JQL : The Java Query Language

Slides created by Darren Willis, David J Pearce, James Noble; used with their permission
JQL is a simple extension for Java.

JQL adds support for **Object Querying**.

**Object Querying** helps us pick sets of objects from other sets.

JQL can simplify working with collections.
Without JQL:

```java
for(Object a : collection1)
    for(Object b : collection2)
        if(a.equals(b))
            results.add(new Object[]{a,b});
```

With JQL:

```java
selectAll(Object a = collection1,
          Object b = collection2
         :a.equals(b));
```
### Session

<table>
<thead>
<tr>
<th>String name;</th>
</tr>
</thead>
<tbody>
<tr>
<td>send_msg(Msg);</td>
</tr>
<tr>
<td>recv_msg(Msg);</td>
</tr>
</tbody>
</table>

### Window

<table>
<thead>
<tr>
<th>String name;</th>
</tr>
</thead>
<tbody>
<tr>
<td>int id;</td>
</tr>
<tr>
<td>show_msg(Msg);</td>
</tr>
<tr>
<td>enter_msg();</td>
</tr>
</tbody>
</table>
We're **joining** the sets of Sessions and Windows.

An easy way – the nested loop join:

```java
ArrayList results = new ArrayList();
for (Session s : sessions)
    for (Window w : windows)
        if (w.name.equals(s.name))
            results.add(new Object[w, s]);
```

This takes $O(|sessions| \times |windows|)$ time.
Another join technique, from databases:

```java
ArrayList results = new ArrayList();
Hashtable nameIndex = new Hashtable();
for (Session s : sessions)
    nameIndex.put(s.name, s);
for (Window w : windows) {
    results.add(names.get(w.name));
}
```

(This is a simplified version)
results = selectAll(Session s = sessions,
    Window w = windows:
    s.name.equals(w.name));

Let the Query Evaluator take care of the details!
Passing in a Domain Variable source

```java
selectAll(Window w = windows : w.ID == 5)
```

Domain Variable Definition without a source

```java
selectAll(Window w : w.ID == 5)
```
w.ID == 5

w.ID == 5 && w.name == s.name

w.ID > 5 && w.ID < 10 && w.name == s.name && s.isActive() && ...

• Method calls are allowed
• Side effects are possible
• These expressions do not short circuit

So what else can we do with queries?
BinTree Aliasing
assert null == selectA(BinTree a, BinTree b :
    (a != b && a.left == b.left) ||
    (a != b && a.right == b.right)) ||
    a.left == b.right);

Uses **Object Tracking** to check all BinTrees.
public class GroupChat{
    private List participants;

    public SomeClass(){
        participants = new ArrayList();
        ...
    }

    public void addChatter(String name){
        ...
        participants.add(new Session(name));
        ...
    }
}
Queries get broken down along `&&`

```
w.ID == 5 && w.name == s.name
```

Becomes two subqueries:

```
w.ID = 5
w.name == s.name
```

JQL arranges these into a *query pipeline*. 
w.ID == 5

w.name == s.name

w.ID == 5 && w.name == s.name

JQL - A Query Pipeline
There are two factors to consider:

- **Cost**
  - Very dependent on input sizes
  - Input sizes are dependent on selectivity…

- **Selectivity**
  - The proportion of results rejected by the stage

  - We estimate it based on the expression used:
    \[ \text{==, < >, !=, aMethod()} \]

  - Or, we can test it with sampling
Two strategies to decide Join Order:

- **Exhaustive Search**
  - Good results but expensive
  - Requires searching $n!$ possible orderings

- **Maximum Selectivity Heuristic**
  - Less optimal order, less overhead.
Four versions of:

```java
selectAll(Integer a=as,
        Integer b=bs,
        Integer c=cs,
        Integer d=ds:
        a == b &&
        b != c &&
        b != c &&
        c < d);
```
HashMap<Integer,ArrayList<Integer>> map;

map = new HashMap<Integer,ArrayList<Integer>>();

for (Integer i1 : array1) {
    ArrayList<Integer> grp = map.get(i1);
    if (grp == null) {
        grp = new ArrayList<Integer>();
        map.put(i1, grp);
    }
    grp.add(i1);
}

ArrayList<Object[]> matches = new ArrayList<Object[]>();

Collections.sort(array4);

for (Integer i2 : array2) {
    int b = i2;
    ArrayList<Integer> grp = map.get(i2);
    if (grp != null) {
        for (Integer i1 : grp) {
            int a = i1;
            for (Integer i3 : array3) {
                int c = i3;
                if (b != c) {
                    for (int x = array4.size(); x != 0; x = x - 1) {
                        int d = array4.get(x - 1);
                        if (c < d) {
                            Object[] t = new Object[4];
                            t[0] = a;
                            t[1] = b;
                            t[2] = c;
                            t[3] = d;
                            matches.add(t);
                        } else {
                            break;
                        }
                    }
                } else {
                    break;
                }
            }
        }
    }
}

return matches;
JQL's performance is pretty good!

The average programmer is more likely to code HANDPOOR than HANDOPT.
Object tracker overhead varies greatly.
$C_\omega$/LINQ(C#)  
Provide similar querying capabilities.

Query-Based Debuggers  
QBD, DQBD by Lencevicius et al.  
Used querying and object tracking for debugging.

Relational Databases  
Much prior work in query optimisation.
Caching
  Both simple, and incrementalised

Other join methods
  Again, from databases – Merge, Sort, etc

Tool Support
  Eclipse/Netbeans/etc
Conclusions

- Queries are neat
- They are a powerful, simple, useful abstraction.
- JQL provides efficient and easy support for querying in Java.
- For more information on JQL:
  [www.mcs.vuw.ac.nz/~darren/jql/](http://www.mcs.vuw.ac.nz/~darren/jql/)