FRAMEWORK

- Framework is ready solution for common problems.
- The different frameworks are
  - i. Akka
  - ii. Struct
  - iii. JSF (java server face)
  - iv. Velocity
  - v. Voadin
  - vi. Spark
  - vii. Rat-pack
  - viii. Vertex
  - ix. GWJ
  - x. Jackson
  - xi. Hibernate
  - xii. Spring

Hibernate:

- To overcome the problems of JDBC, we go for hibernate

DTO(Data Transfer Object):

- DTO is a class
- DTO is a DP(Design Pattern)
- DP is a class but which follows some rules .
- There are 5 rules
  1. Class must implement serializable.
  2. Class must be non-final
  3. All properties in a class must be private.
  4. Class must have at least default constructor.
  5. Properties may have setter and getters.

- DTO is used to transfer the data within the application.

  1. Initializing DTO class.

```
package com.hdb.hibernateDTO;

import java.io.Serializable;

public class WeaponDTO implements Serializable{

	String type;
	double price;
	double range;
	String model;
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}
```

```
	public double getPrice() {
		return price;
	}
	public void setPrice(double price) {
		this.price = price;
	}
	public double getRange() {
		return range;
	}
	public void setRange(double range) {
		this.range = range;
	}
	public String getModel() {
		return model;
	}
	public void setModel(String model) {
		this.model = model;
	}


}


package com.hdb.hibernate.dto.weaponDTO;

import com.hdb.hibernateDTO.WeaponDTO;

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

		WeaponDTO weaponDTO=new WeaponDTO();
		weaponDTO.setType("Rifle");
		weaponDTO.setPrice(10000);
		weaponDTO.setRange(500.00);
		weaponDTO.setModel("AK-47");

	}

}
```

DAO: (Data Access Object)

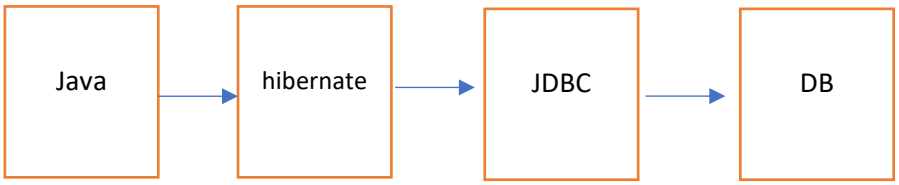
- DAO is a class
- DAO is used to write DB logic.
- DAO will have methods to perform operations on DTO.

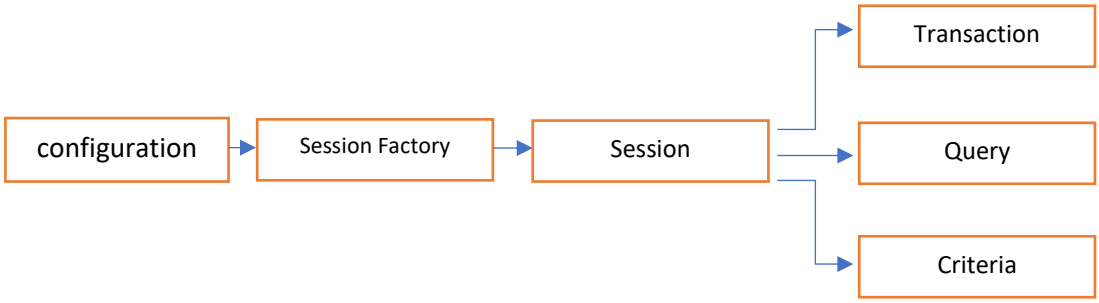
```
package com.hdb.hibernate.dao;

public class WeaponDAO {


	public void save(WeaponDAO weaponDTO) {
		System.out.println("should impl");
	}
```

FRAMEWORK

```
        public void update(WeaponDAO dto) {
                System.out.println("should impl");
        }
        public void delete(WeaponDAO dto) {
                System.out.println("should impl");
        }
        public WeaponDAO getweapon() {
                System.out.println("should impl");
                return null;
        }
}
```

- Hibernate is an open source DB framework
- Hibernate is used to perform CRUD operation.
- Hibernate is an implementation of ORM(Object Relation Mapping).
- Hibernate is a JPA (Java Persistance API)standard.





Configure()
AddAnotationClass(.class)
buildSessionFactory()

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
```

FRAMEWORK

```
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
                <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
                <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/universal</property>
                <property name="hibernate.connection.username">root</property>
                <property name="hibernate.connection.password"></property>

