Hibernate 1

- Persistency: Persistency is the process of storing, managing and retrieving data from a database or file system in effective way called as Persistency.
- In initial days of computer there was a File System database, which is used to store data in it. But problem with file system are managing and retrieving the data.

Hibernate class 1

- There are some drawbacks in file system:
- Data stored in not structured manager.
- No semantics .
- Hard to manage(like insert, delete , update the data etc).
- It is only Human understandable but not computer understandable.
- We can’t reuse data for operation in program .
- It is like keeping records in notebook only.
- Data stored in character basis, B’Z of that it is very hard to read the data char By char.

Hibernate class 2

- To overcome the problems of file system CSV(character separated value) file has been invented.
- CSV is the file system which is used to stored data in character format with value separator using comma(,). It also lead to a problem while reading the data. B'z actual data may contains a comma(,) it's lead to get wrong data input. And also it take more time to read data.
- As same as file System CSV also have the bunch of problems.

Hibernate class 2

- 1st CSV file leads some problems:
- It stored data in Character format .
- Data separated with comma.
- It is understandable by Human only.
- It is very hard to manipulation.
- Data may contains commas.

Hibernate class 3

- In the next version of CSV DisAdvantages:
- Data stored in fixed field size format which leads too many problems for 5 character it take too much memory
- Wastage of memory.
- Its not easy to structured in file.
- Data not reusable it take more time to read char by char
- Data not in semantic format.
- Performance is very low.
- No relation between data.
- Its critical in manipulation of data.
- Even though it is extension of file system but not enough to perform persistency.

Hibernate class 3 and 4

- XML(Extensible markup language)
- It is language used to stored data in tag based format.
- XML is invented by W3C org. and the version is 1.0 in current day also.
- Advantages
- Data will be stored in tag based format.
- Data will have high relationship b’w them.
- it has well structured .
- It has well semantics.
- Anyone can understand the data.
- It provided a validator called dtd (Document type Definition and xsd(xml schema definition).
- Dtd and xsd have protocols about how to write the xml file.
- It is operable(means it can run in any platform).
- It does have any keywords (in the prog. World only XML lang. does have any keywords)

Hibernate class 4 and 5

- Disadvantages of XML
- It is not easy for common people to understand it.
- Without knowing we can’t write the xml file.
- Data will be stored in hierarchical format.
- Before writing xml we have to know dtd or xsd file.
- Xml speak more for storing the data.
- It consume more memory for little data also.
- We can stored data but while reading we’ll get more problems.
- IOStreams read data line by line, it couldn’t identified actual data in xml file. Parsing technology we have to use.
- B’z of above reason xml also failed to persistence the data in it.
- To overcome the problems of xml Java has invented a Serialization concepts which seems to be good compared with earlier file systems.

Hibernate class 5 and 6

- Serialization
- Its concept provided by Java Prog. Lang. to persist the data into file.
- Serialization means change the status of an objects in bit or byte.
- While Serialization use public static final serialVersionUID to match the exact sender or receiver.
- Advantages
- Serialization make an object serializable it will transfer through network.
- Using FileInputStream and FileOutputStream as well as ObjectInputStream and ObjectOutputStream we make an object serializable.
- It contains bits or bytes of information.
- No need to verification while accessing the data.
- Data will be in high security mode while transmission.
- Serializable is the marker interface in Java. It doesn't contains any methods.
- Static and final keywords are not usable in serialization.
- Transient keyword used to hide the data while serialization.
- We can perform operation on the serialized data.

- Disadvantages
- Serialization only used in Java it is not universal.
- We can stored multiple object using collection but it is complex to read and write.
- It is used by only programmer.
- It may fail while transmission the data from different platform.

Hibernate 7

- DBMS(Database management System)
- To overcome the problems DBMS has been invented.
- DBMS is the process of storing, managing and retrieving related data from the DBMS.
- DBMS contains four parts

  -Hierarchical Database

  -Network Database

  -Relational Database

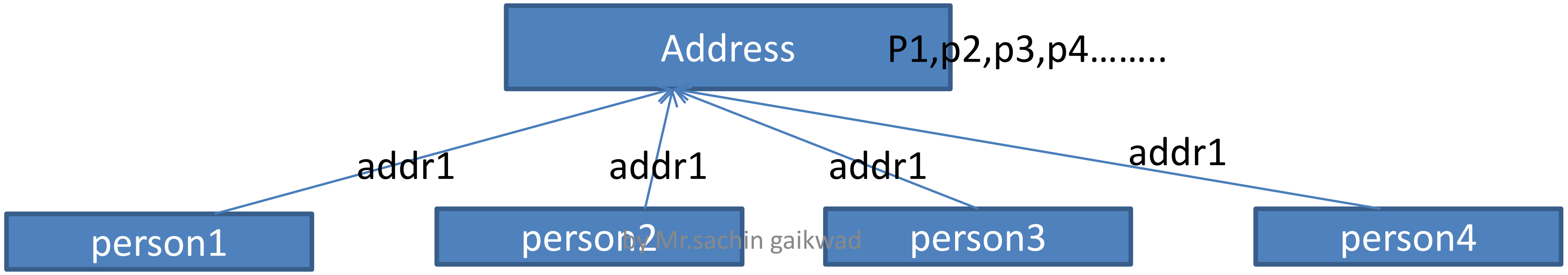
  -Object oriented Database

Hibernate 7

- Hierarchical Database:
- in hierarchical database data will stored in parent and child format which lead to duplication of data.
- We can not relate one data to another data object B’z of hierarchical concept.
- There Is only one bottle neck to access the data it lead to low performance.
- There are many chances to loss of data B’z subsequent data depends on the parent only. once parent deleted means all the data will lost.
- Data may be in structured format but not in semantics and accessing will be complex.

Hibernate 7

- Network Database
- In Network database all data are related to each other by address.
- It is critical to keep all the addresses in memory.
- While manipulation it lead to the problems.
- We can not delete the data without knowing all info. About it.
-



Hibernate 7

• Relational Database

- It is the world wide used database concept.
- Data stored in the form of tables.
- Relationship between data depend on logical addresses.
- Different types of databases supported by Relational database concept(oracle, DB2, mysql, mysqlServer etc.).
- We can stored related data in it .
- To make data clear and consistence we can use Normalization form.
- It also provide different constraints which helps developer to stored data in persistence manner.
- It uses different model to store the data.
- A table also called as an entity.
- A record also called as a tuple.
- Domain means datatype which allow to store accurate data in it.
- It is easy to persist the data in RDBMS.
- Java provided a package called JDBC API which help you to connect with the data bases.

- It is seem to be good while working with RDBMS but there are no. of drawbacks also there.
- In RDBMS primary kay means logical address of the data which is unique to every record in the database.
- RDBMS use hashing indexing technique to fetch the data and stored the data
- Physical address and logical address are different for each record.

Hibernate 8

- **RDBMS and Other Database Drawbacks**
- Java provided a JDBC API which deals with all databases.
- Using JDBC API we can perform operations on the databases but there are no. of databases are available in the market. Like CSV, xml, RDBMS etc.
- For every database we have to write separated business logic to perform operation on it.
- If we want the data from database, we will get data in relational format rather than object format.
- Relational database will retrieve data in rows and columns format.
- CSV, XML, Hierarchical, Network database are structured, semantics, if any change In database make the change in corresponding code in the application. So it is very expensive task , we cant modify our code every time.

Hibernate 9

- Data stored in databases in different format. Relational database can stored data in rows and columns.
- CSV stored data in record format.
- XML stored data in tag based format.
- So it is very difficult to get operational data from the different databases.
- If we are using RDBMS to get operational data then we will get data in relational format, in ResultSet.
- But the problem is that we have to know the internal column names with datatypes to get the operational data. Using rs.next() we access the data.
- we have to expose the internal format of the database to the programmer.
- In future database will change into other database then again we have to rewrite the whole business logic code to perform same operations.
- To access data and perform business logic we have to write code in one class but it may not recommended B,z if database get change means we have to change whole accuser code and business code.

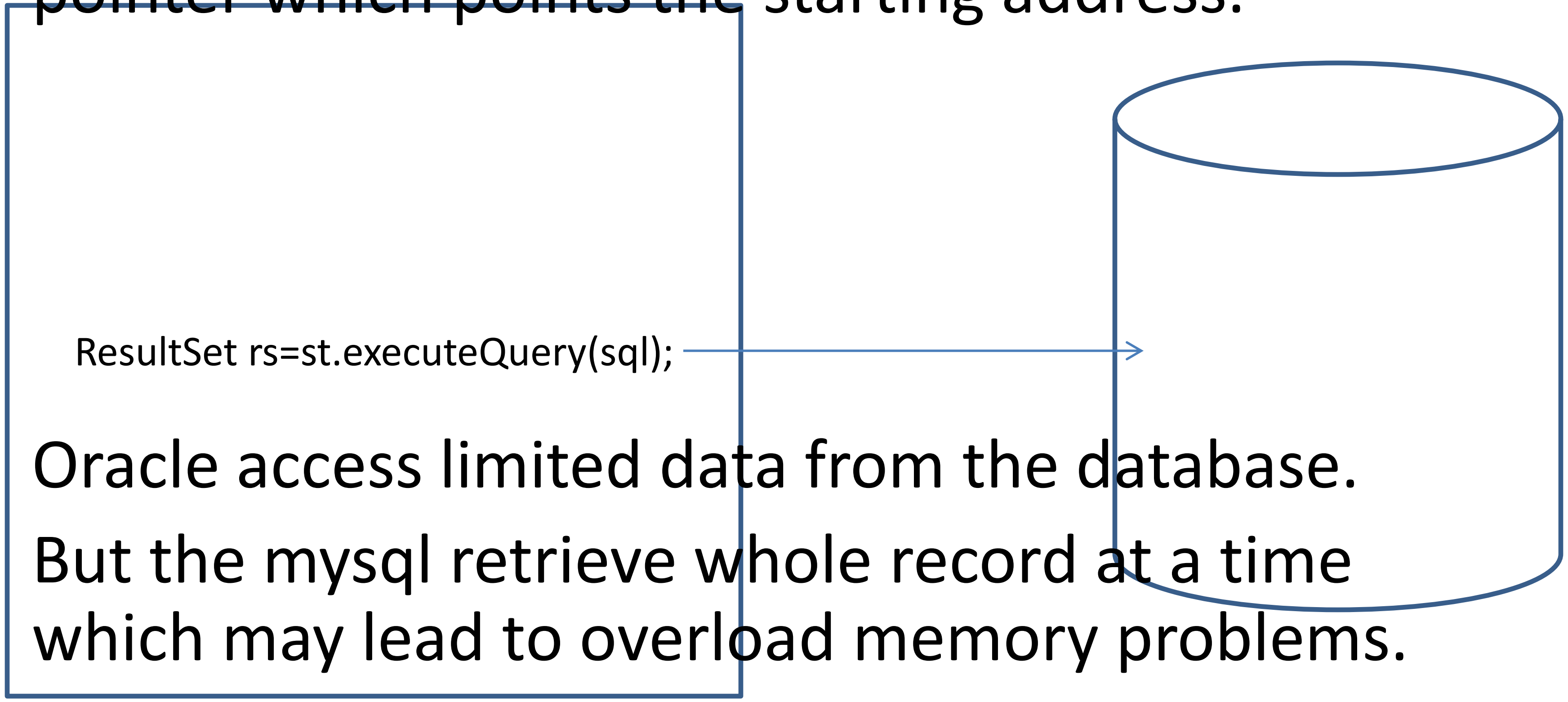
- so to overcome above problem we write code in two different classes, one is for data accuser and other for business logic.
- physically they are different but logically same, B’z we are sending data in relational format only.
- If we want to work independently then we have create POJO or domain or DAO class which separated the data access logic from business logic.
- We want a object which take care about all internal activities and give only required data.
- If we want object as a input then we have to store data as a object only.
- But there are several problems if we stored in object format. We cant store object directly into the RDBMS before that we have to serialize the object and stored into the database using BLOB datatype.
- Again there is problem to persistence the data. It is understandable by only java programmer.
- Serialized data means bits or bytes format and we can’t perform queries on the serialized data.

- If we want to stored object or access object before that as programmer manually stored the data into object and persist into database.
- We hold accessed data into ResultSet but we can’t send ResultSet object as operational data to the business logic. B’z once conn get close how it is possible to access data from the database that is the drawback of ResultSet.

```
Class OrderProcessing{
    Public void dataAccusor(ResultSet ){
        //Business logic
    }
    Class OrderConn{
        connection  con……//operational data
        statement stmt…….
        ResultSet rs=stmt.executeQuery(sql);
        OrderProcessing op = new OrderProcessing();
        op.dataAccusor(rs);
    Close.con;
    Close.st;
    }
```

– Once connection has been closed how it is possible to send ResultSet data to the dataAccusor() method.

- ResultSet cant stored all data in it. There is some internal logic which make ResultSet object to access data from the cursor.
- Cursor which hold logical data in it. It have start pointer which points the starting address.



```
ResultSet rs=st.executeQuery(sql);
```

- Oracle access limited data from the database.
- But the mysql retrieve whole record at a time which may lead to overload memory problems.

- ResultSet fetch data from cursor and keep into buffer memory access data one by one . But once the connection get close we can’t access the data.
- So we can’t use ResultSet as the Object to perform operations.
- If we want data into object form programmer has to stored into object.

Hibernate 10

To overcome the above problems Object Oriented databases has been invented.

- OOD provide platform to stored data in object oriented format. And access data in object format.
- OOD is the NOSQL database means unstructured database.
- B’z of the OOD suffering from problems, in market every one adducted to use only RDBMS no one use OOD.
- OOD don’t have support for relationship between the data, and people want relationship.
- B’z of representation of OOD people don’t want to use OOD database.
- People facing sort of problems while working with OOD.
- TO overcome from OOD the third party JBOSS has invented a Technologies called ORM Technologies.

- ORM(Object Relational Mapping Technologies)
- ORM is the methodology which provided by some people after observing the problems with RDBMS and OOD databases.
- We already discussed which kind of problems we are there with above databases.
- For performing any operation we have to write huge number of code for parsing from RDBMS data to Object format and Object format to RDBMS format.
- If the are 20 records are there in the database the as programmer he have to write 20 lines of code for parsing. Not only getting the data for storing also the have to write 20 lines of code.
- It is not possible in one project we have use only one time data and perform the operation every time we have to deals with the database, if there are 10 classes using the same logic then as programmer the has to write 10 time the same code .
- It will very high problem for programmer. It also reduce the performance of the project , it will take huge time to develop our application, bugs may be high , maintenance may high, the are several problems are available.
- Java hasn't provided any API for ORM technology but in market the are several business people are available for looking learn money.

- Java people thought ORM is open source it may or may not use by the all people, B’z of that java hasn’t provided any API.
- But in market other competitor has developed there own APIs to work with ORM Technologies. Like .net etc.
- B’z of that every one looking to .net technology and migrating from java to .net.
- This situation observed by JBOSS people and they have provided a ORM supported Framework called as HIBERNATE.
- It contains boiler plat logic which make very easy to deals with databases.
- Before ORM technology programmer has to take care of converting the data into Object format and Object format to the RDBMS format, but now Hibernate has provided pre-identified class which can automatically convert data to object and object to data.
- It made programmer job easy, now programmer has to follow the Hibernate provided methodology to work with object Relational mapping.

Hibernate 11

- **Hibernate**
- Hibernate is the third party vendors provided framework which build for ORM technology.
- Hibernate has invented then suddenly most of the Org. adopted the hibernate.
- Like hibernate there the several third parties are available in the market which provide the support for ORM technology.
- But Hibernate has invented first in the market B'z of that most of the org. adapted the hibernate.
- Hibernate adapted by millions of org. even though Java hasn't shown his interest to develop one ORM supported API.
- Eventually java developer been working on Entity bean only.
- It is very hard to live with Entity bean B'z of the org. started using the .net or other technologies.

- B’z of that Java started thinking about to develop one API which can support the ORM technologies.
- It is very hard to java to convince the org. to use java provided API for ORM . B’z of the java played a game and he handshake with the Hibernate.
- For hibernate it is very great thing B’z java is the widely used programming language.
- But the problems is that hibernate can’t change his classes B’z most of the Org. using Hibernate.
- Then java has provided one API called JPA(java Persistence API).
- Which specially build for ORM and java has provided interfaces to the hibernate people to work on it and provide a new API which build top of the JPA and JDBC API.
- At the end of the day hibernate provide a API with version 3.0 which can support independent hibernate people and who want to use JPA with hibernate.
- **Hibernate 3.0[independent + hibernate with JPA].**

- After the hibernate, other vendors also shown there interest in implementing the java provided JPA API for developing ORM tools.
- Java has provided implementation jpa ri interfaces to hibernate. Like there are no. of implementation provided by java.
- Now a days those are want to develop new application they can use hibernate with JPA and old people can use hibernate independently.
- In across the world there the more then 25 tools are available for ORM technologies.

Hibernate 12

- Why hibernate? Why not jdbc?
- As we discussed in previous classes about the JDBC problems. While storing the data, accessing data and lot more…
- 1.One of the new problem with JDBC is JDBC Exceptions are checked exceptions.
- Java has provided classes to work with database, but programmer has to write try catch block in every class. Even it is not required.
- Writing try catch block in every program it is not good practices.

- Lets see the example

```
Class Test {
Connection con= null;
Statement stmt = null;
            Test(){
                        try{
con = DriverManager.gtConnection(url,un,pwd);
Stmt = con.createStatement();
Stmt.executeQuety(update);
            }catch(SQLException e){
                        }
}
```

- To perform any operation connection is very important. If we write the connection logic without try catch block we will get SQLException, But the problem is we can not handle this problem using checked exception. Even if we handle there is no use B'z for closing the conn we using checked exception.
- There are some situation we can handle by checked exception but there is no use. Like creating a conn .
- For very jdbc classes we forcibly writing try catch blocks number of times.
- JDBC is the checked exception.

- Checked Exception means we can handle at the compile time. Java compiler provide some guidelines, for how to handle that exception. Means there is an alternate way to handle the exception.
- But for Unchecked Exception there may be or may not be the chances of handle the Exception.
- Unchecked exception provide the flexibility to handle in two ways, u can if u want if u don't leave it.
- 2.JDBC suffering from more problems to persist the data. If we want to perform any operation we have to know about the whole details about the particular table. About table name, datatypes, column name, whole schema about the table etc.We'll get data in relational format only.
- 3. Most of the time we want data in the object format but JDBC will give data in Relational format, as programmer we have to write for conversation logic object or relational format and relational format to Object format.
- 4.Managing the connection most heavy task B'z many time we failed to close the connection, if there are 100 user want to interact with database and if we din't close the connection then database will not allow other users to access the database until and unless he get resource.

- Most of the people close there connection in try block only but it leads many problems, if any problems with loading the driver or in connection then directly catch block will execute but we wrote closed conn in try block.
- if we close our connection in catch block then also it is a problems B'z a class can contain many catch blocks so we have to write same code in every catch block. Duplication of the code.
- There is the sequence of closing the connection, first ResultSet, second Statement and third connection.
- If we handled in catch block using if condition for example

```
Try{
………}
Catch(SQLException e)
{
          if(stmt != null)
          stmt.close();
          if(conn!= null)
          con.close();
}
```

If while executing the program Statement din't created then it will thorwn that exception to the method, so what about the next if condition. We are unable to handle here.

- Using finally we can handle lets see the example

```
Try{………..
}catch(SQLException e){
…………..
}finally{
            if(stmt ! = null){
            try{
                        stmt.close();
               }catch(Exception e){}

            if(conn!=null){
            try{
                        con.close();
                }catch(Exceptio e){}
}
```

- To close the connection we have to write more lines of code in finally block.

- 5. JDBC face problem in Transaction management
- While dealing with Transaction management we manually set setAutoCommit(false), B’z by default it is AutoCommit mean after every transaction it will commit and if there is any problems with query then it will rollback all the previous queries.
- lets see the example

```
Class A{
            boolean isException = true;
            try{
            loading driver
             Connection……
            Statement………
            Con.setAutoCommit(false);
            stmt.executeUpdate(sql1);
            stmt.executeUpdate(sql2);
            isException = false;
}catch(Exception e )
{
            isException = true;
}
Fanally{
            if(isException==true)
            con.rollback();
            if (isException  == false)
            con.commit();
}
```

- Most of the people going to commit() in try block only but the problems is while updating there may be chances to get exception, then we can not handle. using finally we can handle as per above example.
- 6. There are two Transactions
  - 1. Local Transaction -----JDBC API
  - 2. Global Transaction-----JTA API

  Local Transaction handled by JDBC api but Global Transaction handled by JTA api.

  We can not handle both using JDBC api, if we want to work with Global Transaction we have to learn JTA api which is not good for programmers.

  7. While changing our database from one database to another database it will affect all the SQL queries in the application, B’z JDBC doesn’t have such kind of support.

  Oracle queries syntax are different from mysql queries syntaxes so it will affect all the queries.

  8.We can’t maintain VERSION of the data for maintaining version we have to write individual code.

  9. We can’t find and TimeStamp feature in JDBC API.

  10. In jdbc if we want same data 100 time then jdbc will execute same query 100 time to get data. We have to write same code no. of times.

- Hibernate: Hibernate is the framework which is provided by JBOSS vendors. Hibernate is the most used ORM Tool in the market.
- Hibernate is the framework which contains bunch of classes and classes contains pre-identified code.
- Hibernate provide boiler plat logic to the developers . Framework make programmer job easy, programmer need not to write much coding for development.
- As we discussed in previous classes what are the problems with the JDBC API.
- To overcome all the drawback developer going to use Hibernate.

