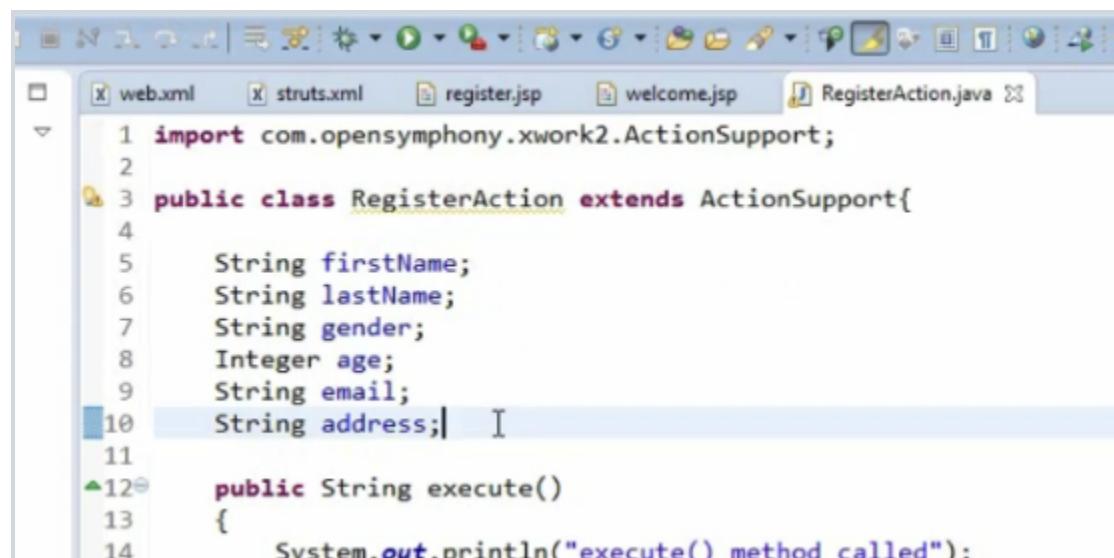


## Struts - 2 Tags:

### 21. TextArea and Reset Tag

```
14
15⑩<s:form action="registerAction">
16    <s:textfield name="firstName" label="First Name" />
17    <s:textfield name="lastName" label="Last Name" />
18    <s:radio name="gender" list="{'Male','Female'}" label="Gender" />
19    <s:textfield name="age" label="Age" />
20    <s:textfield name="email" label="Email" />
21    <s:textarea name="address" cols="30" rows="7" label="Address"/>
22    <s:submit value="Register" />
23 </s:form>
24
-- .
```



```
1 import com.opensymphony.xwork2.ActionSupport;
2
3 public class RegisterAction extends ActionSupport{
4
5     String firstName;
6     String lastName;
7     String gender;
8     Integer age;
9     String email;
10    String address;|  I
11
12⑩    public String execute()
13    {
14        System.out.println("execute() method called");
```

```
□ web.xml struts.xml register.jsp welcome.jsp RegisterAction.java
8 <title>Welcome</title>
9 </head>
10<body>
11 <h2>Welcome</h2>
12 <s:label value="First Name:> />
13 <s:property value="firstName"/><br/>
14
15 <s:label value="Last Name:> />
16 <s:property value="lastName"/><br/>
17
18 <s:label value="Gender:> />
19 <s:property value="gender"/><br/>
20
21 <s:label value="Age:> />
22 <s:property value="age"/>
23
24 <s:label value="Email:> />
25 <s:property value="email"/>
26
27 <s:label value="Address:> />
28 <s:property value="address"/>
29 </body>
30 </html>
```

Registration Form

localhost:8080/Struts2Tags/register.jsp

*First Name:*

*Last Name:*

*Gender:*  Male  Female

*Age:*

*Email:*

*Address:*

### Reset Tag

```
20    <s:textfield name="email" label="Email" />
21    <s:textarea name="address" cols="30" rows="7" label="Address"/>
22    <s:reset value="Reset" />
23    <s:submit value="Register" />
24 </s:form>
```

Registration Form

First Name:

Last Name:

Gender:  Male  Female

Age:

Email:

Address:

### 23. Select Tag: Part 2

```
15<s:form action="registerAction">
16    <s:textfield name="firstName" label="First Name" />
17    <s:textfield name="lastName" label="Last Name" />
18    <s:radio name="gender" list="{'Male','Female'}" label="Gender" />
19    <s:textfield name="age" label="Age" />
20    <s:textfield name="email" label="Email" />
21    <s:textarea name="address" cols="30" rows="7" label="Address"/>
22    <s:select list="{{'Blue','Red','Green','White'}}" name="selectedColor" headerKey="None" headerValue="Select a color" />
23    <s:reset value="Reset" />
24    <s:submit value="Register" />
25 </s:form>
```

```
"7" label="Address"/>
te'"} name="selectedColor" headerKey="None" headerValue="Select a color" label="Favourite Color" />
```

```
10     String address;
11     String selectedColor;
```

Add setters and getters

```
11 <h2>Welcome</h2>
12     <s:label value="First Name:" />
13     <s:property value="firstName"/><br/>
14
15     <s:label value="Last Name:"/>
16     <s:property value="lastName"/><br/>
17
18     <s:label value="Gender:"/>
19     <s:property value="gender"/><br/>
20
21     <s:label value="Age:"/>
22     <s:property value="age"/><br/>
23
24     <s:label value="Email:"/>
25     <s:property value="email"/><br/>
26
27     <s:label value="Address:"/>
28     <s:property value="address"/><br/>
29
30     <s:label value="Favourite Color:"/>
31     <s:property value="selectedColor"/>
32 </body>
33 </html>
```

Registration Form

localhost:8080/Struts2Tags/register.jsp

## Registration Form

First Name:

Last Name:

Gender:  Male  Female

Age:

Email:

Address:

Favourite Color:  Select a color ▾

Welcome

localhost:8080/Struts2Tags/registerAction.action

## Welcome

First Name:

Last Name:

Gender:

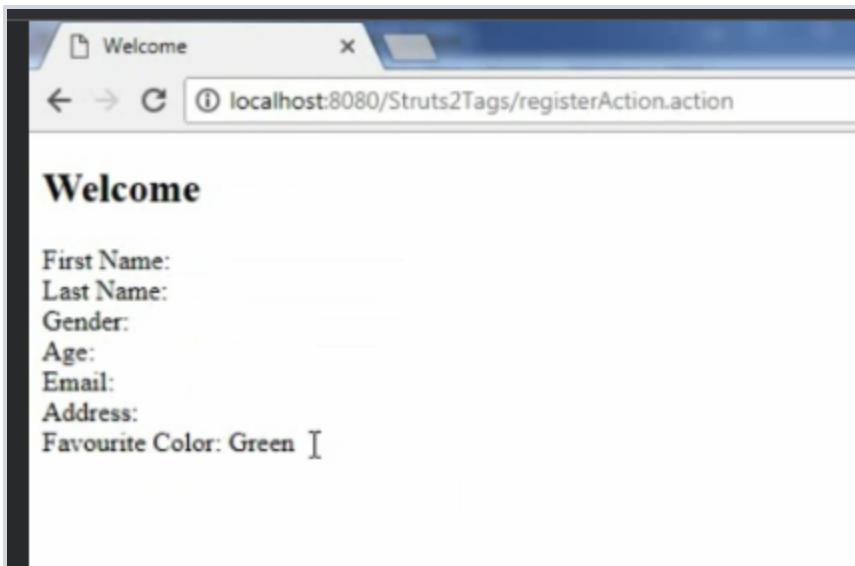
Age:

Email:

Address:

Favourite Color: None

When we wont select then we will see None.