        </session-factory>
</hibernate-configuration>
```

ORM: (Object Relational Mapping or Model)

- ORM are just guidelines
- ORM provides guidelines to map object model with relational model.
- ORM allows to persist an object without converting object into values and vice-verse

ORM Problems:

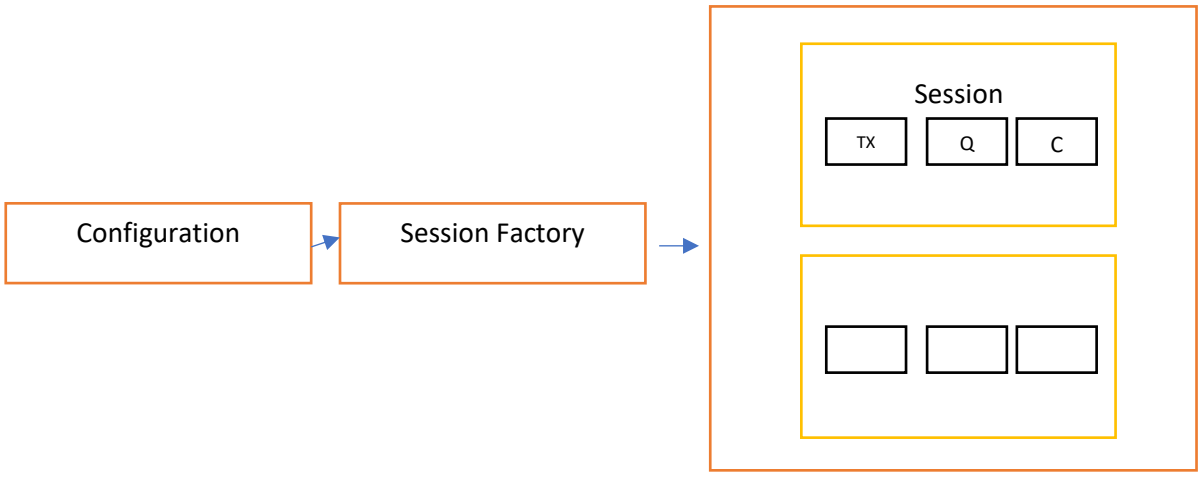
- Problems of relation's
- Problems of inheritance.
- Problems of identity
- Problems of navigation
- Problems of grain

Examples of ORM Tools: Hibernate, IBatis, Toplink etc.

JPA: (Java persistence API)

- JPA is a standard API.
- **JPA** provides standards for ORM problems.
- Hibernate is a JPA standard.
- JPA consists of standard library and annotations.
- **Javax.persistence** is the standard package.

Hibernate Architecture:



By: HDB

FRAMEWORK

Org.hibernate.cfg.Configuration :

- Configuration is a class
- Configuration is the basic component in hibernate.
- There are three uses of configuration

1. **Configure()**
   Hibernate configuration is used to provide config properties
   Note: config object is used to parse hibernate xml file.

```
Configuration configuration=new Configuration();
configuration.configure(); // by default hibernate.cfg.xml
```

OR

```
Configuration configuration=new Configuration();
configuration.configure("mysql.xml");// overloaded method
```

2. **addAnnotatedClass()**
   it is used to provide mapping information
   Note: **Entity** is a class which is mapped to database table.
   configuration.addAnnotatedClass(WeaponDTO.class);
   this method takes Entity.class as an argument
   this method should be invoke for each Entity.

   <Mapping> Tag: is used to provide mapping information
   This tag should be used in hibernate config lines.
   Ex.

```
<mapping class="com.hdb.hibernate.dto.WeaponDTO"/>
```

   Class Attribute should have fully qualified name of the Entity as value.

3. **buildSessionFactory()**
   Configuration is used to create and get reference of session factory

```
SessionFactory factory=configuration.buildSessionFactory();
```

4. **org.hibernate.SessionFactory**
   Session factory is an abstraction
   there are 3 uses of session factory
   a. it is used to configure database
   b. it is used to manage connections and entity
   c. it is used to get reference of session

characteristics of session factory

- session factory should be created only once for an application

FRAMEWORK

- session factory is immutable
- session factory is thread safe

{ Tomcat JNDI, C3PO }

```
package com.hdb.hibernate.dto;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="weapon_table")

public class WeaponDTO implements Serializable{

	@Id
	@Column(name="weapon_id")
	private int wid;
	@Column(name="weapon_type")
	private String type;
	@Column(name="weapon_price")
	private double price;
	@Column(name="weapon_Range")
	private double range;
	@Column(name="weapon_model")
	private String model;

	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}
	public double getPrice() {
		return price;
	}
	public void setPrice(double price) {
		this.price = price;
	}
	public double getRange() {
		return range;
	}
	public void setRange(double range) {
		this.range = range;
```

```
	}
	public int getWid() {
		return wid;
	}
	public void setWid(int wid) {
		this.wid = wid;
	}
	public String getModel() {
		return model;
	}
	public void setModel(String model) {
		this.model = model;
	}
	@Override
	public String toString() {
		return "WeaponDTO [wid=" + wid + ", type=" + type + ",
price=" + price + ", range=" + range + ", model=" + model
	+ "]";
	}

}
```

```
	package com.hdb.hibernate.dao;

	import org.hibernate.Session;
	import org.hibernate.SessionFactory;
	import org.hibernate.Transaction;
	import org.hibernate.cfg.Configuration;

	import com.hdb.hibernate.dto.WeaponDTO;

	public class WeaponDAO {

		public void save(WeaponDTO weaponDTO) {
			System.out.println("should impl");

			Configuration configuration=new Configuration();
			configuration.configure();
	//		configuration.addAnnotatedClass(WeaponDTO.class);
			SessionFactory factory=configuration.buildSessionFactory();
			Session session=factory.openSession();
			Transaction transaction=session.beginTransaction();
			session.save(weaponDTO);
			transaction.commit();
			session.close();
			factory.close();
```

```
        }

        public WeaponDTO readbyId(int id) {

                Configuration configuration=new Configuration();
                configuration.configure();
//              configuration.addAnnotatedClass(WeaponDTO.class);
                SessionFactory factory=configuration.buildSessionFactory();
                Session session=factory.openSession();
                Transaction transaction=session.beginTransaction();
                WeaponDTO dto=session.get(WeaponDTO.class, id);
                session.close();
                factory.close();
                return dto;

        }
        public void update(WeaponDAO dto) {
                System.out.println("should impl");
        }
        public void delete(WeaponDAO dto) {
                System.out.println("should impl");
        }
        public WeaponDAO getweapon() {
                System.out.println("should impl");
                return null;
        }
    }
```

