Ссылки

1. <http://devcolibri.com/4149>
2. <http://javatutor.net/articles/j2ee-pattern-data-access-object>
3. <https://habrahabr.ru/post/268371/>
4. <https://habrahabr.ru/post/60317/>
5. <http://developer.alexanderklimov.ru/android/java/constructor.php>
6. <https://github.com/>

Lesson 2

1. <http://developer.alexanderklimov.ru/android/java/generic.php>
2. <http://metanit.com/java/tutorial/4.3.php>
3. <http://www.quizful.net/post/java-exceptions>
4. <http://citforum.ru/database/oracle/prz/key.shtml>
5. <https://habrahabr.ru/post/168195/>
6. <https://habrahabr.ru/post/169733/>
7. <http://stackoverflow.com/questions/4508172/how-to-properly-clean-up-jdbc-resources-in-java>
8. <https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html>
9. <http://www.javaportal.ru/java/tutorial/tutorialJDBC/statement.html>

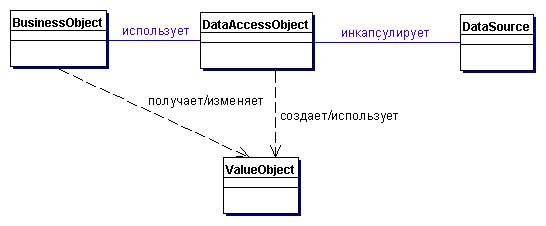
Lesson 1

**Data Access Object (DAO)**

Используйте Data Access Object (DAO) для абстрагирования и инкапсулирования доступа к источнику данных. DAO управляет соединением с источником данных для получения и записи данных.

DAO реализует необходимый для работы с источником данных механизм доступа. Источником данных может быть персистентное хранилище (например, RDBMS), внешняя служба (например, B2B-биржа), репозиторий (LDAP-база данных), или бизнес-служба, обращение к которой осуществляется при помощи протокола CORBA Internet Inter-ORB Protocol (IIOP) или низкоуровневых сокетов. Использующие DAO бизнес-компоненты работают с более простым интерфейсом, предоставляемым объектом DAO своим клиентам. DAO полностью скрывает детали реализации источника данных от клиентов. Поскольку при изменениях реализации источника данных представляемый DAO интерфейс не изменяется, этот паттерн дает возможность DAO принимать различные схемы хранилищ без влияния на клиенты или бизнес-компоненты. По существу, DAO выполняет функцию адаптера между компонентом и источником данных.

Структура



BusinessObject

BusinessObject представляет клиента данных. Это объект, который нуждается в доступе к источнику данных для получения и сохранения данных. BusinessObject может быть реализован как сессионный компонент, компонент управления данными или другой Java-объект, сервлет или вспомогательный компонент.

DataAccessObject

DataAccessObject является первичным объектом данного паттерна. DataAccessObject абстрагирует используемую реализацию доступа к данным для BusinessObject, обеспечивая прозрачный доступ к источнику данных. BusinessObject передает также ответственность за выполнение операций загрузки и сохранения данных объекту DataAccessObject.

DataSource

Представляет реализацию источника данных. Источником данных может быть база данных, например, RDBMS, OODBMS, XML-репозиторий, система плоских файлов и др. Источником данных может быть также другая система (традиционная/мэйнфрейм), служба (B2B-служба или система обслуживания кредитных карт), или какой-либо репозиторий (LDAP).

TransferObject

Представляет собой Transfer Object, используемый для передачи данных. DataAccessObject может использовать Transfer Object для возврата данных клиенту. DataAccessObject может также принимать данные от клиента в объекте Transfer Object для их обновления в источнике данных.

Больше информации по 2-ой ссылке 1-ого Lesson.

DTO vs POCO vs Value Object

Краткие описания Data Transfer Object, Value Object, POJO. А так же их сравнения. Всё это по 3 ссылке 1-ого Lesson.

# Сериализация в Java

*Сериализация -* это процесс сохранения состояния объекта в последовательность байт; *десериализация* это процесс восстановления объекта, из этих байт. Java Serialization API предоставляет стандартный механизм для создания сериализуемых объектов.

