- ClassNotFoundException & NoClassDefFoundError
- Javascript diff b/t apply() & call()
- How to display 100 numbers or so without using loops — use recursion
- Display only unique chars in a str — use regex
- Can we implement autoboxing without any support — use reflection

Javascript - diff b/t call() & apply()

call() — function that takes "this" as an argument and required parameters individually

for ex:

```
var fx = function(arg1, arg2) {
        console.log(this, arg1, arg2);
    }

fx.call(null, "first param", "second param")
```

- apply() — same as call() except that it accepts the parameters as an array.

for ex, fx.apply (null, ["first param", "second param"]);

- bind() - does not call the function but returns a copy of a fn

for ex ; fx.bind (null, "first param", "second param").

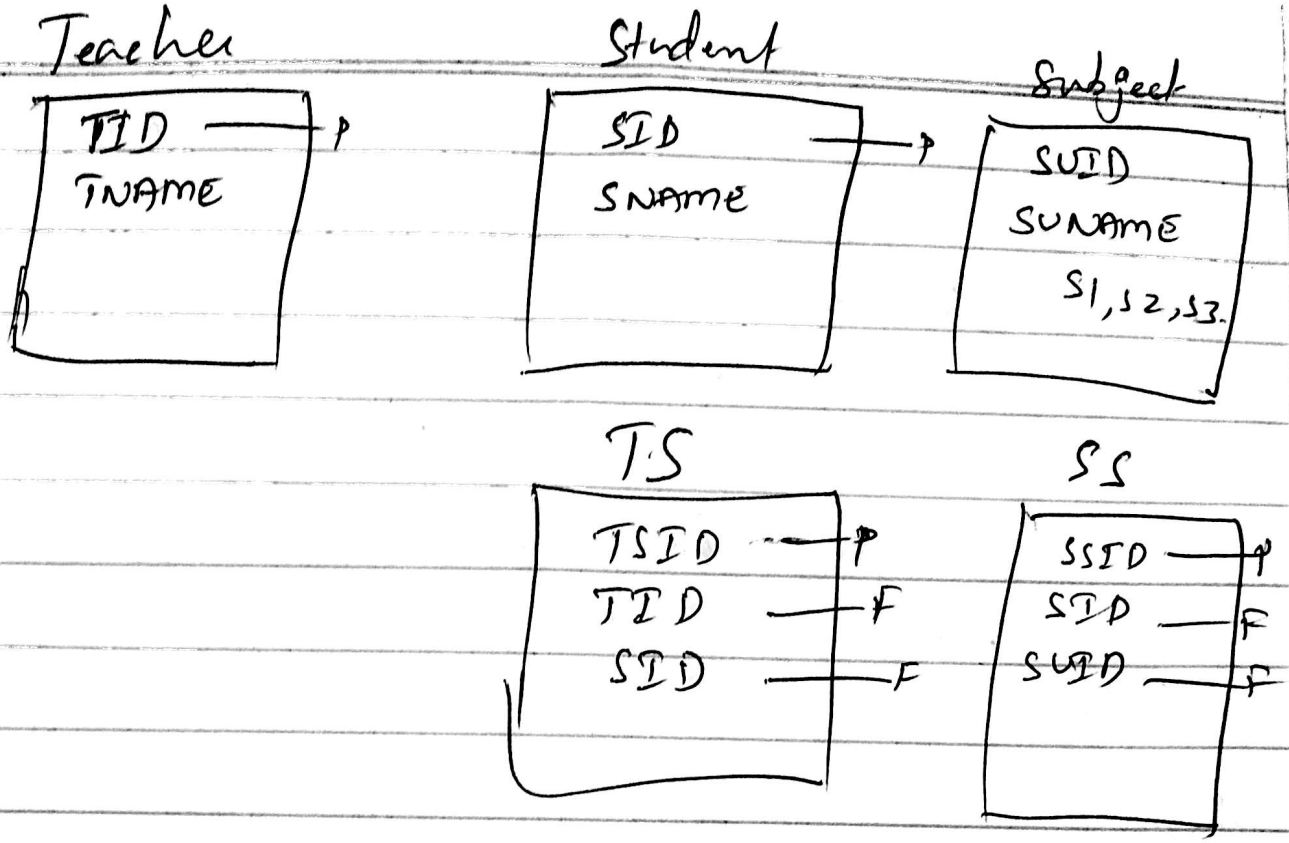
// logs null, "first param", "second param"

- Early & late binding: In an object def^n, data & methods are defined separately, called Encapsulation. These are bind together either lately or early.

```
class A {                          class B {
  public void foo() {                public void foo() {
  }                                  }
}                                  }
```

```
A a = new A()
a.foo(); // early binding. A's foo() is called // compile time
B b = new B()
b.foo(); // " B's " " // compile time
```

```
A ref = null;
ref = b;
ref.foo(); // late binding. B's foo() method is called // Runtime.
```



TS

1, T1, S1
2, T1, S2
3, T2, S1
4, T2, S2

TreeSet – if natural sorting is not-possible (if objects with no compare logic) then only order/ip is preserved.
– if natural sorting is possible (say only strings/numbers present) then they are sorted automatically and listed.

Hibernate

ACID properties — what are the Transaction properties.

Scopes — what are the spring scopes.

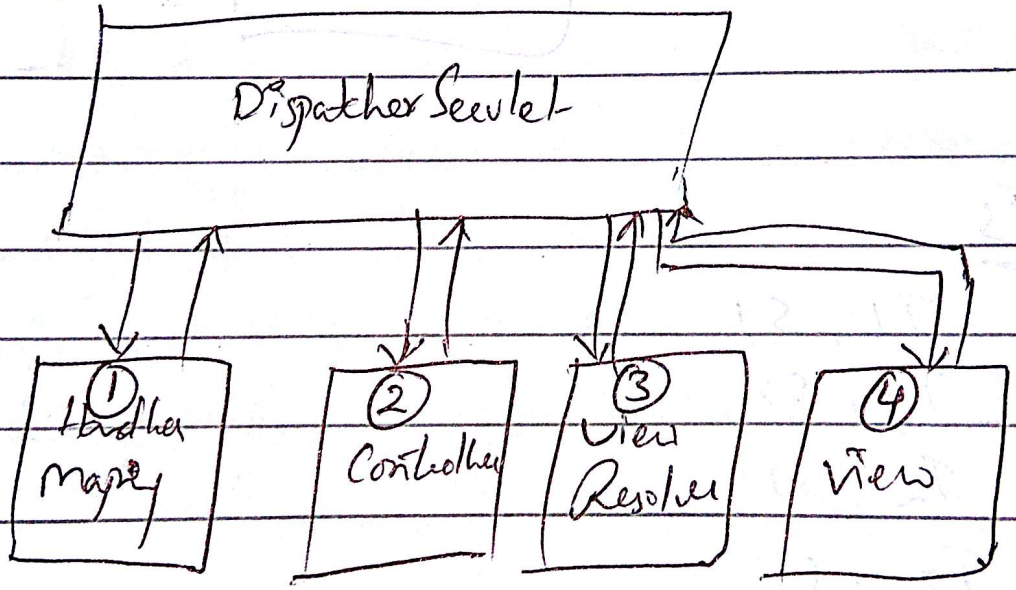
Default Isolation levels.

