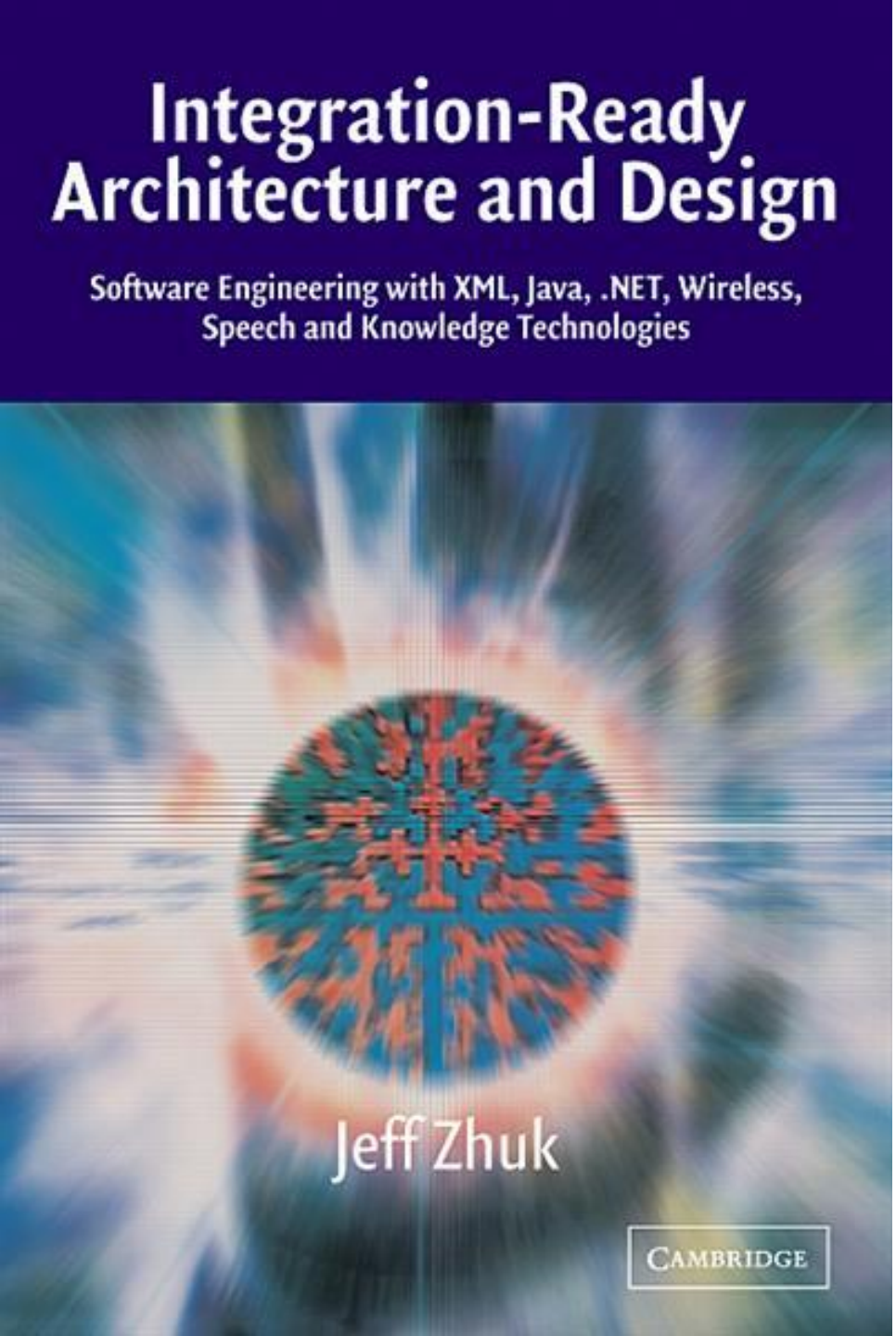

Design and Coding Hints

Jeff (Yefim) Zhuk

From the book

"Integration-Ready Architecture and Design"

Cambridge University Press

JavaSchool.com

Software Engineering With XML, Java, .NET, Wireless, Speech and Knowledge Technologies

Design and Code Hints
Optimize String Manipulations

When new strings are created,
GC has a lot of work to do.

```
System.out.println(

person.getLastName() +

", " + person.getFirstName() +

" lives at " +

person.getAddress().getStreet1() +

"; " + person.getAddress().getCity()

+  ",  "  +

person.getAddress().getState() );
```

Use StringBuffer – better choice!

```
StringBuffer buf = new StringBuffer;
buf.append(person.getLastName());
buf.append(", ");
buf.append(person.getFirstName());
buf.append(" lives at ");
buf.append(person.getAddress().getStreet1());
buf.append("; ");
buf.append(person.getAddress().getCity());
buf.append(",  " );
buf.append(person.getAddress().getState());
System.out.println(buf.toString());
```

Design and Code Hints
Re-use Objects (Especially in loops!)

Checking the length of string on every cycle is extra work

```
public loop control()
{
  String str = “abcdefghi”;
  for (int j = 0; j < str.length(); j++)
  {
  // code doesn’t change the length of the string.
  }
}
```

Extra line – is the better way

```
public loop control()
{
   String str = “abcdefghij”;
   int len = str.length(); // !!!
   for (int j = 0; j < len; j++)
   {

   }
}
```

The JVM is optimized to compare to integers between -1 and +5.

for (int j = len-1; j >= 0; j—) // even better if you compare len to 0

Design and Code Hints
Avoid using synchronized classes

Vector is an example of a synchronized class.
Synchronized classes provide single-thread access to its data using locks.