```
package com.hdb.hibernate.util;

import com.hdb.hibernate.dao.StudentDAO;
import com.hdb.hibernate.dao.WeaponDAO;
import com.hdb.hibernate.dto.StudentDTO;
import com.hdb.hibernate.dto.WeaponDTO;

public class Tester {

    public static void main(String[] args) {

        WeaponDTO weaponDTO =new WeaponDTO();
        weaponDTO.setType("long-1");
        weaponDTO.setModel("hdb");
        weaponDTO.setRange(500.00);
        weaponDTO.setPrice(100);
        WeaponDAO weaponDAO=new WeaponDAO();

        weaponDAO.save(weaponDTO);
```

FRAMEWORK

```
    /*
        StudentDTO studentDTO=new StudentDTO();
        studentDTO.setName("hdb");
        studentDTO.setAge(23);
        studentDTO.setNumber(959041040);

        StudentDAO studentDAO=new StudentDAO();
        studentDAO.save(studentDTO);*/


    }

}
```

org.hibernate.Session:

- ✓ session is an interface
- ✓ session is an abstract
- ✓ session is mainly used to perform CRUD operation.
- ✓ **Session** is used to get references of transaction query and criteria.

Operations	Methods	SQL
C	save()	Insert
R	get()	Select
U	update()	Update
D	delete()	delete

- ✓ Session requires a connection to perform CRUD operations.

org.hibernate.Transaction;

- ✓ transaction is a boundary, which consists of set of operations
- ✓ Transaction should be success or failure.
- ✓ Transaction consists of two functions
  a. Commit()
  b. Rollback()

- ✓ Transaction is required only for write operations
  Note: in hibernate, autoCommit mode is false.

```
Transaction transaction=session.beginTransaction();
```

- ✓ This component is required to commit() or rollback()
- ✓ There are two method of Transaction
  a. transaction.commit();
  b. transaction.rollback();

FRAMEWORK

implementing singleton class of session factory:

i. create java project
ii. create standard package and util package
iii. add hibernate libraries into build path
iv. write singleton class

```
package com.hdb.hibernate.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public final class SessionFactory1 {

	private final static SessionFactory factory;

	public static SessionFactory getFactory() {
		return factory;

	}
	static {
		Configuration configuration=new
Configuration().configure();
		factory=configuration.buildSessionFactory();
	}
}
```

Handling session and Transaction:

```
	try {
		session.save(armyDTO);
		transaction.commit();

	}catch(HibernateException e)
	{
		e.printStackTrace();
		transaction.rollback();
	}
	finally {
		session.close();
	}
```

ID generator:

```
	@Id
	@GenericGenerator(name="hdb", strategy="increment")
	@GeneratedValue(generator="hdb")
	@Column(name="army_id")
		private int id;
```

FRAMEWORK

Generator:

- Generators are used to automatically generate primary key.
- Generators depends on Data base.

Examples:

1. Increment(mysql)
2. Sequence (Oracle)
3. Identity (serval)

- Assigned is the default generator

@GenericGenerator

- This annotation is used to create an object of identifier generator
- GenericGenerator has two attributes
  i. Strategy
  ii. Name

- Value for strategy should be short names predefined by hibernate or fully qualified name of class which implements identifier generator.
- Name attribute acts as reference for the generated object
- This annotation is used to generate and set the primary key
- This annotation has an attribute generator
- Generator should have name of the generic generator as value

Implementing custom generator:

1. Write a class and implement identifier generator

```
package MyGenerator;

import java.io.Serializable;

import org.hibernate.HibernateException;
import org.hibernate.engine.spi.SessionImplementor;
import org.hibernate.id.IdentifierGenerator;

public class mygenerator  implements IdentifierGenerator{

	public mygenerator() {

		System.out.println("Mygenerator created......");
	}

	@Override
	public Serializable generate(SessionImplementor arg0, Object
arg1) throws HibernateException {
```

```
            System.out.println("calling generate method");

            return 786;
        }


}
```

- Provide fully qualified name as value in strategy

```
@Id
@GenericGenerator(name="hdb",strategy="MyGenerator.mygenerator")
@GeneratedValue(generator="hdb")
@Column(name="army_id")
```

Query:

- **There are 4 ways to load data using hibernate**
  i. Session
  ii. [native query] SQL
  iii. Hibernate query language [HQL]
  iv. Criterion API

SQL : Select army.* from army_table army where army_country='india'
HQL: select army from ArmyDTO army where country_name='india'

HQL:
select watch from from WatchDTO watch where watch.barnd='Fossil'
Select watch from WatchDTO watch where watch.price=999;
Select watch.wid from WatchDTO watch where watch.brand='titan';

Update WatchDTO watch set watch.brand='something' where watch.wid=101
Delete from WatchDTO where brand='titan'

HQL (hibernate query language)

- HQL allows to write query based on object model
- HQL uses object model to execute statements.
- HQL is independent of Database
- HQL supports select, update and delete operation

```
package com.hdb.hibernate.dao;

import org.hdb.hibernate.util.SessionFactory1;
import org.hibernate.Query;
import org.hibernate.Session;
```

FRAMEWORK

```
import org.hibernate.SessionFactory;

import com.hdb.hibernate.dto.armyDTO;

public class armyHQLDAO {

	private SessionFactory factory=SessionFactory1.getFactory();
	public armyDTO fetchByCountryName(String cname) {
		/* HQL - 2 steps
		 * 1. create a query
		 * 2. process the query
		 */
		Session session=factory.openSession();

		String hql="select army  from armyDTO army where
army.countryName='"+cname+"'";
		try {
		Query query=session.createQuery(hql);
		armyDTO dto=(armyDTO) query.uniqueResult();
		return dto;

		}finally
		{
			session.close();
		}

	}
}
```

Positional parameters:

```
Session session=factory.openSession();

	String hql="select army  from armyDTO army where
army.countryName=?";

	try {
	Query query=session.createQuery(hql);
	query.setString(0, cname);
	armyDTO dto=(armyDTO) query.uniqueResult();
	return dto;

	}finally
	{
		session.close();
	}
```

FRAMEWORK

Named parameters:

- named parameters should be used instead of positional parameters i.e. ?

