


JHUG March 2011



## Commenting Java code in Enterprise applications

Kapelonis Kostis (kkapelon@gmail.com)

I have issues



Sometimes I cry when I see  
(bad) Java code

# Package by feature



My biggest Java complaint  
(Previous JHUG Presentation)



Code comments

Specifically Java comments in  
Enterprise Applications



Junior programmer

# The Godfather



```
[...create integer i...]
```

```
    i++ //Add 1 to i
```

```
[...use i ...]
```

Java programmer

Are comments better?



The Godfather  
PART II



# Java programmer

**//Search all fields**

```
searchAllFields(true);
```

**//Calculate result**

```
calculateResult();
```

# Senior Java programmer



THE GODFATHER  
SERIE LIMITED

The Godfather  
Part III

Out of sync Javadoc



# Senior Java programmer



S E R I E L I M I T E E

```
/**Searches all fields
 * @param index starting position
 */
public boolean searchFields()
{
    [...]
}
```

# Definition



Enterprise?

# Definition (according to me)

- Lot's of code (e.g. 150K LOC)
- No single developer knows all parts
- Sometimes the team does not include original authors
- Often this is just for maintenance



# Part I

# Javadoc Comments



Attention, attention!

Even experienced Java  
developers forget this!

# Golden Rule for Javadoc



**Javadoc comments can exist without the actual Java code!**

# Javadoc without Java code

```
/**
```

```
* adds the item into the overall
```

```
* ArrayList collectedItems
```

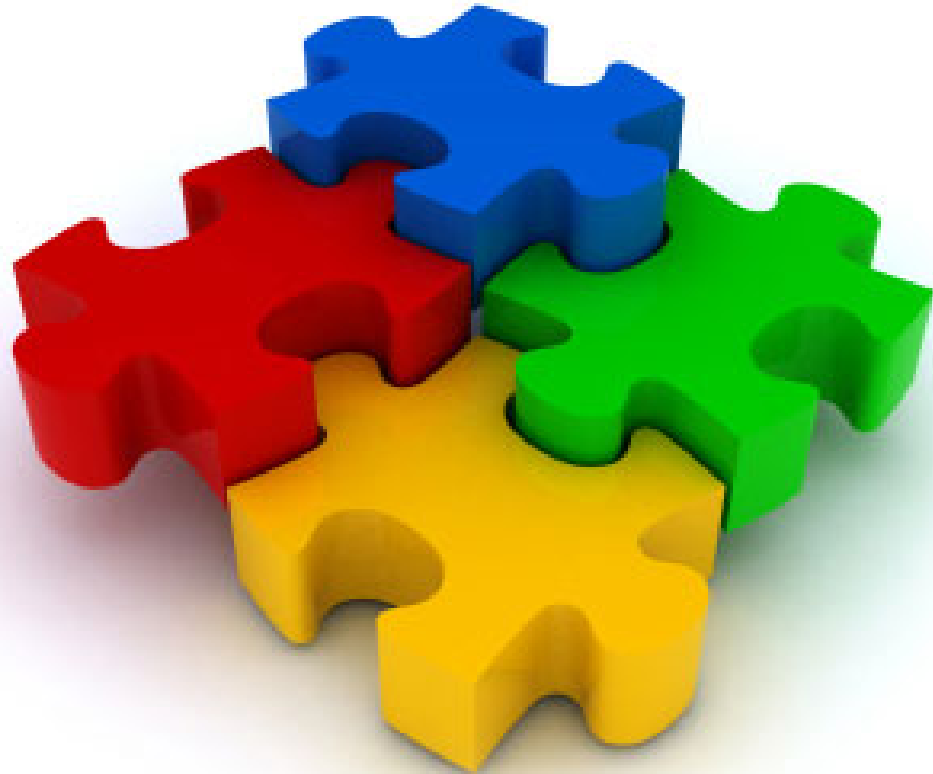
```
* (See top of Collected.java  
* for the definition)
```

```
*/
```

```
public void addItem(Item item)
```



# Javadoc = specification



No implementation details  
should be mentioned



# Javadoc = specification



```
/**
```

```
* Sorts all items
```

```
* using Quick Sort
```

```
*/
```

```
public void sortItems()
```

A photograph of six chocolate cupcakes arranged in a cluster. Each cupcake is topped with a swirl of white frosting, a chocolate heart-shaped decoration, and a small green leaf. The cupcakes are in dark brown paper liners. The background is a plain, light-colored surface.

