ООП

* ***Класс***-это шаблон или образец, по которому будет сделан объект. Конструирование объекта на основе некоторого класса называется получением экземпляра этого класса. ***Объект***-это конкретный экземпляр класса.
* ***Инкапсуляция***(иногда называемая сокрытием данных)-это ключевое понятие для работы с объектами. ***Основной принцип инкапсуляции-запрещение прямого доступа к полям экземпляра данного класса из других классов***. Данные в объекте называются полями экземпляра, а функции и методы, выполняющие операции над данными-его методами. Множество значений полей экземпляра называется текущим состоянием объекта. Вызов любого метода может изменить его состояние.
* Все объекты в Java распологаются в динамической области памяти,называемой кучей(<https://tproger.ru/translations/programming-concepts-stack-and-heap/> это ссылка на статью о том,что такое стек и куча. Как я понял, стек используется для хранения локальных данных(для хранения параметров, передаваемых для вызова подпрограмм ,а также для временного хранения данных),а куча-для хранения динамических объектов(т.е. все вот эти malloc,free и прочее-это по сути прямое выделение памяти в куче).
* Если метод имеет доступ к закрытым полям своего класса,то этот метод имеет доступ к закрытым данным всех объектов данного класса. Пример:

public Boolean equals(Employee other){

return name. equals(other.name);

}

* Метод не может изменять параметры примитивных типов
* Метод может изменять состояние объекта, передаваемого в качестве параметра
* Метод не может делать в своих параметрах ссылки на новые объекты
* Пример: ты хочешь свапнуть два объекта. Ты пытаешься сделать это через промежуточный объект temp,но по сути в конце метода твой temp удалится, а объекты, переданные в качестве параметров, останутся со своими старыми ссылками и значениями параметров.
* Можно присваивать полям класса начальные значения сразу при объявлении. Т.е. например private String name = “”;
* [Метод finalize](https://habr.com/post/183344/). Как я понял, это необходимо, чтобы убедиться, что мы точно освободим все ресурсы(например, потоки и т.д.).
* Любое наследование в Java является открытым. Если что, вот [ссылка](http://www.quizful.net/interview/cpp/class-inheritance-types), где объясняется, какие вообще виды наследования существуют.
* У унаследованного класса нет прямого доступа к закрытым полям суперкласса.
* [Множественное наследование](https://drive.google.com/drive/folders/0B_sYN-EHCNmoMjcxRWVTSGNmZXc)-это иногда зло. Основной пример-это ромбовидное наследование.
* **Ромбовидное наследование** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *diamond inheritance*) — ситуация в [объектно-ориентированных](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [языках программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) с поддержкой [множественного наследования](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), когда два [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) B и C [наследуют](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) от A, а класс D наследует от обоих классов B и C. При этой схеме наследования может возникнуть неоднозначность: если [метод](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D1%8F%D0%B7%D1%8B%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) класса D вызывает метод, определенный в классе A (и этот метод не был [переопределен](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0) в классе D), а классы B и C по-своему переопределили этот метод, то от какого класса его наследовать: B или C?
* Что такое динамическое связывание? [Ссылка на javarush](https://javarush.ru/groups/posts/439-razlichija-mezhdu-rannim-i-pozdnim-svjazihvaniem-v-java)
* Как я понял, динамическое связывание-это привязка к коду после начала компиляции. Т.е. пусть у тебя есть два переопределенных метода в классе-родителе и классе-наследнике. Т.к. компилятор не знает до создания объекта, кто из них кто, то связывание происходит после и зависит от выбора объекта. А вот если у тебя есть перегруженный метод(статический, приватный и т.д.),то компилятор уже знает, к чему его привязывать и где использовать. Это есть раннее или статическое связывание.
* Final-классы-это классы, которые не могут быть унаследованы. Если класс объявлен как final,то конечными автоматически становятся только его методы, но не ПОЛЯ!!!
* Final методы не могут быть переопределены в подклассах.
* Абстрактные классы могут содержать НЕАБСТРАКТНЫЕ МЕТОДЫ.
* Создать экземпляры абстрактного класса нельзя, но можно создавать объектные переменные, но эти переменные должны ссылаться на неабстрактный класс, унаследованный от изначального абстрактного.
* В Java защищенные компоненты программ доступны из всех подклассов, а также из других классов того же самого пакета. Этим Java отличается от C++,где ключевое слово protected имеет несколько иной смысл. Таким образом, в Java ограничения на доступ к защищенным элементам менее строги, чем в C++.
* Если модификатор доступа отсутствует, то область видимости ограничивается по умолчанию(пакетом).
* Как формируется метод equals():

Он должен соответствовать пяти пунктам:

1. Рефлексивность
2. Транзитивность
3. Симметричность
4. Согласованность: если некоторые объекты x и y не изменяются, то вывод equals должен остаться прежним.
5. При вызове x.equals(null) по непустой ссылке x должно возвращаться непустое значение false.