- Using JDBC we going to face below problem:
  - We have to Write more lines of code.
  - More code more bugs.
  - More maintenance .
  - More Budget.
  - More developer.
  - More time.
  - We will not get good quality.

Hibernate 13

- Advantages in Hibernate:
- Hibernate has provided predefined classes which avoid all the above problems.
- 1.We will get more quality code from hibernate. B’z of that we have to write less no. of code.
  - Less code less bugs.
  - Less maintenance.
  - Less Budget.
  - Less time
  - Less developer.
- 2.Hibernate take of underlying conversion of the data to object and object to data.
- 3. we need not to provide internal details schema about the database to the hibernate , we going to create an class and just talk to the hibernate only once .
- Hibernate can get the data in to object format or it can persist the data in relational format.

- 4. Hibernate take care of managing the resources itself we need not wary about the closing the connection…..
- 5. We need not wary about the SQL query language while working with the hibernate. B’z hibernate provided HQL to the programmers, to works with the database.
- Without knowing single query also we can work with hibernate.
- 6. Hibernate use dilate internally B’z as per programmer requirement he can switch databases. From oracle to mysql, mysql to myseversql ….etc.
- Hibernate has provided one criteria class with take care of managing the all HQL queries.
- 7. most powerful advantage is hibernate has a caching memory, it improve the performance of the application .
- It keep your data in caching and as per request he give the data.
- 8.hibernate has version feature which is take of maintaining the version of the modified record.

- If we want know how much time a data has modified or inserted in the database so there is no need to write extra logic, Hibernate has provided VERSION feature which take care of maintaining record.
- 9. TimeStamp it is also one of the feature of hibernate, if we want to know when it is modified or when it is inserted in the database then TimeStamp will going to keep the records.
- 10. Hibernate manages the Transactions internally we need not be wary either it is local transaction or Global Transaction.

Hibernate 14

- As we discussed earlier advantages of hibernate now we will learn how to deals with the application.
- In application contains multiple classes very class have it own uses.
- There are three kinds of classes we will use
  - POJO(plan old java object)
  - Java bean
    - Business objects
    - Entity objects

  --Component class

1. POJO(plan old java object)

POJO class is java class which will compile and execute using underlying JDK without any third party support called as POJO class.

- 2. Java Bean: java bean are java classes which contains attributes and accuser method.
  - Java bean use to hold the data in object format.

  i).Business Objects: A java bean class attributes are used for performing some business operation called as Business Object. But business object contains only required data which is used for performing the operation.

  ii).Entity Objects: A java bean class attributes are used for persist a data also in the relational format.

  A Entity objects contains all the table attributes for persistence.

  We mostly concentrate on Entity Objects B'z hibernate deals with persistency.

  How to map object to data and data to object, we will learn all the concept.

- 3.Component class: It is java class contains attributes and (methods)business logic to perform the operations.
- In Spring we mostly deals with component classes B’z Spring used for Business logic.
- As Hibernate Application one class can depend on other class easily and having multiple relationship between classes. So there are two way to deals with other classes either by inheritance or composition.
- As we know that inheritance means IS-A relationship, one class is same as other class then only we can use inheritance.
- For example child is same as parent use inheritance.

- Composition: Composition means HAS-A relationship, A class may contains multiple attributes and method called as composition.
- A Object is the combination of multiple thing in it called as composition.
- As we know if we want to use one class features into other class we will use inheritance or composition.(placing the relationship between two class.)
- **Relational Object model:**
- Inheritance having multiple problems
  - We can't extends more than one class
  - It forces other class to use all features
  - There is problem or fragile the classes
  - Problems in testability.
- There are two type in inheritance
  - Generalization
  - Realization
  - Specialization

- i).Generalization: It is the process of keeping duplicate method in one class, and extends that class to other subsequent classes.

A
m4()

B	C	D
M1()	m2()	m3()

=>As we see class A contains method m4() which is common for all the concern classes whoever want to use that method extends class A and access that method.

ii).Realization: It deals like mediator for main class and subsequent classes. If a method name is same but implementation will be different then we can use Realization concept.

iii).Specialization: A class want to create a new feature and it is specific to that particular class called Specialization. A may contains it own methods which is not sharable to other classes.

Lets see the example for more understanding.

Realization example

```
Class A{
        private IB ib;
        public static void main(String ars[]){
                ib = new B();
                ib.m1();
        }
}
Interface IB{
Public void m1(){
}
Class B implements IB{
Public void m1(){
        //logic
}
Class C implements IB{
Public void m1(){
        //logic
}
```

Here class B and Class C Realizing the method m1() in it.

Whoever want that method just implements IB interface and provide the implementation to the method.

Hibernate 15

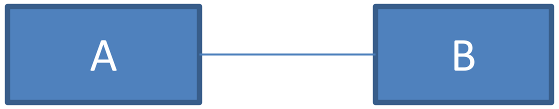
- As we discussed in previous class what are the ways to speak with other classes using inheritance.
- Now we will discuss Composition
- Composition: A class contains other class attributes and method called as Composition.
- A object has a other object data called as Composition.
- Composition classified into three ways
  - Association
  - Aggregation
  - Composition
  - Dependency

- Association: Two classes are associated with each other. One class can survive without other class called as Association.

  In association parent and child relationship is not there both are equal to carry there work.

  -owner :not applicable

  -relationship: independent

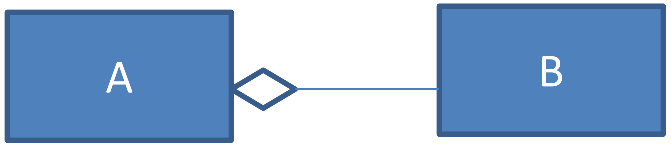


- Aggregation : Two classes are aggregate with each other. One class can survive without other class But here classes has there own parents.

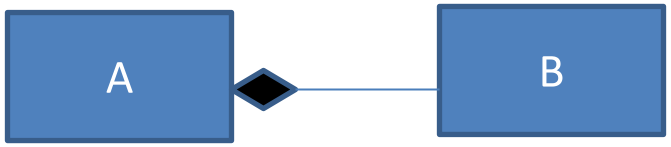
  ex: students are depends on teacher but without student teacher can survive and without teacher student can survive.

  - owner : parent is there

  -relationship: independent

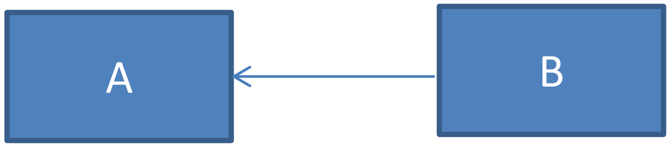


- Composition: one class can not survive without other class that is composition.
- Means class A contains class B attribute to survive if B is not there means class A also not there.
  - owner : parent is there
  - Relationship: dependent



- Dependency: A class method is depend on other class attribute it is called as dependency.

Ex:
```
class A{
        p v m1(B a){}
}
class B{}
```



Class A method depends on Class B object so it is called as dependency.

Hibernate 16

- Relational model
- In relational model means placing the relationship between tables. There are four kinds of relationship
  - One to one
  - One to many
  - Many to one
  - Many to many
- Lets see the diagram and understand

ONE TO ONE RALATIOSNHIL

Parent table primary key has to be included as pronary key in child table referencing parent table.

CUSTOMER	
PK	CUSTONER_ID
	FIRST_NM
	LAST_NM
	GENDER
	DOB

ADDRESS	
PK,FK	CTUSTOME_ID
	ADDRESS_LINE_1
	ADDRESS_LINE_2
	STATE
	CITY
	ZIP
	COUNTRY

ONE TO ONE RALATIOSNHIL

CUSTOMER	
PK	CUSTONER_ID
	FIRST_NM
	LAST_NM
	GENDER
	DOB
UNIQUE,FK	ADDRESS_ID

ADDRESS	
PK	ADDRESS_ID
	ADDRESS_LINE_1
	ADDRESS_LINE_2
	STATE
	CITY
	ZIP
	COUNTRY

ONE TO MANY RALATIOSNHIL

1)Primary key of the one table become foreign key of another table.
2)The table which holds the foreign key always represents many side of the relationship

CUSTOMER	
PK	CUSTOMER_ID
	FIRST_NM
	LAST_NM
	GENDER
	DOB

ADDRESS	
PK	ADDRESS_ID
	ADDRESS_LINE_1
	ADDRESS_LINE_2
	STATE
	CITY
	ZIP
	COUNTRY
FK	CUSTOMER_ID

MANY TO MANY RALATIONSHIP

CUSTOMER	
PK	CUSTONER_ID
	FIRST_NM
	LAST_NM
	GENDER
	DOB

ADDRESS	
PK	ADDRESS_ID
	ADDRESS_LINE_1
	ADDRESS_LINE_2
	STATE
	CITY
	ZIP
	COUNTRY

CUSTOMER_ADDRESS	
PK,FK	CUSTOMER_ID
PK,FK	ADDRESS_ID

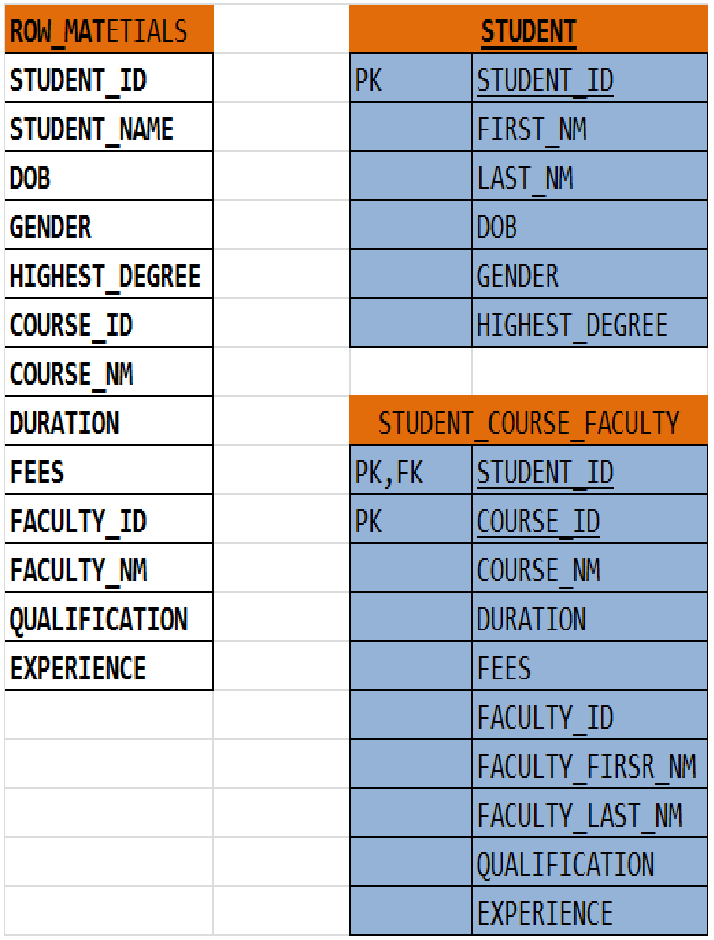
Hibernate 17

- As we discussed relationship between tables i.e. one-to-one, one-to-many, many-to-one and many-to-many.
- Giving the relationship between tables mean analyzing the real world relationship.
- Normal forms are very important while normalizing the data into tables.
- Normal form: It is the procedure to avoid the anomalies/ redundant data form the table.
- There are
  - 1NF
  - 2NF
  - 3NF
  - BYEES CODD RULES
  - 5NF

- 1NF:
  - Avoid redundant data from the table.
  - The row/tuple should have an atomic name.
  - The table should have primary key column.

Here one student can join multiple course, if he join multiple course then we have to store duplication record in the database, for one student course is repeating multiple time, after analyzing we going to devide the Row_materials into two table which will not repeat the same record.

Now one student can join multiple courses. just place student_id into student_course_faculty table and it referencing to the student table.



ROW_MATETIALS
STUDENT_ID
STUDENT_NAME
DOB
GENDER
HIGHEST_DEGREE
COURSE_ID
COURSE_NM
DURATION
FEES
FACULTY_ID
FACULTY_NM
QUALIFICATION
EXPERIENCE

	STUDENT
PK	STUDENT_ID
	FIRST_NM
	LAST_NM
	DOB
	GENDER
	HIGHEST_DEGREE

	STUDENT_COURSE_FACULTY
PK,FK	STUDENT_ID
PK	COURSE_ID
	COURSE_NM
	DURATION
	FEES
	FACULTY_ID
	FACULTY_FIRSR_NM
	FACULTY_LAST_NM
	QUALIFICATION
	EXPERIENCE

- Hibernate 18,19,20,21 are the property usecase which is in excel format. Plz refer excel file.
- Hibernate 22 , 23 are How to create the ER Diagram using MYSQL workbench and discussion of joins.
- Hiberntae 24 Customer support center usecase. Which is in excel format .

Hibernate 25 and 26 and 27

Mapping information:

```
class Customer{//entity class
        private int customerID;
        private String customer_FN;
        private String customer_LM
        private Date dob;
        private String gender;
        //setter and getters
}
```

- Customer is the entity class B’z the class is going to store the exact table related columns as attributes in it.
- Entity class used to hold the data we can’t write any business logic or database logic. If we do so its lead to problems in the application , b’z application contains number of classes and they also want the data. So it again programmer has to write duplicate code for same job.
- Even if we develop a mapping logic its to tricky to write mapping.
- While retrieving the data from relational model it may come from more then one table . To handle and write such logic will lead to the maintenance problems and boiler plat logic.

- Hibernate provided a mapping configuration file which is avoid all the boiler plat logic
- We can avoid connection, preparedStatement logic , closing the connection, try-catch block, so on. It is handled by hibernate mapping file.
- But here is the programmer job to tell what to map with which table and which class.
- Hibernate provided a structure which is hierarchical format and follow the dtd which is provided by the hibernate people.
- Programmer has to map class name , table name, type of the data and also specify the id which is unique to every table i.e. primary key column.
- A project may contains 25 entity classes so programmer has to map all the entity classes in to the mapping file, but the problem is we can write all entity classes in one single mapping file but it lead very huge mapping file .
- Every class contains no. of attributes so it is not possible to write all the entity in the one hibernate mapping file.
- We have to write separate mapping file for each and every class with specific file name corresponding table name and column names.
- We can give any name to mapping file but recommended to use only hibernate provided convention i.e **<className>.hbm.xml .**

- Let see the hibernate mapping file

```
Customer.hbm.xml
=================
<hibernate-mapping>
    <class name=“customer  table=“CUSTOMER”>
            <id=“customerid column=“CUSTOMER_ID” type=“int”>
            <property name=“customerFN” column=“CUSTOMER_FN”
type=“String”>
            <property name=“customerLN” column=“CUSTOMER_LN”
type=“String”>
            <property name=“dob” column=“DOB” type=“Date”>
            <property name=“gender” column=“GENDER” type=“String”>
</hibernate-mapping>
```

Just understand the uses of the tags which is given by hibernate dtd.

Class, name, table, id, property, type column these all are tags and tags properties which is given by hibernate. Using these tags we have to map the entity class to table.

Hibernate 28

- As per the previous class having mapping file is not enough we b'z we are mapping a table with class and columns with attributes but to perform the any operation we want a connection.
- To get the connection have to pass the database information to the mapping file. But if there are 10 classes are there who wanted to perform the database operation. Then we have to write 10 mapping files with same database connection details.
- Means we are end up with duplicating the database information.
- To avoid such duplication hibernate has provided a configuration file I.e hibernate.cfg.xml which is common for all the mapping files. If we are performing operation with one database write one hibernate configuration file and if we are using multiple databases then we have to write multiple hibernate configuration file.
- A every configuration file contains a dialect which is very important while performing the database opertion.
- Dialect is the one of interface which is connect the underlying database and programmer using database.

- In hibernate very database contains a separate dialect to perform the corresponding operation.
- What is the need of dialects in the hibernate?

Ex.

```
<hibernate-mappings>
        <class>….
        <propery name=“dob” column=“DOB” type=“date”>
        ……</class>
</hibernate-mappings>
```

In the above example a property with date datatype which is may not be same to every databases. Every database may varied from another database.

- Oracle may contains date , mysql contains timeStamp, MsAccess may contains different .
- Hibernate using his own provided datatype i.e. date and depends on the database dialects going to convert the date format in to the java understandable format.
- Means if there is changes in database there is no effect on mapping file .
- Dialect works many places we will see later.

- Let see how we are duplicating the database configuration details if we are using multiple mapping files.

Customer.hbm.xml

```
<hibernate-mappings>
        <class>.............</class>
</hibernatemappings>
<hibernate-configuration>
        <session-factory>
        <property name=connection.driver_className>OracleDriver</property>
<property name=connection.url>jdbc:oracle:thin:@localhost:1521:xe</property>
<property name=connection.username>hbm_user</property>
<property name=connection.password>hbm_pass</property>
<property name=hibernate.dialect>oracle10gDialect</property>
</hibernate-configuration>
```

employee.hbm.xml

```
<hibernate-mappings>
        <class>.............</class>
</hibernatemappings>
<hibernate-configuration>
        <property name=connection.driver_className>OracleDriver</property>
<property name=connectionurl>jdbc:oracle:thin:@localhost:1521:xe</property>
<property name=connection.username>hbm_user</property>
<property name=connection.password>hbm_pass</property>
<property name=hibernate.dialect>oracle10gDialect</property>
</hibernate-configuration>
```

To avoid such problem hibernate provided a configuration file we can write database details in one place.

- **Session:**
- Session is a state which contains mapping and configuration information.

Session is the special class which take care of mapping the hibernate file info and configuration file info and creating the connection to perform the DB operation, but it is wrong bz if Session is used for all these thing then there are many classes are want to perform the database operation then every time Session going to created and very new operation hibernate mapping , configuration , connection has to created means if there are 25 classes wanted to perform the database operation means 25 time session going to create and 25 time he has to lead the same file. So it is some what tedious job for session.

If session is doing this job the we end up with duplicating and same mapping and configuration file.

So don’t create a Session your own ask someone to create the session.

- SessionFactory:
- Factories are the classes which going to create the object of other class. But main advantage is hiding the complexity of creating the object of another class.
- In hibernate also there is a Factory class called as SessionFactory which is able to create the session object.
- In SessionFactory we are giving the mapping file , configuration file information, by that sessionFactory going to create the session object. And all the required information with the session Factory there is no need to create multiple sessionFactory. One sessionFactory can share the same information to the created session object.