When we select some thing we will see that value.

```
<s:textarea name="address" cols="30" rows="7" label="Address"/>
<s:select multiple="true" list="{ 'Blue', 'Red', 'Green', 'White' }" name="selectedColor" headerKey="None" headerValue=
<s:reset value="Reset" />
<s:submit value="Register" />
```

Select multiple tag above

Kafka Portal - Login    Folios    gpo    spring    GAP Utilities    stockoverfl

Registration Form    X

localhost:8080/Struts2Tags/register.jsp

## Registration Form

First Name:

Last Name:

Gender:  Male  Female

Age:

Email:

Address:

Favourite Color: Select a color

- Blue
- Red
- Green

Welcome    X

localhost:8080/Struts2Tags/registerAction.action

## Welcome

First Name:

Last Name:

Gender:

Age:

Email:

Address:

Favourite Color: Blue, Green

Otherway of selecting list

A screenshot of a Java IDE interface. The menu bar includes 'File', 'Search', 'Project', 'Run', 'Window', and 'Help'. The toolbar contains various icons for file operations. The tabs at the top show 'web.xml', 'struts.xml', 'register.jsp', 'welcome.jsp', and 'RegisterAction.java'. The code editor displays the following Java code:

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 import com.opensymphony.xwork2.ActionSupport;
5
6 public class RegisterAction extends ActionSupport{
7
8     String firstName;
9     String lastName;
10    String gender;
11    Integer age;
12    String email;
13    String address;
14    String selectedColor;
15    List<String> colors;
16
17    public void initializeColors()
18    {
19        colors = new ArrayList<String>();
20        colors.add("Red");
21        colors.add("Blue");
22        colors.add("Green");
23        colors.add("White");
24    }
}
```

The code editor shows the 'RegisterAction.java' file with the cursor positioned inside the 'initializeFormFields()' method. The code is as follows:

```
    List<String> colors;
}
public String initializeFormFields()
{
    initializeColors(); I
    return "input";
}

public void initializeColors()
{
    colors = new ArrayList<String>();
    colors.add("Red");
    colors.add("Blue");
    ...
}
```

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
4   "http://struts.apache.org/dtds/struts-2.5.dtd">
5
6<struts>
7  <package name="register" extends="struts-default">
8    <action name="registerAction" class="RegisterAction">
9      <result name="success"/>/welcome.jsp</result>
10     <result name="input"/>/register.jsp</result>
11   </action>
12  <action name="formLoadAction" method="initializeFormFields" class="RegisterAction">
13    <result name="input"/>/register.jsp</result>
14  </action>
15 </package>
16 </struts>

```

Add action tag. Inorder call custom method apart from execute method use method attribute.

Replace array in jsp with variable name colors

```

19  <s:textfield name="age" label="Age" />
20  <s:textfield name="email" label="Email" />
21  <s:textarea name="address" cols="30" rows="7" label="Address"/>
22  <s:select list="colors" name="selectedColor" headerKey="None" headerValue="Select a color" />
23  <s:reset value="Reset" />
24  <s:submit value="Register" />
25 </s:form>

```

24. Checkbox Tag

```

21  <s:textarea name="address" cols="30" rows="7" label="Address"/>
22  <s:select list="colors" name="selectedColor" headerKey="None" headerValue="Select a color" />
23  <s:checkbox name="subscription" value="true" label="Subscribe to our newsletter"/>
24  <s:reset value="Reset" />
25  <s:submit value="Register" />

```

23rd line

```

16  List<String> colors;
17  Boolean subscription;
18

```

Add setters and getters

```
18     List<String> hobbies;
19     String selectedHobbies;
20
21     public String initializeFormFields()
22     {
23         initializeColors();
24         initializeHobbies();    I
25         return "input";
26     }
27
```

Introduce hobbies and selected hobbies and add setters and getters and create method initializeHobbies in initializeFormFields Method.

```
35
36     public void initializeHobbies()
37     {
38         hobbies = new ArrayList<String>();
39         hobbies.add("Drawing");
40         hobbies.add("Teaching");
41         hobbies.add("Singing");
42         hobbies.add("Programming");
43     } I
44
```

The screenshot shows a Java IDE interface with two tabs open: 'RegisterAction.java' and 'register.jsp'. The 'RegisterAction.java' tab contains Java code for initializing hobbies and selected hobbies. The 'register.jsp' tab contains JSP code for displaying form fields. The cursor is positioned at the end of the 'selectedHobbies' property in the JSP code.

```
13     <s:property value="firstName"/><br/>
14
15     <s:label value="Last Name:"/>
16     <s:property value="lastName"/><br/>
17
18     <s:label value="Gender:"/>
19     <s:property value="gender"/><br/>
20
21     <s:label value="Age:"/>
22     <s:property value="age"/><br/>
23
24     <s:label value="Email:"/>
25     <s:property value="email"/><br/>
26
27     <s:label value="Address:"/>
28     <s:property value="address"/><br/>
29
30     <s:label value="Favourite Color:"/>
31     <s:property value="selectedColor"/><br/>
32
33     <s:label value="Hobbies:"/>
34     <s:property value="selectedHobbies"/> I
35 </body>
36 </html>
```

Registration Form

First Name:

Last Name:

Gender:  Male  Female

Age:

Email:

Address:

Favourite Color:

Subscribe to our newsletter

Hobbies:  Drawing  Teaching  Singing  Programming

Welcome

First Name:  
Last Name:  
Gender:  
Age:  
Email:  
Address:  
Favourite Color: None  
Hobbies: Teaching, Programming

## 26. Iterator Tag

```
Java EE - Struts2Tags/src/Product.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Navigator Properties Run Test Debug Preferences
Project RegistrationApplication Servers Struts2Tags settings build src Product.java
  Product.java RegisterAction.java struts.xml
  WebContent META-INF WEB-INF register.jsp welcome.jsp
  .classpath .project
Struts2Test

web.xml struts.xml register.jsp welcome.jsp RegisterAction.java *Product.java
1
2 public class Product {
3
4     Integer productId;
5     String productName;
6     Integer productPrice;
7
8     public Product(Integer productId, String productName, Integer productPrice) {
9         super();
10        this.productId = productId;
11        this.productName = productName;
12        this.productPrice = productPrice;
13    }
14
15 }
16
```