* Если ты переопределяешь в подклассе метод equals, то не забывай чекать на равенство еще и родительские классы через сравнивание полей и super.equals(other);
* public final native Class **getClass**()
* public native int **hashCode**()
* public boolean **equals**(Object obj)
* protected native Object **clone**() throws CloneNotSupportedException
* public String **toString**()
* public final native void **notify**()
* public final native void **notifyAll**()
* public final native void **wait**(long timeout) throws InterruptedException
* public final void **wait**(long timeout, int nanos) throws InterruptedException
* public final void **wait**() throws InterruptedException
* protected void **finalize**() throws Throwable
* Автоупаковка и распаковка-это соответсвующее приведение элементарных типов к их классам-оболочкам и наборот.
* Интерфейс в Java не является классом.Он представляет собой множество требований, предъявляемых к классу,который должен соответствовать интерфейсу.Все методы интерфейса являются открытыми.
* В интерфейсе можно объявлять константы, которые тоже будут открытые по определению. Интерфейсы можно наследовать. Интерфейсу можно присваивать ссылку на объекта класса, реализующего данный интерфейс.
* Полиморфизм (polymorphism) (от греческого polymorphos) - это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных.
* В интерфейсе можно описывать методы и поля, но на этом всё.
* Полиморфизм означает способность переменной данного типа, которая ссылается на объекты разных типов, при этом вызывается метод, характерный для конкретного типа ссылки на объект.
* JRE кратко - для работы. Java Runtime Environment (сокр. JRE) - минимальная реализация виртуальной машины, необходимая для исполнения Java-приложений, без компилятора и других средств разработки. Состоит из виртуальной машины - Java Virtual Machine и библиотеки Java-классов.
* JDK кратко - для программирования. Java Development Kit (сокращенно JDK) - бесплатно распространяемый компанией Oracle Corporation (ранее Sun Microsystems) комплект разработчика приложений на языке Java, включающий в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java (JRE).
* Java Virtual Machine (сокращенно Java VM, JVM) - виртуальная машина Java - основная часть исполняющей системы Java, так называемой Java Runtime Environment (JRE). Виртуальная машина Java интерпретирует Байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java (javac). JVM может также использоваться для выполнения программ, написанных на других языках программирования.
* Абстрактный класс используется когда нам нужна какая-то реализация по умолчанию. Интерфейс используется когда классу нужно указать конкретное поведение. Часто интерфейс и абстрактный класс комбинируют, т.е. имплементируют интерфейс в абстрактном классе, чтоб указать поведение и реализацию по умолчанию.
* Куча используется всеми частями приложения в то время как стек используется только одним потоком исполнения программы.

|  |
| --- |
| * Всякий раз, когда создается объект, он всегда хранится в куче, а в памяти стека содержится ссылка на него. Память стека содержит только локальные переменные примитивных типов и ссылки на объекты в куче |
|  |  |

* Объекты в куче доступны с любой точки программы, в то время как стековая память не может быть доступна для других потоков.
* Стековая память существует лишь какое-то время работы программы, а память в куче живет с самого начала до конца работы программы.
* Анонимные классы-это та параша, которая вызывается в скобочках(по типу new Runnable()…)
* Статические методы могу перегружаться и наоборот-в этом нет никаких проблем. А вот в переопределении статического метода тупо нет смысла.
* Что такое статический класс и какие особенности его использования?
* Статическим классом может быть только внутренним. В объекте обычного внутреннего класса хранится ссылка на объект внешнего класса. Внутри статического внутреннего класса такой ссылки нет.

Т.е. для создания внутреннего статического класса не нужно создавать экземпляр внешнего класса. Из объекта статического вложенного класса нельзя обращаться к нестатическим членам внешнего класса напрямую. И еще обычные внутренние классы не могу содержать статические методы и члены.

* Возможно при переопределении метода изменить его модификатор доступа(да, если ты идешь расширяешь ( package- protected -public),возвращаемый тип(да, если ты идешь по низу, т.е. например от String к Object),тип аргумента(нет, т.к. это уже перегрузка).
* В чем отличие между поверхностным клонированием и глубоким(при поверхностном копируются только значения примитивов, а при глубоком при переопределении метода clone буду копироваться уже и ссылочные поля)
* Все типы исключений делятся на проверяемые и непроверяемые. Компилятор требует обязательной проверки проверяемых исключений(блоки try-catch, throws и тому подобное),а непроверяемые по типу деления на ноль и прочие – они не должны проверяться обязательно, т.к. по определению такие исключения не должны возникать, если ты нормально пишешь код.
* Блок try-catch-finally можно использовать и без catch
* [Intern()](https://habr.com/post/79913/)
* Collection расширяет три интерфейса:List, Set, Queue.
* List: LinkedList, ArrayList, Vector. Vector синхронизирован, работает только в одном потоке, а поэтому медленнее. Отличие LinkedList и ArrayList очевидна: один как двусвязный!!! список, второй как массив. В LinkedList проще добавлять и удалять, а в ArrayList удобнее искать(его преимущество в навигации в коллекции).
* Set не содержит повторяющихся элементов. Основные – это TreeSet, HashSet, LinkedHashSet.(TreeSet и HashSet сразу сортируют по значению и по хешам соответственно, а LinkedHashSet просто добавляет элементы).
* Queue : LinkedList, PriorityQueue.
* Map : LinkedMap, HashMap, TreeMap .Тут ключ-значение, аналогичная сортировка по хешам и по значениям. LinkedMap просто добавляет.
* Map не включен в Collections, т.к. методы, определенные в коллекциях, не всегда корректно работают в мапе. Взять то же удаление: в мапе по ключу, а в коллекциях по значению.
* Итератор-это объект, позволяющий перебрать все элементы в колллекции.
* Вместо стека рекомендуют использовать Deque, т.к. Stack уже очень старый.