Hibernate 29

- If a wanted to create the session object must and should mapping information and configuration information is crucial. We have to pass this information to the SessionFactory to create the object of session.
- If create a empty sessionFactory object then there is no use, we want the all mapping and configuration inforamation injected to the sessionFactory object.
- Bz session object required the mapping and configuration information to generate the corrosponding sql query by help of Dialect.
- So it is the programmer responsibility to pass all mapping and configuration information to sessionFactory.
- As per the below example we can able see how the happing information build with the configuration file. There is a reason to build the mapping file with the configuration file.
- One configuration file for one database environment and the all corresponding entities work with same DBEnv, so always database is same tables are same, columns are same and attributes also same.
- Rather to configure separate mapping to the sessionFactory better to configure in the configuration file which is single for every database. Bz there is no need to pass every mapping file to the sessionFactory every time.
- While creating the object of sessionFactory, sessionFactory will take input as a configuration file and store as a metadata in SessionFactory registry for future reference.
- Whenever we create the session object there is no need to read the physical mapping and configuration file rather than we can read it from the SessionFctory Registry .

```
class Customer{
private int customerid;
private String customerName;
....
}
```

Entity class

```
customer.hbm.xml
<!xml version=1.0 encoding="utf-8">
<!DOCTYPE HIBERNATE DTD......./>
<hibernate-mappings>
  <class  name="Customer" table="CUSTOMER">
   <id="Customerid" column="CUSTOMER_ID" type="int"/>
   <property name="CustomerName" column="CUSTOMER_NM" type="string">
     ........
  </class>
</hibernate-mappings>
```

Hibernate mapping File

```
hibernate.cfg.xml
<hibernate-configuration>
  <session - Factory>
   <property name="connection.driverclass_name>OracleDriver</property>
   <property name="connection.url>jdbc:oracle:thin:@localhost:1521:xe</p
   <property name="connection.username>hbm_usr</property>
   <property name="connection.password>hbm_pwd</property>
   <property name="hibernate.dialect>oracle10gDialect</property>
  </session - Factory>
    <mapping resourse="com.hf.entities.Customer.hbm.xml/>
 </hibernate-configuraion>
```

Database Env info

Dialect information

mapping information

Hibernate Configuration File

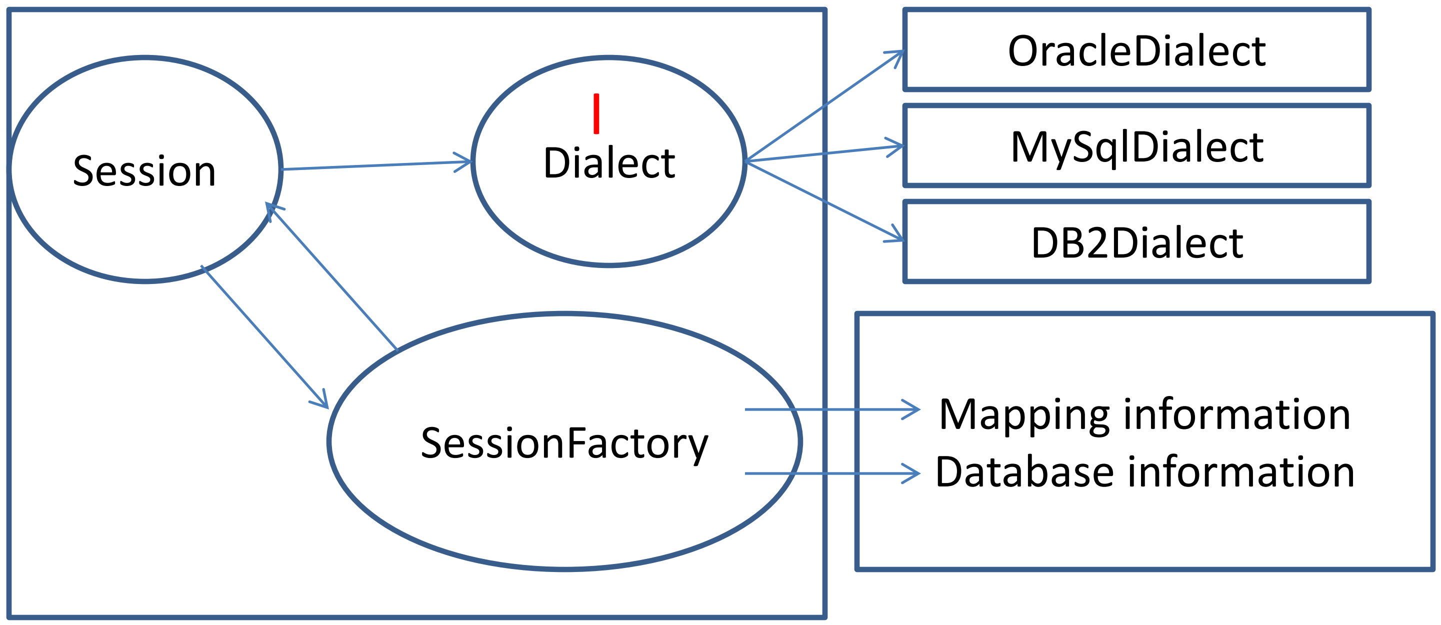
by Mr.sachin gaikwad

How to attach with sessionfactory by example:

```
Class customerTest{
        p s v m ()
        {
          Configuration cong = new Configuaration();
               cong.configure();
        SessionFactory sfactory =
                                cong.buildSessionFactory();
        Session session = sfactory.getSessionFactory();
        ……………………
        }
}
```

- **Dialect**:
- In hibernate dialect is the concept is use to translate HQL to underlying database and underlying database to HQL;
- Hibernate is the database independent framework bz of only dialect.
- Dialect contains all the syntaxes, structure of the databases depends on that dialect act like that.
- Hibernate provided a type date, so depend up on the underlying database, hibernate dialect get a particular date type structure and convert into that type.
- Some time a application may change one database to another database, it is easy to change the configuration details but it is very hard to change the underlying queries. Bz very database may have different query structure at that time we have to change it manual.
- But if you are using hibernate there is no need to change the queries manually just only change your database details and hiberntae-dialect . Hibernate – dialects will automatically perform conversation.

Internal work of Dialects



As per the diagram session object is created by help of sessionFactory. sessionFactory contains all the information about mapping and Configuration information and it is giving to the session object to create the connection and persist the data.

- But session object create the connection and talk to the Dialect to create the query depends upon the configuration file provided database information and convert corresponding database query.
- Here dialect is the interface which having no of implementation classes and depends on configuration file he is going to use particular dialect to generate the query for update, insert, delete, retrieve.

- Factory Design pattern

```
Class  A{
        A(B a){}
}
Class B {
        B(C c){}
}
Class c{
}
Class T1{
        A a = Afactory.createA();
}
Class T2{
        A a = Afactory.createA();
}
```

```
Class Afactory{
Public static A createA(){
C c = new C();
B b = new B(c);
A a = new A (b);
Return a;
}

}
```

In the above design pattern we are avoid the complexity to create the object of another class by this design pattern.

- Builder Design pattern : It is used to create the object of another class. It is same like Strategy design pattern but strategy design pattern going to create empty object, it is depends upon the requirement.
- Here BDP is a special DP which will take the input from the other classes and inject that information with the particular object and return the object.
- For example.
- Session object need the information about mapping and configuration, if we create empty sessionFactory object and depends upon we will create session object then there is no use. Bz sessionFactory don't have mapping and configuration information.
- Here sessionfactory going to take input i.e. mapping and configuration information and depends on that information sessionFactory object inject that information in it.
- While creating the object of Session it will give the internal information to the Session object. Then only session going to inject with Dialect to perform the operation.

- Example

```
Configuration configuration = new Configuration();
configuration.configure();
SessionFactory sFactory =
configuration.buildSessionFactory();
Session session = sFactory.getSessionFactory();
……………………….
```

- If you see the sessionFactory class also taking a configuration file as a input and injecting with the session.
- Configuration and mapping information build with the sessionfactory.
- While createing session object we will get all the required information from the sessionFactory object.

Hibernate 30

- What is sessionFactory?
- What is session?
- Why we have to use sessionFactory?
- What is configuration file? Why we have to use configuration file?
- What is mapping file and why we have to use mapping file?
- How to write the configuration and mapping files in to hibernate?
- What is dialect? What is the use to dialect in hibernate?

- We covered all the basic which is required to demonstrate the first hibernate example.
- Let see the first hibernate example .

- First we have to create the entities which are reflects the same tables which are in the database.
- It is mandatory to have same attributes as same as table columns with there same data types.
- A entity represent the table in the database.
- Let first create the table customer

```
CREATE TABLE CUSTOMER(
        CUSTOMER_ID NUMBER(5) CONSTRAINT PK_CID PRIMARY KEY,
        FIRST_NAME VARCHAR2(25),
        LAST_NAME VARCHAR2(25),
        DOB DATE,
        GENDER VARCHAR2(6)
);
```

- A table must and should contains a primary key column bz, while writing the mapping file id attribute is mandatory.
- While writing the entity class make sure all the type same as table column types only.

```
Class Customer{ => entity class
		private int customerId;
		private String firstName;
		private String lastName;
		private String dob;
		private String gender;
		//setter and getters
}
```

Customer.hbm.xml
================

- <?xml version="1.0"?>
- <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
- "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
- <!-- Generated Sep 29, 2015 9:59:47 AM by Hibernate Tools 3.4.0.CR1 -->
- <hibernate-mapping>
-     <class name="com.fh.entities.Customer" table="CUSTOMER">
-         <id name="customerId" column="CUSTOMER_ID" />
-         <property name="firstName" column="FIRST_NAME"></property>
-         <property name="lastName" column="LAST_NAME"></property>
-         <property name="dob" column="DOB" type="date"></property>
-        <property name="gender" column="GENDER"></property>
-     </class>
- </hibernate-mapping>

Hibernate.cfg.xml ===➔ This is the common configuration file
=================

- <?xml version=*"1.0" encoding="UTF-8"?>*
- <!DOCTYPE hibernate-configuration PUBLIC
- "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
- "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
- <hibernate-configuration>
- <session-factory>
- <property name=*"dialect">org.hibernate.dialect.Oracle10gDialect</property>*
- <property name=*"connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>*
- <property name=*"connection.username">system</property>*
- <property name=*"connection.password">system</property>*
- <property name=*"connection.driver_class">oracle.jdbc.driver.OracleDriver</property>*
- <mapping resource=*"com/fh/entities/Customer.hbm.xml"/>*
- </session-factory>
- </hibernate-configuration>

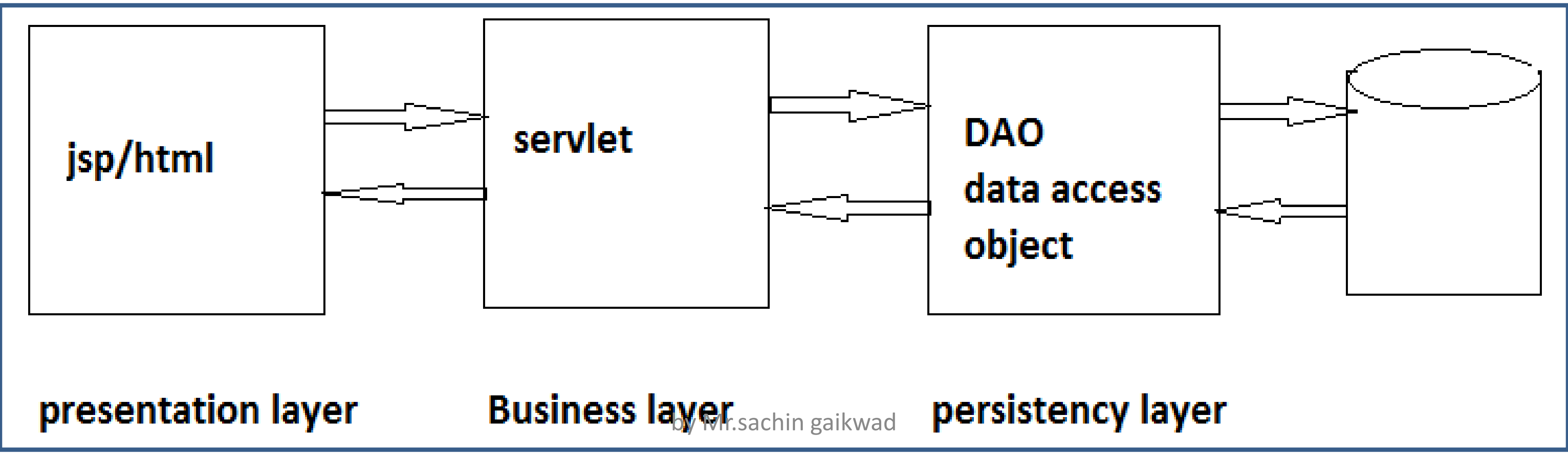
To compile and run the application write test class and configure the configuration file and get the sessionFactory to create Session object.

- **public class FHTest {**
- **public static void main(String[] args) {**

- Configuration configuration = **new Configuration();**
- configuration.configure();//new File("A:\\Spring Data\\Hibernate Data\\HibernateWorkplace\\FirstHibernate\\src\\com\\fh\\entities\\hibernate.cfg.xml"));
- SessionFactory factory = configuration.~~buildSessionFactory();~~
- Session session = factory.openSession();
- session.beginTransaction();
- Customer customer = **new Customer();**
- customer.setCustomerId(5);
- customer.setFirstName("black");
- customer.setLastName("ss");
- customer.setDob(**new Date());**
- customer.setGender("Male");
- session.save(customer);
- System.*out.println(customer);*
- session.getTransaction().commit();
- }
- }

Hibernate 31,32

- while dealing with application development, we have to know about the what are the phase / tier are available
  - Presentation tier layer
  - Business tier layer
  - Persistency tier layer



- Why DAO?

There are several reasons if we write our persistency related logic in business logic which impact our business logic, there is change in table or database then we have to rewrite all the business logic .

If there are 10 servlets available in the project so we have to write persistency logic in each every servlet. In every servlet we have to create connection and close the servlet.

Again there are so many problems ……..

- DAO(Data Access Object)
- As per the above discussion we have to use DAO in our application. To make our application loosely coupled from the persistency logic.
- DAO are the classes which contains only persistency related logic, they going to interact to the database.
- DAO represents one layer which reflects the tables in the database. we can easily interact with the DAO.

- While developing an application we have to follow the separation of concern principle, which divide our application in three layer.
  - Presentation layer
  - Business layer
  - Persistency layer

Hibernate 33

- Singleton Design pattern:

A class is going to allow you to create only one object of a class called as singleton class.

- There are several reasons available why we are creating singleton class.
- In some cases a object or a configuration common to whole the application.
- If every one going to create the object of common thing then is it duplicating among the application, and we are wasting JVM memory.
- If there is common requirement is there then create a singleton class which going to share same object through out the application.
- Let see the example of how to create SessionFactory as singleton , bz every class want sessionFactory object to create a session object and to interact with the database.

Singleton design pattern

```
Class HibernateSessionFactory{
            private SessionFactory sessionFactory;
            private HibernateSessionFactory(){}
            public static SessionFactory getSessionFactory(){
              if(sessionFactory==null){
                 CFG cfg = new CFG().configure();
                sessionFactory = cfg.buildSF();
                }
                        return sessionFactory;
            }
```

While we are dealing with normal application we can use such kind of singleton design pattern.

Let see the singletonDesignPattern example in STS you can understand what are the problems with the singleton DP.

Hibernate 34,35

- We completed a second hibernate application and how to create sessionFactory by using singleton design pattern. To understand second hibernate example refer singletonDesignPattern example in IDE.
- As per previous class we well understood what is the use of singleton Design pattern, when to use, where use and so on.
- In hibernate configuration files are plays vital role to get the database environment to work with the hibernate.
- There are three BootStrapping hibernate configuration files availables in the hibernate
  - Hibernate.cfg.xml
  - Hibernate.properties
  - Programatic approach

These file are very important while working with hibernate. Bz these files contains configuration information.

- Why hibernate has given three configuration files?
- While developing any application there are different types of requirement come across, there are some situation programmer has to use these configuration files.
- 1)hibernate.cfg.xml
- Most of the time industries going to use only cfg.xml file only there are sort of benefits available
  - Cfg.xml file has hierarchical structure, we can easily read and write it.
  - Pre-validation is available.
  - We can protect our application before executed, bz hibernate going to check each and every attribute in your cfg.xml file.
- There are some drawbacks also available
  - We should have to remember all dtd type.
  - All the complicated attributes and there values
  - And structure, how to represent in the cfg.xml.
- By default a configuration name is hibernate.cfg.xml but as per programmer requirement we can change name of the configuration file.
- There may be chance to create multiple cfg.xml file also , bz it may possible one project deal with multiple database. And we can place cfg.xml file in other package also but while configuration we have to mention the package path to the configure() method.
- Ex; configuration cfg = new configuration.configure("com/bs/common/customer.cfg.xml);

- 2). Hibernate.properties
- By help of hibernate.properties we can configure the configuration details to the hibernate.
- When to use hibernate.properties approach in the project. When there is piece of code (POC) development then we can use hibernate.properties file.
- In this approach we will get any dtd and pre-validation feature.
- Hibernate.properties should be the name to the file.
- It should be placed under classpath location only.
- While configuration there is no need to configure the hibernate.properties file.
- Bz hibernate automatically find and fetch the hibernate.properties file into classpath location and load it.
- If hibernate.properties file is not available even though hibernate create empty configuration object which never tell you properties file has been loaded or not.
- We can not add mapping information into the hibernate.properties file bz we can't store with same key multiple value.
- So we have to add externally mapping file to the configuration object .
- There is a method in configuration class i.e. addResource("mapping file path");//addClass("classname.class");
- If there is small application or for testing purpose we can use hibernate.properties approach .
- Properties file contains key and value

- For example :

```
hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.url=jdbc:oracle:thin:@localhost:1521:xe
hibernate.connection.username=hibernate
hibernate.connection.password=hibernate
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
```

Hibernate 36

- 3).Programmatic approach:
- This approach is rarely used while development of application.
- By example we can understand this concept.

If there is old legacy application which is developed in old language and my requirement is to add the new components in the legacy application. But now I want to use hibernate instead of old technology.

But in that legacy application there is a property file containing some sort of data. For example

Db.properties

```
============
con.driverClass=oracle.jdbc.driver.OracleDriver
con.url=jdbc:oracle:thin:@localhost:1521:xe
con.username=hibernate
con.password=hibernate
```

So I wanted to use same properties file with hibernate keys.

If I write another properties file it will may duplicated in the same project.

So by help of programmatic approach I can use db.properties file..

- For example:

```
static {
Properties props = null;
Configuration configuration = null;
props = new Properties();
try {
props.load(HibernateUtil.class.getClassLoader()
.getResourceAsStream("db.properties"));
configuration = new Configuration();
configuration.setProperty("hibernate.connection.driver_class",
props.getProperty("con.driverClass"));
configuration.setProperty("hibernate.connection.url",
props.getProperty("con.url"));
configuration.setProperty("hibernate.connection.username",props.getProperty("con.username"
));
configuration.setProperty("hibernate.connection.password",
props.getProperty("con.password"));
configuration.addResource("com/bsp/entities/TicketBooking.hbm.xml");
sessionFactory = configuration.~~buildSessionFactory();~~
} catch (IOException e) {
e.printStackTrace();
}
}
```

- Even without cfg.xml and hibernate.properties file we can configure the hibernate configuration by help of programmatic approach.
- As per given example we can easily deals with the programmatic approach.
- But most of the time industries and programmer follows the cfg.xml configuration.
- Programmatic approach hasn’t provide any validation, or dtd for configuration.

Hibernate 37,38

- **Get Vs Load:**
- Session class contains number of methods and every method used for specific purpose.
- Get and Load both are session class method used for retrieving the data from the database.
- Get and Load method going to return data in the object format.
- There are number of differences available in get and load methods.
- **Get() :** get() is the eager initializer method.
- Syntax : get(class, primary key id);
- Get() is the special method which is used for retrieving the data from the database, get going to take two parameter as input i.e. class name and a primary key id.
- Internally hibernate going to take that class name and going to find the canonical name of the class, after that hibernate matches that class name with the mapping file which is available into the classpath.
- according to the mapping information it will go to the table and fetch the one record and return to the session object, session object going to return the object to the corresponding class.
- But there are several disadvantages available.

- 1) get() method is eager initializer mean once it get the call it will go to the underlying database and fetch all the records, rather than particular records.
- If there is a association between two classes and if we call the get method, it will load current class data as well as associative class data also, even we are not certainty sure we will use that data or not.
- 2).It will forcibly consume more amount of memory in the JVM.
- 3).it will decrease the performance.
- 4).it will consume the more bandwidth while transformation.
- Get() method will return the actual class object to the session.
- For example:

```
Customer  customer  =(Customer)
                      session.get(Customer.class,101);
```

- Here session class going to return the actual class object. By type casting we are able to use the Customer class object.

```
System.out.println(customer);
```

- There are sort of procedure available how the session class get() method mapping the Customer.hbm.xml and there corresponding attributes with columns.

- **Load():** it is lazy initializer.
- Load() method also present in the session class only, both get() and load() method used for retrieving the data from the database.
- Why load method is called as lazy initializer?
- Syntax: load(class, primary key id);
- Both get() and load() method having same parameter but load() method will not go directly to the database.
- Load() method will take the input and hibernate will return the object to the load method which contains prepopulated id value.
- But internally hibernate will call the javaassists package, which is executed while bytecode optimization time. Hibernate will take help of javaassists and generate one proxy class which is same as original class.
- Proxy class contains all the original class properties.
- When programmer call the session class load () method it will return proxy class object which is look like same as original one with prepopulated id.
- Actually load () method fetch the data on demand, mean when we call a particular method , then load() method going to fetch the data from the database, but fetching the data will going to happen into the proxy class and we are calling proxy class methods only.

- For exampele

  Event event = (event)session.load(Event.class,10);

In the above statement a load() returning the proxy class object which contains prepopulated data i.e. id. After getting event object we are calling the methods

For example :

```
event.getId();
event.getEventName();
```

Here we called the methods but these method belongs to the Event$Proxy class.

- After getting call Event$Proxy class going to execute

Some special method these method will going to fetch the data from database and going to set the value to the Event$Proxy class method.

For example:

```
Class Event$Proxy extends Event{
            private object id;
            private Event event;
            public Event$Proxy( object id){
                        this.id= id;
                        event = new Event();
            }
            //setter and getter
            //business logic which is going to connect to the mapping file and
validating the data has been populated or not .
            //special methods which contains logic related to separating the
association between the classes.
}
```

- Load() method going to fetch only current class records even association presents into the classes.
- In performance load() method is good.
- It will not consume more memory into the JVM.
- Depends on the demand it will fetch the data.
- It will fetch one record at a time.

Hibernate 39,40

- Internally how the proxy going to work lets see the example.
- There is InvocationHandler interface which is present under java.lang.reflect.InvocationHandler.
- Which contains only one method i.e.

public object invoke(object obj,Method m1,Object[] args)

The above method used by the proxy to invoke the proxy method.

There are three parameters are available which going to take as following

1. Object of the current classLoader
2. object of the interface (for all method)
3. It take all arguments .

Example:

while creating the proxy object we going to pass these parameters to the InvocationHandler.

Account proxy = Proxy.newProxyInstance(Account.class.getClassLoader(),account.class.getClass.getInterface(),new Invocationhandler (new object));

Lets see the example:

•

```
package com.pa.entities;
public interface Account {
    float getBalance();
    void setBalance(float balance);
}
```

```
package com.pa.entities;
public class SavingAccount
implements Account{
    private float balance;
    @Override
    public float getBalance() {
        return balance;
    }
    @Override
    public void setBalance(float
balance) {
        this.balance = balance;
    }
}
```

```
package com.pa.helper;
public class AccountHelper implements InvocationHandler{
        private Object target;
        public AccountHelper(Object target) {
                this.target = target;
        }
        @Override
        public Object invoke(Object proxy, Method method, Object[]
arg1)
                        throws Throwable {
        System.out.println(proxy.getClass().getName());
                System.out.println(method.getName());
                System.out.println( Arrays.toString(arg1));

                if(method.getName().equals("getBalance")){
                        System.out.println("Attached to the database");
                        System.out.println("fetch the data");
                        System.out.println("set to the original
method");
                        ((Account) target).setBalance(10.0f);
                }
                Object result=method.invoke(target, arg1);
                return result;
        }
}
```

```
public class ASTest {
public static void
main(String[] args) {

    Account account = new
SavingAccount();
    Account proxy =
(Account)
Proxy.newProxyInstance(Savi
ngAccount.class.getClassLoa
der(), new Class[]
{Account.class},new
AccountHelper(account));
    System.out.println("Usi
ng proxy call to the
getBalance");
    System.out.println(prox
y.getBalance());
    }
}
```

Proxy example

- Some internal details about how the proxy going generate

```
Class session{
	public Object Load(Class classType, Object id) {
	return newProxyInstance(classType.class, new
Class[]{classType.class},new InvocationHanlder(new Event(id));
}
```

The above code is internal code….for understanding purpose.