Add getters and setter methods for the above.

```
9
10    String firstName;
11    String lastName;
12    String gender;
13    Integer age;
14    String email;
15    String address;
16    String selectedColor;
17    List<String> colors;
18    Boolean subscription;
19    List<String> hobbies;
20    String selectedHobbies;
21    List<Product> products;
22
23    public String initializeFormFields()
24    {
25        initialize();
26
27        if (selectedColor == null)
28        {
29            selectedColor = "Red";
30        }
31
32        if (subscription == null)
33        {
34            subscription = false;
35        }
36
37        if (selectedHobbies == null)
38        {
39            selectedHobbies = "Gardening";
40        }
41
42        if (products == null)
43        {
44            products = new ArrayList<Product>();
45        }
46
47        return SUCCESS;
48    }
49
50    private void initialize()
51    {
52        firstName = "John";
53        lastName = "Doe";
54        gender = "Male";
55        age = 30;
56        email = "john.doe@example.com";
57        address = "123 Main Street";
58        selectedColor = "Red";
59        colors = Arrays.asList("Red", "Green", "Blue");
60        subscription = true;
61        hobbies = Arrays.asList("Gardening", "Cooking", "Reading");
62        selectedHobbies = "Gardening";
63        products = new ArrayList<Product>();
64    }
65
66    public void addProduct(Product product)
67    {
68        products.add(product);
69    }
70
71    public void removeProduct(Product product)
72    {
73        products.remove(product);
74    }
75
76    public void clearProducts()
77    {
78        products.clear();
79    }
80
81    public void updateProduct(Product product)
82    {
83        products.set(products.indexOf(product), product);
84    }
85
86    public void setFirstName(String firstName)
87    {
88        this.firstName = firstName;
89    }
90
91    public void setLastName(String lastName)
92    {
93        this.lastName = lastName;
94    }
95
96    public void setGender(String gender)
97    {
98        this.gender = gender;
99    }
100   public void setAge(Integer age)
101   {
102       this.age = age;
103   }
104   public void setEmail(String email)
105   {
106       this.email = email;
107   }
108   public void setAddress(String address)
109   {
110       this.address = address;
111   }
112   public void setSelectedColor(String selectedColor)
113   {
114       this.selectedColor = selectedColor;
115   }
116   public void setColors(List<String> colors)
117   {
118       this.colors = colors;
119   }
120   public void setSubscription(Boolean subscription)
121   {
122       this.subscription = subscription;
123   }
124   public void setHobbies(List<String> hobbies)
125   {
126       this.hobbies = hobbies;
127   }
128   public void setSelectedHobbies(String selectedHobbies)
129   {
130       this.selectedHobbies = selectedHobbies;
131   }
132   public void setProducts(List<Product> products)
133   {
134       this.products = products;
135   }
136
137   public String getFirstName()
138   {
139       return firstName;
140   }
141
142   public String getLastName()
143   {
144       return lastName;
145   }
146
147   public String getGender()
148   {
149       return gender;
150   }
151
152   public Integer getAge()
153   {
154       return age;
155   }
156
157   public String getEmail()
158   {
159       return email;
160   }
161
162   public String getAddress()
163   {
164       return address;
165   }
166
167   public String getSelectedColor()
168   {
169       return selectedColor;
170   }
171
172   public List<String> getColors()
173   {
174       return colors;
175   }
176
177   public Boolean getSubscription()
178   {
179       return subscription;
180   }
181
182   public List<String> getHobbies()
183   {
184       return hobbies;
185   }
186
187   public String getSelectedHobbies()
188   {
189       return selectedHobbies;
190   }
191
192   public List<Product> getProducts()
193   {
194       return products;
195   }
196
197   public void setProduct(Product product)
198   {
199       products.add(product);
200   }
201
202   public void removeProduct(Product product)
203   {
204       products.remove(product);
205   }
206
207   public void clearProducts()
208   {
209       products.clear();
210   }
211
212   public void updateProduct(Product product)
213   {
214       products.set(products.indexOf(product), product);
215   }
216
217   public String getErrorMessage()
218   {
219       return errorMessage;
220   }
221
222   public void setErrorMessage(String errorMessage)
223   {
224       this.errorMessage = errorMessage;
225   }
226
227   public void validate()
228   {
229       if (firstName == null || firstName.trim().isEmpty())
230       {
231           errorMessage = "First name is required";
232       }
233
234       if (lastName == null || lastName.trim().isEmpty())
235       {
236           errorMessage = "Last name is required";
237       }
238
239       if (gender == null || gender.trim().isEmpty())
240       {
241           errorMessage = "Gender is required";
242       }
243
244       if (age == null || age < 18)
245       {
246           errorMessage = "Age must be at least 18";
247       }
248
249       if (email == null || !isValidEmail(email))
250       {
251           errorMessage = "Invalid email address";
252       }
253
254       if (address == null || address.trim().isEmpty())
255       {
256           errorMessage = "Address is required";
257       }
258
259       if (selectedColor == null)
260       {
261           errorMessage = "Color is required";
262       }
263
264       if (colors == null || colors.isEmpty())
265       {
266           errorMessage = "At least one color is required";
267       }
268
269       if (subscription == null)
270       {
271           errorMessage = "Subscription status is required";
272       }
273
274       if (hobbies == null || hobbies.isEmpty())
275       {
276           errorMessage = "At least one hobby is required";
277       }
278
279       if (selectedHobbies == null)
280       {
281           errorMessage = "Selected hobby is required";
282       }
283
284       if (products == null || products.isEmpty())
285       {
286           errorMessage = "At least one product is required";
287       }
288
289       if (errorMessage != null)
290       {
291           return false;
292       }
293
294       return true;
295   }
296
297   private boolean isValidEmail(String email)
298   {
299       try
300       {
301           InternetAddress address = new InternetAddress(email);
302           address.validate();
303           return true;
304       }
305       catch (Exception e)
306       {
307           return false;
308       }
309   }
310
311   public void setErrorMessage(String errorMessage)
312   {
313       this.errorMessage = errorMessage;
314   }
315
316   public String getErrorMessage()
317   {
318       return errorMessage;
319   }
320
321   public void setProduct(Product product)
322   {
323       products.add(product);
324   }
325
326   public void removeProduct(Product product)
327   {
328       products.remove(product);
329   }
330
331   public void clearProducts()
332   {
333       products.clear();
334   }
335
336   public void updateProduct(Product product)
337   {
338       products.set(products.indexOf(product), product);
339   }
340
341   public String getErrorMessage()
342   {
343       return errorMessage;
344   }
345
346   public void setErrorMessage(String errorMessage)
347   {
348       this.errorMessage = errorMessage;
349   }
350
351   public void validate()
352   {
353       if (firstName == null || firstName.trim().isEmpty())
354       {
355           errorMessage = "First name is required";
356       }
357
358       if (lastName == null || lastName.trim().isEmpty())
359       {
360           errorMessage = "Last name is required";
361       }
362
363       if (gender == null || gender.trim().isEmpty())
364       {
365           errorMessage = "Gender is required";
366       }
367
368       if (age == null || age < 18)
369       {
370           errorMessage = "Age must be at least 18";