Brevity

Keep Javadoc  
short and sweet

# Javadoc should be sort

```
/**
```

- \* Clears the internal caches.
- \* Closes all open file descriptors.
- \* Flushes the network output.
- \* Brings the StringWriter back to 0
- \* Washes the dishes/takes garbage out.

```
*/
```

```
public void reset()
```



# Javadoc should be sort



```
/**  
 * Brings the reader to its  
 * initial state.  
 */  
public void reset()
```

# Synchronization



Be careful after copy-paste

# Javadoc out of sync




```
/**
```

```
* Searches all items
```

```
* @param index
```

```
*/
```

```
public boolean searchItems()
```



My advice  
to you

**Write Javadoc comments for your  
Business Analyst**



## Part II

# Inline Comments



# Golden Rule for comments



Write WHY something is done  
Not HOW it is done.

# Documenting WHY



```
/* The first item is already  
* processed, skip it.  
*/  
  
i++;  
searchItems(i);
```

Less is more



Comments are not TODO lists

# No Todo Lists

```
public void searchItems()
{
//validate all items
for (Item item:items)
{ validateItem(item);}
// calculate average price
for (Item item: items)
{ average = examineAverage(item);}
}
```



# No TODO lists



```
public void searchItems()  
{  
    validateAllItems(items);  
    calculateAveragePrice(items);  
}
```

# Documenting WHY



Magic numbers and everything that is strange should be commented.

# Documenting WHY



```
public void locatePrice()  
{  
    //locate 6th field  
    calculatePrice(fields[5]);  
}
```

# Documenting WHY



```
public void locatePrice()  
{  
    /*The financial library we use  
     * has the price in the 6th field */  
    calculatePrice(fields[5]);  
}
```



# Documenting WHY



Write why choices are made, not what they do different

# Documenting WHY

```
if(discount == 0) {  
    [...lots of java code...]  
    calculatePrice(fields[5]);  
}  
else //if there is discount  
{  
    [...lots of java code...]  
    calculatePrice(fields[6]);  
}
```



# Documenting WHY

```
if(discount == 0) {  
    calculatePrice(fields[5]);  
}
```



```
else
```

```
{
```

```
    /* discount counts as field 0
```

