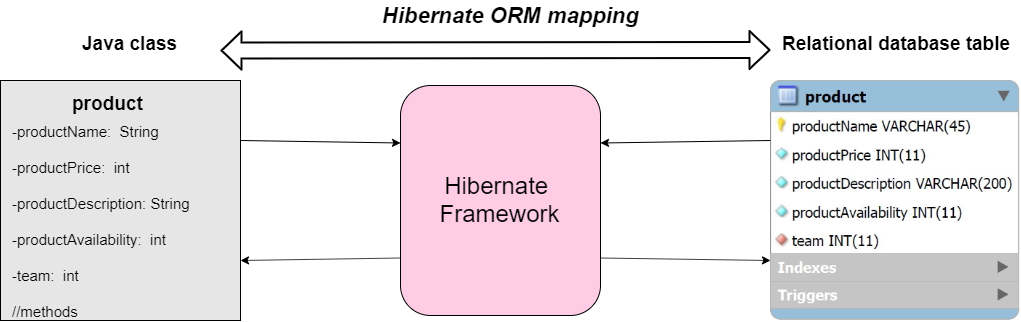
**Tutorial JPA**

The Java Persistence API (JPA) is the Java standard for mapping Java objects to a relational database. It provides specification for the so called ORM (Object-relational mapping). JPA can be implemented, for instance, using the Hibernate framework.

Motivation: why use JPA?

Let’s suppose to have a table PRODUCT, which may represents the many products assembled in a company, in a relational database, and that we want to use a Java application to manage the table. Without JPA, we need to implement all the operation related to the table using MySQL programming language. JPA allows us to manage the creation (if not already present) of the table and all the basic operations (Read, Update, Delete) directly using Java code, by creating a java class “product” that represents the corresponding table in the relational database. This programming technique maps the relational database tables to the application domain objects (Java in this case) is known as ORM. The following figure exhibits the concept of ORM:



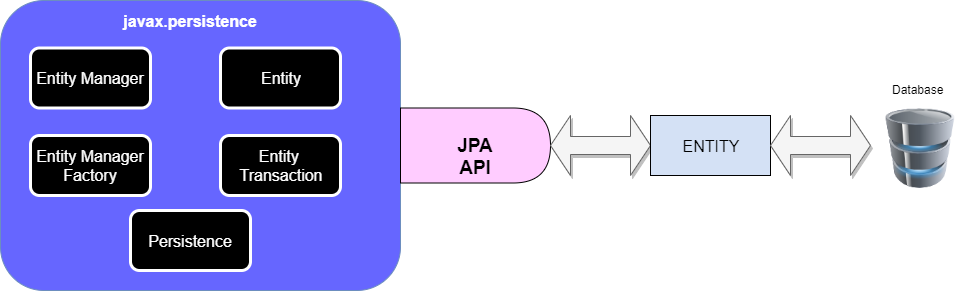
Advantages of this approach:

* The programmer no longer perceives the frustration of having a mismatch between the objects in his data and the relational representation in the database: using JPA each table in the relational database has a correspondent class in the Java code.
* The programmer can create and manage (with the exception of the most complicated operations) the entire database using only Java code without ever writing a single line in SQL. The framework replaces all the database accesses with high level object handling functions, ensuring the persistence of the data in the database.
* The application becomes easier to read. Moreover, by using an object oriented language to map the database objects, the programmer is able to perform functions that wouldn’t be possible in a relational model, like inheritance and handling public/private data.

**JPA Architecture**

The JPA architecture is constructed by the following units:

* Entity:  the persistence objects, that represents a table in the database
* EntityManager: manage the persistence operations of an entity
* EntityManagerFactory: the factory class of EntityManager.
* EntityTransaction: mantains operations for each EntityManager entity.
* Persistence: this class contains static methods regarding EntityManagerFactory instances.



**How to use JPA**

The following steps need to be followed each time you want to manage an object with JPA:

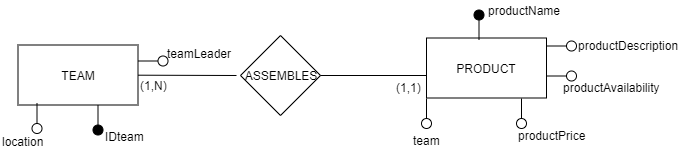
* Introduce the persistence.xml file, where the name of the database is setted
* Define the @Entity that represents the object (if not already defined)
* Use an EntityManagerFactory instance to create a EntityManager in order to handle the object
* Open the connection with the database with a transaction
* Manage the entity (with query or CRUD operations: Create, Read, Update, Delete)
* Commit the transaction
* Close the EntityManager and EntityManagerFactory instances.

To configure the JPA behaviour of the objects, one can use annotations (like @Entity). There are simple and expressive means of decorating java source code with metadata, and are compiled into the corresponding java class files by, for instance, Hibernate to handle the JPA behaviour. Annotations are present in the javax.persistence package. There are many types of annotations.

**Example: managing a DB using JPA**

Suppose we want to manage the database for an application used by a tech company to handle its many products, that are assembled by teams of employees. The following figures represent a possible class diagram and ER diagram for the application:





So we are going to implement two classes: team and product.