```
    public armyDTO fetchByCountryNPName(String cname) {
          /* HQL - 2 steps
           * 1. create a query
           * 2. process the query
           */
          Session session=factory.openSession();

          String hql="select army  from armyDTO army where
army.countryName=:cn";
          try {
          Query query=session.createQuery(hql);
          query.setParameter("cn", cname);
          armyDTO dto=(armyDTO) query.uniqueResult();
          return dto;

          }finally
          {
                session.close();
          }

    }
```

Unique Result: query.uniqueResult()

- This method should be used when select statements returning an object.
- Unique Result will return different types.

Note:

- Unique results an object and casted into below types
  a. Entity - select army
  b. String - army.armyType
  c. Number - army.id (with help of wrapper class such as Integer, Float etc.)
  d. [] - army.id,army.armyType,army.countryName

States of Entity:

- There are three states of an Entity
  a. Transient
  b. Persistent
  c. Detached

a. Transient
   - ✓ This is a state where an entity is newly created and **not associated** with session
   - ✓ In this state entity will not have primary key

b. Persistent

- ✓ When an entity is invoked using session.save(Entity), here entity is **associated** with session.
- ✓ In this state, entity will have primary key

c. Detached

- ✓ This is the state where it was **previously associated** with session
- ✓ So in this state, Entity is **not associated** with session.
- ✓ In this state, entity may or may not have primary key. It had primary key

```
package com.hdb.hibernate.dao;

import org.hdb.hibernate.util.SessionFactory1;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.hdb.hibernate.dto.armyDTO;

public class armyHQLDAO {

	private SessionFactory factory=SessionFactory1.getFactory();

	public armyDTO fetchByCountryName(String cname) {
		/* HQL - 2 steps
		 * 1. create a query
		 * 2. process the query
		 */
		Session session=factory.openSession();

//		String hql="select army  from armyDTO army where
army.countryName=?";

		String hql="select army  from armyDTO army where
army.countryName=:cn";
		try {

		Query query=session.createQuery(hql);

		query.setParameter("cn", cname);

		armyDTO dto=(armyDTO) query.uniqueResult();

		return dto;

		}finally
		{
```

```
                        session.close();
                }


        }



        public void fetchById(int id) {
Session session=factory.openSession();

                String hql="select army.armyType  from armyDTO army where
army.id=:id";
                try {
                Query query=session.createQuery(hql);
                query.setParameter("id", id);
                String type=(String) query.uniqueResult();
                System.out.println(type);
                }finally
                {
                        session.close();
                }

        }
        public armyDTO fetchByCountryNPName(String cname) {
                /* HQL - 2 steps
                 * 1. create a query
                 * 2. process the query
                 */
                Session session=factory.openSession();

                String hql="select army  from armyDTO army where
army.countryName=:cn";
                try {
                Query query=session.createQuery(hql);
                query.setParameter("cn", cname);
                armyDTO dto=(armyDTO) query.uniqueResult();
                return dto;

                }finally
                {
                        session.close();
                }

        }
        //select * from amry where id=?;

        public Object[] fetchByIdAndCountryNameByNoOfRec(int rec) {
                /* HQL - 2 steps
                 * 1. create a query
                 * 2. process the query
```

```
                 */
                Session session=factory.openSession();
                String hql="select army.id, army.countryName  from armyDTO army
where army.noOfRec=:nor";
                try {
                Query query=session.createQuery(hql);
                query.setParameter("nor", rec);
                        Object[] arry= (Object[]) query.uniqueResult();
                        System.out.println(arry.length);
                        return arry;

                }finally
                {
                        session.close();
                }


        }
        public String fetchByCountryNPName(int id) {
                /* HQL - 2 steps
                 * 1. create a query
                 * 2. process the query
                 */
                Session session=factory.openSession();

                String hql="select army.countryName  from armyDTO army where
army.id=:cn";
                try {
                Query query=session.createQuery(hql);
                query.setParameter("cn", id);
                String  d=(String) query.uniqueResult();
                return d;

                }finally
                {
                        session.close();
                }
        }
        public Object [] fetchtyperecbyCname(String cname) {
                /* HQL - 2 steps
                 * 1. create a query
                 * 2. process the query
                 */
                Session session=factory.openSession();

                String hql="select army.noOfRec,army.armyType  from armyDTO
army where army.countryName=:cn";
                try {
                Query query=session.createQuery(hql);
                query.setParameter("cn", cname);
                Object[]  d=(Object[]) query.uniqueResult();
```

```
                return  d;

                }finally
                {
                        session.close();
                }

            }


        }


query.list();
```

```
package com.hdb.hibernate.dao;

import java.util.List;

import org.hdb.hibernate.util.SessionFactory1;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.hdb.hibernate.dto.WeaponDTO;

public class WeaponHQLDAO {

    SessionFactory factory=SessionFactory1.getFactory();

    public List<WeaponDTO> fetchAll(){

        String hql="select weapon from WeaponDTO weapon";
        Session session=factory.openSession();

        try {
            Query query=session.createQuery(hql);
        List<WeaponDTO> list=query.list();
        return list;
        } finally {
            session.close();
        }