Diff bet^n Inner join & left outer join.

left outer join — returns all the rows of the left table, regardless of whether there is a row that matches on the right table.

full outer join —

Spring MVC Flow :



1st level & 2nd level cache —

Isolation levels —

Diff b/n load() & get() —

Spring

- if same bean is defined twice in the same bean xml then it throws msg "There are multiple occurrences of ID value 'helloworld'
- if defined in diff xml's, then no error is thrown

Two ways:

① BeanFactory factory = new XmlBeanFactory(
new FileSystemResource("spring.xml")

② ApplicationContext app = new ClassPathXmlApplicationContext(new String[]{"spring.xml", "beans.xml"});

BeanPostProcessor – used to write our own instantiation logic + logic to run after bean instantiation, configuration & initialization by plugging multiple BeanPostProcessor impl's.

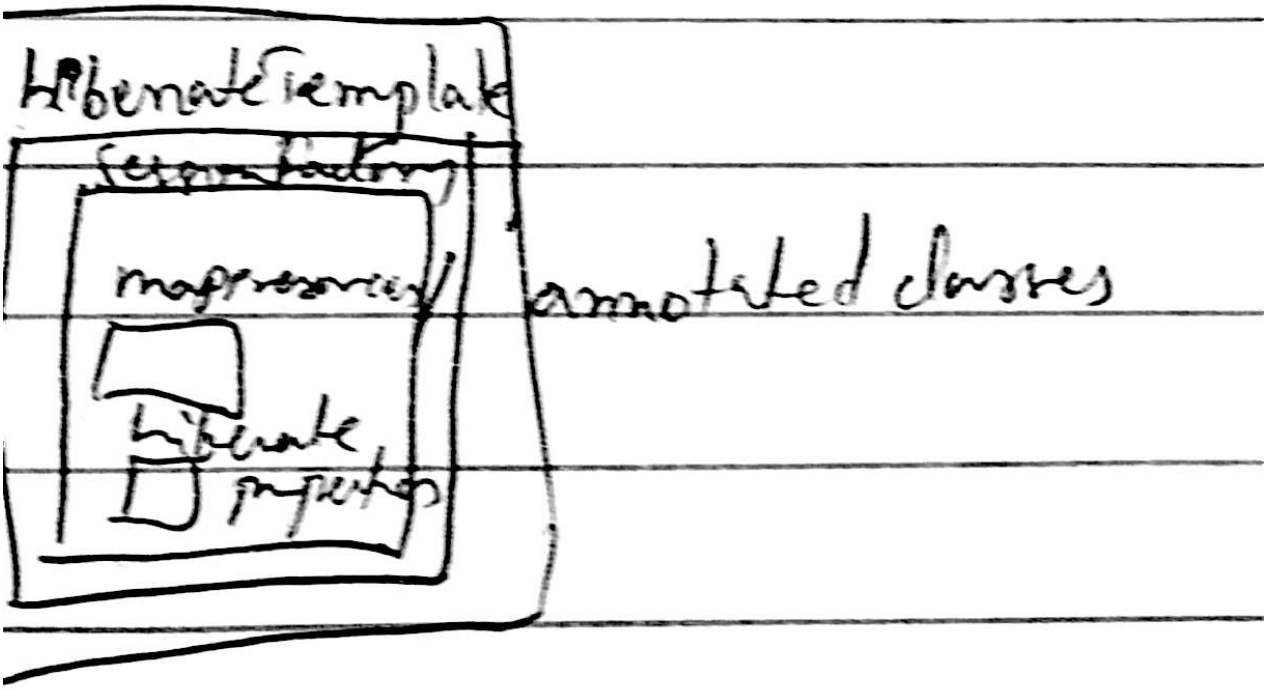
- needs the bean to implement init() & destroy() methods & caller to use registerShutdownHook() method.

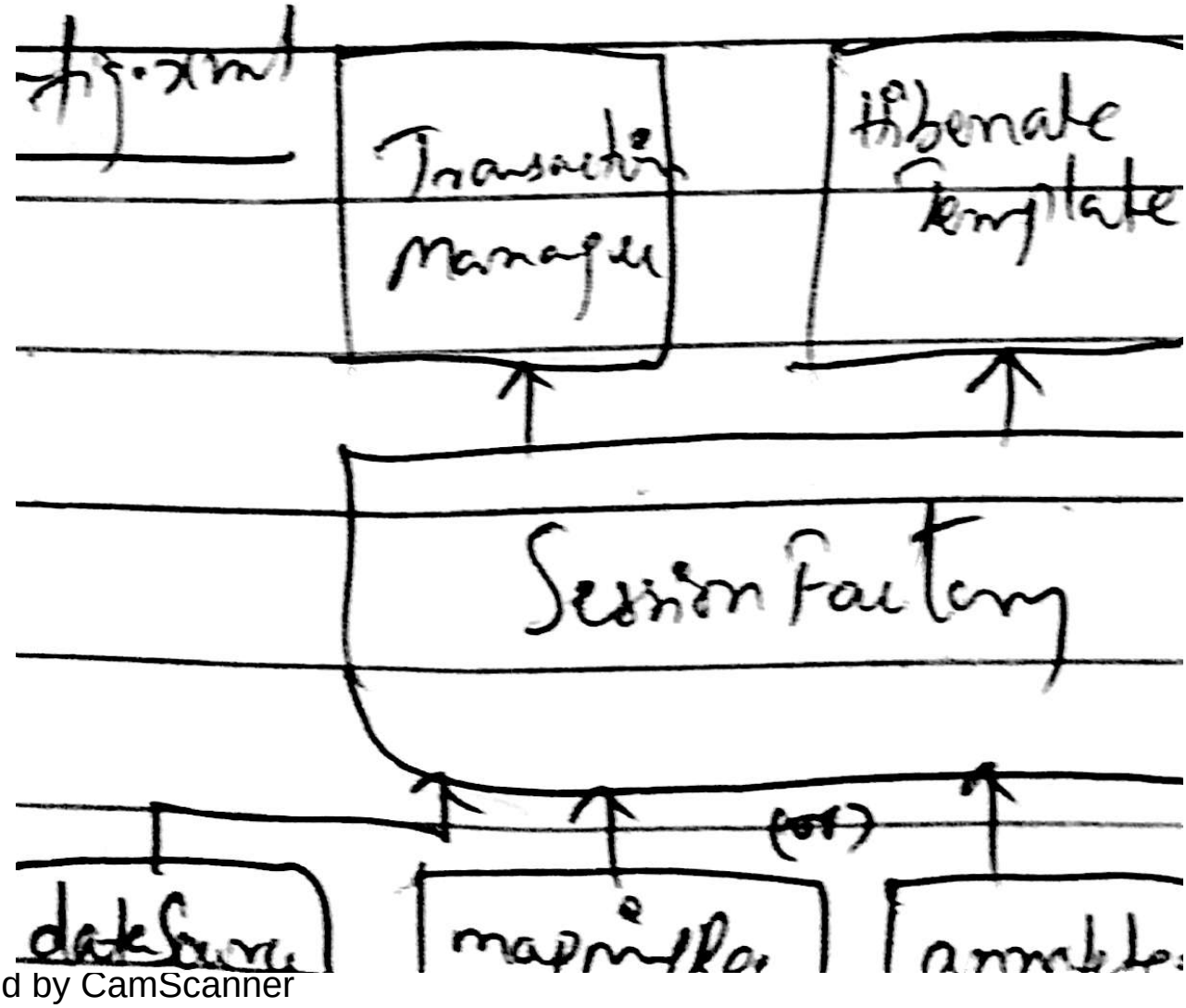
Inner bean injection :- <bean/> element inside <property/> or <constructor-arg/> elements is called inner bean.