Proxy used by pre-validation, to avoid boiler plat logic, etc.

So better understanding just refer the examples which are in the eclipes IDE.

- if data is not available into the table then get() method will return null . But load() method throws an exception.
- In table data is there but some column data are not available and we are trying to fetch the data then we will get the runtime exception.

- Rule while writing the entities
  - Entity must contains default constructor
  - Entity contains only accusers method it should not contains business logic or other logic.
  - Entity should not be final.
  - Entity should not contains parameterized constructor, if it is there then we can not perform save operation.
  - Entity should not contains private constructor, hibernate can create the object but we can not create the object.
  - Entity contains equals() and hashCode() method.
  - Entity should be implemented by Serializable interface. Even we are not implements, we can write the entity no problem.
  - Even if we declared all attributes as private no problems we can easily access and set the values to these attributes form outside class , b'z of setter and getter (accusers method).

Hibernate 41

- We have to follow certain rules while writing the entities classes. If our entities are final then hibernate will not create proxy for the entities, rather it will return original class object only.
- Even our entity class final we can create the proxy, but we have to follow certain procedure.
- Create one interface and your entity should be implements to the interface and override all the methods.
- While calling session.load(.,.) create the interface reference and typecast into interface variable only.

```
Interface Ilaptop{
		//methods declaration
}
Final Class Laptop{
		//override  all the methods
		}
Class LaptopDao{
		//create sessionFactory
		//create session
		Ilaptop laptop;
		public void getData(int id){
		laptop = (Ilaptop)session.load(Laptop.class,id);
		syso(laptop.getName());
		}
}
```

- After creating Laptopdao class make sure while creating your mapping file write one extra tag which help hibernate to create the proxy of your class.
- Ex:

```
        <class name=“com.gl.Laptop” table =“LAPTOP”
proxy=“com.gl.Illaptop”>

        ………..

        </class>
```

For more understanding just see GetAndLaod example in eclipse.

Hibernate 42

- Hibernate Tools
  - Graphical user interface
    - Hibernate Configuration Tools
    - Hibernate Mapping Tools
    - hibernate console Tools / Query Editor
    - Reverse Engineering Tools
  - Command-line utilities
    - Schema Export
    - Schema Visibility
    - Schema update
- Hibernate configuration Tools:
  - We are manually writing the configuration information information into the cfg.xml file , But hibernate has provided the hibernate configuration tool, so we can write configuration file using graphical user interface.
- Hibernate Mapping tools :
  - Even we can create mapping file using graphical user interface but it will assume that your entity attributes are same as the table column names.
- Hibernate console file/tools:
  - Hibernate has provided the HQL console to query the data from the table but we have to query data in HQL format.
- Reverse Engineering Tools
  - It's a tool which help more efficiently to the programmers. If you given table to this tool it will generate mapping file as well as entities class related to the table column attributes.

Hibernate 43

by Mr.sachin gaikwad

Hibernate 44

by Mr.sachin gaikwad

Hibernate 45

Hibernate 46

by Mr.sachin gaikwad

Hibernate 47

- Most of the object oriented languages use XML to make application loosely coupled.
- But XML speak more and it has its own limitations
- If we want to use the xml we have to know the structure of the xml.
- We have to remember the all the tags.
- We have to follow some xsd or dtd type structure.
- Because of the above reasons peoples don't show there interest in xml.
- So third party community has provided a concept called Xdoc. Which is used for providing the configuration but it doesn't carry long period to time. within short period it has fails.
- So most of the third party community observed these thing and they come with a new concept called annotation.
- Annotation is the language specific configuration feature.
- Now there are two ways to configuration one is xml based configuration and second one is annotation based configuration.

- From Initial days onwards java doesn't has support for annotation, but from JDK 5 java has supported a concept called Annotation.
- Why Annotation came in feature?
- =>As we discussed above the pit point about the xml.
- JPA(java persistence API) : first java does not has support for ORM technology, but rest of the all programming languages shown there interest to the ORM technology, so people started loosing the interest on java. B'z of that java has provided one API called JPA(java persistence api). Which has the support for ORM technology.
  - JPA –ri(implementation): JPA-RI is the implementation provided by java people to the other people to add the support for jpa.
- Annotation type:
  - Logical mapping annotation
  - Physical mapping annotation
- Logical mapping annotation :
  - Ex. : @Entity, @ID , an annotation talks about logical thing it doesn't has any physical existence called logical annotation.
- Physical Mapping Annotation:
  - Ex.: @Table(name="Event"), @Column(name="Event_Id"), these two annotation talks about the physical existence , means table is present into the database and columns also present into the table.
- Native hibernate library
- Annotation with native JPA
- Annotation with Hibernate library

- Some of the Annotations in hibernate:
  - @Entity:(To make our class pertistence and pojo)
  - @Table(name="tableName"): (map corresponding table to the pojo class)
  - @Id: (to specify primary column attribute)
  - @Basic (assume attribute name as column name)
  - @Column(name="columnName", size=" XXX" sqlType="XXX") : To map with specific column with different column name)
  - @Unique : (To apply unique constraints to the column)
  - @UniqueConstraint() : (if we want to give unique constraints more than one column we can use this annotation)
  - @Basic(fetch=fetchType.Lazy) : (we can make a particular column as lazy loading column by specifying this annotation.
  - @Transient : if we don't want to stored column into table so we can use this annotation, actually for reference purpose we can use.
- Two ways to provide the annotation
  - Attribute level
  - Method level
- One of the way we can use only , we cant use both the way to configure our class.
- If we use both then it will take one of them, but it will not take both.
- Mostly Attribute level annotations are used.
- The mandatory annotations are @Entity and @Id, if we provide these two then others attributes names are considered as column names by default.

- jpa vs hibernate
- =================
- Jap is the java provided api which is used to work with ORM technology.
- Hibernate has greater support for JPA.
- most of the annotations of the jpa and hibernate are same.

- Drawbacks with ORM Technologies
- Coming to the annotation concept every one has there own annotation.
- every framework provided the support for annotation but they are not same across the technology.
- every framework has to provide the xml support for configuaration and mapping.
- but every one has there owm standard to provide the configuaration and mapping.

- Because of that we can not easily switch from one frameworks to another frameworks.

- But now a days every one has the support for JPA annotation and JPA annotation is the standard acorss the platform. we no need to change the mapping or configuation when we are using JPA Annotation.

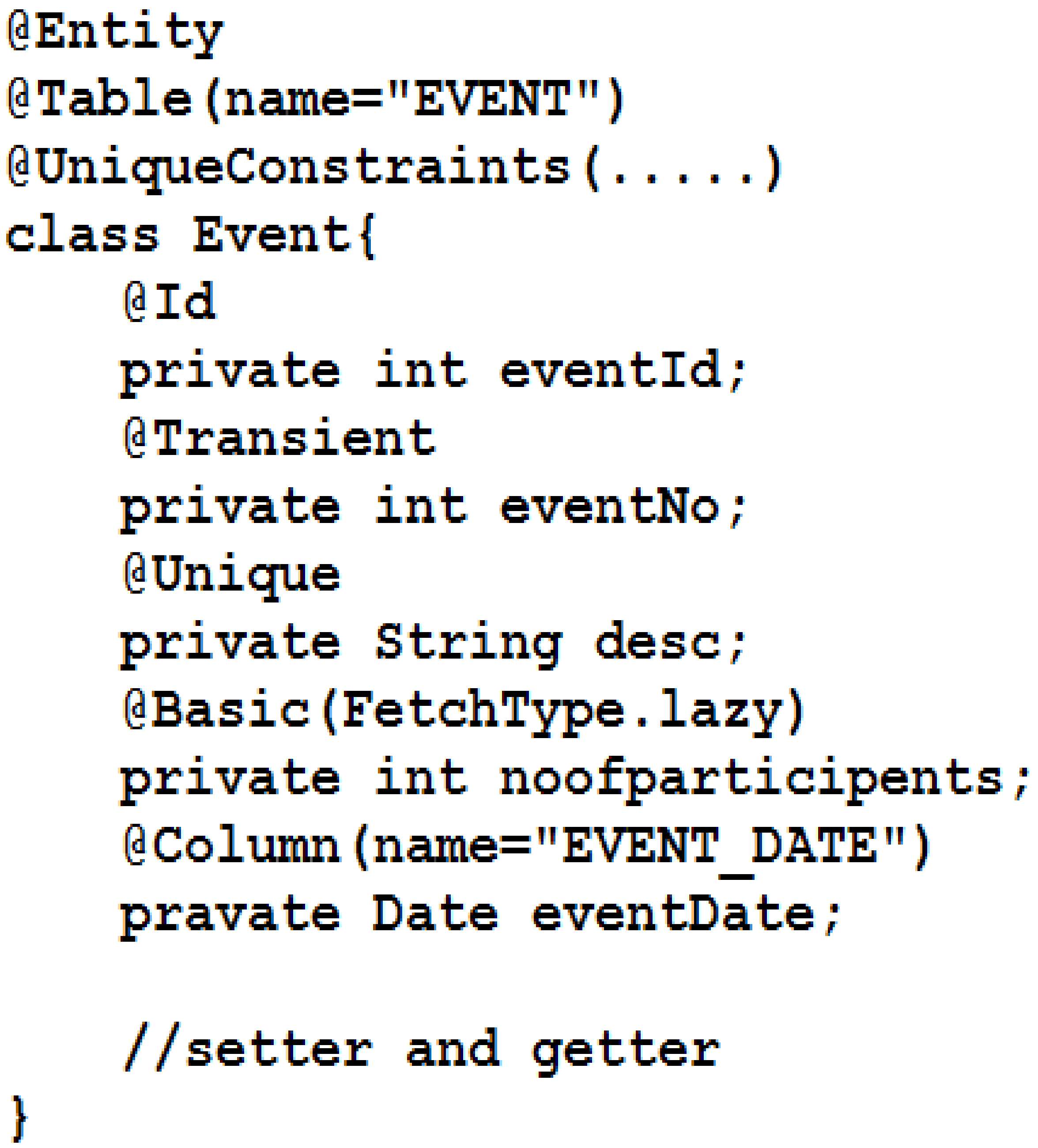
- while working with the hibernate annotation we can mix JPA annotation, also we can mix annotation and configuaration information. both we can use.

- without writing configuaration and mapping information we can work with the hibernate, but we have to use the programmatic approach to pass the cfg and mapping information to hibernate.
-
- Both way we can provide the mapping information to the hibernate,

- 1). By annotation
- 2). By mapping file(hbm.xml)
- if we specified both into the configuration then hibernate by default take hbm.xml only.
- To make hibernate to take annotation entity class the specify the precedence into the configuration file.

- Let see the example

```
(1)
@Entity
@Table(name="EVENT")
Class Event{
        @Id
        private int eventId;
        @Column(name="DESCR")
        private String description;
        ……
        //setter and getters
}
(2)
 @Entity
@Table(name="EVENT")
Class Event{
        private int eventId;
        private String description;
        ……
        @Id
        public int getEventID(){}
        public int setEventID(){}
        @Column(name="DESCR")
        public String getDescription(){}
        public void setDescription(){}

}
```



```
@Entity
@Table(name="EVENT")
@UniqueConstraints(.....)
class Event{
    @Id
    private int eventId;
    @Transient
    private int eventNo;
    @Unique
    private String desc;
    @Basic(FetchType.lazy)
    private int noofparticipents;
    @Column(name="EVENT_DATE")
    pravate Date eventDate;

    //setter and getter
}
```

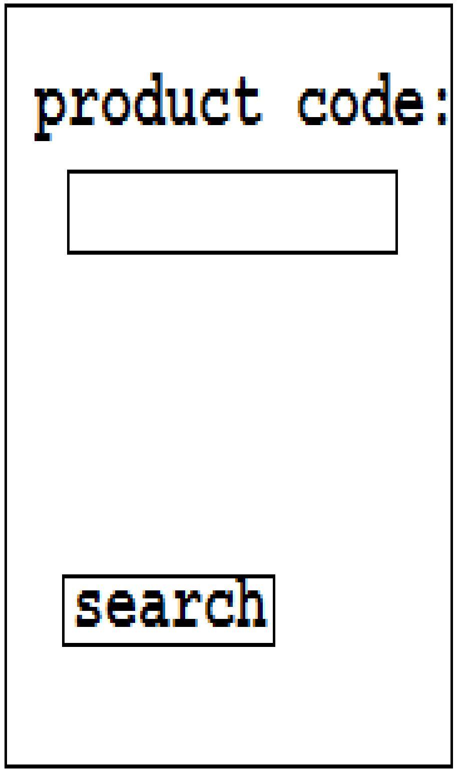
Annotation based Example

Hibernate 48 & 49

-
- First level Cache
- ==================
- Hibernate support three level of cache,
- 1) First level cache
- 2) Second level cache
- 3) Query level cache
- Q. what are the drawbacks, if we develop our application with our cache?
- ==> There are several problems we have to face if we wont follow the cache concept
- 1) Performance issue
- 2) Resource management issue
- 3) network bandwidth problems

  4) Database Scalability problems
- Ex:==>
- If we want to develop an e-commerce site, so we have to provide the product information to the customers, if customer choose a product and he requested our servlet then servlet container going to execute our servlet, but problems are if multiple users going to request to the servlet for the same data then it is end up with accessing the data from the database.

•



searchProduct.jsp

scalability problem
performance problem
bandwidth problem
resource mngt prblm

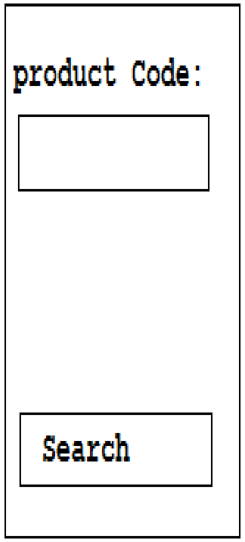
```
class searchRequestProduct extends
HttpServlet{


    public void service(req, resp){

    String productCode=
req.getParameter("productCode");

    //connect to the database
    //search for the data
    //Iterate the data and retrieve it
    // bind to the req scope
    // send to the product_details jsp
}
```

by Mr.sachin gaikwad

- In the above application user going to find the product , after entering the productName or productCode user going to click on search button, and the request received by the servlet Container and container going call appropriate servlet.
- and call the service method of the SearchProductServlet , service method read the jsp page data by request object using getParameter() method . Once the data available with the service method it will connect to the database and perform the query and retrieve the data to the servlet.
- But there are lacs of user want the same data then each and every request end up with executing the same the query one database.
- Every time same query going to execute here creating the database connection is expensive job, and we are sacking the performance of the application.
- Because of executing same query CPU resources not used properly even we are killing the scalability of the application.

- Because of repetitive execution of same query we are wasting bandwidth.
- To overcome we have to use the cache concept into the application.
- Cache will improve the performance of your application.
- According to the above example if we use cache concept to get the product from the database then we easily improve the performance to the application.
- Instead of going every time to the database, go once and get the data and store into the attribute, for feature reference.
- If other request will request for same product then easily we can utilizes the previous data.
- If the coming request is new the it will check and if it is not available it will get the data from the database and store into the cache and sent to target page.

- Cache General example



```
class SearchProductServlet extends HttpServlet{

    public void service(req, resp){

    String productCode = req.getParameter("prodectCode");

        Cache ch = Cache.getInstance();
        if(ch.ContainsKey(productCode)==true){
            product p = product(ch.get(productCode));
        }else
        {
            //connect to the database
            //query the data
            Product p = (Product)//iterate data
            ch.put(productCode,p)
        }
            //bind the data to the request scope
            //send to product details to jsp page
    }

}
```

```
class Cache{
        private static Cache instance;

    Map<Object,Object>  productData;
    private Cache(){}
    public static synchronized Cache getInstance()
{
            if(instance==null){
                instance = new Cache();
        }
        return instance;
    }
    public void put(Object key, Object value){
        productData = new HashMap<Object,Object>
();
        productData.put(key,value);
    }
    public boolean containsKey(Object key){
        return productData.containsKey(key);
    }
    public Object get(Object key){
        return  productData.get(key);
    }
}
```

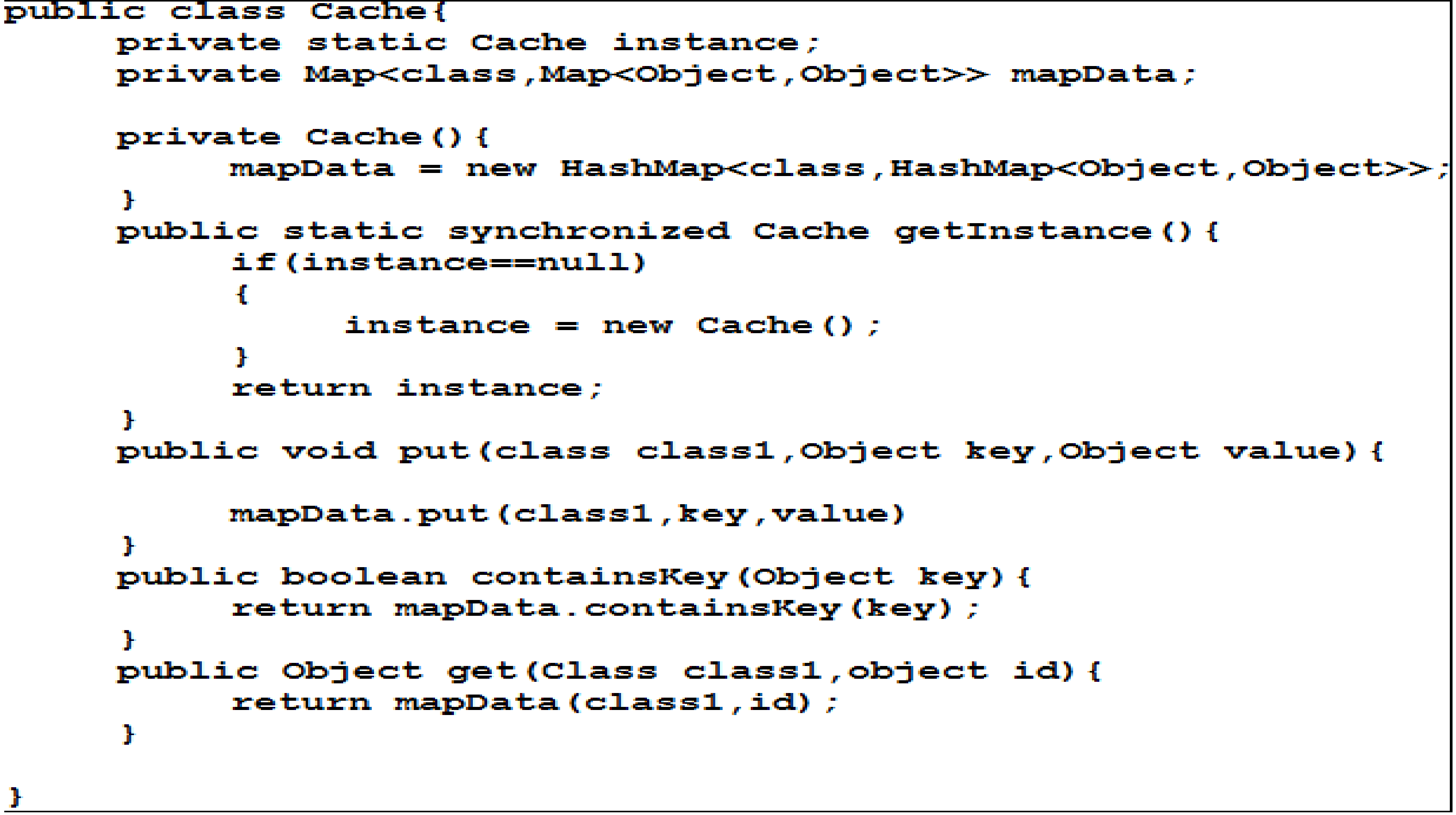
by Mr.sachin gaikwad

Hibernate 50

- In the above example we implemented the simple cache which work with one product item or one similar type of data only, we can’t store heterogeneous data into the HashMap.
- For example a project contains multiple tables and if we want to store data into HashMap it may overlap.
- Ex:
  - HashMap(1,product1)
  - HashMap(2,product2)
  - HashMap(1,customer1)
  - HashMap(2,customer2)
  - HashMap(1,seller1)

  Here in HashMap data overlapping if user want a particular data then it is difficult to get the corresponding data.