    }
}

package com.hdb.hibernate.util;

import java.util.List;
```

FRAMEWORK

```
import com.hdb.hibernate.dao.WeaponHQLDAO;
import com.hdb.hibernate.dto.WeaponDTO;

public class WeaponHQLTester {

public static void main(String[] args) {
      WeaponHQLDAO hqldao=new WeaponHQLDAO();
                  List<WeaponDTO> list=hqldao.fetchAll();

                  list.forEach(a->System.out.println(a.getModel()));
}
}
```

OR

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

      new WeaponHQLDAO().fetchAll().forEach(dto-
>System.out.println(dto.getPrice()));


      }
```

@NamedQuery

@NamedQuery(name="fetchAll",query="select weapon from WeaponDTO weapon")

@NamedQueries

```
@NamedQueries({@NamedQuery(name="fetchAll",query="select
weapon from WeaponDTO weapon"),
@NamedQuery(name="fetchAllTypes",query="select weapon from
WeaponDTO weapon")})
```

1. **Problems of Relations**

- in java, one can be related with other classes
- if one class related with another class of different type, we refer as Has a Relation.
- Has a can be called associations or composition.
- There are two factors to decide type of relations.
  - i. Numbers
  - ii. Directions

Types of Has a Relations

1. One to one
2. One to many
3. Many to one

FRAMEWORK

4. Many to many

1. One to One

➢ When a class is having reference of another class once we call it as One to One.

```
Public class MovieDTO{
Private int movieID;
Private String producer;
Private double budget;

}
Public class movieDAO{
{
Public Integer saveAndReturnId(MovieDTO dto)
{
Return 0;
}
Public void updateBudgetByName(String name,double budget)
{
}
Public List<MovieDTO> fetchAll() (){return null;}
Public String fetchProducerNameByMovieNmae(String mname) return “”;
Public long fetchCount(){}
Public Decimal fetchMaxBudget(){}
```

Lazy Loading:

- Lazy loading is used to load entities only when the properties of an entity is used
- By default lazy is enabled in session.
- Example: session.load()

load()	get()1
1. Lazy loading	
2. When we use property	
3. Proxy object or entity	
4.	

1. Difference b/w get and load
2. Diff b/w flush and clear
3. Diff b/w clear and evict
4. Diff b/w clear and close
5. Diff b/w flush and commit
6. Diff b/w session and persist
7. Diff b/w update and merge

FRAMEWORK

8. Diff b/w save and saveOrUpdate

- Hibernate has two levels of cache
  1. First level (session cache)
  - Can not be configured
  2. Second level (session factory cache)
  - Can be configured and be shared b/w session object.

Cascade:

- Cascade is used perform operations on associated entities
- Cascade can be used with entity associations or relations
- It can be configured with below relation
  1. One to one
  2. One to many
  3. Many to one
  4. Many to many
- @OneToOne(cascade=CascadeType.ALL)
- Cascade should have cascade type as value.
- Values : ALL,PERSIST MERGE, REMOVE, REFRESH, DETACH

```
Public enum CascadeType{
        ALL,
        PERSIST,
        MERGE,
        REMOVE,
        REFRESH,
        DETACH
}
```

Annotations:

@OneToOne :

this is used to relate an entity with another entity.

This annotation is bidirectional

@PrimaryKeyJoinColumn

This annotation is used to relate primary of owning entity as join column.

This annotation should be used on owning entity.

Note:

This annotation will consider the column used with @Id as primary key.

@JoinColumn

Join column means foreign key

Join column will be present in associated table.

In one to one, this annotation should be used on associate entity.

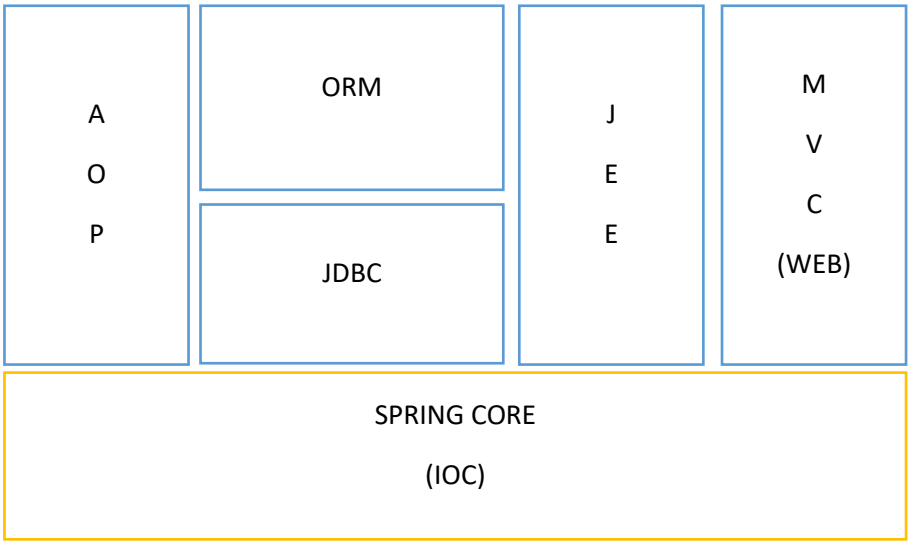
Spring framework

- It is an open source enterprise application framework.
- Spring is an important integration technology.
- Spring provides infrastructure for building applications.

FRAMEWORK

- Spring is an implementation of IOC.
- Spring is used to manage components of application.