## Зачем сериализация нужна?

В сегодняшнем мире типичное промышленное приложение будет иметь множество компонентов и будет распространено через различные системы и сети. В Java всё представлено в виде объектов; Если двум компонентам Java необходимо общаться друг с другом, то им необходим механизм для обмена данными. Есть несколько способов реализовать этот механизм. Первый способ это разработать собственный протокол и передать объект. Это означает, что получатель должен знать протокол, используемый отправителем для воссоздания объекта, что усложняет разработку сторонних компонентов. Следовательно, должен быть универсальный и эффективный протокол передачи объектов между компонентами. Сериализация создана для этого, и компоненты Java используют этот протокол для передачи объектов.

## Алгоритм сериализации Java

 запись метаданных о классе ассоциированном с объектом

 рекурсивная запись описания суперклассов, до тех пор пока не будет достигнут java.lang.object

 после окончания записи метаданных начинается запись фактических данных ассоциированных с экземпляром, только в этот раз начинается запись с самого верхнего суперкласса

 рекурсивная запись данных ассоциированных с экземпляром начиная с самого низшего суперкласса

Примеры и описание формата сериализованного объекта можно найти по 4 ссылке 1-ого Lesson

Конструктор

Конструктор - это специальный метод, который вызывается при создании нового объекта. Не всегда удобно инициализировать все переменные класса при создании его экземпляра. Иногда проще, чтобы какие-то значения были бы созданы по умолчанию при создании объекта. По сути конструктор нужен для автоматической инициализации переменных.

Конструктор инициализирует объект непосредственно во время создания. Имя конструктора совпадает с именем класса, включая регистр, а по синтаксису конструктор похож на метод без возвращаемого значения.

В отличие от метода, конструктор никогда ничего не возвращает.

Конструктор определяет действия, выполняемые при создании объекта класса, и является важной частью класса. Как правило, программисты стараются явно указать конструктор. Если явного конструктора нет, то Java автоматически создаст его для использования по умолчанию.

Больше информации, а также примеры создания конструктора можно найти по 5-ой ссылке 1-ого Lesson

Lesson 2

# Generic

Generic - это параметризованные типы. С их помощью можно объявлять классы, интерфейсы и методы, где тип данных указан в виде параметра. Обобщения добавили в язык безопасность типов.

В угловых скобках используется **T** - имя параметра типа. Это имя используется в качестве заполнителя, куда будет подставлено имя реального типа, переданного классу при создании реальных типов. То есть параметр типа **T** применяется в классе всякий раз, когда требуется параметр типа. Угловые скобки указывают, что параметр может быть обобщён. Сам класс при этом называется обобщённым классом или параметризованным типом.

Вместо **T** подставится реальный тип, который будет указан при создании объекта класса. Объект будет объектом типа, переданного в параметре типа **T**. Если в параметре **T** передать тип **String**, то экземпляр будет иметь тип **String**.

Больше информации и примеров можно получить, перейдя по 1-ой ссылке 2 Lesson

# Исключения в Java

**Исключение**- это проблема(ошибка) возникающая во время выполнения программы. Исключения могут возникать во многих случаях, например:

1. Пользователь ввел некорректные данные.
2. Файл, к которому обращается программа, не найден.
3. Сетевое соединение с сервером было утеряно во время передачи данных.

И т.п.

Обработка исключительных ситуаций (exception handling) — механизм языков программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма.

## Синтаксис

В Java есть пять ключевых слов для работы с исключениями:

1. **try**- данное ключевое слово используется для отметки начала блока кода, который потенциально может привести к ошибке.
2. **catch**- ключевое слово для отметки начала блока кода, предназначенного для перехвата и обработки исключений.
3. **finally**- ключевое слово для отметки начала блока кода, которое является дополнительным. Этот блок помещается после последнего блока 'catch'. Управление обычно передаётся в блок 'finally' в любом случае.
4. **throw**- служит для генерации исключений.
5. **throws**- ключевое слово, которое прописывается в сигнатуре метода, и обозначающее что метод потенциально может выбросить исключение с указанным типом.

Примеры можно найти перейдя по 3 ссылке 2-ого Lesson

## Создание своих классов исключений

Чтобы увидеть краткую информацию, а также пример по созданию своего Exception, перейдите по 2-ой ссылке 2-ого Lesson

Работа в OracleDB

Перейдя по 4-ой ссылке 2-ого Lesson, можно увидеть немало полезной информации по OracleDB (Например задание FK через консоль )

# hashCode() и equals()

Если очень просто, то хеш-код — это число. На самом деле просто, не так ли? Если более точно, то это битовая строка фиксированной длины, полученная из массива произвольной длины

В терминах Java, хеш-код — это целочисленный результат работы метода, которому в качестве входного параметра передан объект.