- There is one more alternate we can store data into has map but if kill the performance of the application. And to get the data haspMap has to go line by line to search the data.
  - Ex:
    - Product.class+1,productObject1
    - Product.class+2,productObject2
    - Customer.class+1, CustomerObject1
    - Customer.class+2,CustomerObject2
    - Seller.class+1,SellerObject1
  - Here to get the seller object HashMap has to check from first line to the last line to get the data.
  - It is linear search so it will kill the performance of the application.
- So to make over cache more intelligence we have to rewrite the cache with new logic.



```
public class Cache{
        private static Cache instance;
        private Map<class,Map<Object,Object>> mapData;

        private Cache(){
                mapData = new HashMap<class,HashMap<Object,Object>>;
        }
        public static synchronized Cache getInstance(){
                if(instance==null)
                {
                        instance = new Cache();
                }
                return instance;
        }
        public void put(class class1,Object key,Object value){

                mapData.put(class1,key,value)
        }
        public boolean containsKey(Object key){
                return mapData.containsKey(key);
        }
        public Object get(Class class1,object id){
                return mapData(class1,id);
        }

}
```

Above example describe the generalized view of the hibernate

Hibernate 51

- By the generalized example we can optimize the solution over the roundtrip to the database. But the problems is if more then one class want to the same data then every class has to write checking logic in it.
- It has to check data is available into the cache or not and if it is not then go to the database and fetch the data and stored into the cache.
- When to use the cache
  - If data is same around the application and it will change every rarely then go and implements the cache.
  - But if data is changing frequently we should not use cache here .
  - But within a second 200 time same data required and after one second it will change even though we have to implements the Cache. B’z we cant allow the application to go and get the data from the DB for 200 time within second. so it is better to implements the cache.
  - Again there are several places we can use the cache concept.

- Problems with the cache is we cannot maintains two copy of data to work with cache.
- It must be one an the same only then only use going to get correct data.
- If we directly update the data into the DB it will not reflect to the cache b’z both are two different copies of the data.it will change into the cache.
- if user access the data then a class going to check data into the cache and user will get the steal/old data. which is wrong.
- In application perspective both cache data and DB data must be same, if user want to modify the data then user has to modify through the application and as programmer has to write the intelligence logic to make sure both the copies of the data will going modify. i.e. cache data and DB data.
- Actually for every operation we need identifier without identifier we can not deal with cache to database.
  - Fetching the data (need identifier)
  - Update the data(need identifier)
  - Delete the data (need identifier)
  - Save the data(need identifier)
- For each operation we need identifier means identifier is the common for cache as well as DB.
- So we need some one who take the identifier and perform the operation.
- But the operation first reflected into the cache then database because user will not get wrong or steal data.

- We do such kind of operation we need the persistanceManager

```
class PersistanceManager {

      private Cache cache;
      //DB attributes

      public
PersistenceManager(driverManager,con,url,un,pwd)
      {
            //assignment
            cache = getInstance();
      }
      public object get(Class class,object id){
            if(cache.containsKey(id)==true){
            return cache.get(class,id);
            }
            else{
                  //goto the DB and query the data
                  //add into entity object
                  //store into cache
                  return cache.get(class,id);
            }
      }
}
```

- In the above program we can see persistenceManager contains cache object and DB data.
- PM handle the modification of the data will be reflected into two places i.e. cache and DB.
- A get() method in PM will take two parameter entity name and second the identifier.
- It will first check data is available into the cache with corresponding identifier or not, if it is available it will return the same data.
- If data is not available then it will go to the DB and query the data using identifier and stored into the cache and return the entity object along with data.
- Not only get() even it will manages the update, save, delete operation.
- By this we can avoid all the problems while implementing the cache which we discussed earlier.

- How the hibernate session is working?
  - Hibernate session has its own session manager to manage the cache.
  - First level cache or session level cache or object level cache these three represent the same cache concept.
  - When we use session.get(class,id); automatically hibernate open the first level cache and will manage the operations.
  - If subsequently if we call the session.get(class,id);
  - Session will check into session cache and perform the operation.

Ex:

Product p1= (Product)session.get(Product.class,1);

we will get the product data, and session perform internal operation like enabling the cache. Fetching the data, storing in to the cache so on.

If next time we will call the same product then will get the same data from the cache.

Ex:

```
p1.setProductName("LCD Color TV");
session.update(p1);
```

when we modified the data of the product then changes reflected into the session cache only, it will not modified into the database .but next time when we call the session.get(product.class,1); we will get modified data.

It means modification done into the cache but not into the DB. Actually it will modified into the database after calling the commit() method.

Ex: session.getTransaction().commit();

after calling above method change will going to reflected into the database.

- What is the reason it will not update each and every time?
  - if a user updating the data for every one second with in the same product class then for every second hibernate has to connect to the DB and update the data update and cache and close the connection.
  - It is very costly to do such kind of operation.
  - B’z of that hibernate session cache will not update the data very time it will update will we session.getTransaction().commit().
  - Internally commit() method going to call the flush() method.
  - Flush() : it will store all updated data into the database.

- What the method available work with session object ?
  - There are three method are available
    - Evict(object): It will remove the particular object from the cache .
    - Clear(): it will remove all the cache object of that particular session.
    - Flush(): it will add the updated data into the database.

- Evict(object):
  - For example if I create two session object work with product class, one session is getting the data and other session is updating the data then first session can not get updated data because both sessions are different copies of the cache.
  - To get second session updated data we have to use evict()method.
  - Evict() method will the product object and it will delete the product data, then next time when we call that product object session will get updated data.
  - Ex:

```
Product p1 = (Product)session1.get(product.class,1);
Product p2 = (Product)session2.get(product.class,1);
p2.setProductName(“LCD color TV”);
Session2.update(p2);


Product p3 = (Product)session1.get(product.class,1);
```

here we will not get updated data b’z one session object is different then other session object. To get the upgraded data.

```
session.evict(p1);
 Product p3 = (Product)session1.get(product.class,1);
```

now we’ll get ungraded data.

- **Clear():**

  if we want to clear all the session cache then we can use clear()tmethod as part of the application.

  – Session.clear();

- **Flush():** it is used to stored updated data into the DB. This method called by commit() method internally .

- Is session object is singletone or not?why?
  - No, session cache or first level cache not an singletone class, b’z many time in application we can not relay on one single object we need multiple session object.
  - Application has multiple classes and multiple classes want to interact with session cache , then it is necessity to allow multiple object creation.

Hibernate 53

- Flush() Vs commit():
- Flush():
  - If we tried to update the data , it will updated into the cache but not into the database.
  - If there are 10 update methods are there then in general in jdbc 10 update query has been generated, but when we use hibernate Flush() method it will generate only one update query but it will not updated into the database. To update into the database we have to call the commit() method.
  - Flush() method will cross check the database constraint when we are applied flush() method.
  - It will check database table constraint and hibernate inserted data if both are valid then only flush() method will generate the query, neither is will throw the exception.
  - Flush() method used for inserting , updating, deleting the records into the cache. As well as database once the validation happen.
- Commit():
  - In general we know the usage of the commit(), in hibernate also until we call the session.getTransaction().commit(); the updated data will not reflected into the database.
  - Once we call commit() method it will update the data into database but it will not reflect into the active session cache.

```
Ex: product p1 = (product)session1.get(product.class,1);
      p1.setProductName("LCD COLOR TV");
      session1.update(p1);
      session.getTransaction().commit();
product p2 =  (product)session2.get(product.class,1);
```

– Here session1 is updating the data into the database and even committing also session2 wont get the updated data b'z one session is different from another session.
– To get the updated data we have to call evict() method, then we will get updated data.
– If we call flush method after update the data then only query go generated.

First level cache working example

```
class Test{
    p s v main(){
    //configuration and get the sessionFactoty
    //get the session1 and session2 and open the session

    Product p1 = (product)session1.get(Product.class,1);
    Product p2 = (product)session2.get(Product.class,1);
    p1.setProductName("LCD Color TV");
    session1.update(p1);// it will update the cache not DB data
    p1.setDesctiption("52 inch high hd");
    session1.update(p1);

    session1.flush();//it will not generate seperate update query
it will genarete one update query taking all the changes into
consideration.

    session1.getTransaction().commit();// it will innernally call
the flush() method to generate update query and it will reflect
into the database

    product p3 = (product)session1.get(product.class,1);// even
we are accessing data after commit() also we will get old data b'z
already old data available into the cache, to get the new data use
following method
    session1.evict(p1);
    product p3 = (product)session1.get(product.class,1);//now we
will get update data.
```

```
class Product{
    //attributes
    //setters and getters
}

//mapping file (Product.hbm.xml)
//configuration file(hibernate.cfg.xml)
```

=>if a cache contains old data and if we updated(commited) the database data even though we will get old data only, b'z session1 cache old data to get the new/upgrated data hibernate has provide evict(),clear() method.
evict() method will delete the current object and it will get new data from the DB.

Hibernate 54

- ID Generator:
  - Before we discussing about ID Generator, we should have know why ID generator came into feature.
  - In olden days people used to store the data into the DB, but they unable to distinguish the primary column into the DB.
  - They used to use natural key as a primary key into the DB. But when there is change into the natural key column they has to rewrite whole table.
  - They unable to hold the relationship between tables.
  - So they are failing while storing the data, even they relay on composite key column they can not distinguish the primary column.
  - Because of the above reasons people thinks about primary key column, which is unique and easily hold the ralationship.

- It is common requirement every has to use the primary key into the database, but when we relay on relational model then we have to write complex logic to generating the primary keys.
- In oracle we use to use sequences to generated the primary key, but writing the sequence it is not complicated but when we migrate from oracle to mysql or any other DB, they does not contains sequence feature then we have to rewrite the logic.
- It is common requirement for every DB to have a Identity column.
- Id is the more important into the DB table, b’z of id we can easily hold the relationship with other DB tables.
- So hibernate has provided multiple key generator, to overcome the problems of above reasons.

- Hibernate provided KEY Generator
  - Assigned
  - Identity
  - Sequence
  - Holi
  - Holi_seq
  - Select
  - native
  - Increment
  - guid
  - Uuid
  - Foreign

- ID generator / key generator / surrogate key / identity generator these are name for hibernate ID generators.
- Hibernate is the ORM Tech. at each and every place he has provide the flexibility to avoid the boiler plat logic.
- Here also hibernate has provided id generator to avoid the duplicate the logic and it will make our application portable from on DB to another DB.
- But when we deal with RDBMS and we can not migrate from oracle to Mysql, b’z both DB has there own syntaxes.

Hibernate 55

- Every ID generator has there own implemented classes which is internally called by session object.
- Assigned, identity, native, sequence are the short name of the id generators.
- Even we can create our own id generator by overriding the internal class/interfaces.
- We can call custom id generator.
- Every id generator having there own strategy to generate a primary key value.
- Every id generator different from each other and there implementations also.
- actually session never generate the primary key for table, it will take the metadata and pass to the corresponding class to generate the primary key value.
- A class going to generate the primary key value and return to the session object and session will add to the cache.

- 1)assigned:
- How does it work?
  - Hibernate will not generate any surrogate key for assigned ID generator, programmer has to provide the primary key value to the hibernate.
  - Session object will take the provided class name it will go to the sessionFactory , it will get the metadata for corresponding class from mapping information, it look to id generator, if it is ASSIGNED Id generator then it will take programmer provided value and computed into the cache.
- Data- type : int , short , long
- Benefit :
  - It is database independent
- Syntax:

```
<ID name="att_name" column="col_name">
    <generator class="assigned"/>
```

2)Increment :

- How does it work? :=> MAX(ID)+1;
- It will going to fire query on the DB table
  - Ex: select max(AGENT_ID) from AGENT;
  - Session object will take the provided class name, it will go to the sessionFactory and it will get the metadata for corresponding class from mapping information, it look to id generator, if it is INCREMENT Id generator then it will generate one query to get the max value from the database and it will add +1.
  - And it will give to the session object, session object going to perform rest of the operation.
- Data-type : int , short , long.
- Benefits:
  - It is database independent
  - It will work with any database.
- Drawback:
  - It will not generate unique primary key in multi-threaded environment or cluster environment.
  - If two or more classes wanted to persist the data at a time then it may chance to generate duplicate primary key.

- Syntax:

```
<ID name=“att_name” column=“col_name”>
     <generator class=“increment”/>
```

- 3) Identity:
- How does it work?
  - Identity one of the id generator which relay on the database auto-generated elements.
  - Here hibernate will not do any thing, session object will play the role for getting the primary key value from the corresponding database.
  - Session will get the metadata from the mapping file and it will go to the specific DB and will run the auto-generate column which is available into DB.
  - Here programmer and hibernate will not have any work everything is takes care by session, and it will automatically generate the primary key values.
- Data-type: int , short , long.
- Benefits:
  - It will work with auto- sequence database.
- Drawback:
  - It only work with auto-generated value databases.
- Ex: mysql, ms-sql server.
- What is the difference between assigned and identity?b'z both the places hibernate will not do anything ?
  - Assigned : here programmer is the responsible for providing the primary key value to the session to persist into the database.
  - Identity: hare identity specific class will automatically execute the corresponding database sequence and it will generate the primary key value.
  - Both the places hibernate will not do anything.
- Syntax:

```
<ID name="att_name" column="col_name">
        <generator class="identity"/>
```

- 4) sequence:
- How does it work?
  - Sequence id generator relay on the specific DBs which having the sequences concept.
  - Session will get the metadata from the mapping file and it will go to the specific DB and will run the sequence which is available into DB.
- Data-types: int , short, long.
- Benefits:
- Drawback:
  - It will work with only those DBS which are having the concept called sequence.
  - If we are using sequence and if we are not specifying any name to the sequence the hibernate going to give the default name to the sequence.
  - But if there are more then one entities are there and we haven't specified sequence name then it lead to generate incorrect primary key value .
- Syntax:
  - `<generator class="sequence">`
    ```
    <param name="sequence" > agent_ID</param>
    </generator>
    ```
  - `<generator class="sequence">`
    ```
    <param name="sequence" > property_ID</param>
    </generator>
    ```
- Ex: create sequence hibernate_sequence
- Ex: select hibernate_sequence.nextval from dual

Hibernate 56

- 5) Hilo :
  - How does it works?
    - Hilo work on one of the formula
      - (Max_lo * next_hi + next_hi)
  - Hilo is the one to the mathematical algorithm which work with batch of records.
  - Hibernate going to generate one default table with the name (hibernate_unique_key) with one column and column name is next_hi.
  - If we don’t want to work with the default generated DB table or column then we can configure our own table name and column.
  - Hilo works with the batch of records.
  - Already other ID generators are there then what is the need for going to word Hilo id generators
    - Actually every id generator have there own drawbacks while generating the id of the table.
    - They have to do maximum around trips while generating the primary value.

- Hilo will not make maximum around trips with the database for generating the primary key value.
- Actually session object will take the class name with that it will go to the sessionFactory and get the metadata for corresponding class with table and it will check which id generator has been used for generating the primary key.
- If it is hilo it will check any table and column has been configured or not along with max_lo value.
- If it is already configured it will use existing table and column to generate the primary key value, if table and column not configured then hibernate will generate its own table with the name hibernate_unique_key and the column name next_hi.
- Max_lo is the value which is talk about how much records inserted at one short. E.g. if we provide 10 means it will insert 10 records.

- Internally hilo work as fallows
- Ex <generator class="hilo">

```
<param name="max_lo">10</param>
[optional]
<param name="table">tbl_name</param>
<param name="column">col_name</param>
</generator>
```

  - Now hibernate will generator one table with name hibernate_unique_key with the column name next_hi and it will assign the value as 1 .(next_hi = 1).
  - We already configured max_lo with the value 10.
  - Hibernate will compute the formula as below

    (max_lo *next_hi * next_hi)

    10 * 1 * 1 = 11
  - While taking the next_hi from the DB it will increase the next_hi with 1, means next time if other session object want to insert the batch of record, then it will get 10 * 2 * 2 = 22 and they will insert the records.
  - No one session object will get duplicate next_hi value.
  - It will start inserting the records with id 11, 12 , 13…20. means it will not around trips on the DB it will compute first time and for next 10 records it will get it from hibernate only.
  - After inserted 10 records it will go to the DB and get the current next_hi and increments by 1. and perform the tansaction.

- Data-types : int , short , long.
- Benefits:
  - It will work with any application environment.
  - And it will not around trips to the DB maximum time.
  - It will use internally locking system while working, b’z in jee application more possibilities are there more then one person can insert the records into the DB, making sure every no one object will get the same hilo value.
- Drawbacks:
  - It will work with JEE apllication only.

- 6) Seq_hilo:
  - Seq_hilo same as the hilo id generator, hilo id generator relay on the formula but seq_hilo id generator work with the sequence generator.
  - While inserting the data it will generate one of the sequence in to the database.
  - It is also used to insert the batch of records, only the difference is, we will get primary value by executing the seq_hilo.
  - It will also perform same operation like hilo only.
  - How does it works?
    - Same as hilo but it will get the value from the sequence.
  - Benefits:
  - It will work with any application environment.
  - Data-types: int , short , long.
  - Drawback:
    - It will work with oracle database only.
    - It will support all the application environment.

Syntax:

```
<generator class=“seq_hilo”>
 <param name=“max_lo”>10</param>
 <param name=“sequence”>agent_id_seq</param>
</generator>
```

- 7)GUID: (global uid):
  - How does it works?
    - Actually session object will take the class name with that it will go to the sessionFactory and get the metadata for corresponding class with table and it will check which id generator has been used for generating the primary key.
    - Actually GUID work with JEE environment, and it will uses one algorithm which going to generator primary key value.
    - GUID will take number of parameter to generator the primary key value. It will generator the value in String format with max length is 256 character.
    - It will unique across the universe.

    Ex: select sys_guid() from dual;
  - Benefits:
    - We will get unique id in the universe.
  - Data-types : string
  - Drawbacks:
    - It will work with GUID generator databases only.
    - It will consume more memory space for storing the primary key value.
    - It is very hard to remember and query the data.
    - It is used in special environment only.

Hibernate 57

- UUID
- Foreign
- Select
  - Legacy based application
  - Internally trigger has been used
  -

Hibernate 62

- Update and marge

Hibernate 63

- Working with entity lifecycle states:
  - In programming world every object has state while transition from one place to another place.
  - In normal java object lifecycle there are infinite state are available for one object, but coming to the hibernate there are finite set of states are available.
  - Actually hibernate object deals with persistency related context so it has finite set of states.
  - There are four states where hibernate object will go through to perform the persistency.
    - Transient state
    - Persistent state
    - Detached state
    - Removed state
- Transient state :
  - When object has been created then that state called as transient state.
  - If we perform any operation also it will not affect into the database or session cache.
  - To perform operation on the database or first level cache then we have to change the state of the object. By using sort of method which are provided by the hibernate.

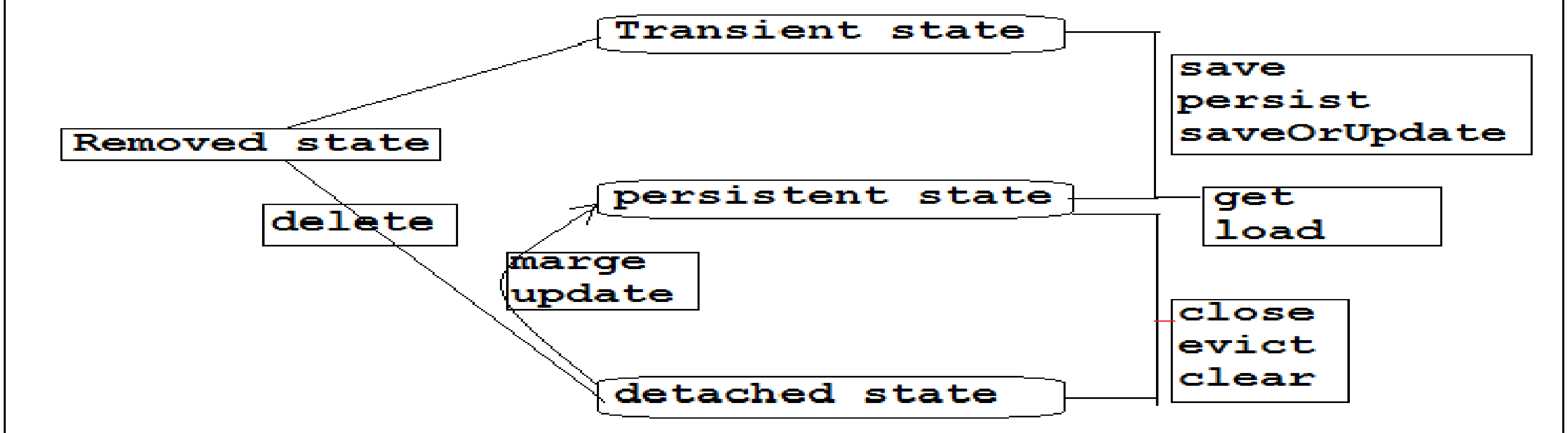
- Persistent state:
  - In this state object has some data and it is attached with the session or database to perform some operation.
  - Once object entered into the persistent state then there is no need to use additional method to perform the persistency.
  - Object itself is responsible for persisting the data.
    - Ex:

```
Address addr = (Address)session.get(Address.class,1);
Addr.setAddressLine1("solapur");
Session.getTransaction().commit();
```

  - There are different method provided by hibernate to get object into persistency state.
    - Save
    - Persist
    - saveOrUpdate
  - Get and load are two method which directly make the object persistent state.
  - A object deals with session cache or database then that object should be in persistency state only.

- Detached state:
  - Once object has been removed the relationship with session or database then that state called as detached state.
  - There are several method which can easily detach the object from persistency state to detached state.
    - Close
    - Evict
    - Clear
  - These three method make object to detached from persistent state to detached state.

  - There are two methods available were we can transmit detached state to persistent state.
    - Update
    - Marge: marge can transmit detached state object to attached state.