- from slide 2-5.

Hibernate Integration

DAO





- ClassNotFoundException – class is not found in the classpath – compile time
- NoClassDefFoundError – class was present during compile-time but not during runtime – runtime.

(4)

Spring config files can be loaded from: (Example for each)
- classpath
- filesystem
- FTP location
- HTTP location

lazy-init = "true" – By default spring impl^n of ApplicationContext does eager instantiations of singleton beans at startup. To make them lazy we use this.

ApplnContext	BeanFactory
1. multiple config files	only 1 config file
2. can publish events to beans registered as listeners	Doesn't support
3. (i18n) messages	"
4. appln life-cycle events & validation	"
5. JNDI, EJB integration, remoting	"

Spring 2.0 –

Spring 2.5 – Annotations, Auto detecting components,
Java 1.4

Annotations on web mvc

@Autowire and @Resource have the same purpose (byType)

@Autowire – injection happens thr' type

@Resource – " " " name

@Controller – to make a custom controller in web mvc.

Spring 3 – Java 5

Diff injections in Spring

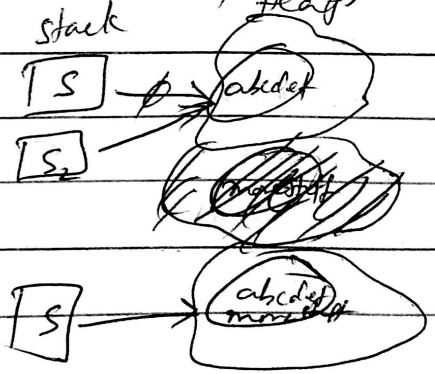
① annotation (*)

@Autowired – can be used on setter / on properties.

String s = "abcdef";

String s2 = s;

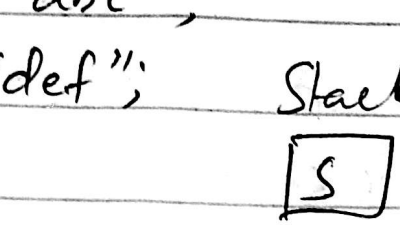
s = s.concat("more stuff");

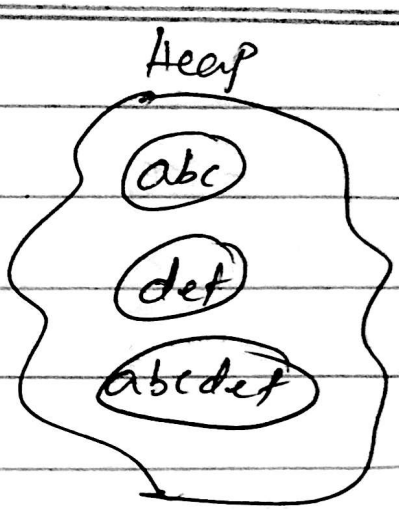


② config. – xml

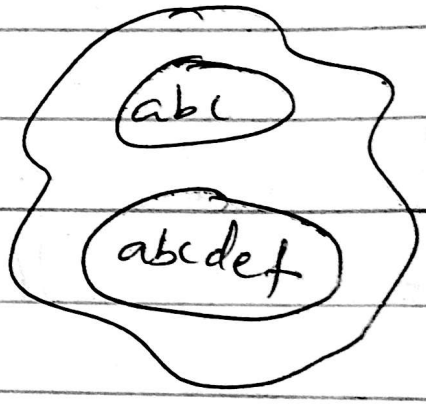
③ Java based

String s = "abc";
s = s + "def";





String s = "abc";
s = s.concat("def");



JSF

JSF 2.0 – annotation based beans.

Scopes – request, session, application, view and custom scope, None scope.

Root class of view – UIViewRoot

JSF Managed beans:

```
@ManagedBean(name = "someBean")
@RequestScoped
public class SomeBean { }
```

- dont use them for Java EE 6 and using CDI.
- provides simple DI mechanism.
- less powerful than CDI.

Context & DI Beans: (Bean Mgmt + DI framework)

- introduced with Java EE 6.
- more flexible than JSF beans.
- can make use of interceptors, conversation scope, Events, type safe injection, decorators, stereotypes and producer methods.

How to do:

1. place beans.xml in META-INF folder. (jar/war)
   then every bean in the package becomes CDI bean.

- Scopes - request, Conversation, session and app^lly.
- To use CDI bean in JSF:

```
@Named("someBean")
@RequestScoped
public class SomeBean {
    @Inject
    private SomeService someService;
}
```

- use @Named annotation (javax.inject.Named)
- to inject a bean into another use @Inject.

