# Лекция 6. Объектно-ориентированное программирование

**Объектно-ориентированное программирование –** это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования*.*

1) ООП использует в качестве базовых элементов ***объекты****;*

2) каждый объект является ***экземпляром*** какого-либо определенного ***класса****;*

3) классы организованы ***иерархически*** (***«is-а»***).

**Объектно-ориентированное проектирование** *–* это методология проектирования, соединяющая в себе процесс объектной ***декомпозиции*** и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

Объектная модель характеризуется следующими свойствами:

* абстрагирование;
* инкапсуляция;
* модульность;
* иерархия.

Абстрагирование является одним из основных способов решения сложных задач, суть которого заключается в нахождении **сходств** между группами объектов, ситуациями или процессами реального мира, и в принятии решений на основе этих сходств, отвлекаясь на время от имеющихся различий*.* Абстрагирование концентрирует внимание на внешних особенностях объекта и позволяет отделить самые существенные особенности поведения от несущественных.

|  |  |
| --- | --- |
| Вид  абстракции | Особенности |
| Абстракция сущности | Объект представляет собой полезную модель некой сущности в предметной области |
| Абстракция поведения | Объект состоит из обобщенного множества операций |
| Абстракция виртуальной машины | Объект группирует операции, которые либо вместе используются более высоким уровнем управления, либо сами используют некоторый набор операций более низкого уровня |
| Произвольная абстракция | Объект включает в себя набор операций, не имеющих друг с другом ничего общего |

Внешнее поведение объекта называется ***контрактом***. Контракт фиксирует все обязательства, которые объект-сервер имеет перед объектом-клиентом. Каждая операция, предусмотренная этим контрактом, однозначно определяется ее формальными параметрами и типом возвращаемого значения.

Полный набор операций, которые клиент может осуществлять над другим объектом, вместе с правильным порядком, в котором эти операции вызываются, называется ***протоколом****.* Протокол отражает все возможные способы, которыми объект может действовать или подвергаться воздействию.

|  |
| --- |
| http://vmk.ugatu.ac.ru/book/buch/pg057.gif |
| Рис. 1. Пример фокусировки абстракции на существенных с точки зрения наблюдателя характеристиках объекта |

**Инкапсуляция –** один из методов скрытия информации, то есть маскировкой всех внутренних деталей, не влияющих на внешнее поведение. Обычно скрываются и внутренняя структура объекта и реализация его методов. Инкапсуляция, таким образом, определяет четкие границы между различными абстракциями.

**Интерфейс** является реализацией понятия инкапсуляции в программировании и отражает внешнее поведение объекта, описывая абстракцию поведения всех объектов данного класса. Внутренняя *реализация* описывает представление этой абстракции и механизмы достижения желаемого поведения объекта. Принцип разделения интерфейса и реализации соответствует сути вещей: в интерфейсной части собрано все, что касается взаимодействия данного объекта с любыми другими объектами; реализация скрывает от других объектов все детали, не имеющие отношения к процессу взаимодействия объектов.

|  |
| --- |
| http://vmk.ugatu.ac.ru/book/buch/pg063.gif |
| Рис. 2. Инкапсуляция скрывает детали реализации объекта |

**Инкапсуляция** – это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации. Абстрагирование и инкапсуляция приводит программу в набор взаимодействующих модулей.

**Модульность**– это свойство системы, которая была разложена на внутренне связные, но слабо связанные между собой модули.

**Иерархия** *–* это упорядочение абстракций, расположение их по уровням.

Основными видами иерархических структур применительно к сложным системам являются структура классов (иерархия ***«is-a»***). **Наследование** – это отношение типа родитель/потомок, когда потомок заимствует структурную или функциональную часть родителя. При этом потомок может как достраивать, так и переписывать компоненты родителя. Эта особенность называется **полиморфизмом***:* одно и то же имя может означать объекты разных типов, но, имея общего предка, все они имеют и общее подмножество операций, которые можно над ними выполнять. Полиморфизм возникает там, где взаимодействуют наследование и динамическое связывание.

# Организация иерархии классов в Java

Наследование является неотъемлемой частью Java. При этом используется принцип «is-a»: новый класс похож на тот класс, которому он наследует.

В Java для реализации наследования используется ключевое слово **extends**, после которого указываете имя базового класса. Результатом наследования является получение доступа ко всем полям и методам базового класса. Используя наследование, можно создать общий класс, который определяет характеристики, общие для набора связанных элементов. Затем, после наследования от этого класса, появляется возможность добавить свои уникальные характеристики. Главный наследуемый класс в Java называют суперклассом или родителем. Наследующий класс называют подклассом или потомком. Иначе говоря, что подкласс – это специализированная версия суперкласса, которая наследует все члены суперкласса и добавляет свои собственные уникальные элементы.

|  |
| --- |
| [модификатор\_доступа] Имя\_класса extends Имя\_суперкласса  {  // тело класса-потомка  } |
| Формула 1. Организация наследования классов |

В Java не существует множественного наследования, соответственно, невозможно наследовать более чем одному классу. Сделано это во избежание ромбовидного наследования. Дело в том, что при наследовании одного класса другому, поля и методы родителя становится доступны потомку. В ситуации, когда возможно наследование двум и более классам, есть вероятность встретить методы, имеющие одинаковое название и список параметров и потому, невозможно предсказать, какой из них будет выбран во время исполнения.

|  |
| --- |
| **public** **class** Animal { // класс-родитель  **int** age;  **int** weight;  **int** hight;    **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** **int** getWeight() {  **return** weight;  }  **public** **void** setWeight(**int** weight) {  **this**.weight = weight;  }  **public** **int** getHight() {  **return** hight;  }  **public** **void** setHight(**int** hight) {  **this**.hight = hight;  }  } |
| **public** **class** Dog **extends** Animal { // класс-потомок  String name;  String breed;  } |
| Пример 1. Организация наследования |

# Инициализация объекта по иерархии

Помимо инициализации описанной в самом классе, во время создания объекта происходит инициализация всех полей всех его суперклассов. Порядок такой: во время вызова конструктора заданного класса первой строчкой (явно или неявно) вызывается конструктор суперкласса. При этом существует одна особенность: если в суперклассе есть хоть один явный конструктор, необходимо указывать его явно из потомка.

Для вызова любого конструктора суперкласса используется ключевое слово super.

|  |
| --- |
| [модификатор\_доступа] Имя\_конструктора([список\_параметров])  {  super([список\_параметров]); // вызов конструктора суперкласса  // инициализация полей класса  } |
| Формула 2. Объявление конструктора класса в Java |

|  |
| --- |
| **public** **class** Animal {  **int** age;  **int** weight;  **int** hight;    {  System.***out***.println("init block Animal");  }  **public** Animal() {  System.***out***.println("Animal()");  }  } |
| **public** **class** Dog **extends** Animal {  String name;  String breed;  {  System.***out***.println("init block Dog");  }  **public** Dog() {  **super**();//может быть явно и неявно указано  System.***out***.println("Dog()");  }  } |
| **public** **class** Engine {  **public** **static** **void** main(String[] args) {  Dog businka = **new** Dog();  }  } |
|  |
| Пример 2. Инициализация объектов в иерархии |

Пример 2 наглядно демонстрирует, что во время инициализации первыми инициализируются поля суперкласса (в порядке инициализации для любого класса), а после происходит инициализация в классе-потомке.