Entity lifecycle states

```
class Test{
      SF sf;
      Session s;
      Address addr;
      addr = new Address(); Transient state

      addr = (Address)session.get(A.class,1);
 persistent state
      addr.setStreet("ameerpet x road");
not required save or update method for persist
bz object state is persistent
      session.getTransition().commit();
      addr = (Address)session.get(A.class,1);
      //session.evict(addr);
      session.clear();                    detached state
      session.delete();
}
```

Hibernate 64

- **Relational Object model:**
- Inheritance having multiple problems
  - We can’t extends more than one class
  - It forces other class to use all features
  - There is problem of fragile the classes
  - Problems in testability.
- There are two type in inheritance
  - Generalization
  - Realization
  - Specialization
- i).Generalization: It is the process of keeping duplicate method in one class, and extends that class to other subsequent classes.

A
m4()

B
M1()

C
m2()

D
m3()

=>As we see class A contains method m4() which is common for all the concern classes whoever want to use that method extends class A and access that method.

ii).Realization: It deals like mediator for main class and subsequent classes. If a method name is same but implementation will be different then we can use Realization concept.

iii).Specialization: A class want to create a new feature and it is specific to that particular class called Specialization. A may contains it own methods which is not sharable to other classes.

Relationship model

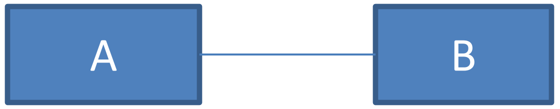
- As we discussed in previous class what are the ways to speak with other classes using inheritance.
- Now we will discuss Composition
- Composition: A class contains other class attributes and method called as Composition.
- A object has a other object data called as Composition.
- Composition classified into three ways
  - Association
  - Aggregation
  - Composition
  - Dependency

- Association: Two classes are associated with each other. One class can survive without other class called as Association.

  In association parent and child relationship is not there both are equal to carry there work.

  -owner :not applicable

  -relationship: independent

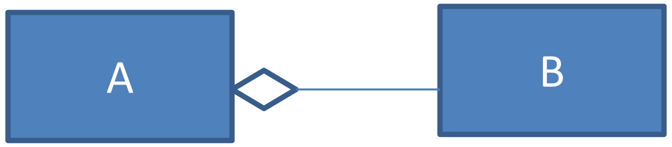


- Aggregation : Two classes are aggregate with each other. One class can survive without other class But here classes has there own parents.

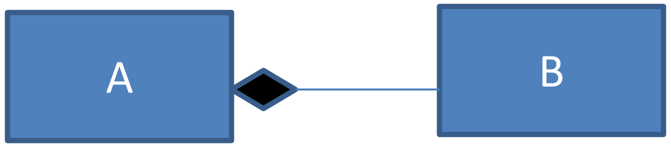
  ex: students are depends on teacher but without student teacher can survive and without teacher student can survive.

  - owner : parent is there

  -relationship: independent

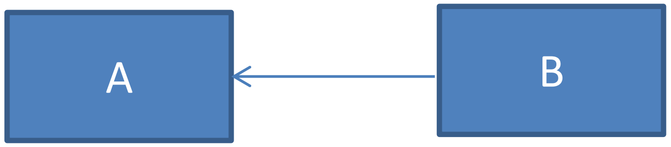


- Composition: one class can not survive without other class that is composition.
- Means class A contains class B attribute to survive if B is not there means class A also not there.
  - owner : parent is there
  - Relationship: dependent



- Dependency: A class method is depend on other class attribute it is called as dependency.

Ex:
```
class A{
        p v m1(B a){}
}
class B{}
```



Class A method depends on Class B object so it is called as dependency.

- Relationship Mapping Model:
  - In general there are several problems while persistency the data into the database because one entity class may contains relational data it is every hard to predict the relationship with underlying database.
  - We can not judge or decide which kind of relationship available between the classes, like one-one , one-many, many-many..etc.
  - ORM Technology has founded five most common problems while dealing with the relationship with object.
    - Granularity
    - Inheritance
    - Navigation
    - Identity
    - Association

Hibernate 65

- Granularity:
  - Granularity is the problem which come while we are accessing the data from the underlying database.
  - If data is stored in table per class hierarchy then it is very hard to find the appropriate data from the corresponding table. B’z a table contains multiple class data so we unable to find appropriate data from the table .

- Inheritance (Subclass type):
  - Object never talks about relationship like primary key , foreign key.
  - If there is parent and child relationship then it is very hard to identified which kind of relationship they going to carry while storing the data into the table .
  - If child is extending the parent data while storing the child, parent data has to be stored, but the problems is which relation he has to carry there is impedance miss match.

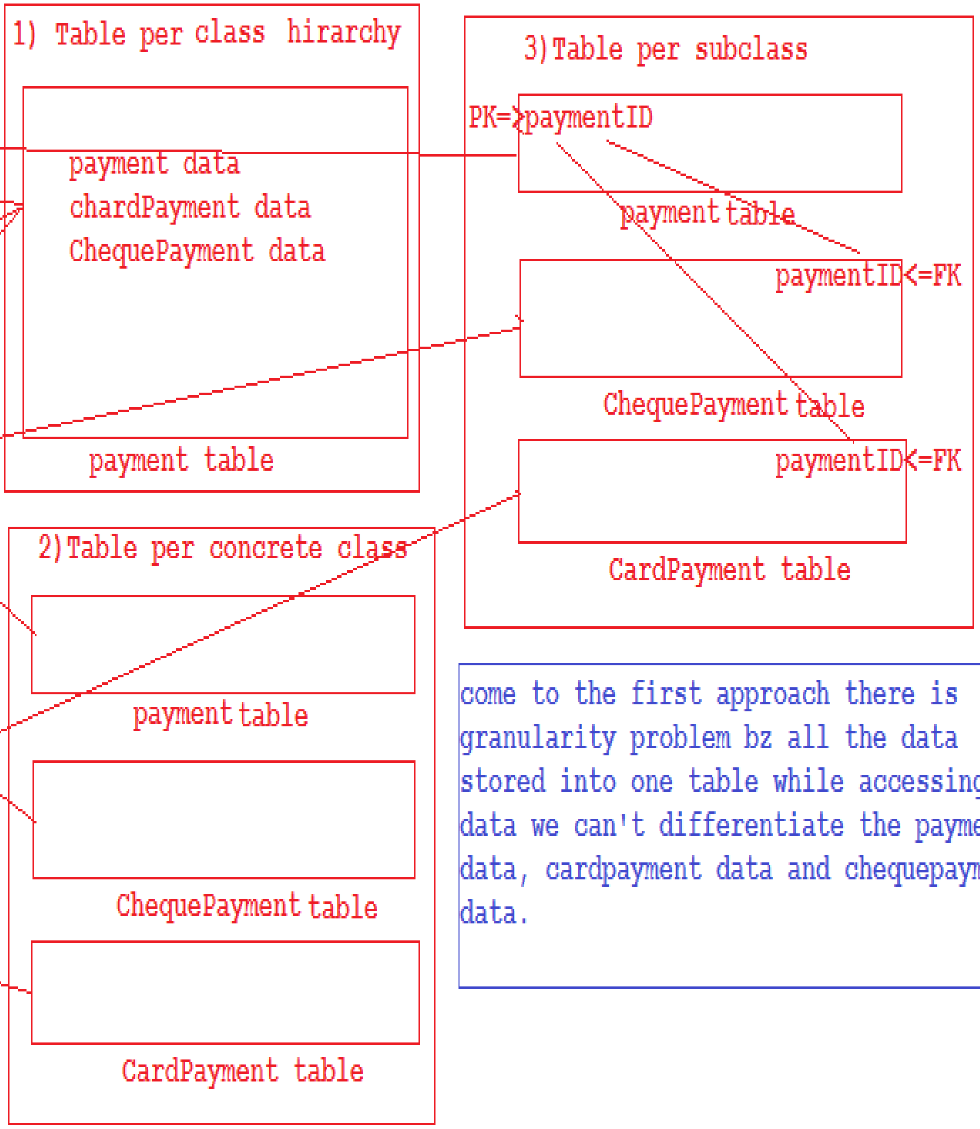
- Navigation:
  - In database data will be store into relational format with some relationship.
  - While accessing the data it should come from multiple tables following relationship. So to get it has to navigate from multiple table.
  - Object never talks about the relationship between the tables.

- Identity : identity talks about the uniqueness .

- Association :

- 1)Inheritance Mapping Model
  - IMM going to store data into database using following four ways
    - Table per class hierarchy
    - Table per concrete class
    - Table per subclass
    - Implicit polymorphism

```
class Payment{
     int paymentId;
     string marchand;
     float amount;
     //setter && getter
}
class CardPayment extends Payment{
     int paymentId;
     string marchand;
     float amount;
     String bankName;
     String cardno;
     String expiry;
     int cvv;
     //setter && getter
}
class chequePayment extends Payment{
     int paymentId;
     string marchand;
     float amount;
     String chqNo;
     String bank;
     //setter && getter
}
```



come to the first approach there is granularity problem bz all the data stored into one table while accessing the data we can't differentiate the payment data, cardpayment data and chequepayment data.

Inheritence mapping model

Hibernate 66

- Inheritance Mapping model:
  - As per the above discussion we come to know what are the problems are available while storing the data into relational database.
  - Actually impedance miss-match talks about the problems while storing the object data into the relational format.
  - One of the problem is subclass(inheritance)
  - To solve the inheritance problem there are three ways we have to apply
    - Table per class hierarchy
    - Table per concrete class
    - Table per subclass

- 1) Table per class hierarchy:
  - Actually class hierarchy talks about the parent-child relationship . Number of classes can inherit the properties of parent class to child class for whole hierarchy there is only of table which going to handle the all classes data.
  - As per the above diagram payment is the parent class and cardpayment and chequepayment is the other subclass of payment class.
  - And as per the above diagram how actually whole hierarchy going to stored into one table.
- Procedure of mapping the inheritance mapping model using class per hierarchy.
  - All the Payment class attribute are inherited by CardPayment and ChequePayment.
  - While mapping such kind of relationship we have to little conscious . Bz other classes are inherited from one class.

- First write the payment.hbm.xml along with all the configuration details.
- Ex:

```
<hibernate-mapping package="com.imm.entities">
 <class name="Payment" table="PAYMENT_HISTORY" discriminator –
class="PAYMENT">
    <id name="paymentId" column="PAYMENT_ID">
    <generator class="increment"/>
    </id>
    <discriminator column="PAYMENT_TYPE"/>
    <property name="merchant" column="MERCHANT"/>
    <property name="amount" column="AMOUNT"/>
</class>
 </hibernate-mapping>
```

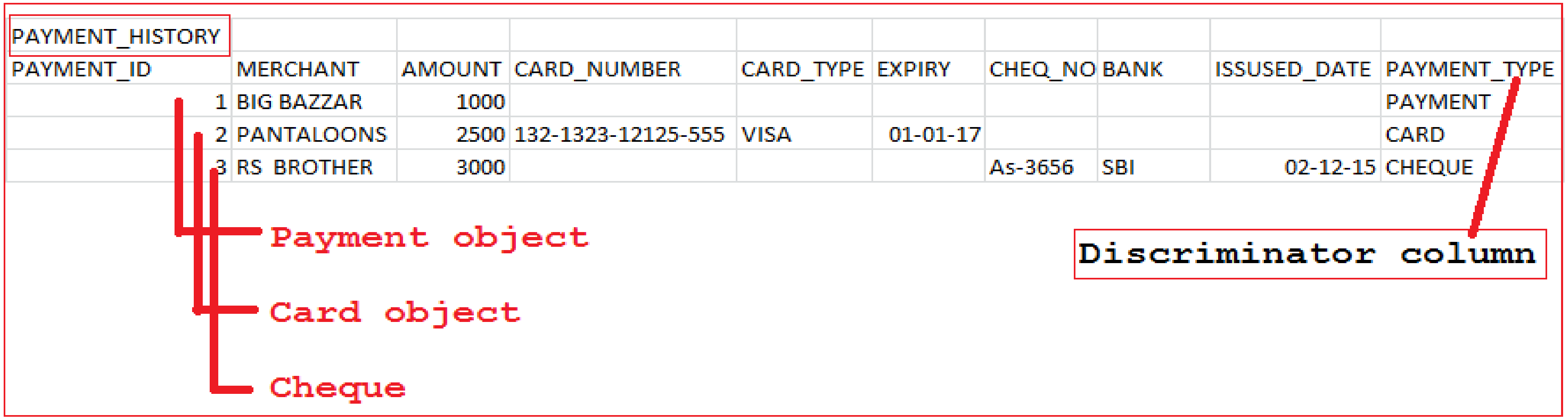
- In above payment.hbm.xml we configured Payment.class attributes to the mapping file. But along with that we configured one new element and attribute i.e. discriminator and discriminator-value.

- If we stored all classes relational data into one table, then while retrieving how we going to identified the particular data from the table.
- B’z one table having multiple classes data so it is difficult to identify which data we are retrieving. Bz of the discriminator we going to identify clearly where the data coming from.
- Discriminator means differentiate the classes with passed identities. While storing the object we can pass the discriminator value to clearly differentiate from other class object.
- Now map the subclasses with there mapping files, but subclasses are the classes which inherits the payment mapping also.
- Now we can not directly write subclass mapping file we have to extends the payment mapping file and along with that we have to pass discriminator values.

- Ex:

```
CardPayment.hbm.xml
<hibernate-mapping  package="com.imm.entities">
    <subclass="CardPayment" extends="Payment"
discriminator-value="CARD">
<property name="cardNumber" column="CARD_NM"/>
……
</hibernate-mapping>
```

- Now create another mapping for ChequePayment with same above procedure.
- Now create the table with all the columns and add one more column i.e. discriminator column.
- Lets we the diagrammatic presentation.



PAYMENT_HISTORY

PAYMENT_ID	MERCHANT	AMOUNT	CARD_NUMBER	CARD_TYPE	EXPIRY	CHEQ_NO	BANK	ISSUSED_DATE	PAYMENT_TYPE
1	BIG BAZZAR	1000							PAYMENT
2	PANTALOONS	2500	132-1323-12125-555	VISA	01-01-17				CARD
3	RS BROTHER	3000				As-3656	SBI	02-12-15	CHEQUE

Payment object
Card object
Cheque
Discriminator column

Hibernate 66

- Table per class hierarchy using annotation approach:
  - As we know how to configure the class with the annotation driven approach, now we will convert above table per class hierarchy example into annotation approach.
  - To configure table per class hierarchy approach to annotation approach we have to use new annotations let see the example.

```
@Entity
@Table(name="PAYMENT_HISTORY")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="PAYMENT_TYPE")
@DiscriminatorValue("payment")
public class Payment {
            @Id
            @GenericGenerator(strategy="increment",name="hib_inc")
            @GeneratedValue(strategy=GenerationType.AUTO,
                                    generator="hib_inc")

            //setters && getters
}
```

- Above annotation will be added to the superclass only and other subclasses are having only @discriminator-value ="value" and @Entity.
- For ex:

```
@Entity
@Descriminator-value="value"
Class cardPayment extends Payment{
    //attributes
    //setters && getters
}
```

Hibenate 67

- 2) Table per concrete class:
  - It is one of the technique were we can store data into database, actually tables having the parent child relationship.
  - But we can store every entity as separate table into the database. Lets see the diagram for better understanding.
  - How the payment is inherited to other class and what are the column are available.
  - Table per concrete class pays polymorphic behavior which is take care by hibernate only.
  - Polymorphic behavior means if we requested child class object using parent class reference then hibernate automatically convert child class ref to parent class and it will return the output.
  - Here base class can store any child reference bz of that there no need to warry while accessing the data by parent reference we can access any child data.

- But in table per concrete class technique we have to configure addition configuration to work with table per concrete class.
- While mapping our classes with mapping file we have to configure who is the base class and id but while configuring child classes we have to be little conscious.bz child table also have there own table per perform the operation.
- To make sure by base reference to get the data we have to make our child class as subclasses.
- But while configuring subclasses we have to make they are union-subclass only.bz while using parent reference if we are accessing data hibernate has to check all the table where the actual data is available for that hibernate will union all the table and it will return the corresponding record to the user.

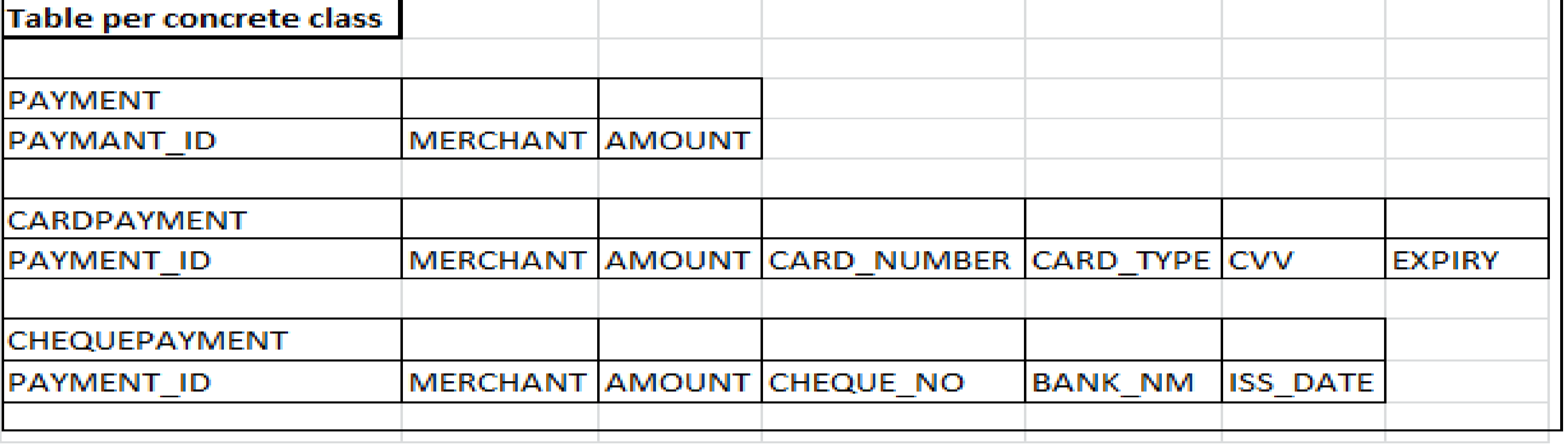
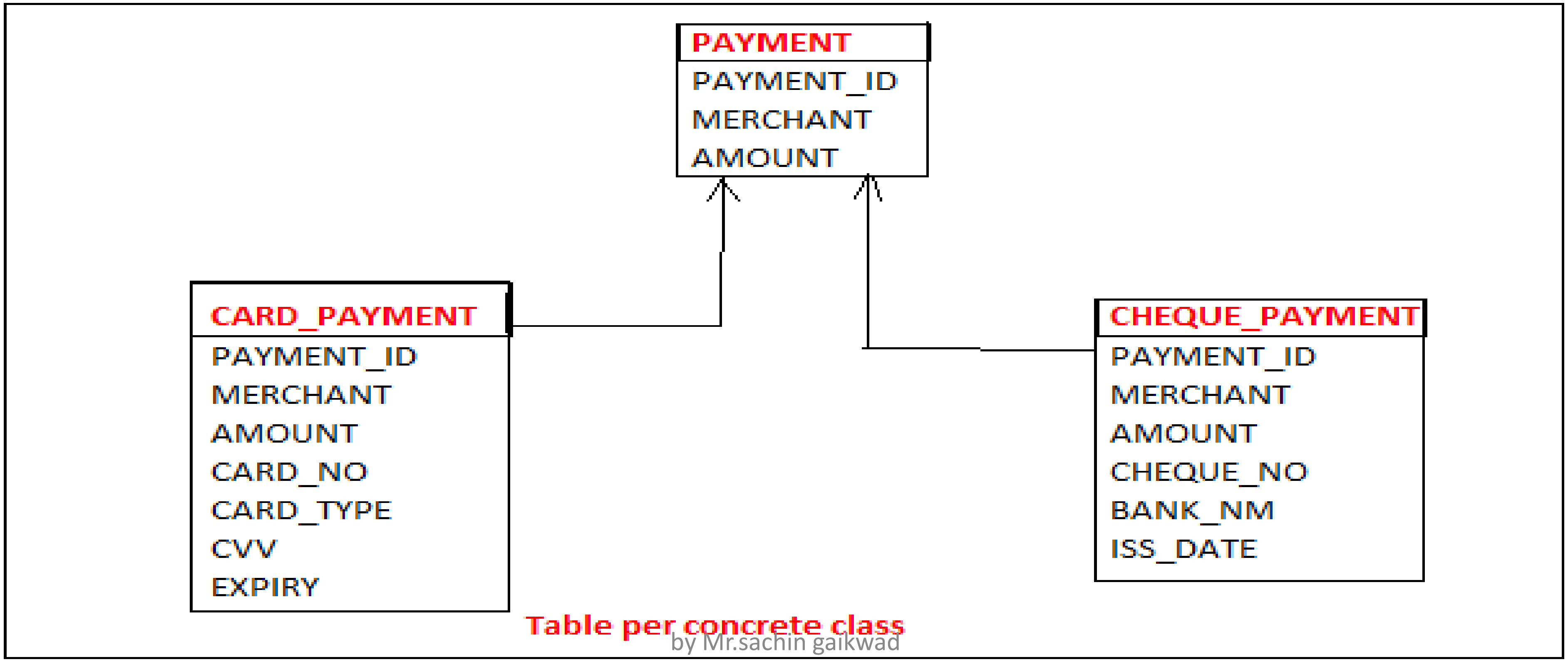


Table per concrete class

PAYMENT		
PAYMANT_ID	MERCHANT	AMOUNT

CARDPAYMENT						
PAYMENT_ID	MERCHANT	AMOUNT	CARD_NUMBER	CARD_TYPE	CVV	EXPIRY

CHEQUEPAYMENT					
PAYMENT_ID	MERCHANT	AMOUNT	CHEQUE_NO	BANK_NM	ISS_DATE



- Only base class mapping file having the ID element rest of the child classes we can not configured ID element bz these child are inherited by base class only.
- Even we are trying to store child class data into child table then hibernate well know to the provided relationship and hibernate going to generate the key for our next record.
  - Ex:

    If parent table contains one record and child want to insert the new record then hibernate enough intelligence to get the id from executing parent ID generator and it will assign 2 as the id for next record into child table.