Spring Modules:

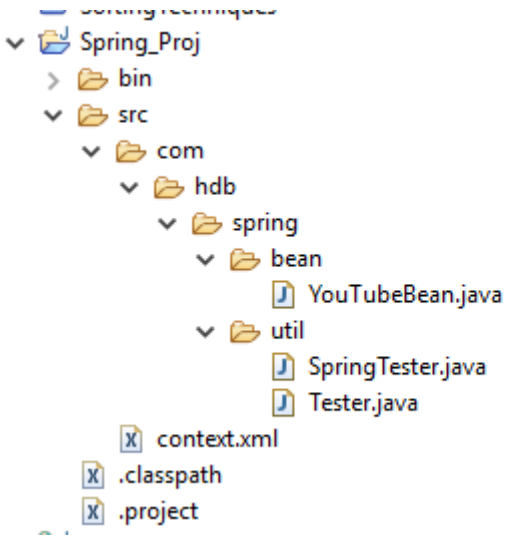


AOP- aspect oriented programming
ORM- object relational mapping

Spring CORE:-

- Sprig core module is used to manage components of an application
- Spring core is the base module in spring framework.
- Spring core can also be referred to as core container.

1. create a java project
2. create standard package, bean and util packages
3. copy and add jar to build path
4. copy string configuration file into class path



FRAMEWORK

```
package com.hdb.spring.bean;


public class YouTubeBean {

	public YouTubeBean() {
		System.out.println(this.getClass().getSimpleName()+"
created");
	}
	public void play(String name) {
		System.out.println("playing "+name);
	}

}


package com.hdb.spring.util;


import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.hdb.spring.bean.YouTubeBean;



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

		ApplicationContext spring=new
ClassPathXmlApplicationContext("context.xml");

		YouTubeBean
youTubeBean=spring.getBean(YouTubeBean.class);
		youTubeBean.play("hdb");



	}

}
```

FRAMEWORK

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

      <bean id="youTubeBean"
class="com.hdb.spring.bean.YouTubeBean"></bean>


      </beans>
```

Bean:

- bean is an object which is managed by spring framework.
- To declare a bean, we need to use bean tag in spring configuration file.
- Spring will create a bean object by invoking default constructor of the class.
  Note: spring uses reflection API.

Starting and initializing spring framework:

```
ApplicationContext spring=new
ClassPathXmlApplicationContext("context.xml");
```

To initialize we need to pass spring configuration location or name which is present classpath as string.

Note : spring configuration file should be present in class path.

.getBean()
This method is present in application context.
This method is used to get reference of the bean object which is managed by spring framework.
YouTubeBean youTubeBean=spring.getBean(YouTubeBean.class);

getBean() method takes .class as argument and return reference of the bean.

Container:
Container is an object which manages life cycle of other objects.
Examples : JEE container, EJB container, JMS container and Spring container.

Spring container:
Spring container is used to manage life cycle of the object.
Examples of beans: DAO connection, Session Factory.

FRAMEWORK

Types of Spring Container:

1. BeanFactory
   ↑ (I)
2. ApplicationContext
   ↑ (I)
3. webApplicationContext

1. **BeanFactory :**
   BeanFactory is the core container.
   BeanFacory is inherited by applicationContext

2. ApplicationContext:
   applicationContext is a BeanFactory
   applicationContext is a container.
   applicatiionConext support AOP.

   Note:
   it is suitable for developing web applications as it supports AOP.
   It supports secondary concern like security, Transaction, Multilanguage's . Themes etc.

   Differences:
   Bean Factory – functional or primary concern
   Application Context –Non- functional or Secondary concern

   Creating and initializing the Spring container :

```
ApplicationContext spring=new
ClassPathXmlApplicationContext("context.xml");
```

   There are two ways to configure spring framework
   1. Xml
   2. Java configurations (@)

```
ApplicationContext spring=new
ClassPathXmlApplicationContext("context.xml");

YouTubeBean youTubeBean=spring.getBean("youTubeBean",
YouTubeBean.class);
```

Initializing Spring Bean:

FRAMEWORK

- There are two ways to initialize property of a bean
  1. Constructor
  2. Setter

  Constructor initialization:

```
public CameraBean(String pixel) {
this.pixel=pixel;
System.out.println(this.getClass().getSimpleName()+"
created");
}
```

```
<bean id="CameraBean"
class="com.hdb.spring.bean.CameraBean">
<constructor-arg value="100"></constructor-arg>
</bean>
```

- <constructor-arg> tag is used to pass an argument to a constructor
- This tag should be used depending number of parameter in the constructor
- Value attribute is used whenever the parameter type is a string or a number

```
public CameraBean(double price) {
this.price=price;
System.out.println(this.getClass().getSimpleName()+" created");
}
```

```
<bean id="CameraBean" class="com.hdb.spring.bean.CameraBean">
<constructor-arg value="67" type="double"></constructor-arg></bean>
```

- type attribute is used to specify the argument type for constructor
  Note:
  Spring container by default gives or considers String as the argument type.

Two argument constructor calling:

```
public TrimBean(String batteryBrand, boolean working)
{
this.batteryBrand = batteryBrand;
this.working = working;
}
```

```
</bean>

<bean id="trimmerBean" class="com.hdb.spring.bean.TrimBean">
<constructor-arg value="excide"></constructor-arg>
<constructor-arg value="true"></constructor-arg>
</bean>
```

```
public void trim(String trim)
{
if(this.working)
System.out.println("trimming "+trim+" with battery"+batteryBrand);
else
System.out.println("not working");
}
```

FRAMEWORK

Dependency injection:

```
package com.hdb.spring.bean;

public class HotSpot {

            private String operator;
            private Power power;
            public HotSpot() {

      System.out.println(this.getClass().getCanonicalName()+"
created.");
            }

            public void connect() {
                  if(power!=null) {
                        System.out.println(" connecting.....");
                        power.generate();
                  }else {
                        System.out.println("No power");
                  }

            }

            public String getOperator() {
                  return operator;
            }

            public void setOperator(String operator) {
                  System.out.println("calling set method");
                  this.operator = operator;
            }
```