CDI allows injecting beans with mismatched scopes thro' proxies. Yes we can inject a request-scoped bean into a session scoped bean

EJB's - there from before CDI
- EJB's are transactional
- Remote and local
- can be async. etc.

When to use what?

Manage Beans — If working in a servlet Container & dont want to try CDI

CDI — if you need advanced fn^ns available

in EJB's like transactional fty.
- we can write our own interceptors to make CDI transactional.

EJBs - provide transactional fns in built.

beans.xml :

CDI needs an beans.xml file to be in
- META-INF of your jar (or)
- classpath (or)
- WEB-INF of your war.

CDI scans for beans.xml file even if it's 0 bytes.

JSF Scopes: (6)

@RequestScoped - bean lives until request-response lives.

@SessionScoped - lives as long as the user session lives

@ApplicationScoped - as " " webappln lives

@ViewScoped - as long as the user is interacting with the same JSF view.

@NoneScoped - as long as a single EL lives. starts when EL evaluation starts & destroyed after EL " .

@CustomScoped - as long as the beans entry in the custom map which is created for this scope lives.

AJAX & JSF:

AJAX is a technique to use HTTPXMLObject & Javascript to send data to server & receive data asynchronously.

f:ajax tag is used.

```
<f:ajax execute = "input-component-name"
        render = "output-component-name" />
```

execute & render support space-separated list of ID's for components that will be updated.

Exception handling in JSF:

```
FacesContext.getCurrentInstance().addMessage(
        null, facesMessage);
FacesMessage facesMessage = new FacesMessage(msg);
```

view

```
<h:messages style = "color:red" />
```

locale:

Locale l = FacesContext.getCurrentInstance().
getViewRoot().getLocale();

ServletContext from FacesContext:

ServletContext sc = (ServletContext) FacesContext
.getCurrentInstance().getExternalContext().
getContext();

JSF form posttype:

JSF uses post-back technique — to submit form data back to the page that contains the form.

- javax.faces.CONFIG_FILES
- what happens when none of the navigation rules matches — same page is rerendered.

faces-config.xml:

```
<faces-config>
    <navigation-rule>
    <from-view-id>...
        <navigation-case>
            <from-outcome>...
            <to-view-id>...
        </navigation-case>
    </navigation-rule>
```

<managed-bean>
<managed-bean-name>...
<managed-bean-class>...
<managed-bean-scope>...
</managed-bean>
</faces-config>

JSF 1.2 & JSF 2.0:

Facelets:

1.2 – JSF tags were written in JSP

2.0 – " - jsf page (facelets)

JSP > servlets > component tree
Facelet > component tree.

AJAX support:

1.2 – needs added libraries.

2.0 – has builtin support – provides f:ajax tag

Groovy support:

2.0 has it 1.2 doesn't

Annotations

2.0 has it. like

@FacesComponent
@FacesRenderer
@ " converter
@FacesValidator
@ " Behavior

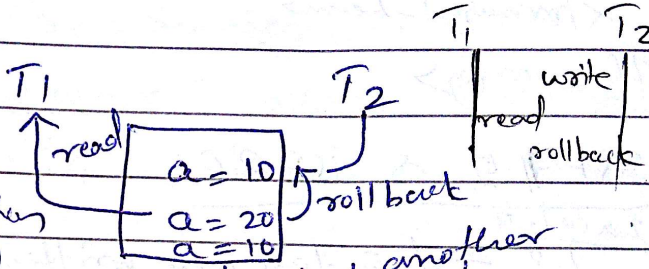
New Scopes:

2.0 — flash and new scopes.

Problems in transactions:

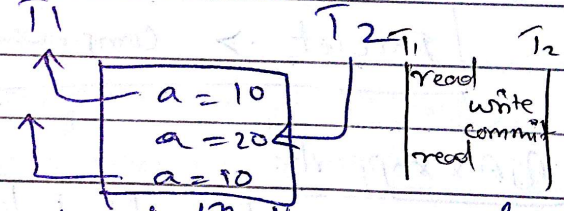
① DIRTY READS:

When data is read by one transaction which may later be rolled back by another transaction.



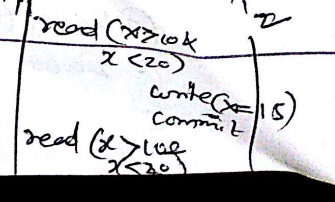
② NON-REPEATABLE READS:

When a transaction reads the same row of data twice, but another transaction changes it bet^n the 2 reads



③ PHANTOM READS:

A transaction searches and gets results after which another transaction adds a new entry satisfying the search condition and for the next search an extra result is displayed.



Isolation levels:

	DR	NRR	PR
① TRANSACTION_READ_UNCOMMITTED	Y	Y	Y
② TRANSACTION_READ_COMMITTED	N	Y	Y
③ TRANSACTION_REPEATABLE_READ	N	N	Y
④ TRANSACTION_SERIALIZABLE	N	N	N

① allows to read uncommitted data
② guarantees that data you are getting is committed
③ " " all reads will be same during the trans
④ all trans" are performed serially.

Transaction attributes:

Required, Requires New, Supported, Not Supported, Mandatory, Never.

Legacy collections VS New collection alternatives

HashTable	=	HashMap
vector	=	ArrayList
Stack	=	LinkedList
properties	=	TreeMap
BitSet	=	Not available.

Diff betn PrintWriter & JspWriter:

JspWriter - some functionality from BufferedWriter and PrintWriter.

- throw IOException from its print methods

PrintWriter - does not throw exceptions

Custom Tag lib Base class:

javax.servlet.jsp.tagext.TagSupport.

JSP :- why there is no main() method

- JSP is actually a class. If we define a class in JSP, it is an inner class. Inner class cannot have static methods. Hence we can't declare main() in JSP.

@CookieParam – to get the parameters from cookie

ex:- @CookieParam(value = "User-Agent")

@FormParam – to get HTML form parameter values

@Produces – to specify the MIME media types a resource can produce and send back.

ex:- @Produces("text/plain")

@Consumes – to specify the MIME types that a resource can consume.

ex:- @Consumes("text/plain")

Javascript:

Event Bubbling —

Dom elements can be nested inside each other. The handlers of the parent works even if you click on its child.

Hibernate

persist()

1. void return type
2. executes inside a transaction scope.

save()

1. return type is Serializable id object
2. may execute outside of a transaction scope

Save()

1. inserts the record & generates & returns a new identifier Serializable obj
2. returns id obj

saveOrUpdate()

1. checks if record already exists if so inserts else updates.
2. void return type.

get()

1. returns a completely initialized object
2. returns null if obj not found.

load()

1. returns a proxy which can be lazily initialized
2. throws ObjectNotFound Exception if obj not found.

