очень быстро сможет любая блондинка, окончившая двухнедельные курсы вождения. С другой стороны, чтобы освоить управление современным пассажирским самолётом, необходимо несколько месяцев, а то и лет упорных тренировок. Не хотел бы я находиться на борту Боинга, которым управляет человек, имеющий двухнедельный лётный стаж. С другой стороны, вы никогда не заставите автомобиль подняться в воздух и перелететь из Москвы в Вашингтон.

#### **Инкапсуляция**

Представим на минутку, что мы оказались в конце позапрошлого века, когда Генри Форд ещё не придумал конвейер, а первые попытки создать автомобиль сталкивались с критикой властей по поводу того, что эти коптящие монстры загрязняют воздух и пугают лошадей. Представим, что для управления первым паровым автомобилем необходимо было знать, как устроен паровой котёл, постоянно подбрасывать уголь, следить за температурой, уровнем воды. При этом для поворота колёс использовать два рычага, каждый из которых поворачивает одно колесо в отдельности. Думаю, можно согласиться с тем, что вождение автомобиля того времени было весьма неудобным и трудным занятием.  
  
Теперь вернёмся в сегодняшний день к современным чудесам автопрома с коробкой-автоматом. На самом деле, по сути, ничего не изменилось. Бензонасос всё так же поставляет бензин в двигатель, дифференциалы обеспечивают поворот колёс на различающиеся углы, коленвал превращает поступательное движение поршня во вращательное движение колёс. Прогресс в другом. Сейчас все эти действия скрыты от пользователя и позволяют ему крутить руль и нажимать на педаль газа, не задумываясь, что в это время происходит с инжектором, дроссельной заслонкой и распредвалом. Именно сокрытие внутренних процессов, происходящих в автомобиле, позволяет эффективно его использовать даже тем, кто не является профессионалом-автомехаником с двадцатилетним стажем. Это сокрытие в ООП носит название инкапсуляции.  
  
**Инкапсуляция** – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали  
реализации от пользователя.  
  
Инкапсуляция неразрывно связана с понятием интерфейса класса. По сути, всё то, что не входит в интерфейс, инкапсулируется в классе.

#### **Абстракция**

Представьте, что водитель едет в автомобиле по оживлённому участку движения. Понятно, что в этот момент он не будет задумываться о химическом составе краски автомобиля, особенностях взаимодействия шестерён в коробке передач или влияния формы кузова на скорость (разве что, автомобиль стоит в глухой пробке и водителю абсолютно нечем заняться). Однако, руль, педали, указатель поворота (ну и, возможно, пепельницу) он будет использовать регулярно.  
  
**Абстрагирование** – это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, **абстракция** – это набор всех таких характеристик.  
  
Если бы для моделирования поведения автомобиля приходилось учитывать химический состав краски кузова и удельную теплоёмкость лампочки подсветки номеров, мы никогда бы не узнали, что такое NFS.

#### **Полиморфизм**

Любое обучение вождению не имело бы смысла, если бы человек, научившийся водить, скажем, ВАЗ 2106 не мог потом водить ВАЗ 2110 или BMW X3. С другой стороны, трудно представить человека, который смог бы нормально управлять автомобилем, в котором педаль газа находится левее педали тормоза, а вместо руля – джойстик.  
  
Всё дело в том, что основные элементы управления автомобиля имеют одну и ту же конструкцию и принцип действия. Водитель точно знает, что для того, чтобы повернуть налево, он должен повернуть руль, независимо от того, есть там гидроусилитель или нет.  
Если человеку надо доехать с работы до дома, то он сядет за руль автомобиля и будет выполнять одни и те же действия, независимо от того, какой именно тип автомобиля он использует. По сути, можно сказать, что все автомобили имеют один и тот же интерфейс, а водитель, абстрагируясь от сущности автомобиля, работает именно с этим интерфейсом. Если водителю предстоит ехать по немецкому автобану, он, вероятно выберет быстрый автомобиль с низкой посадкой, а если предстоит возвращаться из отдалённого маральника в Горном Алтае после дождя, скорее всего, будет выбран УАЗ с армейскими мостами. Но, независимо от того, каким образом будет реализовываться движение и внутреннее функционирование машины, интерфейс останется прежним.  
  
**Полиморфизм** – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.  
  
Например, если вы читаете данные из файла, то, очевидно, в классе, реализующем файловый поток, будет присутствовать метод похожий на следующий: *byte[] readBytes( int n );*  
Предположим теперь, что вам необходимо считывать те же данные из сокета. В классе, реализующем сокет, также будет присутствовать метод *readBytes*. Достаточно заменить в вашей системе объект одного класса на объект другого класса, и результат будет достигнут.  
  
При этом логика системы может быть реализована независимо от того, будут ли данные прочитаны из файла или получены по сети. Таким образом, мы абстрагируемся от конкретной специализации получения данных и работаем на уровне интерфейса. Единственное требование при этом – чтобы каждый используемый объект имел метод *readBytes*.

#### **Наследование**

Представим себя, на минуту, инженерами автомобильного завода. Нашей задачей является разработка современного автомобиля. У нас уже есть предыдущая модель, которая отлично зарекомендовала себя в течение многолетнего использования. Всё бы хорошо, но времена и технологии меняются, а наш современный завод должен стремиться повышать удобство и комфорт выпускаемой продукции и соответствовать современным стандартам.  
  
Нам необходимо выпустить целый модельный ряд автомобилей: седан, универсал и малолитражный хэтч-бэк. Очевидно, что мы не собираемся проектировать новый автомобиль с нуля, а, взяв за основу предыдущее поколение, внесём ряд конструктивных изменений. Например, добавим гидроусилитель руля и уменьшим зазоры между крыльями и крышкой капота, поставим противотуманные фонари. Кроме того, в каждой модели будет изменена форма кузова.  
  
Очевидно, что все три модификации будут иметь большинство свойств прежней модели (старый добрый двигатель 1970 года, непробиваемая ходовая часть, зарекомендовавшая себя отличным образом на отечественных дорогах, коробку передач и т.д.). При этом каждая из моделей будет реализовать некоторую новую функциональность или конструктивную особенность. В данном случае, мы имеем дело с наследованием.  
**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.  
  
Необходимо отметить, что производный класс полностью удовлетворяет спецификации родительского, однако может иметь дополнительную функциональность. С точки зрения интерфейсов, каждый производный класс полностью реализует интерфейс родительского класса. Обратное не верно.  
  
Действительно, в нашем примере мы могли бы произвести с новыми автомобилями все те же действия, что и со старым: увеличить или уменьшить скорость, повернуть, включить сигнал поворота. Однако, дополнительно у нас бы появилась возможность, например, включить противотуманные фонари.  
  
Отсутствие обратной совместимости означает, что мы не должны ожидать от старой модели корректной реакции на такие действия, как включения противотуманок (которых просто нет в данной модели).