```
            public Power getPower() {
                  return power;
            }

            public void setPower(Power power) {
                  this.power = power;
            }
}
package com.hdb.spring.bean;

public class Power {

      private int volts;

      public Power() {

            System.out.println(this.getClass().getName()+"
created..");
      }

      public void generate() {
            System.out.println("genereted..");
      }

      public int getVolts() {
            return volts;
      }

      public void setVolts(int volts) {
            System.out.println("calling set volt method");
            this.volts = volts;
      }
}

package com.hdb.spring.util;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.hdb.spring.bean.HotSpot;

public class DependencyTester {

      public static void main(String[] args) {

            ApplicationContext context=new
ClassPathXmlApplicationContext("dependency.xml");
```

```
        HotSpot hotSpot=(HotSpot) context.getBean("hotSpot");
        hotSpot.connect();

    }
}
```

```
<bean id="hotSpot" class="com.hdb.spring.bean.HotSpot">
<property name="operator" value="Airtel"></property>
<property name="power" ref="power"></property>
</bean>

<bean id="power" class="com.hdb.spring.bean.Power">
<property name="volts" value="12"></property>
</bean>
```

OUTPUT:

```
com.hdb.spring.bean.HotSpot created.

com.hdb.spring.bean.Power created..
calling set volt method
calling set method
connecting.....
genereted..
```

- dependency injection is a process of referring an object.
- There are two types of dependency injection
  1. Constructor injection
     Injecting an object using constructor.
  2. Setter injection
     Injecting an object using setter.
- For dependency injection, we should use **ref** attribute
- Ref attribute can be used in both <constructor-arg> and <property>
- The value for attribute should be id of the dependent bean.

Life cycle of spring bean:

- Spring bean life cycle is managed by spring container
- Life cycle methods are invoked only once by the container.

init-method="initResource"

- This attribute is used to invoke the initialization method of the bean post construction.

```
<bean id="browser" class="com.hdb.spring.bean.Browser" init-method="initResource">
<property name="name" value="chrome"></property>
</bean>
```

- destroy-method:

FRAMEWORK

- this attribute is used to invoke a method to clear resources of the bean.
- This method will be invoked by the container only once on shutdown.

```
<bean id="browser" class="com.hdb.spring.bean.Browser" init-method="initResource" destroy-method="destroyResource">
<property name="name" value="chrome"></property>
</bean>
```

- ClassPathXmlApplicationContext ctx=(ClassPathXmlApplicationContext) context;
  ctx.close();

  OR
- ClassPathXmlApplicationContext context=**new** ClassPathXmlApplicationContext("springB.xml");
  context.registerShutdownHook();

  Constructor → setters → init-method → destroy-method

IOC:

- **IOC is an abstract design principle.**
- **IOC is a process of giving control to an external entity.**
- **Inversion of control is implemented by spring framework.**

  Note:
- **Dependency injection is implementation of IOC.**

  Java configuration for configuring spring framework:
- **There are two way to configure spring framework.**
  1. Xml
  2. Java configuration(annotation)

  To configure using annotation, we need to use context namespace

  There are two steps to configure annotation:
  1. Declare <context:component-scan base-package="com.hdb.spring"></context:component-scan>
  2. Base package should have standard package name as value
     Use @component annotation above the bean class declaration.

```
import org.springframework.stereotype.Component;
@Component
public class Rocket { }
```

@component :
This annotation is used to create a bean object
The annotation can be declared above the class
This annotation invokes default constructors.

FRAMEWORK

@value

This annotation is used to initialize property of the component of type String or Number.

This annotation can be used in three places

```
@Value(value = "100")
public void setThrust(int thrust)
 {
      System.out.println("calling...... "+thrust);
      this.thrust = thrust;
 }
```

The value can be used in three places,

1. Above set method
2. Above value
3. Above constructor

```
@Value(value = "100")
Private int thrust;
```

Constructor initialization:

```
@Autowired
public Fuel(@Value("65") double cost) {
      // TODO Auto-generated constructor stub
      System.out.println(this.getClass().getSimpleName()+"
created");
      System.out.println("cost : "+cost);
      this.cost=cost;
}
```

@Autowired

- Autowired is a process of injecting an object automatically by container.
- Autowired can also be referred as dependency lookup.
- Autowired can be used for both constructor and setter injection.
- This annotation can be used in three places
    1. Above constructor
    2. Above set method
    3. Above property

Example:

```
package com.hdb.spring.bean;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
```

FRAMEWORK

```
import org.springframework.stereotype.Component;
@Component
public class Rocket {

	private int thrust;
	@Autowired
	private Fuel fuel;
	@Autowired
	private astronaut ast;

	public Rocket() {
		// TODO Auto-generated constructor stub
		System.out.println(this.getClass().getSimpleName()+"
created");
	}

	public void launch(Date date) {
		if(date!=null&&fuel!=null) {
			System.out.println("launched "+date);
			fuel.burn();
			ast.Drive();
		}else {
			System.out.println("cound not lanuch");
		}
	}
	@Value(value = "100")
	public void setThrust(int thrust) {
		System.out.println("calling...... "+thrust);
		this.thrust = thrust;
	}
}
```

- Id can be assigned in the component as follows as