Hibernate 68,69

- To perform the above operation we have to write the mapping file. Which carry the operations.

```
*PAYMENT*
<hibernate-mapping>
 <class name="com.tpcc.entities.Payment" table="PAYMENT">
  <id name="PaymentId" type="int">
   <column name="PAYMENTID" />
    <generator class="increment" />
  </id>
     //all the properties
 </class>
</hibernate-mapping>
```

```
*CARDPAYMENT*
<hibernate-mapping package="com.tpcc.entities">
    <union-subclass name="CardPayment" extends="Payment" table="CARDPAYMENT">
         <property name="cardNumber" type="int">
            <column name="CARDNUMBER" />
        </property>
          //all the properties
    </union-subclass>
</hibernate-mapping>
```

```
*CHEQUEPAYMENT*
<hibernate-mapping package="com.tpcc.entities">
    <union-subclass name="ChequePayment" extends="Payment" table="CHEQUEPAYMENT">
        <property name="chaqueNumber" type="java.lang.String">
            <column name="CHAQUENUMBER" />
        </property>
          //all the propertiues
    </union-subclass>
</hibernate-mapping>
```

- As we seem how to program the application using table per concrete class by mapping approach, now we will see how to program the same application using the annotation based approach.
- It's more easy when compared to the mapping approach, just add one annotation it will work with annotation based approach.

ex:

```
@Entity
@Table(name="PAYMENT")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
Class Payment{
  @Id
   int paymantid;
…………
}
```

- just adding @Inheritance(strategy=InheritanceType.TABLE_PER_CLASS) hibernate will perform the table per concrete class operation.
- Once we configured the base class with the above annotation there is no need to add another annotation to the subclasses automatically hibernate will recognize and perform the corresponding operation.

- 3)Table per subclass:
  - As we learnt table per class hierarchy and table per concrete class now we will see the third approach that is table per subclass.
  - In table per subclass parent/base class data shared to the other subclasses, In table per concrete class duplicate columns present across multiple table but in table per subclass we put the relation with the parent class with child classes To make fully normalized table.
  - Actually parent table primary key will become the primary key column for the child class. b’z to avoid the duplication column among subclasses.
  - while storing the data hibernate is intelligence in separating parent class data and child class data and it will stored into the respective tables.
  - But while accessing hibernate has to follow join strategy b’z while storing the data sub tables will get the parent primary key id.
  - To find out were the actually data is coming from hibernate has to use join queries, comparing parent key id value with sub table id value and retrieve the data.

- In table per subclass if we directly accessing the data from child table then it will use equi-join to retrieve the data.
- Table per subclass is the fully normalized tables.
- It also support polymorphism queries while retrieving the data.
- Using super class reference we can retrieve the data but it is every costly b’z to get the data all table has to be join using left outer join.
- To map these classes we have to use the joined-subclass element while configuring the mapping file.
- It is recommended to configure the key attributes in each and every child table bz super class primary key column value become the foreign key and primary key for the child tables. So to retrieve the data it is mandatory to provide the key attributes.
- Lets see the mapping files.

```
<hibernate-mapping>
  <class name="com.tpsc.entities.Payment" table="PAYMENT">
   <id name="paymentId" type="int">
    <column name="PAYMENTID" />
     <generator class="increment" />
   </id>
     // all the properties
  </class>
</hibernate-mapping>
```

```
<hibernate-mapping>
    <joined-subclass name="com.tpsc.entities.CardPayment"
     extends="com.tpsc.entities.Payment" table="CARDPAYMENT"
      lazy="false">
        <key>
            <column name="PAYMENTID" />
        </key>
     // all the properties
    </joined-subclass>
</hibernate-mapping>
```

```
<hibernate-mapping>
    <joined-subclass name="com.tpsc.entities.ChequePayment"
extends="com.tpsc.entities.Payment" table="CHEQUEPAYMENT" lazy="false">
        <key>
            <column name="PAYMENTID" />
        </key>
     //all the properties
   </joined-subclass>
</hibernate-mapping>
```

Hibernate 70,71-miss

- As per the above discussion we can run the application using mapping file but there is another approach which give rest to the mapping approach i.e. annotation driven approach.
- To write the same application using the annotation based approach we have to use below procedure.

Ex:

```
@Entity
@Table(name="PAYMENT")
@Inheritance(strategy=InheritanceType.JOINED)
@PrimaryKeyJoinColumn(name="paymentId")
public class Payment {
@Id
@Column(name="PAYMENT_ID")
@GenericGenerator(strategy="increment",name="hib_inc")
@GeneratedValue(strategy=GenerationType.AUTO,generator="hib_inc
")
private int paymentId;
private String merchant;
private float amount;
…
}
```

- Above class is the base class which has all the required annotation but for child class or derived class we have to add one extra annotation along with the regular annotation.
- Ex:

```
@Entity
@Table("CARD_PAYMENT")
@PrimaryKeyJoinColumn(name="card_payment
Id")
public class CardPayment extends Payment {
…….
}
```

by this we completed the annotation driven approach with table per subclass.

- Implicit polymorphism:
  - This technique has provided by the hibernate people to work with inheritance mapping approach.
  - In this approach we have to write the classes only and the mapping files as normal.
  - But classes should be inherited into hierarchy.
  - For every class provide the separate mapping file along with the corresponding tables.
  - While persisting the data hibernate has to identify the relation between the classes and hibernate good enough in storing the data in corresponding tables.
  - But the problem is that while retrieving the data hibernate may not retrieve correct data, bz each and every table has its primary key column so hibernate got confuse while retrieving the data, Which record programmer asking for.

    ex:

    just write the normal classes and there corresponding mapping and classes should be in inherited hierarchy. And execute the application.

Hibernate 71

by Mr.sachin gaikwad

Hibernate 72

- Association Mapping model:
  - We can access properties of one class to another class by two ways
    - Inheritance
      - Generalization
      - Realization
    - Association
      - Association
      - Aggregation
      - Composition
  - As we learnt how to work with inheritance mapping model, Now we will learn how to work with the Association Mapping model.
  - Association represents the HAS-A relationship between the classes.
  - means one class contains multiple attributes of own class and other class also.
  - Association does not have any owner and they are independent, one can live without other one.
  - Composition mean it has the owner and they are completely dependent on each other, one can not survive without other one.

- In relation database we can represent the relation using primary key and foreign key. But coming to the java we can represent the relationship using inheritance or association.
- In relational DB there are cardinalities available for representing the relationship, in java also we can represent the cardinalities using association mapping.
- Let see the below representation.
- As per the below representation we can associate one table to another table.

one to one relationship in Relational model

#1

	Person	Address
	personId	addressId
	firstName	address_line_1
	lastName	address_line_2
	gender	city
	age	state
fk,Unique	addressId	zip
		country

#3

Person	person_address		Address
personId	personId	fk,unique	addressId
firstName	addressId	fk,unique	address_line_1
lastName			address_line_2
gender			city
age			state
			zip
			country

#2

Person	Address	
personId	addressId	
firstName	address_line_1	
lastName	address_line_2	
gender	city	
age	state	
	zip	
	country	
	personId	FK,unique

#4

Person	Address
personId	personId
firstName	address_line_1
lastName	address_line_2
gender	city
age	state
	zip
	country

by Mr.sachin gaikwad

- In java also we can represent the association between the classes, to represent the association we have to take other class object as a attribute of the another class.
  - Ex:

```
Class person{
        private personID;
        ….
        Address address;
}
Class Address {
        int addressId;
        …..
}
```

- Once we taken another class object as a attribute in another class it is not enough again we have to map that class with the relational table internally using mapping files.
- While persisting the person data address should be persist into the address table. And while accessing the person record ,address should be fetch from the DB.

Hibernate 73,74

- Association using Many-to-One as One to One:
- As per the above diagram representation we can guess how actually we can make many to one to one to one.
- Always foreign side represents the many side but to write it one to one make that the foreign key column as unique.
- We can make many-to-one and one-to-many as one-to-one making foreign key column as unique. If we make not null also then it will allow only restricted data.
- Writing the classes with relationship as well as making sure how to persist that relationship data into the relational table, and while retrieving how to retrieve parent table data along with relation data.

- To perform such kind of operation hibernate has provided different kind of additional configuration tag which help in persisting and retrieving the relational data into the database.
- Just we have to write general mapping configuration file along with that we have to tell to the hibernate which attribute is caring the relationship.
- For example person class contains one attribute i.e. address which carry the relationship, here person class has a other class object as attributes means person class represents the many side.
- It is uni-direction relational mapping methodology.

- To configure Many-to-one hibernate has provided one tag called as <many-to-one>, it will contains number of attributes which specifies how to relate the relationship with another table primary key column to our class foreign key column.

- Ex:
  - <many-to-one name="attr_name" column="col_name" class="Atrri_class_name" unique="true"/>
  - As per the above example many-to-one is establish the relationship between two classes. i.e. person and address.
  - Name: attribute represent the name of the attribute whichever we defined inside the person class .
  - Column : it represent the where actually we going to store the values into the table.
  - Class: Actually person class hold the address class attribute but coming to the relational tables person table has foreign key column relation with the address table which represent the many to one relationship.
  - Unique: To make many-to-one as one-to-one we have used the unique as true.

- Ex:

  <many-to-one name="address" column="address_Id" class="Address"/>

  - address is the attribute for person class but it is the object type attribute, one we configured as address as attribute hibernate will can join the relationship between ADDRESS table and PERSON table by placing the column name as address_id as the foreign key in the person table.
  - Address_id of PERSON table is the primary key of the ADDRESS table .
  - While accessing the data hibernate will join PERSON and ADDRESS table according to there relationship means address_id of PERSON table and the address_id of ADDRESS table ,it will compare and if both are equal then it will return person and address data.
  - By using class hibernate will get the corresponding table for that class and it is easy to relate the relationship.

```java
public class Person {
    private int personId;
    private String firstName;
    private String lastName;
    private int age;
    private String gender;
    private Address address ;
    //setters and getters
}
```

```java
public class Address {
    private int addressId;
    private String addressline1;
    private String addressline2;
    private String city;
    private String state;
    private String zip;
    private String country;
    //setters and getters
}
```

```xml
<hibernate-mapping>
    <class name="com.oto.entities.Person" table="PERSON">
    <id name="personId" type="int">
      <column name="PERSONID" />
       <generator class="increment" />
    </id>
     //properties
     <many-to-one name="address"
        class="com.oto.entities.Address" fetch="join"
          lazy="false">
        <column name="ADDRESS" />
     </many-to-one>
    </class>
</hibernate-mapping>
```

```xml
<class name="com.oto.entities.Address" table="ADDRESS">
        <id name="addressId" type="int">
            <column name="ADDRESSID" />
            <generator class="increment" />
        </id>
      //propertis
</class>
```

```java
public class OTOTest {
        public static void main(String[] args) {
                SessionFactory factory;
                Session session = null;
                factory = HibernateUtil.getSessionFactory();
                session = factory.openSession();
                session.beginTransaction();
                //savePerson(session);
                getPerson(session, 1);
                session.getTransaction().commit();
```

Hibernate 75

- **One to one association mapping:**
- It is one of the association technique which allow us to map the two tables.
- Here one table primary key will become the primary key of the other table as a foreign key.
- For such kind of mapping hibernate has provided a tag called one-to-one.

  ex:

  <one-to-one name="house" class="House"/>
- In the above example house will be the object type attribute which is declared in the layout class.
- Hibernate will take house and it will get the corresponding type, using that it will go to the particular class and it will get the primary key column and value and it will inject in to the layout table id column.
- Bz in layout mapping file we configured id generator as foreign and property name as house.
- Means house table primary key value will be injected to the primary key value to the layout table.

- Lets see the example

```
HOUSE
======
HOUSE_NO(PK)
TYPE
SURVEY_NO
PLOT_NO
LOCATION
PRICE
```

```
LAYOUT
======
HOUSE_NO  (PK)
AREA
UNITTYPES
FACING
WIDTH
LENGTH
```

```
class House{
     int houseNo;
     String type;
     ....
     //setter and getters
}
```

```
House.hbm.xml
==============
<hibernate-mapping>
  <class name="House" table="HOUSE">
     <id name="houseNo" column="HOUSE_NO">
      <generator class="increment"/>
     </id>
     <property name="type" column="TYPE"/>
     ....
    </class>
</hibernate-mapping>
```

```
class Layout{
     int layoutNo;
     String survayNo;
     String platNo;
     House house;
     ....
     //setters and getters
}
```

```
Layout.hbm.xml
===============
<hibernate-mapping>
  <class name="Layout" table="LAYOUT">
     <id name="layoutNo" column="LAYOUT_NO">
      <generator class="foreign"/>
      <param name="property">house</param>
     </id>
     <property name="type" column="TYPE"/>
     ....
     <one-to-one class="House"/>
    </class>
```

=> First house will be stored and while persisting Layout, id generator has mapped with foreign then hibernate will take the property attribute which is declared in param tag and it will get the type of the house and it will connect to that corresponding table column and fetch the primary key of that column and inject to the foreign key column.

ONE TO ONE ASSOCIATION

Hibernate 76,77,78

- One-to-Many Association:
  - While working with one-to-one we made many-to-one as one-to-one by making many side as unique and not-null.
  - Many side represent more then one value going to associate with the one side.
    - Ex:
      1) A person can have multiple address (means in address table we have to make person id as many side)(set of address)
      2) In multiple choice question exams ,one question have multiple answer (means in answer table we have to make question_id as foreign key column)(set of answer)
      3) A property can have multiple approvals (means approval table property_id became foreign key column)(set of approvals)
      4) A order can have multiple Items (means in items table order_id became foreign key column)(set of items)

- As per the example we will derive the class which will give more clarity.
- Lets see below

```
class Property{
        int propertyId;
        Stirng type;
        ....
        Set<Approval> approvals;
        setters and getters
}
```

property have multiple approvals

```
class Approval{
        int approvalId;
        Date apprDate;
        ....
        //setters and getters
}
```

```
class Question{
        int questionId;
        String que_type;
        Set<Answer> answers;
        //setter and getter
}
```

```
class Answer{
        int answerId;
        String postedName;
        String answer;
        ...
        //setter and getters
}
```

```
class Order{
        int orderId;
        String orderName;
        ..
        Set<Item> items;
        //setter and getter
}
```

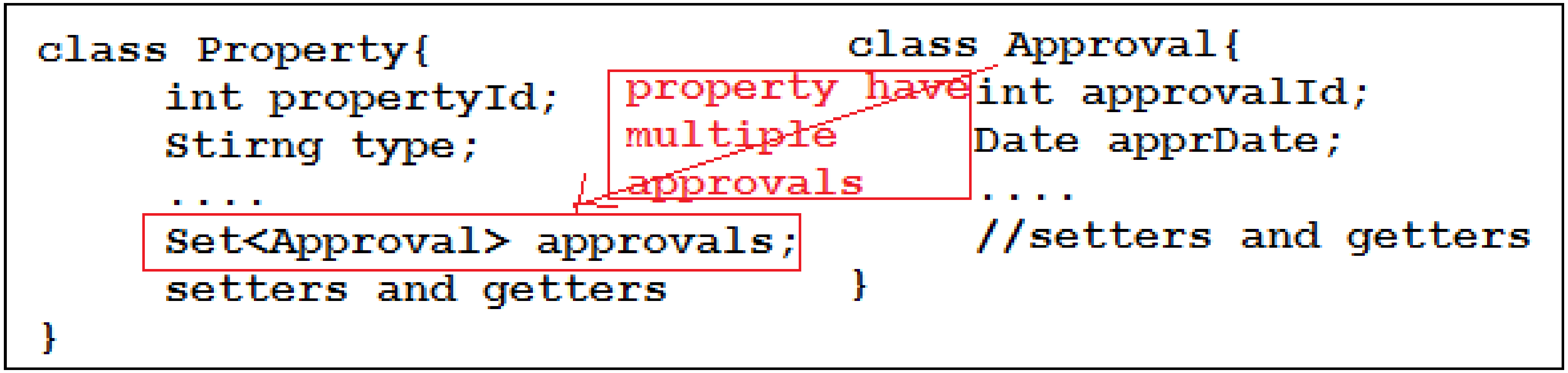
```
class Item{
        int itemId;
        String itemName;
        ..
        //setter and getters
}
```

```
class Person{
        int personId;
        String firstName;
        String lastName;
        ....
        Set<Address> address;
}
```

```
class Address{
        int addressId;
        String street;
        ...
        //setters and getters
}
```

- As per the above example we can easily derive the mapping file.
- Hibernate has provided multiple tags which going to represents the set of values for a specific attributes.
  - Set :
    - We can use set when insertion order is not required and duplicate are not allowed.
    - Set represents the attribute name and where that attribute is belongs to and we can configure cascade, inverse also.
    - Within set we have configure the foreign key column inside the key tag.
  - List
    - We can use list when insertion order will be required and duplicate will be allow.
    - While configuring the list we have to tell to the hibernate take this column and starting index no and maintaining the insertion order.
    - In list also we have to configure the foreign key column under key tag only.
    - While configuring list we have to write and attribute name and corresponding table and cascade , inverse and so on.

- Let see how actually we can able to derive the mapping for property and approval.
- As per the above class one property can have multiple approvals for example electricity approval, water board approval , land cert approvals.... Etc.



```
class Property{
      int propertyId;
      Stirng type;
      ....
      Set<Approval> approvals;
      setters and getters
}
```

```
class Approval{
     int approvalId;
     Date apprDate;
     ....
     //setters and getters
}
```

property have multiple approvals

- Now propertyId will become the foreign key in Approval table. Which represent the many side.
- Let see how we derive the mapping file..

```
Property.hbm.xml
=================
<hibernate-mapping>
    <class name="com.otm.entities.Property" table="PROPERTY">
        <id name="propertyId" type="int">
            <column name="PROPERTYID" />
            <generator class="increment" />
        </id>
     ....//properties....
      <set name="approvals" table="APPROVAL" cascade="all">
            <key>
                <column name="PROPERTYID"/>
            </key>
            <one-to-many class="com.otm.entities.Approval" />
      </set>
    </class>
</hibernate-mapping>
```

```
Approval.hbm.xml
================
<hibernate-mapping>
    <class name="com.otm.entities.Approval" table="APPROVAL">
        <id name="approvalId" type="int">
            <column name="APPROVALID" />
            <generator class="increment" />
        </id>
     //properties
    </class>
</hibernate-mapping>
```

- In property.hbm.xml we configured with set tag which contains
  - Name: it represents the approval which is attribute of type Approval and declared in property class.
  - Table: it is option but for more clarity we have to write, it means property class attribute approval going to inject to Approval table with specified column which is declared inside the key tag.
  - Cascade: there are different ways for persisting data inside the tables with there associate relationship but when we configured cascade as all then it will not consider the association, individually hibernate will persist the data and at last it will it will fire one update for establishing the association.
  - Key : it represent primary kay of property and inject to the foreign key column of approval table which is declared in key tag.
    - Ex:

      -
        ```
        <key>
              <column name="approval_property_id"/>
        </key>
        ```
  - As per our requirement we can make mentioned column as not-null as true.
  - One-to-many : here we are configuring the association by declaring association class name.

- Now there are sort of usecases are there we have to understand how actually hibernate internally performing the opertion.
- 1) one-to-many(unidirectional)
  - If we are not declaring cascade as all then first we have to store the approval then property bz without approval there is no properties available and to update property id with approvals, approvals should be save first.
  - Approval
  - Property
  - Cascade = none
  - Inverse = false
  - Foreign column = null
    - Save(approval)(if you can't then you will get transient exception)
    - Save (property)
    - At last hibernate will check the association and it updates approval_property_id with property_id in approval table.

- 2) one-to-many(unidirectional)
- -Approval
- -Property
- -cascade = all
- -inverse = false
- -foreign column=nullable
  - Save (property)
    - Now cascade will not check association directly it will persist property first and approval next ,after that it will check association and it updates approval_property_id with property_id in approval table.
- 3) one-to-many(unidirectional)
- -Approval
- -Property
- -cascade = all
- -inverse = false
- -foreign column=not – null = true
  - Save (property)
    - Now cascade will not check association directly it will persist property first and while inserting approval hibernate will take property_id column value and stored into approval_property_id column , after that it will check association and it updates approval_property_id with property_id in approval table.
    - Here hibernate duplicating the update query.