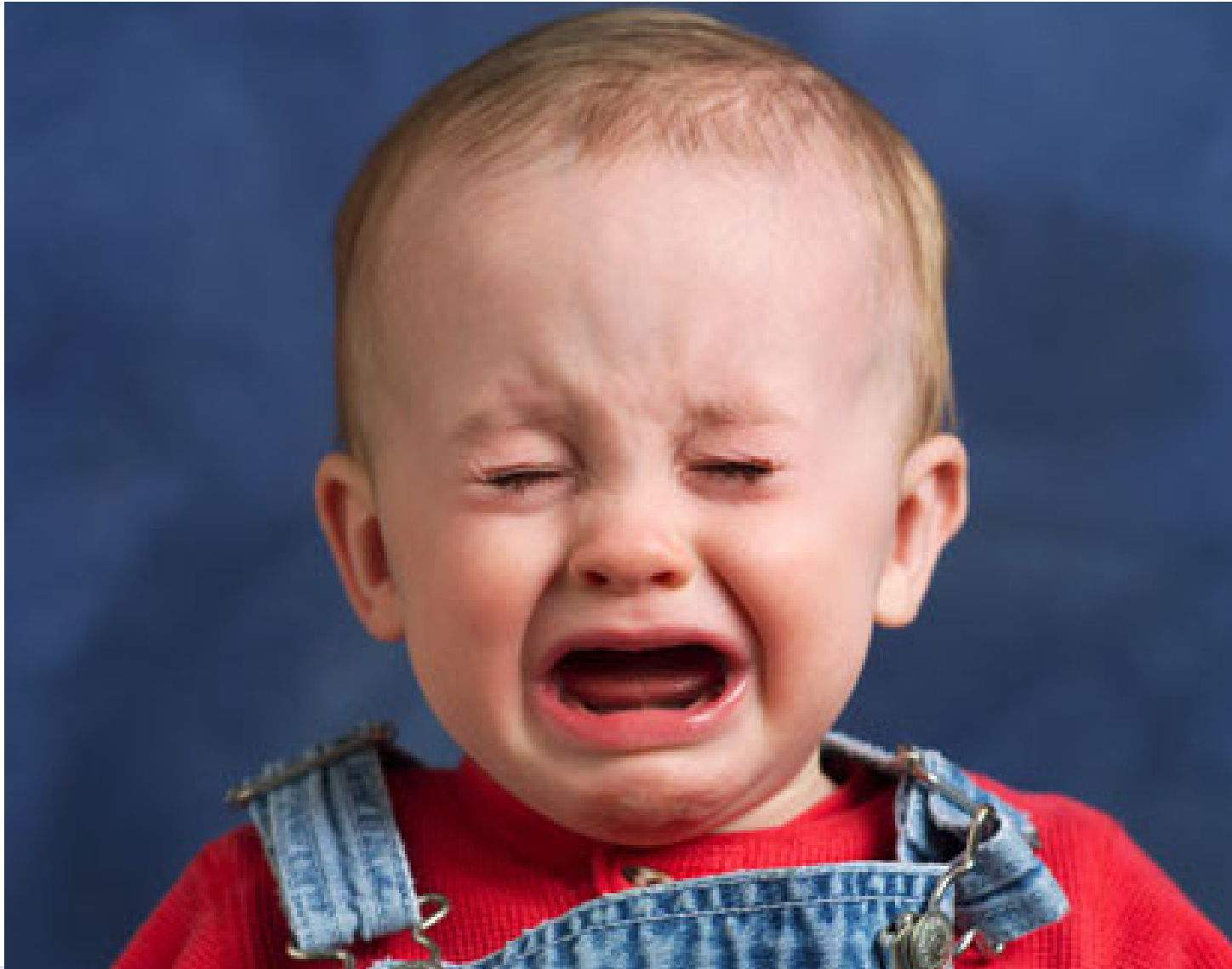
```
    * therefore price is located in the next
```

```
    * field 7 instead of 6
```

```
    calculatePrice(fields[6]);
```

```
}
```

# Documenting WHY



**My personal big complaint about comments**

# Documenting WHY



//See JIRA issue 5867

```
calculatePrice(fields[5]);
```

# Documenting WHY



```
/* See JIRA issue 5867
```

```
* The price field is in position 6 in
```

```
* version 1.3 of the library.
```

```
*/
```

```
calculatePrice(fields[5]);
```

# Useless comments



Territorial pissing

# No territorial pissing



```
//code added by John  
calculatePrice(fields[6]);  
findResult(fields);  
// end of code additions by John
```



# Useless comments



Doubt and confusion

# Doubt and confusion



```
//I am not sure why this works  
calculatePrice(fields[6]);  
// Does this belong here?  
findResult(fields);  
// Maybe there is a better way for this  
applyDiscount(order);
```

# Useless comments

Inside jokes



# Inside jokes



```
//John told me to implement this  
calculatePrice(fields[6]);  
findResult(fields);  
// If this breaks ask Nick. He wrote it.
```

# Useless comments



Swearing

# Swearing



```
// The client is fucking crazy for this  
calculatePrice(fields[6]);  
// Stupid specification says that  
findResult(fields);
```

# The end



Questions?