Cascade types

CascadeType.PERSIST:

" .MERGE:

" .REMOVE:

" .REFRESH:

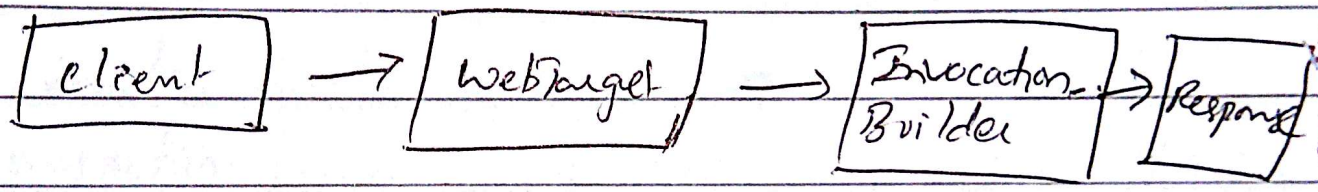
" .DETACH:

" .ALL:

@BeanParam – can be used to hold any/all other parameters as a single entity. It is used to aggregate many request parameters into a single bean.

@HeaderParam – to get info from the HTTP headers.

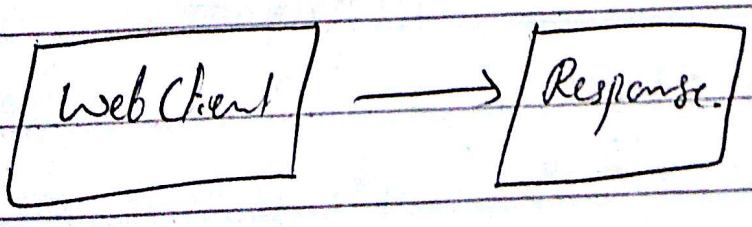
JAX-RS client (Pure JAX-RS i.e. javax.ws.rs.client)



Ex:-

```
Client client = clientBuilder.newClient();
WebTarget wt = client.target("http://abc.com/hello");
InvocationBuilder invBuilder = wt.request("text/plain");
Response res = invBuilder.get();
Strip msg = res.readEntity(Strip.class);
```

Apache CXF client:-

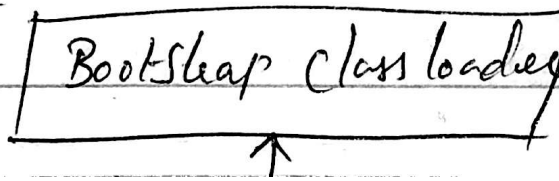


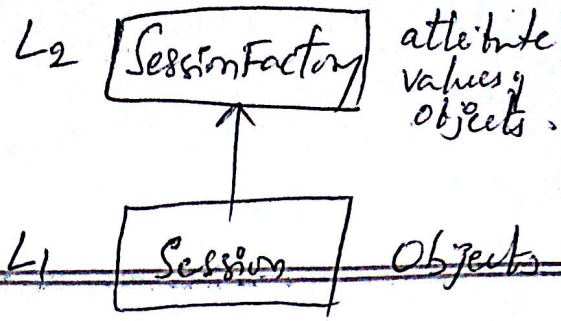
```
WebClient client = WebClient.create("http://abc.com/
Hello");
Book book = client.path("book/"+id).accept("text/xml")
            .get(Book.class);
```

path → get a specific resource

accept → mime type.

Classloaders:





Hibernate Cache levels:

Has 2 levels of caching.

(1) first level – L1 cache – is the session cache. Objects are cached within the current session and they live as long as session lives. – enabled by default

(2) second level – L2 cache – exists as long as the session factory is alive. In hibernate, in second level, ~~objects~~ objects are not cached, instead it stores attribute values.

ex:-

L2 Cache:

[

id = 1, {name = 'Spain', population = 1000; ...}

id = 2, {name = 'Germany', population = 2000, ...}

]

Query Cache: – stores combination of query and parameters as key and list of identifiers of obj's retrieved as values.

key

[{from Country where population >: number, 1000},

{id: 2}]

Value

To configure Second level cache,

In hibernate.properties;

hibernate.cache.provider_class = org.hibernate.cache.
EhCacheProvider.

hibernate.cache.use_structured_entries = true

" . " . use_second_level_cache = true

using annotations,

@Cacheable

@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_
READ_WRITE)

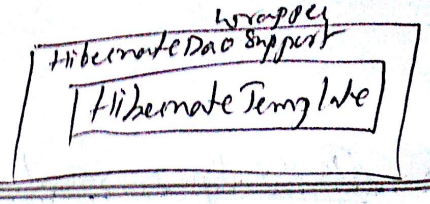
Diff b/n getCurrentSession() & getSession():

① getHibernateTemplate().getSessionFactory().getCurrentSession()

② getSession() method of HibernateDaoSupport

- Each subsequent call to getCurrentSession() will return the same session object that is in ThreadLocal
- getSession() always creates a new Session for each call.

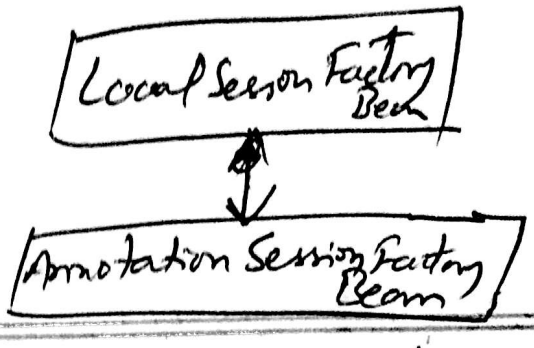
NOTE:- if you are using hibernate 3.1 or higher, you should not use the HibernateTemplate (or) HibernateDaoSupport. You just get coupling of your code to spring, which spring does not want you to do.



HibernateTemplate - helps in accessing the db via hibernate
- maps hibernate exceptions to DataAccess Exceptions (checked) (unchecked)
- takes care of obtaining & releasing Sessions.
- SessionFactory is injected via a LocalSessionFactory Bean.
- provides methods such as find, saveOrUpdate, persist, delete etc. that corresponds to hibernate API but manages sessions & exceptions.

HibernateDaoSupport — convenient class for hibernate based db access.
- wrapper over HibernateTemplate.
- can be initialized using a SessionFactory.
- creates HibernateTemplate. Subclasses can use getHibernateTemplate() to use it.
- can create a DAO and extend this class.