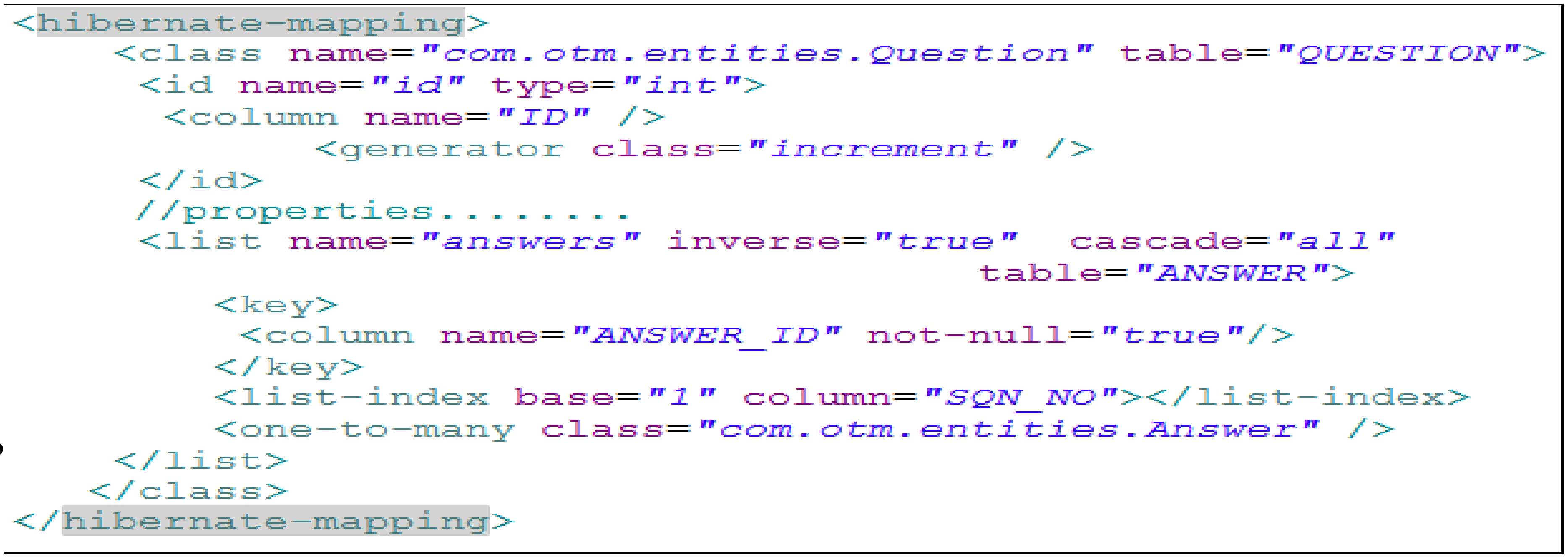
- 4) one-to-many(unidirectional)
- -Approval
- -Property
- -cascade = all
- -inverse = true
- -foreign column=not – null = nullable
  - Save (property)
    - Now cascade will not check association directly it will persist property first and approval , here we enabled the inverse is true mean hibernate will not consider the association which is given into property mapping, it will insert approval with null in approval_property_id column.
    - Now it will not fire update query bz that association owner is other side.
    - Inverse represents I'm not the owner of this association the owner is with my reverse side go and get it from the other side.

- 5) one-to-many(Bi-directional)
- -Approval
- -Property
- -cascade = all
- -inverse = true
- -foreign column=not – null = true
- Before inserting both the table we have to map these classes with bidirectional ways.
- Now approval also have the attribute of property which is the type of Property.
- Write the setters and getters and while storing approval inject property to the approval.
- Approval mapping file now configure one-to-many association.
  - Save (property)
    - Now cascade will not check association directly it will persist property first and approval , here we enabled the inverse is true mean hibernate will not consider the association which is given into property mapping, while inserting approval it will get the property_id from approval and persist the approval with approval_property_id with property_id.
    - Now it will not fire update query bz that insertion happen while inserting only.

- One-to-many with List:
  - List:
  - We can use list when insertion order will be required and duplicate will be allow.
  - While configuring the list we have to tell to the hibernate take this column and starting index number to maintaining the insertion order.
  - In list also we have to configure the foreign key column under key tag only.
  - While configuring list we have to write attribute name and corresponding table and cascade , inverse and so on.
  - If you take an same example which we declared above then the list mapping will look below.

Hibernate 79

Question.hbm.xml



```
<hibernate-mapping>
        <class name="com.otm.entities.Question" table="QUESTION">
          <id name="id" type="int">
            <column name="ID" />
                  <generator class="increment" />
          </id>
          //properties.........
          <list name="answers" inverse="true"  cascade="all"
                                                table="ANSWER">
             <key>
              <column name="ANSWER_ID" not-null="true"/>
             </key>
             <list-index base="1" column="SQN_NO"></list-index>
             <one-to-many class="com.otm.entities.Answer" />
        </list>
      </class>
</hibernate-mapping>
```

- We need to add one more tag i.e. list-index which will talks about the insertion preservation .
- In list-index column talks about the column name where the insertion values will going to store.
- And base is the attribute which will represent the starting of the insertion order. We can give any value but default value is ‘0’.
- Lets see the example.

by Mr.sachin gaikwad

```java
public class Answer {
     private int id;
     private String answername;
     private String postedBy;
     private Question question ;

     //setter and getters
}
```

```xml
Answer.hbm.xml
==============
<hibernate-mapping>
    <class name="com.otm.entities.Answer" table="ANSWER">
        <id name="id" type="int">
            <column name="ID" />
            <generator class="increment" />
        </id>
     //all properties configured
    </class>
</hibernate-mapping>
```

```java
public class Question {
     private int id;
     private String qname;
     private List<Answer> answers;

     //setter and getters
}
```

```xml
Question.hbm.xml
================
<hibernate-mapping>
    <class name="com.otm.entities.Question" table="QUESTION">
        <id name="id" type="int">
            <column name="ID" />
            <generator class="increment" />
        </id>
     //all the properties
     <list name="answers" inverse="true"  cascade="all"
                         table="ANSWER">
         <key>
          <column name="ANSWER_ID" not-null="true"/>
         </key>
        <list-index base="1" column="SQN_NO"></list-index>
        <one-to-many class="com.otm.entities.Answer" />
     </list>
    </class>
</hibernate-mapping>
```

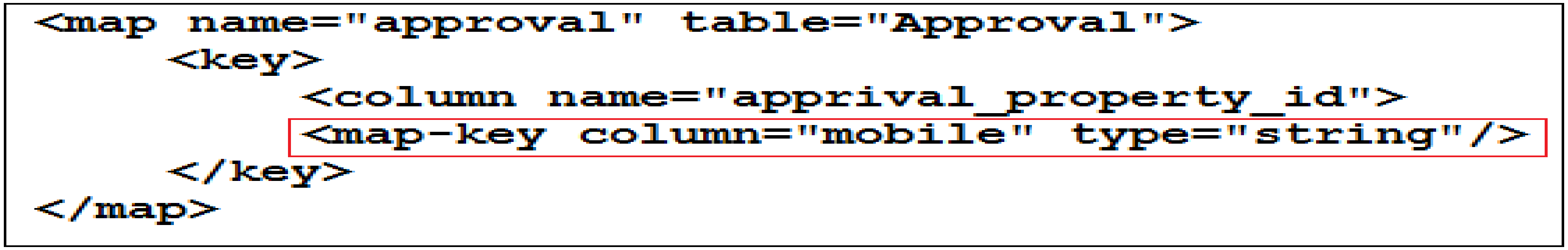
```java
public class OTMTest {
        static List<Answer> answers;
        public static void main(String[] args) {
                SessionFactory sfactory = null;
                Session session = null;
                sfactory =
HibernateUtil.getSessionFactory();
                session = sfactory.openSession();
                session.beginTransaction();
                //save(session);
                getData(session,1);
        session.getTransaction().commit();
        HibernateUtil.closeSessionFactory();
        }
        public static void save(Session session){
                Answer answer1 = new Answer();
                Answer answer2 = new Answer();
                Question question = new Question();
        answers = new ArrayList<Answer>();
                answer1.setAnswername("sachin");
                answer1.setPostedBy("mr.sachin
gaikwad");
                answer2.setAnswername("Rama");
                answer2.setPostedBy("john k");
                        question.setQname("what is ur
first name?");
                answers.add(answer1);
                answers.add(answer2);
                question.setAnswers(answers);
                session.save(question);
        }
}
```

One-to-Many Using List

- While working with association we should have to override the hashCode() and equals() method.
- To make sure if we access same data from the database it should not create the duplicate object for that data.
- While overriding hashcode and equals method make sure you should not generate your hashcode by taking collection attributes and primary key attributes.
- Bz at the time of object creation primary key may be transient and collection may contains multiple data.
- Just take normal attribute which is unique for that object and generate the hashcode.

Hibernate 80

- One-to-Many with Map collection:
- Map also same as other collection tags only just slit different is there.
- Map contains key and value so we have to map the kay which is unique and that column generated by hibernate. But we have to map that column while making the Map collection with the corresponding table.
- Lets see the example.



```
<map name="approval" table="Approval">
    <key>
        <column name="apprival_property_id">
        <map-key column="mobile" type="string"/>
    </key>
</map>
```

- Current table primary key map with provided column i.e. approval_property_id .
- Map-key column which is the key of Map collection created by hibernate.

- When we map map-key into mapping file we should not take that attribute in corresponding class.
- Hibernate is intelligence in binding the key to the corresponding map collection.
- Lets see the example.
-

```
property.hbm.xml
================
<hibernate-mapping>
    <class name="com.otm.entities.Property" table="PROPERTY">
        <id name="propertyId" type="int">
            <column name="PROPERTYID" />
            <generator class="increment" />
        </id>
        //configure all the properties
        <map name="worker" table="WORKER" cascade="all">
            <key>
                <column name="PROPERTYID" />
            </key>
            <map-key column="MOBILE" type="java.lang.String"></map-key>
            <one-to-many class="com.otm.entities.Worker" />
        </map>
    </class>
</hibernate-mapping>
```

{fk}

{map key}

these two columns created into worker table

```
public class Property {

      private int propertyId;
      private String description;
      private String type;
      private String location;
      private String price;
      Map<String,Worker> worker;

      //setter and getter
}
```

```
Worker.hbm.xml
==============
<hibernate-mapping package="com.otm.entities">
      <class name="Worker" table="WORKER">
            <id name="workerId" column="WORKER_ID">
                  <generator class="increment"></generator>
            </id>
            <property name="firstName" column="FIRSTNAME"></property>
            <property name="lastName" column="LASTNAME"></property>
            <property name="workingDate" column="WORK_DT"></property>
      </class>
</hibernate-mapping>
```

```
public class Worker {
      private int workerId;
      private String firstName;
      private String lastName;
      private Date workingDate;

      //setter and getters
}
```

- One-To-Many with Bag:
  - Bag is collection type tag which is provided by hibernate people.
  - Why bag collection has provided by hibernate people?
  - Set collection will store the data in unordered manner but it will not allow duplicate .
  - List collection will store the data in ordered manner but it will allow duplicate value.
  - But sometime requirement is data should be unordered and duplicate also allow bz of that hibernate has provided a collection called bag collection.
  - Bag collection is configured same as set just we need to change the tag name i.e. bag.
  - Bag collection is implementation of collection we can use any collection to map the bag collection data.

- We can map bag as set only lets see the example

```
<bag name="aprovals"  cascade="all" inverse="false">
    <key>
            <column name="P_property_id" />
    </key>
</bag>
```

Hibernate

- Many-to-many using set
- Many-to-many using list
- Many-to-may using map

- Annotation

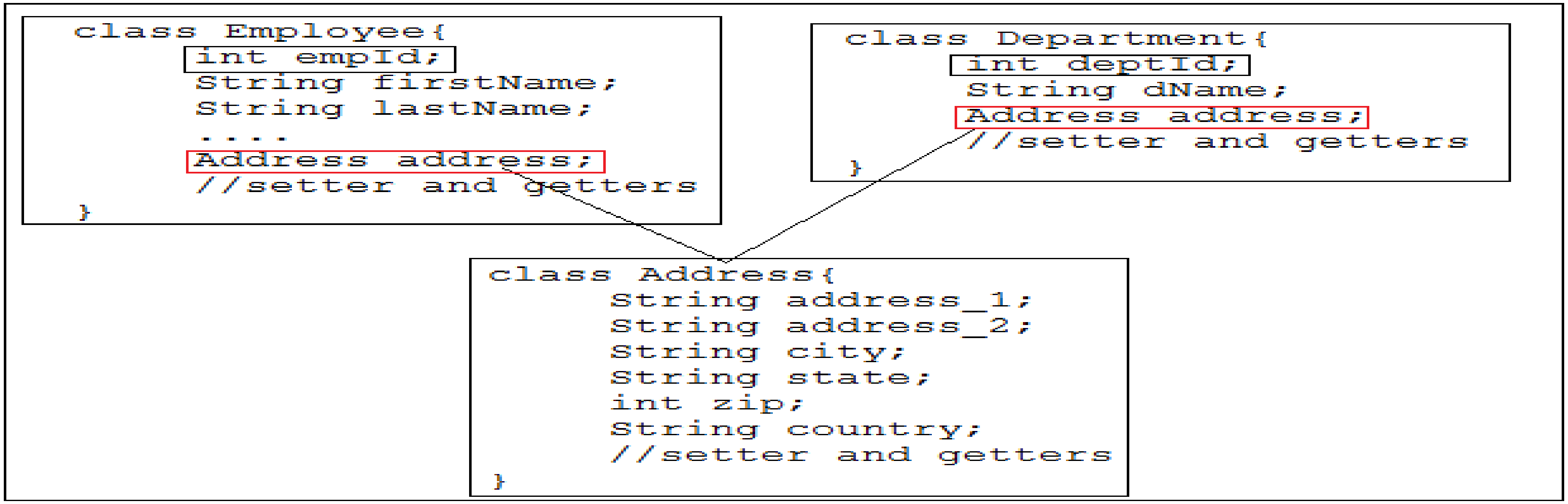
Hibernate 87

- Component Mapping:
  - In association relationship mapping there are different aspects are involved
    - Association mapping
    - Aggregation mapping and
    - Composition mapping
  - As per mapping we covered all the concept which going to overcome the impedance miss match, navigation , identity, inheritance , association problems.
  - Component mapping is one of the important relational mapping which talks about composition.
  - Composition means a object have multipart in it. In other way one object can hold the other object within it called as composition.

- In composition owney is depends on the owner , if owner will die then automatically owney also die.
- If parent is not there means child also not there.
- While persisting the data into database using component mapping along with parent data, child data also get persisted.
- In component mapping child totally depends on the parent. Child doesn’t have any separate ID column to persist it, its totally relay on parent ID only.
- Event though they written in separate classes, while persisting hibernate will marge child attributes into parent and persist the data.
- To make hibernate understand we have to write a mapping file with additional configuration.

- When we query the parent id then corresponding child also get fetched, and when parent get destroyed corresponding child also get destroyed.
- If child data will going to persist into parent table then why we should write separate class for child we can declared into parent class itself.
- For example employee has a address, department has a address and other classes also there which want the address, if we write address attributes into every class then we duplicating the address into several classes.
- To avoid duplicate declaration of the attributes we can write one place and we can easily refer that object to each and every class.
- Lets see the example

component mapping

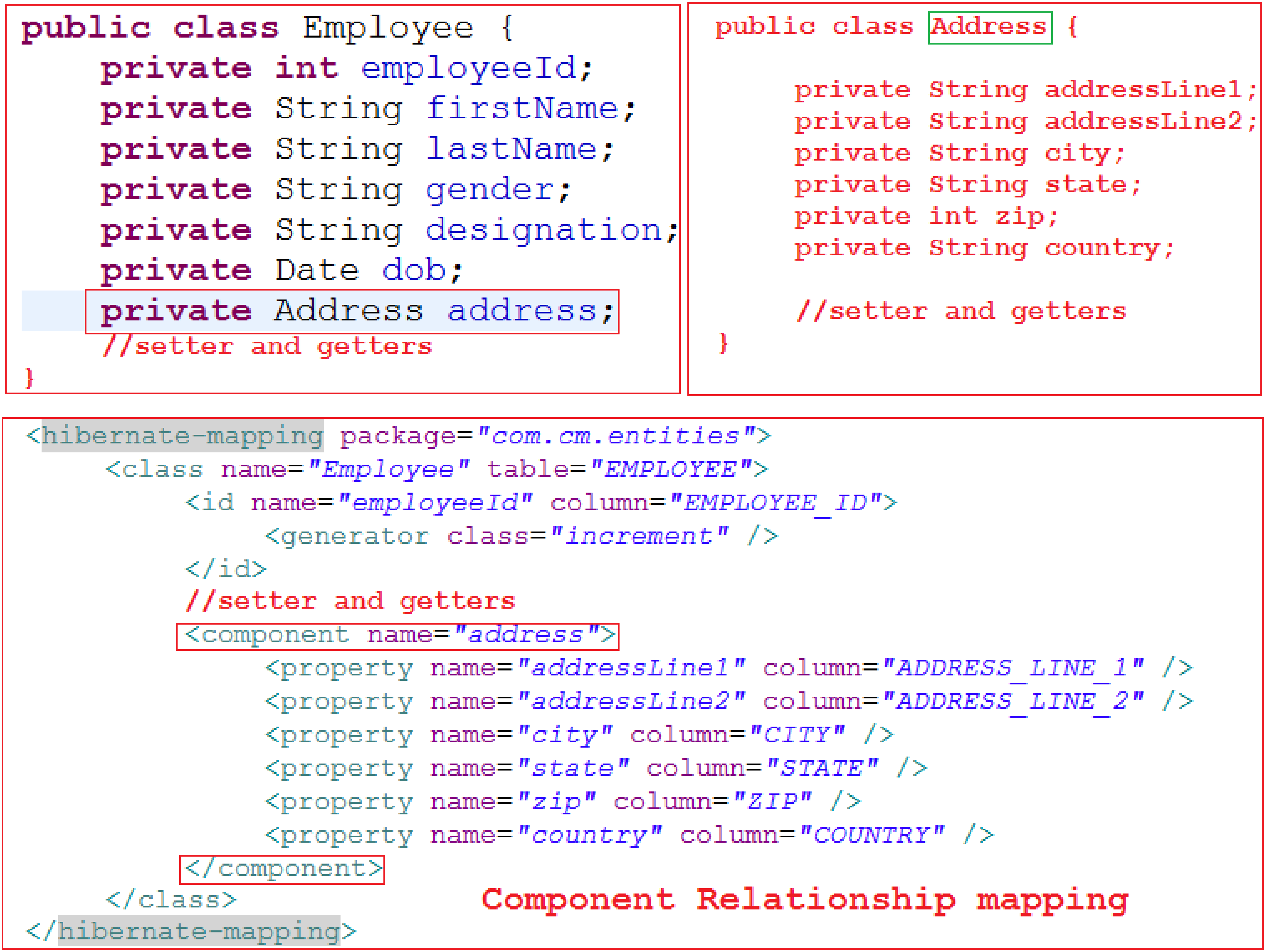


```
class Employee{
        int empId;
        String firstName;
        String lastName;
        ....
        Address address;
        //setter and getters
}
```

```
class Department{
        int deptId;
        String dName;
        Address address;
        //setter and getters
}
```

```
class Address{
        String address_1;
        String address_2;
        String city;
        String state;
        int zip;
        String country;
        //setter and getters
}
```

- As per above example Address class not duplicated into Employee class and Department class, only object reference has declared into both the classes.
- Lets see the full example how to write mapping for component mapping relationship.



```
public class Employee {
    private int employeeId;
    private String firstName;
    private String lastName;
    private String gender;
    private String designation;
    private Date dob;
    private Address address;
    //setter and getters
}
```

```
public class Address {

    private String addressLine1;
    private String addressLine2;
    private String city;
    private String state;
    private int zip;
    private String country;

    //setter and getters
}
```

```
<hibernate-mapping package="com.cm.entities">
    <class name="Employee" table="EMPLOYEE">
        <id name="employeeId" column="EMPLOYEE_ID">
            <generator class="increment" />
        </id>
        //setter and getters
        <component name="address">
            <property name="addressLine1" column="ADDRESS_LINE_1" />
            <property name="addressLine2" column="ADDRESS_LINE_2" />
            <property name="city" column="CITY" />
            <property name="state" column="STATE" />
            <property name="zip" column="ZIP" />
            <property name="country" column="COUNTRY" />
        </component>
    </class>
</hibernate-mapping>
```

Component Relationship mapping

- Using annotation we can write the same application every easily.
- Lets see the example.

```
@Entity
@Table(name="EMPLOYEE")
public class Employee {
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	private int employeeId;
	private String firstName;
	private String lastName;
	private String gender;
	private String designation;
	private Date dob;
	@Embedded
	@AttributeOverrides({@AttributeOverride(name="addre
ssLine1",column=@Column(name="Address1")),
	@AttributeOverride(name="addressLine2",column=@Colu
mn(name="Address2"))})
	private Address address;

	//setter and getters
}
```

```
@Embeddable
public class Address {
	@Column(name="Address_Line_1")
	private String addressLine1;
	@Column(name="Address_Line_2")
	private String addressLine2;
	private String city;
	private String state;
	private int zip;
	private String country;

	//setter and getter
}
```

```
class Test{
	main(){
	//create sessionFactory
	//create session
	//create employee and address
	object
	//set address value
	//set employee values
	//set address object to
employee
	//session.save(employee);
	//commit();
	}
}
```

Annotation based Component mapping

hibernate

- Transaction Management
- Contextual Session
  - JtaSessionFactory
  - ThreadLocalSessionFactory
  - ManagedSessionFactory