371       }
372
373       if (email == null || !isValidEmail(email))
374       {
375           errorMessage = "Invalid email address";
376       }
377
378       if (address == null || address.trim().isEmpty())
379       {
380           errorMessage = "Address is required";
381       }
382
383       if (selectedColor == null)
384       {
385           errorMessage = "Color is required";
386       }
387
388       if (colors == null || colors.isEmpty())
389       {
390           errorMessage = "At least one color is required";
391       }
392
393       if (subscription == null)
394       {
395           errorMessage = "Subscription status is required";
396       }
397
398       if (hobbies == null || hobbies.isEmpty())
399       {
400           errorMessage = "At least one hobby is required";
401       }
402
403       if (selectedHobbies == null)
404       {
405           errorMessage = "Selected hobby is required";
406       }
407
408       if (products == null || products.isEmpty())
409       {
410           errorMessage = "At least one product is required";
411       }
412
413       if (errorMessage != null)
414       {
415           return false;
416       }
417
418       return true;
419   }
420
421   private boolean isValidEmail(String email)
422   {
423       try
424       {
425           InternetAddress address = new InternetAddress(email);
426           address.validate();
427           return true;
428       }
429       catch (Exception e)
430       {
431           return false;
432       }
433   }
434
435   public void setProduct(Product product)
436   {
437       products.add(product);
438   }
439
440   public void removeProduct(Product product)
441   {
442       products.remove(product);
443   }
444
445   public void clearProducts()
446   {
447       products.clear();
448   }
449
450   public void updateProduct(Product product)
451   {
452       products.set(products.indexOf(product), product);
453   }
454
455   public String getErrorMessage()
456   {
457       return errorMessage;
458   }
459
460   public void setErrorMessage(String errorMessage)
461   {
462       this.errorMessage = errorMessage;
463   }
464
465   public void validate()
466   {
467       if (firstName == null || firstName.trim().isEmpty())
468       {
469           errorMessage = "First name is required";
470       }
471
472       if (lastName == null || lastName.trim().isEmpty())
473       {
474           errorMessage = "Last name is required";
475       }
476
477       if (gender == null || gender.trim().isEmpty())
478       {
479           errorMessage = "Gender is required";
480       }
481
482       if (age == null || age < 18)
483       {
484           errorMessage = "Age must be at least 18";
485       }
486
487       if (email == null || !isValidEmail(email))
488       {
489           errorMessage = "Invalid email address";
490       }
491
492       if (address == null || address.trim().isEmpty())
493       {
494           errorMessage = "Address is required";
495       }
496
497       if (selectedColor == null)
498       {
499           errorMessage = "Color is required";
500       }
501
502       if (colors == null || colors.isEmpty())
503       {
504           errorMessage = "At least one color is required";
505       }
506
507       if (subscription == null)
508       {
509           errorMessage = "Subscription status is required";
510       }
511
512       if (hobbies == null || hobbies.isEmpty())
513       {
514           errorMessage = "At least one hobby is required";
515       }
516
517       if (selectedHobbies == null)
518       {
519           errorMessage = "Selected hobby is required";
520       }
521
522       if (products == null || products.isEmpty())
523       {
524           errorMessage = "At least one product is required";
525       }
526
527       if (errorMessage != null)
528       {
529           return false;
530       }
531
532       return true;
533   }
534
535   private boolean isValidEmail(String email)
536   {
537       try
538       {
539           InternetAddress address = new InternetAddress(email);
540           address.validate();
541           return true;
542       }
543       catch (Exception e)
544       {
545           return false;
546       }
547   }
548
549   public void setProduct(Product product)
550   {
551       products.add(product);
552   }
553
554   public void removeProduct(Product product)
555   {
556       products.remove(product);
557   }
558
559   public void clearProducts()
560   {
561       products.clear();
562   }
563
564   public void updateProduct(Product product)
565   {
566       products.set(products.indexOf(product), product);
567   }
568
569   public String getErrorMessage()
570   {
571       return errorMessage;
572   }
573
574   public void setErrorMessage(String errorMessage)
575   {
576       this.errorMessage = errorMessage;
577   }
578
579   public void validate()
580   {
581       if (firstName == null || firstName.trim().isEmpty())
582       {
583           errorMessage = "First name is required";
584       }
585
586       if (lastName == null || lastName.trim().isEmpty())
587       {
588           errorMessage = "Last name is required";
589       }
590
591       if (gender == null || gender.trim().isEmpty())
592       {
593           errorMessage = "Gender is required";
594       }
595
596       if (age == null || age < 18)
597       {
598           errorMessage = "Age must be at least 18";
599       }
600
601       if (email == null || !isValidEmail(email))
602       {
603           errorMessage = "Invalid email address";
604       }
605
606       if (address == null || address.trim().isEmpty())
607       {
608           errorMessage = "Address is required";
609       }
610
611       if (selectedColor == null)
612       {
613           errorMessage = "Color is required";
614       }
615
616       if (colors == null || colors.isEmpty())
617       {
618           errorMessage = "At least one color is required";
619       }
620
621       if (subscription == null)
622       {
623           errorMessage = "Subscription status is required";
624       }
625
626       if (hobbies == null || hobbies.isEmpty())
627       {
628           errorMessage = "At least one hobby is required";
629       }
630
631       if (selectedHobbies == null)
632       {
633           errorMessage = "Selected hobby is required";
634       }
635
636       if (products == null || products.isEmpty())
637       {
638           errorMessage = "At least one product is required";
639       }
640
641       if (errorMessage != null)
642       {
643           return false;
644       }
645
646       return true;
647   }
648
649   private boolean isValidEmail(String email)
650   {
651       try
652       {
653           InternetAddress address = new InternetAddress(email);
654           address.validate();
655           return true;
656       }
657       catch (Exception e)
658       {
659           return false;
660       }
661   }
662
663   public void setProduct(Product product)
664   {
665       products.add(product);
666   }
667
668   public void removeProduct(Product product)
669   {
670       products.remove(product);
671   }
672
673   public void clearProducts()
674   {
675       products.clear();
676   }
677
678   public void updateProduct(Product product)
679   {
680       products.set(products.indexOf(product), product);
681   }
682
683   public String getErrorMessage()
684   {
685       return errorMessage;
686   }
687
688   public void setErrorMessage(String errorMessage)
689   {
690       this.errorMessage = errorMessage;
691   }
692
693   public void validate()
694   {
695       if (firstName == null || firstName.trim().isEmpty())
696       {
697           errorMessage = "First name is required";
698       }
699
700       if (lastName == null || lastName.trim().isEmpty())
701       {
702           errorMessage = "Last name is required";
703       }
704
705       if (gender == null || gender.trim().isEmpty())
706       {
707           errorMessage = "Gender is required";
708       }
709
710       if (age == null || age < 18)
711       {
712           errorMessage = "Age must be at least 18";