LocalSessionFactoryBean — factory bean that creates hibernate's SessionFactory.
- main purpose is to setup hibernate SessionFactory in a spring context



AnnotationSessionFactory - subclass of LocalSessionFactoryBean but supports annotation based mappings.

Hibernate 3.x :-

Dont use HibernateTemplate / HibernateDaoSupport bcos both couples your code tightly to spring. Just inject sessionFactory.

- HibernateTemplate provides single line to run a query + no need to catch a checked exception + begin / commit / rollback of trans^{n}
- All these are automatically provided in hibernate 3.x by default.

evict() → method removes the cached object from L1 cache (session)

clear() → " " all the " " "

Syntax:- session.evict(department);
session.clear();

Java 8

Lambda expns : (λ)

functional interface – any interface with only 1 abstract method.

Let say person is a functional interface

```
interface Person {
    boolean test(Person p);
}
```

Using anonymous class:

```
printPersons(roster, new Person() {
        public boolean test(Person p) {
            return p.getGender == Person.Sex.MALE
                && p.getAge() >= 18;
        }
});
```

Using Lambda expn.:

```
printPersons(roster, (Person p) ->
        p.getGender() == Person.Sex.MALE
        && p.getAge() >= 18
);
```

Lambda exprs - treat functionality as method argument, or code as data.

- can omit the datatype of the parameters.

Streams — are computations sequence of steps (monads)
— is a feature of functional programing

ex:- List<Stip> myList =
Arrays.asList("a1", "a2", "b1", "c2", "c1");

myList.stream()
.filter(s -> s.startsWith("c"))
.map(Stip::toUppercase)
.sorted()
.forEach(System.out::println);
class1 method

output // C1
// C2

.stream() - works on single thread
.parallelStream() - " parallel " .

New Date/time API's — completely rewritten API from scratch.
— earlier was using JODA API

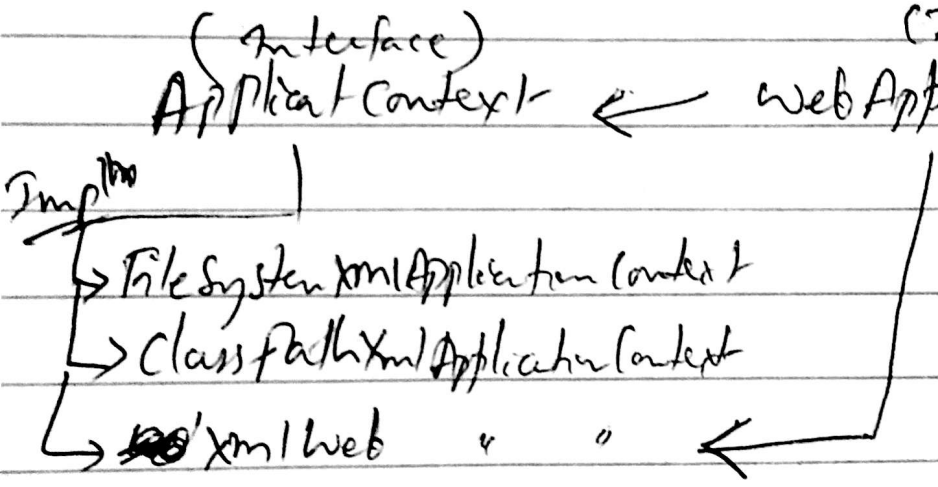
WebApplicationContext:
is used in webapplications
This has more features than ApplicationContext.

In webapps, appln contexts are provided at 2 levels.

(1) Root appln context — ContextLoaderListener

(2) appln context specific to each servlet defined in web.xml — DispatcherServlet.

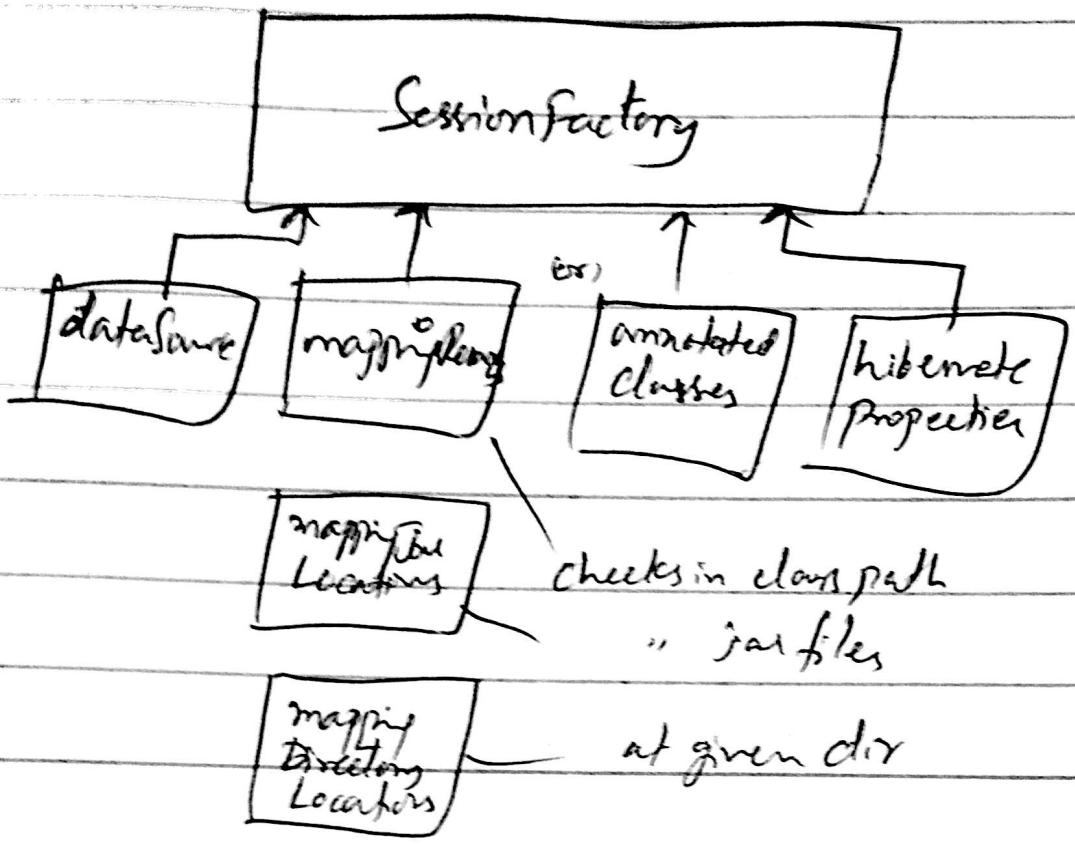
- There is a single root context per appln
- each servlet including dispatcher servlet has its own child context.