For the product class we define the class product.java:

1. **package** net.codejava.hibernate;
2. **import** javax.persistence.\*;
4. @entity
5. @table( name = "product" )
6. **public** **class** Product{
8. @Id
9. @Column( name = "productName", length = 45, nullable = **false** )
10. **private** String productName;
12. @Column( name = "productPrice", nullable = **false** )
13. **private** **int** productPrice;
15. @Column( name = "productDescription", length = 200, nullable = **false** )
16. **private** String productDescription;
18. @Column( name = "productAvailability", nullable = **false** )
19. **private** **int** productAvailability;
21. @ManyToOne( optional = **false** )
22. @JoinColumn( name = "team" )
23. **private** Team team;
25. //----------------------------------------------------------------------------------------------------------
26. //                                          CONSTRUCTORS
27. //----------------------------------------------------------------------------------------------------------
29. **public** Product(){}
31. **public** Product( String name, **int** price, String description, **int** availability ){
33. **this**.productName = name;
34. **this**.productPrice = price;
35. **this**.productDescription = description;
36. **this**.productAvailability = availability;
37. }
39. //----------------------------------------------------------------------------------------------------------
40. //                                          GETTERS
41. //----------------------------------------------------------------------------------------------------------
43. **public** String getProductName(){
45. **return** productName;
46. }
48. **public** **int** getProductPrice(){
50. **return** productPrice;
51. }
53. **public** String getProductDescription(){
55. **return** productDescription;
56. }
58. **public** **int** getProductAvailability(){
60. **return** productAvailability;
61. }
63. **public** Set getProductStock(){
65. **return** productStock;
66. }
68. **public** Team getTeam(){
70. **return** team;
71. }
73. //----------------------------------------------------------------------------------------------------------
74. //                                          SETTERS
75. //----------------------------------------------------------------------------------------------------------
77. **public** **void** setProductName( String productName ){
79. **this**.productName = productName;
80. }
82. **public** **void** setProductPrice( **int** productPrice ){
84. **this**.productPrice = productPrice;
85. }
87. **public** **void** setProductDescription( String productDescription ){
89. **this**.productDescription = productDescription;
90. }
92. **public** **void** setProductAvailability( **int** productAvailability ){
94. **this**.productAvailability = productAvailability;
95. }
97. **public** **void** setProductStock( Set productStock ){
99. **this**.productStock = productStock;
100. }
102. **public** **void** setTeam( Team team ){
104. **this**.team = team;
105. }
107. }

And the class productManager.java

1. **import** javax.persistence.EntityManagerFactory;
2. **import** javax.persistence.EntityManager;
3. **import** javax.persistence.Persistence;
5. **public** **class** ProductManagerEM{
7. **private** EntityManagerFactory factory;
8. **private** EntityManager entityManager;
10. **public** **void** setup(){
12. factory = Persistence.createEntityManagerFactory("InnovativeSolutionsDB");
13. }
15. **public** **void** exit(){
17. factory.close();
18. }
20. **public** List<Product> getTeamProducts( **int** team ){
22. }
24. **public** **int** insertProduct(){
26. }
28. **public** List<Product> getAvailableProducts(){
30. }
32. **public** List<Product> searchTeamProducts( **int** team, String name ){
34. }
36. **public** List<Product> searchProducts( String value ){
38. }
40. **public** **int** updateProductAvailabiility( **int** product , **int** value ){
42. }
44. }

For the class Team, we define the class team.java:

1. **package** net.codejava.hibernate;
2. **import** javax.persistence.\*;
4. @entity
5. @table(name="team")
6. **public** **class** Team{
8. @Id
9. @GeneratedValue( strategy = GenerationType.IDENTITY ) //GenerationType.IDENTITY relies on an auto-incremented database column to generate the primary key
10. @Column( name="IDteam", nullable = **false** )
11. **private** **int** IDteam;
13. @Column( name="location", length = 45, nullable = **false** )
14. **private** String location;
16. @OneToOne( cascade = CascadeType.ALL, optional = **false** )
17. @JoinColumn( name = "teamLeader", referencedColumnName = "IDemployee")
18. **private** Employee teamLeader;
20. //----------------------------------------------------------------------------------------------------------
21. //                                          CONSTRUCTORS
22. //----------------------------------------------------------------------------------------------------------
24. **public** Team(){}
26. **public** Team( String location ){
28. **this**.location = location;
29. }
31. //----------------------------------------------------------------------------------------------------------
32. //                                          GETTERS
33. //----------------------------------------------------------------------------------------------------------
35. **public** **int** getIDteam(){
37. **return** IDteam;
38. }
40. **public** String getLocation(){
42. **return** location;
43. }
45. //----------------------------------------------------------------------------------------------------------
46. //                                          SETTERS
47. //----------------------------------------------------------------------------------------------------------
49. **public** **void** setIDteam( **int** IDteam ){
51. **this**.IDteam = team;
52. }
54. **public** **void** setLocation( String location ){
56. **this**.location = location;
57. }
58. }

And the class teamManager.java:

1. **import** javax.persistence.EntityManagerFactory;
2. **import** javax.persistence.EntityManager;
3. **import** javax.persistence.Persistence;
5. **public** **class** TeamManagerEM{
7. **private** EntityManagerFactory factory;
8. **private** EntityManager entityManager;
10. **public** **void** setup(){
12. factory = Persistence.createEntityManagerFactory("InnovativeSolutionsDB");
13. }
15. **public** **void** exit(){
17. factory.close();
18. }