713       }
714
715       if (email == null || !isValidEmail(email))
716       {
717           errorMessage = "Invalid email address";
718       }
719
720       if (address == null || address.trim().isEmpty())
721       {
722           errorMessage = "Address is required";
723       }
724
725       if (selectedColor == null)
726       {
727           errorMessage = "Color is required";
728       }
729
730       if (colors == null || colors.isEmpty())
731       {
732           errorMessage = "At least one color is required";
733       }
734
735       if (subscription == null)
736       {
737           errorMessage = "Subscription status is required";
738       }
739
740       if (hobbies == null || hobbies.isEmpty())
741       {
742           errorMessage = "At least one hobby is required";
743       }
744
745       if (selectedHobbies == null)
746       {
747           errorMessage = "Selected hobby is required";
748       }
749
750       if (products == null || products.isEmpty())
751       {
752           errorMessage = "At least one product is required";
753       }
754
755       if (errorMessage != null)
756       {
757           return false;
758       }
759
760       return true;
761   }
762
763   private boolean isValidEmail(String email)
764   {
765       try
766       {
767           InternetAddress address = new InternetAddress(email);
768           address.validate();
769           return true;
770       }
771       catch (Exception e)
772       {
773           return false;
774       }
775   }
776
777   public void setProduct(Product product)
778   {
779       products.add(product);
780   }
781
782   public void removeProduct(Product product)
783   {
784       products.remove(product);
785   }
786
787   public void clearProducts()
788   {
789       products.clear();
790   }
791
792   public void updateProduct(Product product)
793   {
794       products.set(products.indexOf(product), product);
795   }
796
797   public String getErrorMessage()
798   {
799       return errorMessage;
800   }
801
802   public void setErrorMessage(String errorMessage)
803   {
804       this.errorMessage = errorMessage;
805   }
806
807   public void validate()
808   {
809       if (firstName == null || firstName.trim().isEmpty())
810       {
811           errorMessage = "First name is required";
812       }
813
814       if (lastName == null || lastName.trim().isEmpty())
815       {
816           errorMessage = "Last name is required";
817       }
818
819       if (gender == null || gender.trim().isEmpty())
820       {
821           errorMessage = "Gender is required";
822       }
823
824       if (age == null || age < 18)
825       {
826           errorMessage = "Age must be at least 18";
827       }
828
829       if (email == null || !isValidEmail(email))
830       {
831           errorMessage = "Invalid email address";
832       }
833
834       if (address == null || address.trim().isEmpty())
835       {
836           errorMessage = "Address is required";
837       }
838
839       if (selectedColor == null)
840       {
841           errorMessage = "Color is required";
842       }
843
844       if (colors == null || colors.isEmpty())
845       {
846           errorMessage = "At least one color is required";
847       }
848
849       if (subscription == null)
850       {
851           errorMessage = "Subscription status is required";
852       }
853
854       if (hobbies == null || hobbies.isEmpty())
855       {
856           errorMessage = "At least one hobby is required";
857       }
858
859       if (selectedHobbies == null)
860       {
861           errorMessage = "Selected hobby is required";
862       }
863
864       if (products == null || products.isEmpty())
865       {
866           errorMessage = "At least one product is required";
867       }
868
869       if (errorMessage != null)
870       {
871           return false;
872       }
873
874       return true;
875   }
876
877   private boolean isValidEmail(String email)
878   {
879       try
880       {
881           InternetAddress address = new InternetAddress(email);
882           address.validate();
883           return true;
884       }
885       catch (Exception e)
886       {
887           return false;
888       }
889   }
890
891   public void setProduct(Product product)
892   {
893       products.add(product);
894   }
895
896   public void removeProduct(Product product)
897   {
898       products.remove(product);
899   }
900
901   public void clearProducts()
902   {
903       products.clear();
904   }
905
906   public void updateProduct(Product product)
907   {
908       products.set(products.indexOf(product), product);
909   }
910
911   public String getErrorMessage()
912   {
913       return errorMessage;
914   }
915
916   public void setErrorMessage(String errorMessage)
917   {
918       this.errorMessage = errorMessage;
919   }
920
921   public void validate()
922   {
923       if (firstName == null || firstName.trim().isEmpty())
924       {
925           errorMessage = "First name is required";
926       }
927
928       if (lastName == null || lastName.trim().isEmpty())
929       {
930           errorMessage = "Last name is required";
931       }
932
933       if (gender == null || gender.trim().isEmpty())
934       {
935           errorMessage = "Gender is required";
936       }
937
938       if (age == null || age < 18)
939       {
940           errorMessage = "Age must be at least 18";
941       }
942
943       if (email == null || !isValidEmail(email))
944       {
945           errorMessage = "Invalid email address";
946       }
947
948       if (address == null || address.trim().isEmpty())
949       {
950           errorMessage = "Address is required";
951       }
952
953       if (selectedColor == null)
954       {
955           errorMessage = "Color is required";
956       }
957
958       if (colors == null || colors.isEmpty())
959       {
960           errorMessage = "At least one color is required";
961       }
962
963       if (subscription == null)
964       {
965           errorMessage = "Subscription status is required";
966       }
967
968       if (hobbies == null || hobbies.isEmpty())
969       {
970           errorMessage = "At least one hobby is required";
971       }
972
973       if (selectedHobbies == null)
974       {
975           errorMessage = "Selected hobby is required";
976       }
977
978       if (products == null || products.isEmpty())
979       {
980           errorMessage = "At least one product is required";
981       }
982
983       if (errorMessage != null)
984       {
985           return false;
986       }
987
988       return true;
989   }
990
991   private boolean isValidEmail(String email)
992   {
993       try
994       {
995           InternetAddress address = new InternetAddress(email);
996           address.validate();
997           return true;
998       }
999       catch (Exception e)
1000      {
1001          return false;
1002      }
1003  }
```

```
46     }
47
48     public void initializeProducts()
49     {
50         products = new ArrayList<Product>();
51         products.add(new Product(111,"Mobile Phone",10000));
52         products.add(new Product(222,"Camera",7000));
53         products.add(new Product(333,"TV",20000));
54         products.add(new Product(444,"Laptop",30000));
55     }
56
57     public String execute()
```

```
22
23@    public String initializeFormFields()
24    {
25        initializeColors();
26        initializeHobbies();
27        initializeProducts();
28        return "input";
29    }
```

```
27 </s:form>
28@   <table border="1" width="300">
29@     <tr>
30@       <th>Product ID</th>
31@       <th>Product Name</th>
32@       <th>Product Price</th>
33@     </tr>
34@     <s:iterator value="products" var="product">
35@       <tr>
36@         <td>
37@           <s:property value="#product.productId"/>
38@         </td>
39@         <td>
40@           <s:property value="#product.productName"/>
41@         </td>
42@         <td>
43@           <s:property value="#product.productPrice"/>
44@         </td>
45@       </tr>
46@     </s:iterator>
47   </table>
```