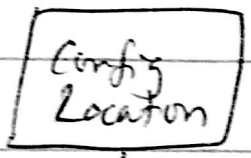


@Transactional — to get benefits of Spring transaction mgt.
— can be used both at method/class level.

Session class of hibernate

- save() or persist() — INSERT
- delete() — DELETE
- update() or merge() — UPDATE
- saveOrUpdate() & replicate() — INSERT/UPDATE

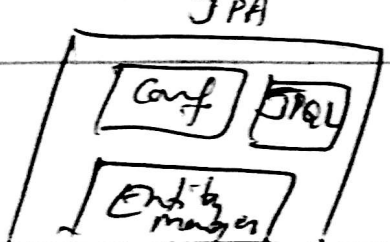




mapping + properties (old fashion)

JPA - Specification

- allow, an object's object-relational mapping to be defined thr' XML or standard annotations
- also defines a runtime EntityManager API for processing queries & transactions.
- provides a query lang JPQL, to query the db.



Spring MVC

@Controller – is used to specify the controller.
– controller takes the request, calls service methods, sets data model & returns view to the DispatcherServlet.

@RequestMapping – to map a URL to a class/method

@ResponseBody – return value to HTTP response

@RequestBody – HTTP request to parameter type

@RequestMapping with method –

@RequestMapping(value = "/method")
@" (value = {"/method1", "method1/sec"})
@" (value = "method2", method = RequestMethod.POST)
@" (value = " ", headers = "name=abc")
@" (" ", produces = {"application/json", "application/xml"}, consumes = "text/xml")

@PathVariable – to get values in URL / dynamic URL parameters.

@RequestMapping(value = "/method7/{id}")
public String method7(@PathVariable("id") int id) {
return "method 7" + id;
}

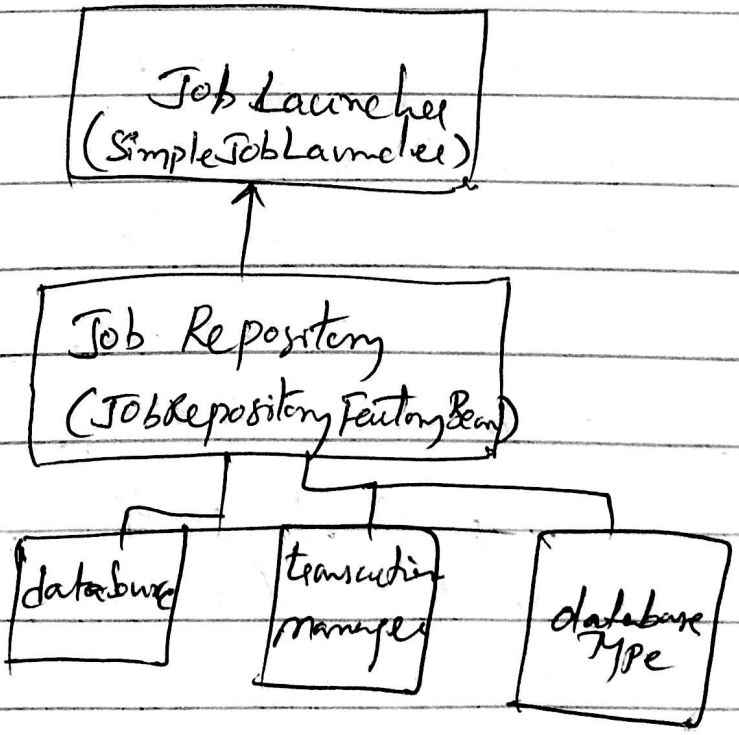
@RequestParam – to get parameters from the request URL.

```
@RequestMapping(value = "/method9")
public String method9(@RequestParam("id") int
                      id) {
    return ...;
}
```

@RequestMapping() – default method – when mapping on class is called.

@ " ("*") – fallback " – if no handler matches

Spring Batch



Chunk

Commit Interval – batch size. No of items that are committed at once.

Skip Limit – no of error records to ignore. If it is set to 2 then 2 error records are allowed but fails on 3rd error.

Job – collection of steps

Step – can be a READ-PROCESS-WRITE or a style tasklet

JobInstance – represents a uniquely identifiable job.
– can be run multiple times in case of failures
– lifecycle ends with 1st successful exec

JobExecution – each run of a JobInstance is a JobExecution

StepExecution – represents each execution of a step. unlike JobExecution, this has additional parameters like commit count etc.

ExecutionContext – is a map of data that batch components may interact during job run, add data to map / read data from map.
– there is one ExecutionContext for JobExecution

and another for StepExecution. i.e, one map for job and one map for each step

Spring creates these maps & adds to ApplicationContext

How to read: (use Spring EL)

① To get property from job parameters:

#{jobParameters[xy]}

② To get property from job's execution context:

#{jobExecutionContext[xy]}

③ To get property from step's execution context:

#{stepExecutionContext[xy]}

@StepScope – new step level scope in Spring Batch. (for parallel processing)

Partitioning :- create a thread pool of x threads and complete the job.

Ex:- Single Thread – processes records 1 to 100 in 10 minutes

Thread 1 – " " 1 to 10 } 1 minute.

:

Thread 10 " " 91 to 100

JobLauncher – are responsible for starting the jobs with the given job parameters

@Autowired and @Inject

1. matches by Type
2. " Qualifiers
3. " Name

@Resource

1. Matches by Name
2. " Type
3. " Qualifiers (ignored if match found by name)

JobRepository : provides CRUD operations for JobLauncher, Job & Step impl^n's.

- JobExecution is obtained from repository
- " & StepExecution are persisted by passing them to repository.

ItemReader — an abstraction representing input for a Step

ItemWriter — " " output for a step.

ItemProcessor — " " business processing an item.

Two phase Locking (2PL):

It is a system in which there are 2 phases, the growing phase where the locks are acquired only and not released and a shrinking phase where the locks are released only and not acquired.

Two phase Commit (2PC):

- It is an atomic commit protocol for distributed systems. This has 2 phases:

1. commit-request phase (or voting phase) - the transaction manager asks all participants to either commit or abort by voting yes: commit or no: abort.
2. Commit phase - transaction manager decides to either commit/abort depending on the votes.