Use HashMap and ArrayList classes when synchronized access to data is not required to improve performance.

Design and Code Hints
Optimize Exception Handling

Try/Catch is Expensive!

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


  try
  {
      int[] numbers = new int[args.length];
      for(int i = 3; i <args.length; i++)
      {
         numbers[i] =
              Integer.parseInt(args[i]);
      }
   } catch(Exception e) {
      System.err.println(e);
   }
}
```

Minimize Try/Catch content

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

{
  int[] numbers = new int[args.length];
 for(int i = 3; i <args.length; i++)
  {

    try
    {
       numbers[i] =
          Integer.parseInt(args[i]);
     } catch(Exception e) {

         System.err.println(e);


    }
  }
```

Design and Code Hints
Static is good for reusable objects!

```
// Reuse of Exception Object
public static Exception REUSABLE_EX = new Exception();
public void method1() throws EXCEPTION
{
        if (I == 1)
                throw REUSABLE_EX;
}

//  "throw new Exception();" // 50 –100 times slower
```

Design and Code Hints
Prevent Out Of Memory Crash

```
Select allThatMatchesCritirea
from bigDataSet

ResultSet rs = tooBig; // crash?

// Create a list of object-records

// User Interaction Loop

// Request to display a page

// Display a selected page
```

→

```
Select rowIdForCritirea from
bigDataSet

// User Interaction Loop

Select properRowIDs from
bigDataSet


ResultSet rs = perPage


// Create a list of object-records
// Display a selected page
```

Unlimited data records often cause out of memory exception. In most cases we need this data to display and user will only observe a portion of data at the time.

Select only reference indexes (rowID) instead of full records and use this array of indexes to select a page of data records.

Design and Code Hints
Foreign Char Set Might Crash

```
SimpleDateFormat sdf = new
SimpleDateFormat(

"dd-MMM-yyyy”);

String date =
sdf.format(userData); // crash?
```

```
SimpleDateFormat sdf = new
SimpleDateFormat(

"dd-MMM-yyyy”,

// enforce US char set

java.util.Locale.US); // !!!

String date =
sdf.format(userData);
```

Foreign character set can be installed on the machine .

In this case the attempt to parse the date (by SimpleDateFormat) might cause Parse Exception.

Use two argument constructor for SimpleDateFormat with java.util.Locale.US to prevent the crash.

Design and Code Hints
Setting Eclipse/JBoss Environment

Goal: create EAR and WAR files transparent to Dev, Test, and Production, avoid changing lines inside EAR/WAR files

Set in the JBoss ../deploy folder all environment variables that are different for Development/Test/Production

data - in oracle-ds.xml or applicationName-ds.xml
URLs - in applicationName-url.properties
other environment dependencies - in applicationName.properties

Use the web.xml and similar files inside EAR or WAR space to set the variables that are the same for Development/Test/Production

Data Sources are bound once when application started via a listener-class assigned in the **web.xml** and connections are handled by common DataService methods.

The line in the web.xml file to assign initialization procedure:

<listener-class>**yourPackage.MyServetContextListener**</listener-class>

See http://javaschool.com/school/public/JZ.Train.Reuse.htm for more details

Design and Code Hints

Use common data services, avoid code duplications, and focus more on a business side of applications.

WEB-INF/sql/getUser.sql

Select * from users where loginName = ':loginName'

```
keys.put("loginName", form.getLoginName()); // common HashMap keys
List records = DataService.getData("getUser", keys, User.class);
User user = (User) records.get(0);
```

Don't mess with SQL in Java code. Keep it in separate files in the SQL – directory.

Connections, Pooling, ResultSet Processing, and all trivial data code is collected in DataService methods.

Don't duplicate but focus on your business!

Design and Code Hints
On-The-Fly Query 1

```
String q = “select something from someTable where “;
if(userProvidedDataField) {
    q += “providedDataField = “ + providedValue;
}

// more if/else cases to build query

// more lines of code to connect, execute query, and process the result set
```

WEB-INF/sql/fullQueryWithAllConditions.sql

```
select something from someTable where name1=‘:value1’ and name2=:value2
```

```
List records = DataService.getData(“fullQueryWithAllConditions”, keys,
RecordBean.class);
```

All data provided by a client via the web page form will be collected in the HashMap keys (automatically).

The DataService will parse the query and remove conditions that are not supported by values in the keys.

Design and Code Hints
On-The-Fly Query 2

```
String q = “select something from someTable where “;
if(userProvidedDataField) {
    q += “providedDataField = “ + providedValue;
}

// more if/else cases to build query

// more lines of code to connect, execute query, and process the result set
```

// DataService.getData() can take ready-to-go SQL statement as an argument

```
List records = DataService.getData(q, RecordBean.class);
```

// It is possible to use multiple data sources in the application

// DataService.getData() can take a dataSourceName as an argument

// All data sources are bound automatically during application start up

```
List records = DataService.getData(q, RecordBean.class, “SpecificDataSource”);
```

Design and Code Hints
On-The-Fly Query 3

```
String q = “select something from someTable where “; // starting query string
if(userProvidedDataField) {
    q += “providedDataField = “ + providedValue;
}

// more if/else cases to build query

// more lines of code to connect, execute query, and process the result set
```

```
// Simple cases of data retrieval
String role = DataService.getString(q); // select a string, for example, a user’s role

int count = DataService.getInt(q);       // select a number, for example, a count

// We might not know the resulting structure because it’s run-time dependent
// The method below retrieves multiple records with multiple fields using rs.getMetaData
Vector records = DataService.getFlexTextRecords(q); // each record is a vector
```