Этот метод реализован таким образом, что для одного и того-же входного объекта, хеш-код всегда будет одинаковым. Следует понимать, что множество возможных хеш-кодов ограничено примитивным типомint, а множество объектов ограничено только нашей фантазией. Отсюда следует утверждение: “Множество объектов мощнее множества хеш-кодов”. Из-за этого ограничения, вполне возможна ситуация, что хеш-коды разных объектов могут совпасть.   
  
Здесь главное понять, что:

* Если хеш-коды разные, то и входные объекты гарантированно разные.
* Если хеш-коды равны, то входные объекты не всегда равны.

Ситуация, когда у *разных*объектов *одинаковые*хеш-коды называется — коллизией. Вероятность возникновения коллизии зависит от используемого алгоритма генерации хеш-кода.

## Алгоритм (в вольном пересказе)

1) Взять число, отличное от нуля, к примеру 17. Для удобства, назовем его p. Присвоить переменной result это начальное значение (result = p).  
2) Для каждого поля вычислить hashCode c по некоторым правилам. Эти правила, нас сейчас не очень интересуют, они не влияют на результат. Для простоты будем работать с целыми числами (int) и будем принимать hashCode равным самому значению числа…  
3) Скомбинировать result и полученный hashCode с: result = result \* q + c, где q = 37, потому что, как мы помним, оно нечетное и простое.  
4) Вернуть result.

Описание алгоритма с примерами можно увидеть перейдя по 6-ой ссылке 2-ого Lesson

## Try-catch-finally

Способ закрытия Connection, Statement и ResultSet в finally можно увидеть, перейдя по 7-ой ссылке 2-ого Lesson. Данный способ обрабатывает каждый запрос в отдельном блоке try (каждый последующий блок является вложенным в предыдущий). Затем каждый запрос закрывается в отдельных блоках finally (т.е. у каждого try будет свой блок finnaly).

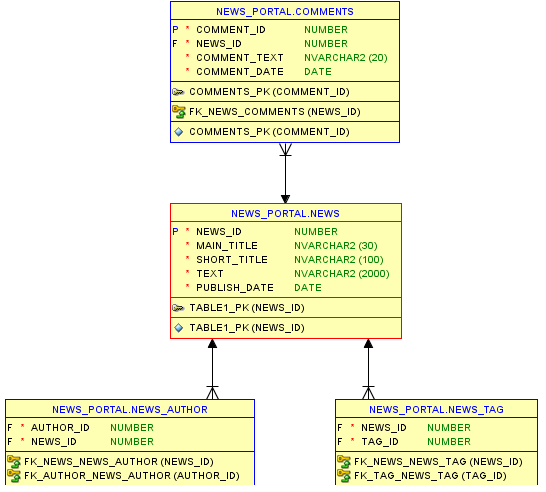
Существует более современный способ закрытия запросов - try-with-resources. Данный способ появился с Java SE 7, и представляет собой заключение запросов в параметры try. Более точную и подробную информацию можно найти перейдя по 8 ссылке 2-ого Lesson

# Запрос (Statement)

Объект Statement используется для выполнения SQL-запросов к БД. Существует три типа объектовStatement. Все три служат как бы конейнерами для выполнения SQL-выражений через данное соединение:Statement, PreparedStatement, наследующий от Statement, и CallableStatement, наследующий отPreparedStatement. Они специализируются на различных типах запросов: Statement используется для выполненияпростых SQL-запросов без параметров; PreparedStatement используется для выполнения прекомпилированных SQL-запросов с или без входных (IN) параметров; CallableStatement используется для вызовов хранимых процедур.

Интерфейс Statement предоставляет базовые методы для выполнения запросов и извлечения результатов. Интерфейс PreparedStatement добавляет методы управления входными (IN) параметрами;CallableStatement добавляет методы для манипуляции выходними (OUT) параметрами.

Метод execute должен использоваться тогда, когда возможно возвращение нескольких объектов ResultSet, более одного счетчика обновлений или комбинации объектов ResultSet и счетчиков обновлений. Такие множественные результаты, хоть и редки, но возможны при вызове некоторых хранимых процедур или динамическом вызове неизвестного на этапе компиляции SQL-запроса.



NEWS\_PORTAL.TAG

NEWS\_PORTAL.AUTHOR