Spring Trans^n Isolation levels:

```
@Transactional (isolation = Isolation. READ_COMMITTED)
                           . READ_UNCOMMITTED)
                           . REPEATABLE_READ)
                           . SERIALIZABLE )
```

Spring Trans^n propagation levels:

- how the Trans^n should behave in terms of propagation.

(1) @Transactional(Propagation = Propagation.REQUIRED)

- if already opened tran^n is present will be used.
- if not new one is created.
- if nested tran^n, are present, if inner method rollsback, the outer method will fail to ~~rollback~~ complete and will also rollback — virtual tran^n is created for inner.

(2) " " Propagation.REQUIRES_NEW

- a new physical tran^n will be created always.
- inner tran^n may commit/rollback independently of the outer tran^n.
- diff/new physical tran^n each time

(3) " " Propagation.NESTED

- uses same physical tran^n but sets savepoints
- inner tran^n may comit/rollback independently of the outer tran^n. - same as JDBC savepoints

(4) " " MANDATORY

- an open tran^n must be present always. If not an exception is thrown.

(5) .NEVER

- an open tran^n must NOT always be present. If exists an exception is thrown.

(6) · NOT_SUPPORTED

- executes outside of a transⁿ scope. If open transⁿ already present it will be paused.

(7) · SUPPORTED

- will execute inside the scope of transⁿ. if open transⁿ already ~~exists~~ exists. If no open transⁿ exists it executes in a non-transactional way.

Readonly:

@Transactional (readOnly = "true")

- specifies that the transⁿ is only going to read data from a database.
- applies only to propagations that starts a transⁿ. i.e., REQUIRED, REQUIRES_NEW and NESTED.

Time out

@Transactional (timeout = 0)

- maxᵐ time allowed for a transⁿ to run.

Declarative transaction Management:

2 types:

① XML based

② Annotations based.

Annotation based:

activate annotations

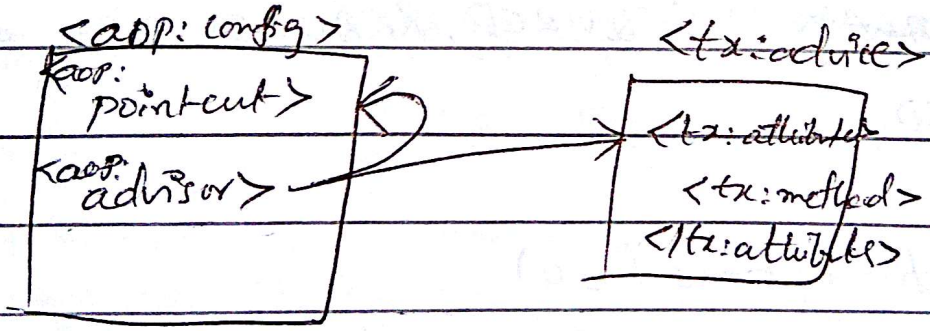
- @Transactional + <context:annotation-config/>
- <tx:annotation-driven/>

XML based:

- <tx:advice/> - advice
- @Transactional(readOnly = "true") will now be written as:

<tx:method name="select*" read-only="true"/>

- <aop:config/> - to apply the advice.



Programmatic Transⁿ management:

① using PlatformTransactionManager.

platformTransactionManager.commit(status)
. rollback(status)

② Using TransactionTemplate

TransactionTemplate transactionTemplate =
new TransactionTemplate(platformTransactionManager);

- TransactionTemplate takes in a platform
- transaction manager as an argument.

Spring Batch tables:

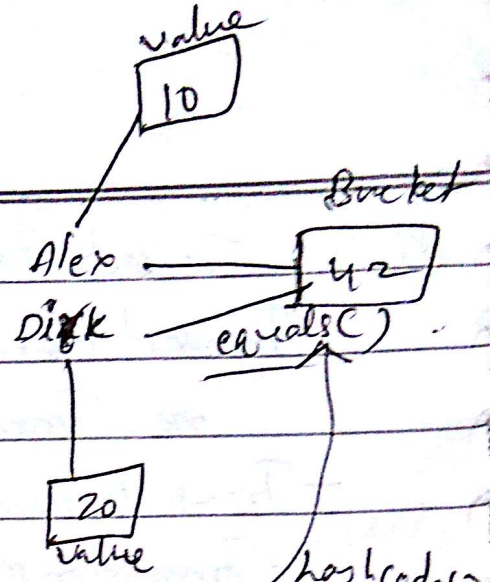
1. BATCH_STEP_EXECUTION — StepExecution
2. " _STEP_EXECUTION_CONTEXT — ExecutionContext
3. " _JOB_EXECUTION — JobExecution
4. " _JOB_ " _CONTEXT_ ExecutionContext
5. " _JOB_INSTANCE — JobInstance
6. * _JOB_EXECUTION_PARAMS. — JobParams.

JobRepository is responsible for storing each Java object into its correct table

→ find the element → find the bucket

equals() & hashCode()

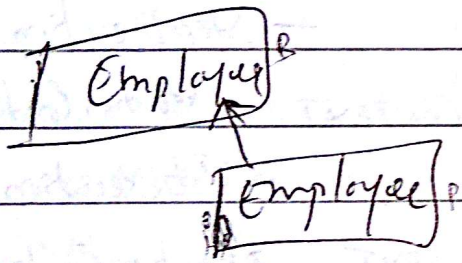
— == ==



map.put("Alex", 10);
map.put("dirk", 20);

map.get("alex");

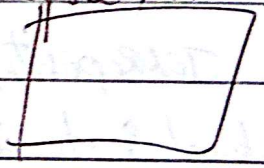
many to one = ignore exception



HQL Criteria - dynamic query Building,

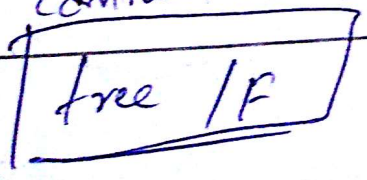
Types of advices:

user registration form.



1. Joins
2. Hiberate.
3. Spring transaction properties, isolation levels.
4. 2PC.
5.

Command to list the files as a tree.



39

39

hashCode()

?

map.put("amy", 10);

map.put("may", 20);

map.get("may");

Imagine there are 2 pigs namely Amy & May

the red angry bird wants to hit the May

(1) Can we write innerclass inside an interface

(2) How to access concrete methods & abstract class using static keyword.

(1) Hashtable — initial capacity = 11
load factor = 0.75

HashMap — initial capacity = 16
load factor = 0.75

(2) ArrayList — IC = 10
- increases by 50%

vector — IC = 10
increases by 100%.