```
Component("rock")
public class Rocket{}
```

- **To change the Scope:**

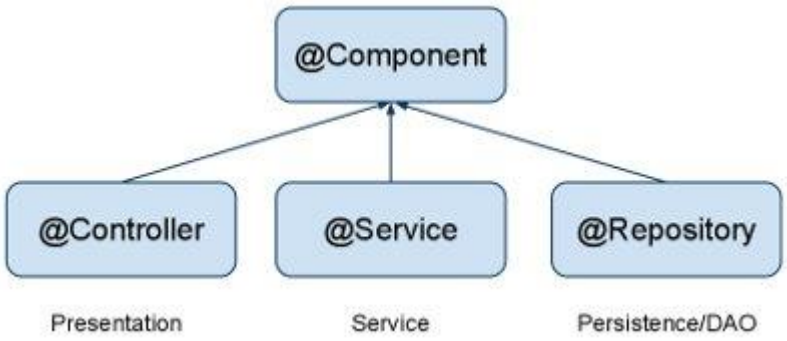
```
@Component("rocket")
@Scope("prototype")
public class Rocket {}
```

Task:

Public abstract class DataAccessObject
{
Public void save();
}
Public class JDBCDAO extends DataAccessObject
{
Private String version;
Public void save(){

```
Sop("jdbc save");
}
}
Public class HibernateDAO extends DataAccesObject
{
Private String version;
Private String author;
Public void save(){
{
Sop("hibernate save");
}
}
Public class Manager
{
Private DataAccesObject dao;
Public void save(){
This.dao.save();
}
```

Annotation	Meaning
@Component	generic stereotype for any Spring-managed component
@Repository	stereotype for persistence layer
@Service	stereotype for service layer
@Controller	stereotype for presentation layer (spring-mvc)



MVC:

- MVC is a design architecture.
- MVC is suitable for developing enterprise web applications or frameworks.
- MVC is the common architecture for developing web application.
- MVC is layer based.
- MVC can be also referred as n-layer architecture.

FRAMEWORK

View	Controller- RPL, NL	Model	
1. Html, 2. CSS- bootstrap 3. JSP-JSTL 4. JS- jQuery, Angular JS 5. JSON	1. Servlet 2. Struts 2 3. JSF 4. Spring MVC	Service	DAO
		Validation	Jdbc
		Java mail	JPA
		Web services	

- View layer should have UI logic.
- Controller will have RPL- request processing logic and NL-navigation logic
- Model is sub layered into service and DAO.
- Service will have business logic and validation logic.
- DAO should have persistence or Database logic.

- View can interact with the client and controller
- Controller can interact with a view and service
- Service can interact with controller and Dao
- Dao can interact with database and service.

Spring and hibernate Integration:

1. Create a java project
2. Create standard package , dao ,dto and util.
3. Add jars files from hibernate, spring ioc and spring orm modules.
4. Copy hibernate.cfg.xml and spring.xml configuration into classpath

Spring-ORM:

- This module is used to integrate spring and ORM implementation.
- This module can be used to integrate with ORM Tools like hibernate, ibatiss etc.
- This module provides implementation for session factory.
- LocalSessionFactoryBean is an implementation of SessionFactory

```
<bean id="factory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryB
ean">
<property name="configLocations"
value="hibernate.cfg.xml"></property>
</bean>
```

- *configLocations is used to pass name and location of hibernate configurations file.*

```
@Component
public class MedicalStoreDAO {

    @Autowired
    SessionFactory factory;
```

FRAMEWORK

- once factory is created by spring container, we need to inject into DAO using @Autowired
- spring framework avoids developer from writing singleton design pattern.
- ----------
- ```
  package com.hdb.integration.dao;
  
  import java.io.Serializable;
  
  import org.hibernate.Session;
  import org.hibernate.SessionFactory;
  import org.hibernate.Transaction;
  import org.springframework.beans.factory.annotation.Autowired;
  import org.springframework.stereotype.Component;
  
  import com.hdb.integration.dto.MedicalStoreDTO;
  @Component
  public class MedicalStoreDAO {
  
  	@Autowired
  	SessionFactory factory;
  	public Serializable save(MedicalStoreDTO DTO) {
  
  		Session session=factory.openSession();
  		Transaction transaction=session.beginTransaction();
  		int pk=(int) session.save(DTO);
  		transaction.commit();
  		session.close(); // must use try-catch .....
  		System.out.println(factory.getStatistics());
  		return pk;
  	}
  }
  ```
- ----------
- ```
  package com.hdb.integration.service;
  
  import org.springframework.beans.factory.annotation.Autowired;
  import org.springframework.stereotype.Component;
  
  import com.hdb.integration.dao.MedicalStoreDAO;
  import com.hdb.integration.dto.MedicalStoreDTO;
  @Component
  public class MedicalStoreService {
  	@Autowired
  	private MedicalStoreDAO medicalStoreDAO;
  
  	public MedicalStoreService() {
  
  ```

```
System.out.println(this.getClass().getName()+" created");

}
public void saveMedicalStore(MedicalStoreDTO dto) {
System.out.println("saving from service...");
medicalStoreDAO.save(dto);

}

}
```

```
MedicalStoreDTO dto=new MedicalStoreDTO();
dto.setName("Homeo-world");
dto.setOwnerName("hdb");
dto.setType("HP");
dto.setLicenseNo("LIC123");


ApplicationContext context=new ClassPathXmlApplicationContext("spring.xml");

MedicalStoreService service=context.getBean(MedicalStoreService.class);
service.saveMedicalStore(dto);
```

Spring Jdbc:

- Spring jdbc is a template provided by spring framework to integrate spring with jdbc.
- There are 4 standard approaches in spring jdbc template.
  1. jdbcTemplate
  2. NamedParamterJdbcTemplate
  3. simpleJdbcTemplate
  4. simpleJdbcCell

  DataSource:
- it is a interface present in javax.sql
- datasource provides connection objects to connect to database.
- It allows connection object to connection pool.

  Examples: BasicDataSource, DriverManagerDataSource
- Implementation of DataSource.

FRAMEWORK

By: HDB

FRAMEWORK

By: HDB