Registration Form

localhost:8080/Struts2Tags/formLoadAction

**Registration Form**

*First Name:*

*Last Name:*

*Gender:*  Male  Female

*Age:*

*Email:*

*Address:*

*Favourite Color:*

Subscribe to our newsletter

*Hobbies:*  Drawing  Teaching  Singing  Programming

Product ID	Product Name	Product Price
111	Mobile Phone	10000
222	Camera	7000
333	TV	20000
444	Laptop	30000

27. If Else Tag

```
1 web.xml    struts.xml    register.jsp    welcome.jsp    RegisterAction.java    Product.java
22 <s:property value="age"/><br/>
23
24 <s:label value="Email:"/>
25 <s:property value="email"/><br/>
26
27 <s:label value="Address:"/>
28 <s:property value="address"/><br/>
29
30 <s:label value="Favourite Color:"/>
31 <s:property value="selectedColor"/><br/>
32
33 <s:label value="Hobbies:"/>
34 <s:property value="selectedHobbies"/>
35
36<s:if test="%{subscription == true}"> I
37     <div>You are a subscriber</div>
38 </s:if>
39<s:else>
40     <div>You are NOT a subscriber</div>
41 </s:else>
42 </body>
43 </html>
```

Favourite Color:

Subscribe to our newsletter

Hobbies:  Drawing  Teaching  Singing  Program

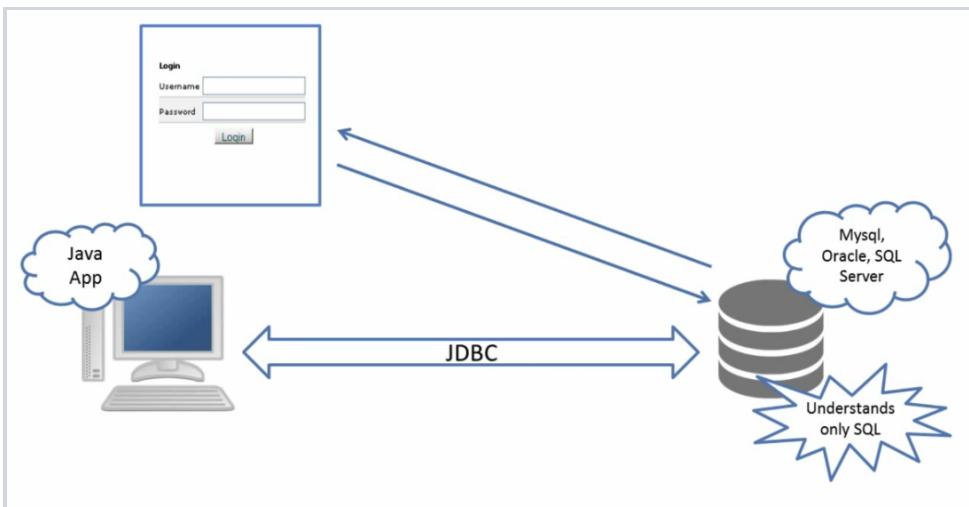
Welcome

First Name:  
Last Name:  
Gender:  
Age:  
Email:  
Address:  
Favourite Color: None  
Hobbies:  
You are a subscriber

When un check

A screenshot of a web browser window. The address bar shows the URL `localhost:8080/Struts2Tags/registerAction.action`. The page content is titled "Welcome" and contains the following text:

First Name:  
Last Name:  
Gender:  
Age:  
Email:  
Address:  
Favourite Color: None  
Hobbies:  
You are NOT a subscriber



## What is JDBC?

- JDBC (Java Database Connectivity) is an API which allows our Java applications to interact with an RDBMS(Relational Database Management System).
- It is independent of any particular database.

## Why do we need a database driver?

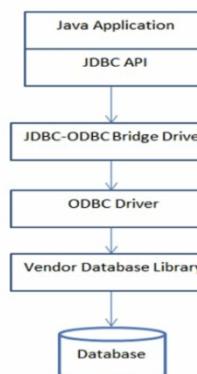
- We can't directly talk to the database server.
- Driver converts JDBC calls to database-specific calls.
- For every database, we need a driver.

## Types of JDBC Drivers

- Type-1: JDBC-ODBC Bridge Driver
- Type-2: Native API Driver
- Type-3: Network Protocol Driver
- Type-4: Thin Driver

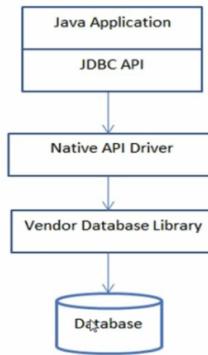
## JDBC-ODBC Bridge Driver

- Converts JDBC calls to ODBC calls.



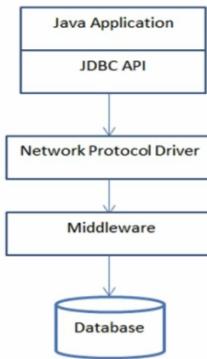
## Native API Driver

- Converts JDBC calls to native calls of the database API.



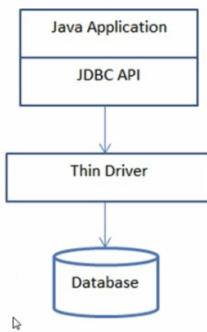
## Network Protocol Driver

- Uses a middleware between the calling program and database.
- Middleware converts JDBC calls to vendor-specific database protocol.



## Thin Driver

- Converts JDBC calls directly to vendor-specific database protocol.



## Commonly used classes/interfaces

- DriverManager
- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet
- ResultSetMetaData
- DatabaseMetaData

Meta data meaning data about data.

### Step 1 : Load and Register the driver

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

### Step 2 : Create connection

```
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

### Step 3 : Create statement

```
Statement stmt=con.createStatement();
```

## Step 4 : Execute query

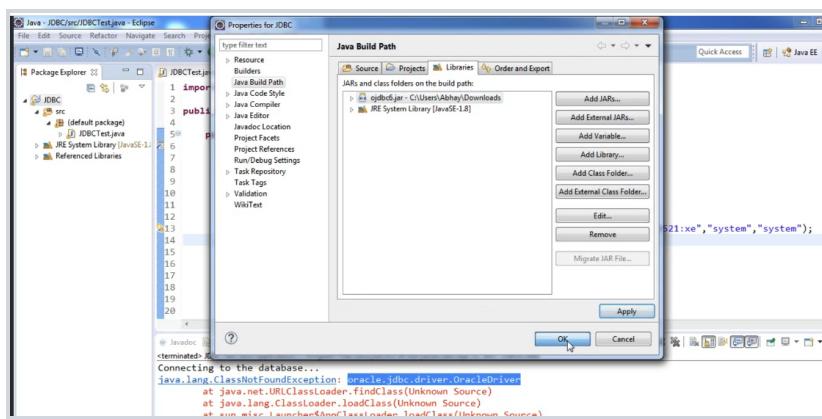
```
ResultSet rs=stmt.executeQuery("select * from emp");
```

## Step 5 : Close connection

```
con.close();
```

## JDBC Connectivity Steps:-

1. Load and Register the driver.
2. Create connection.
3. Create statement.
4. Execute query.
5. Close connection.



The screenshot shows the Oracle Technology Network Applications page. On the left, there's a sidebar with links like AIA Pre-Built Integrations, E-Business Applications Technology, Oracle Policy Automation, Autovue, Construction and Engineering, Oracle Commerce, Utilities, and Communications. The main content area displays the "Oracle Database 11g Release 2 JDBC Drivers" page. It includes a note about accepting the OTN License Agreement, a "Accept License Agreement" button, and a "Decline License Agreement" link. Below this, it lists "Oracle Database 11g Release 2 (11.2.0.4) JDBC Drivers". It shows three files: simplefan.jar (20.365 bytes), simplefan.policy (10.591 bytes), and simplefan.html. There's also a note about classes for RAC events via ONS. To the right, there's a "Popular Downloads" sidebar with links to Berkeley DB, Enterprise Manager, Database EE and XE, Developer VMs, Enterprise Pack for Eclipse, and Java. At the bottom right is an "Oracle Cloud" advertisement.

Jar location in internet.

The screenshot shows the Eclipse IDE interface. The top bar has "Java - JDBC/src/JDBCTest.java - Eclipse" and standard menu items. The left sidebar shows a "Package Explorer" with a project named "JDBC" containing a "src" folder with "JDBCTest.java" and a "JRE System Library [JavaSE-1.8]". The main editor window displays the following Java code:

```

1 import java.sql.*;
2
3 public class JDBCTest {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8         try {
9             System.out.println("Connecting to the database...");
10            Class.forName("oracle.jdbc.driver.OracleDriver");
11            Connection conn=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
12
13            System.out.println("Connection established!!!!");
14        } catch(Exception e) {
15            e.printStackTrace();
16        }
17    }
18 }
19
20

```

The "Console" tab at the bottom shows the output of the program's execution:

```

@ Javadoc Declaration Console
<terminated> JDBCTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (Apr 23, 2017, 2:31:42 AM)
Connecting to the database...
Connection established!!!

```

The screenshot shows a "Run SQL Command Line" window. The title bar says "Run SQL Command Line". The window displays the following SQL\*Plus session:

```

SQL*Plus: Release 11.2.0.2.0 Production on Mon Apr 24 00:22:29 2017
Copyright (c) 1982, 2014, Oracle. All rights reserved.

SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> select * from student;
      ID NAME
----- 
      1 studentA
      2 studentB
      3 studentC
SQL> 

```

The "Console" tab at the bottom shows the output of the session:

```

19
20

```

The screenshot shows the Eclipse IDE interface with the JDBCTest.java file open. The code connects to an Oracle database using JDBC and prints student names from a table named 'student'. The output window shows the results: 'studentA', 'studentB', and 'studentC'.

```
4
5@ public static void main(String[] args) {
6     // TODO Auto-generated method stub
7
8     try
9     {
10         System.out.println("Connecting to the database...");
11
12         Class.forName("oracle.jdbc.driver.OracleDriver");
13         Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
14
15         Statement st = conn.createStatement();
16         ResultSet rs = st.executeQuery("SELECT * FROM student");
17
18         while(rs.next())
19         {
20             System.out.println(rs.getInt("ID")+" --> "+rs.getString("NAME"));
21         }
22     }
23 }
```

Console Output:

```
<terminated> JDBCTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (Apr 24, 2017, 12:27:02 AM)
Connecting to the database...
1 --> studentA
2 --> studentB
3 --> studentC
```

## Create Query

The screenshot shows the Eclipse IDE interface with the CRUDTest.java file open. The code creates a new table named 'employee' with columns 'emp\_id', 'emp\_name', and 'emp\_salary'. The output window shows the message 'Number of rows affected: 1'.

```
1 import java.sql.*;
2
3 public class CRUDTest {
4
5@ public static void main(String[] args) {
6     // TODO Auto-generated method stub
7     try
8     {
9
10        Class.forName("oracle.jdbc.driver.OracleDriver");
11        Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
12
13        Statement st = conn.createStatement();
14        int n = st.executeUpdate("CREATE TABLE employee(emp_id int, emp_name varchar2(30), emp_salary int)");
15        System.out.println("Number of rows affected: "+n);
16
17        conn.close();
18
19    }
20    catch(Exception e)
21    {
22        e.printStackTrace();
23    }
24 }
```

Console Output:

```
<terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (Apr 30, 2017, 12:42:26 AM)
```

The screenshot shows the Oracle SQL\*Plus interface. It first runs a 'select \* from employee' query which returns 'no rows selected'. Then it runs a 'desc employee' command, which displays the table structure with three columns: 'EMP\_ID' (NUMBER(38)), 'EMP\_NAME' (VARCHAR2(30)), and 'EMP\_SALARY' (NUMBER(38)).

```
SQL> select * from employee;
no rows selected
SQL> desc employee;
Name          Null?    Type
-----          ----
EMP_ID          NUMBER(38)
EMP_NAME        VARCHAR2(30)
EMP_SALARY      NUMBER(38)
SQL>
```

## Insert Query

```

1 import java.sql.*;
2
3 public class CRUDTest {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         try {
8             {
9                 Class.forName("oracle.jdbc.driver.OracleDriver");
10                Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
11
12                Statement st = conn.createStatement();
13                int n = st.executeUpdate("INSERT INTO employee values(101,'employeeA',1000)");
14                System.out.println("Number of rows affected: "+n);
15
16                conn.close();
17            }
18        } catch(Exception e) {
19            e.printStackTrace();
20        }
21    }
22
23

```

Console output:

```

<terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (May 2, 2017, 12:38:04 AM)
Number of rows affected: 1

```

```

SQL> select * from employee;
EMP_ID EMP_NAME          EMP_SALARY
----- -----
      101 employeeA           1000
SQL> -

```

### Insert Query - only specific fields

```

1 import java.sql.*;
2
3 public class CRUDTest {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         try {
8             {
9                 Class.forName("oracle.jdbc.driver.OracleDriver");
10                Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
11
12                Statement st = conn.createStatement();
13                int n = st.executeUpdate("INSERT INTO employee(emp_id,emp_name) values(104,'employeeD')");
14                System.out.println("Number of rows affected: "+n);
15
16                conn.close();
17            }
18        } catch(Exception e) {
19            e.printStackTrace();
20        }
21    }
22
23

```

Console output:

```

<terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (May 2, 2017, 12:38:50 AM)
Number of rows affected: 1

```

104 employee has no salary.

```

103 employeeC           7000
SQL> select * from employee ORDER BY emp_id;
EMP_ID EMP_NAME          EMP_SALARY
----- -----
      101 employeeA           1000
      102 employeeB           3000
      103 employeeC           7000
      104 employeeD

```

### Update Query

```
JDBCTest.java CRUDTest.java
1 import java.sql.*;
2
3 public class CRUDTest {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         try {
8             {
9                 Class.forName("oracle.jdbc.driver.OracleDriver");
10                Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
11
12                Statement st = conn.createStatement();
13                int n = st.executeUpdate("UPDATE employee SET emp_salary = 11000 where emp_id = 104");
14                System.out.println("Number of rows affected: "+n);
15
16                conn.close();
17            }
18        } catch(Exception e) {
19            e.printStackTrace();
20        }
21    }
22}
23
```

@ Javadoc Declaration Console <terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0\_91\bin\javaw.exe (May 2, 2017, 1:09:55 AM)

## One more update statement

```
JDBCTest.java CRUDTest.java
1 import java.sql.*;
2
3 public class CRUDTest {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         try {
8             {
9                 Class.forName("oracle.jdbc.driver.OracleDriver");
10                Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
11
12                Statement st = conn.createStatement();
13                int n = st.executeUpdate("UPDATE employee SET emp_salary = 5000 where emp_salary > 1000");
14                System.out.println("Number of rows affected: "+n);
15
16                conn.close();
17            }
18        } catch(Exception e) {
19            e.printStackTrace();
20        }
21    }
22}
23
```

@ Javadoc Declaration Console <terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0\_91\bin\javaw.exe (May 2, 2017, 1:15:05 AM)

Number of rows affected: 3

## Delete Query

```
2
3 public class CRUDTest {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         try {
8             {
9                 Class.forName("oracle.jdbc.driver.OracleDriver");
10                Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
11
12                Statement st = conn.createStatement();
13                int n = st.executeUpdate("DELETE FROM employee where emp_id = 104");
14                System.out.println("Number of rows affected: "+n);
15
16                conn.close();
17            }
18        } catch(Exception e) {
19            e.printStackTrace();
20        }
21    }
22}
23
```

@ Javadoc Declaration Console <terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0\_91\bin\javaw.exe (May 2, 2017, 1:22:36 AM)

Number of rows affected: 1

## One more delete

```
Statement st = conn.createStatement();
int n = st.executeUpdate("DELETE FROM employee");
System.out.println("Number of rows affected: "+n);
```

Statement Vs PreparedStatement

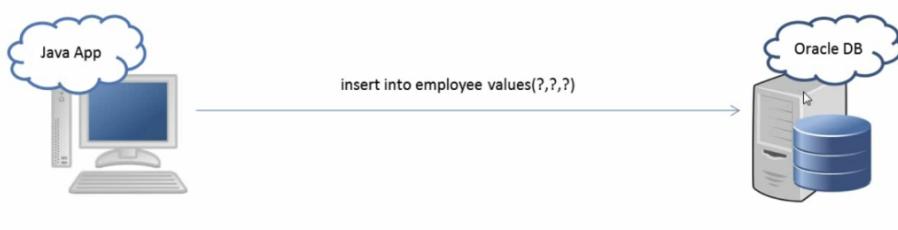
## Stages of processing a query

1. Parse the query.
2. Generate an execution plan.
3. Execute the query.

In case of a statement, object with hardcoded values it will compile db again and again for every insert statement. It will load/affect memory of the oracle database server.



Incase of prepare statement, compile will happens only once.



