ORM

Object – Relational Mapping

Нещо което ще прави връзката между нашия джава код и базата данни.

Таблиците стават класове, а колоните на таблиците стават полета в класа (дефиниращи поведението на класа).

* Във ООП, управлението на данни е базирано върху задачи които действат върху обекти които почти винаги са -> Не скаларни стойности – ( Базови типове - скаларна **величина**Това е стойностна величина(има стойност – логическа , числова, символна и т.н.), не скаларни са тези които не можем да изразим със една стойност – масиви , листове и друг тип обекти които обединяват характерни за обекта стойностни характеристики в множество).
* Много база данни могат да съхраняват и да манипулират (боравят) само със скаларни стойности , организирани в таблици.

Работната рамка на ОРМ предоставя следните функционалности:

* Автоматично да генерира базови SQL заявки за базови операции(delete, update, create Query and so on ) не се генерират автоматично а ние си ги подготвяме така че да се изпълнят за различни обекти.
* Изграждане на Обектно ориентиран модел от схемата на базата данни (DB first model)

(с горните две ще се занимаваме сега)

* Схема на базата данни от обектно ориентиран модел (Code first model).

Singleton – При създаването на класа така правим връзката с конструктора че да може да се създаде само един обект от класа, в случая става въпрос за connection с цел да няма много връзки.

<https://www.baeldung.com/java-singleton>

Annotation , reflection – да ги преговоря общо взето цялото ООП.

@Retention(RetentionPolicy.*RUNTIME*)

Всяка анотация остава до различен момент – по време на компилация или по време на изпълнение на нас ни трябва по време на изпълнение това и значи горното.  
public @interface Entity {

String name();

За име на таблицата, колоната

RetentionPolicy.SOURCE: се отхвърлят по време на изпълнение.

RetentionPolicy.CLASS: се записват във файла .class, но се изхвърлят по време на изпълнение. CLASS е политиката за задържане по подразбиране в Java.

RetentionPolicy.RUNTIME: се запазват по време на изпълнение и могат да бъдат достъпни в нашата програма по време на изпълнение.

Поясненията се използват за предоставяне на допълнителна информация за програма.

* Поясненията започват с „@“.
* Поясненията не променят действието на компилирана програма.
* Анотациите помагат да се асоциират метаданни (информация) с програмните елементи, т.е. променливи на екземпляра, конструктори, методи, класове и т.н.
* Анотациите не са чисти коментари, тъй като могат да променят начина, по който програмата се третира от компилатора. Вижте например кода по-долу.
* Анотациите основно се използват за предоставяне на допълнителна информация, така че могат да бъдат алтернатива на XML и Java интерфейсите за маркери.

Като цяло има 5 категории анотации, както са изброени:

Marker Annotations - Пояснения към маркери

Single value Annotations - Пояснения с една стойност

Full Annotations - Пълни анотации

Type Annotations - Тип Анотации

Repeating Annotations - Повтарящи се анотации

Marker Annotations - Пояснения към маркери :

Единствената цел е да се маркира декларация. Тези пояснения не съдържат членове и не се състоят от никакви данни. Следователно присъствието му като анотация е достатъчно. Тъй като интерфейсът на маркера не съдържа членове, достатъчно е просто да се определи дали той присъства или липсва. @Override е пример за анотация на маркера.

@TestAnnotation()

Single value Annotations - Пояснения с една стойност:

Тези пояснения съдържат само един член и позволяват съкратена форма за определяне на стойността на члена. Трябва само да посочим стойността за този член, когато се приложи анотацията и не е необходимо да указваме името на члена. Въпреки това, за да се използва тази стенография, името на члена трябва да бъде стойност.

@TestAnnotation(“testing”);

Full Annotations - Пълни анотации

Тези пояснения се състоят от множество членове на данни, имена, стойности, двойки.

@TestAnnotation(owner=”Rahul”, value=”Class Geeks”)

Type Annotations - Тип Анотации

Тези пояснения могат да се прилагат на всяко място, където се използва даден тип. Например, можем да коментираме типа на връщане на метод. Те са декларирани анотирани с анотация @Target.

@Target(ElementType.*FIELD*)

Repeating Annotations - Повтарящи се анотации

Това са поясненията, които могат да бъдат приложени към един елемент повече от веднъж. За да може дадена анотация да бъде повторяема, тя трябва да бъде анотирана с анотацията @Repeatable, която е дефинирана в пакета java.lang.annotation. Неговото поле за стойност определя типа на контейнера за повтарящата се анотация. Контейнерът е посочен като анотация, чието поле за стойност е масив от повтарящия се тип анотация. Следователно, за да се създаде повторяема анотация, първо се създава анотацията на контейнера, а след това типът анотация се посочва като аргумент на анотацията @Repeatable.

<https://www.geeksforgeeks.org/annotations-in-java/>