```
1 import java.sql.*;
2
3 public class CRUDTest {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         try {
8             {
9
10             Class.forName("oracle.jdbc.driver.OracleDriver");
11             Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
12
13             PreparedStatement ps = conn.prepareStatement("INSERT INTO employee values(?, ?, ?)");
14             ps.setInt(1, 101);
15             ps.setString(2, "employeeA");
16             ps.setInt(3, 1000);
17             int n = ps.executeUpdate();
18             System.out.println("Number of rows affected: "+n);
19
20             conn.close();
21
22         } catch(Exception e)
23     }
24 }
```

Output window:

```
<terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (May 3, 2017, 1:00:11 AM)
Number of rows affected: 1
```

```
Connected.
SQL> select * from employee order by emp_id;
no rows selected
SQL> select * from employee order by emp_id;
EMP_ID EMP_NAME          EMP_SALARY
----- -----
      101 employeeA           1000
SQL>
```

Update with PreparedStatement.

```
1
2     public static void main(String[] args) {
3         // TODO Auto-generated method stub
4         try {
5             {
6
7             Class.forName("oracle.jdbc.driver.OracleDriver");
8             Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
9
10            PreparedStatement ps = conn.prepareStatement("UPDATE employee SET emp_salary = ? WHERE emp_id = ?");
11            ps.setInt(1, 20000);
12            ps.setInt(2, 102);
13            int n = ps.executeUpdate();
14            System.out.println("Number of rows affected: "+n);
15
16            conn.close();
17
18        } catch(Exception e)
19        {
20            e.printStackTrace();
21        }
22    }
23 }
```

Output window:

```
<terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (May 3, 2017, 1:26:30 AM)
```

```
EMP_ID EMP_NAME          EMP_SALARY
----- -----
      101 employeeA           1000
      102 employeeB           20000
      103 employeeC          10000
SQL>
```

The screenshot shows the Eclipse IDE interface. In the top-left, there are two tabs: 'JDBCTest.java' and 'CRUDTest.java'. The code in 'JDBCTest.java' is as follows:

```
4
5  public static void main(String[] args) {
6      // TODO Auto-generated method stub
7      try {
8          {
9
10             Class.forName("oracle.jdbc.driver.OracleDriver");
11             Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
12
13             PreparedStatement ps = conn.prepareStatement("DELETE FROM employee where emp_salary >= 1000");
14             ps.setInt(1, 103);
15             int n = ps.executeUpdate();
16             System.out.println("Number of rows affected: "+n);
17
18             conn.close();
19
20         }
21         catch(Exception e)
22         {
23             e.printStackTrace();
24         }
25     }
26 }
```

In the bottom-right panel, under the 'Console' tab, the output is:

```
<terminated> CRUDTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (May 3, 2017, 1:46:26 AM)
Number of rows affected: 1
```

### Callable Statement:

Create store procedure in oracle db as shown below.

The screenshot shows the Eclipse IDE interface with a 'Run SQL Command Line' window open. The window title is 'SQL\*Plus: Release 11.2.0.2.0 Production on Sun May 7 23:30:57 2017'. Inside the window, the following SQL code is being run:

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> CREATE OR REPLACE PROCEDURE insertEmployee
2   eid IN NUMBER,
3   ename IN VARCHAR2,
4   esal IN NUMBER
5   IS
6   BEGIN
7   INSERT INTO employee VALUES(eid,ename,esal);
8   END;
9 /
```

And callable statement in program.

The screenshot shows an IDE interface with the following details:

- Java Editor:** The code for `CallableStatementTest.java` is displayed. It imports `java.sql.*` and defines a `CallableStatementTest` class with a `main` method. The code uses `DriverManager.getConnection` to connect to an Oracle database via JDBC. It prepares a call to `call insertEmployee(?, ?, ?)`, sets parameters (101, "employeeA", 9000), executes the update, and prints the number of rows affected (1). Finally, it closes the connection.
- Console:** The output of the Java application is shown. It starts with the JavaDoc links, then displays the command `<terminated> CallableStatementTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (May 7, 2017, 11:41:54 PM)`. Below this, the message `Number of rows affected: 1` is printed.

This screenshot illustrates a combined Java and Oracle environment. It features:

- Java Editor:** The same `CallableStatementTest.java` code is shown, with the cursor positioned at the end of the `conn.close()` statement.
- SQL\*Plus Session:** An Oracle SQL prompt (`SQL>`) is visible, displaying the result of a `select * from employee order by emp_id;` query. The output shows one row: `101 employeeA 9000`.
- Console:** The Java application's output is identical to the previous screenshot: `<terminated> CallableStatementTest [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (May 7, 2017, 11:41:54 PM)` followed by `Number of rows affected: 1`.

One more example with select statement

```

SQL*Plus: Release 11.2.0.2.0 Production on Mon May 8 00:51:10 2017
Copyright (c) 1982, 2014, Oracle. All rights reserved.

SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> CREATE OR REPLACE PROCEDURE getEmployeeById(
  2     eid IN employee.emp_id%TYPE,
  3     ename OUT employee.emp_name%TYPE,
  4     esal OUT employee.emp_salary%TYPE
  5 IS
 6 BEGIN
 7
 8     SELECT emp_name, emp_salary
 9     INTO ename, esal
10    FROM employee
11   WHERE emp_id = eid;
12 END;
13 /

```

Above we have one input parameter and 2 output parameters in our store procedure and its program below.

```

File Edit Source Refactor Search Project Run Window Help
File Edit Source Refactor Search Project Run Window Help
Quick Access Java EE Java
Package Explorer JDBC CallableStatementTest.java
src (default package)
  CallableStatementTest.java
  CRUDTest.java
  JDBCCTest.java
JRE System Library [JavaSE-1.8]
Referenced Libraries
1 import java.sql.*;
2
3 public class CallableStatementTest {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         try {
8             Class.forName("oracle.jdbc.driver.OracleDriver");
9             Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
10            CallableStatement st = conn.prepareCall("{call getEmployeeById(?,?)}");
11            st.setInt(1,101);
12            st.registerOutParameter(2, java.sql.Types.VARCHAR);
13            st.registerOutParameter(3, java.sql.Types.INTEGER);
14            st.executeUpdate();
15
16            System.out.println("Name: "+st.getString(2)+" Salary: "+st.getInt(3));
17
18        } catch (Exception e) {
19            e.printStackTrace();
20        }
21        conn.close();
22    }
}

```

JavaDoc Declaration Console

<terminated> CallableStatementTest [Java Application] C:\Program Files\Java\jre8.0\_91\bin\javaw.exe (May 8, 2017, 12:55:00 AM)

Name: employeeA Salary: 9000