

Productive Coder

Dr Heinz M. Kabutz

The Java Specialists' Newsletter

<http://www.javaspecialists.eu>



JavaSpecialists.EU

Productive Coder

- **How you can have more fun interacting with your machine ...**
- **... and make your computer less frustrated with having you as operator 😊**

Background

- **Dr Heinz M. Kabutz**
 - ***German South African living in Europe***
 - ***The Java Specialists' Newsletter***
 - ***For Java Experts***
 - ***Sign up sheet on clipboards***
 - ***Actively coding Java***
 - ***Teaching Java courses to companies:***
 - **Java Patterns Course**
 - **Java 5 Delta Course**
 - **<http://www.javaspecialists.eu/courses>**
 - ***Java Champion***



Become One With Your Machine

- **Typical programmer works 60 hours per week**
 - We all want *maximum* of 40 hours
- **Programmer and machine should be one**
 - Feel the machine
 - Understand the machine
 - Speak nicely to the machine 😊
- **Human Computer Interaction is progressing slowly**
 - You should be able to type this whilst at the same time watching TV.
 - When you make a typing error, you should know that you have made it without looking at the screen.

Keyboard Skills

- **Not all programmers can touch type**
- **But it is so easy:**
 - Each keyboard has dimple for index fingers on “F” and “J”
 - From there, each finger controls the buttons above and below it
- **Initial investment of about 20 hours**
- **Try to mainly use the keyboard – minimize mouse use**
 - Menu driven copy & paste ...
- **European keyboard layouts bad for coding**
 - Semicolon and curly braces

Keyboard Magic

- **Back to the basics of working with computers**
- **Applies to any language, not just Java**
- **But, Java's IDEs make this approach even more productive**

Keyboard Shortcuts

- **Memorise as many as possible**
- **Use them frequently**
- **Try to minimize mouse usage**
- **Every IDE is different**
 - Sometimes on purpose it seems
 - **CTRL+D in IntelliJ & Eclipse**
- **Learn vim**
 - **Productive for small jobs**
 - **Good discipline in keyboard use**



Know Your IDE

- **Intelij my favourite**
 - **Netbeans 6 and Eclipse narrowing gap**
- **Short demo of how to create three classes:**
 - **Flower, RarityStatus, FynbosFan**



Orothamnus
zeyheri
(Marsh Rose)

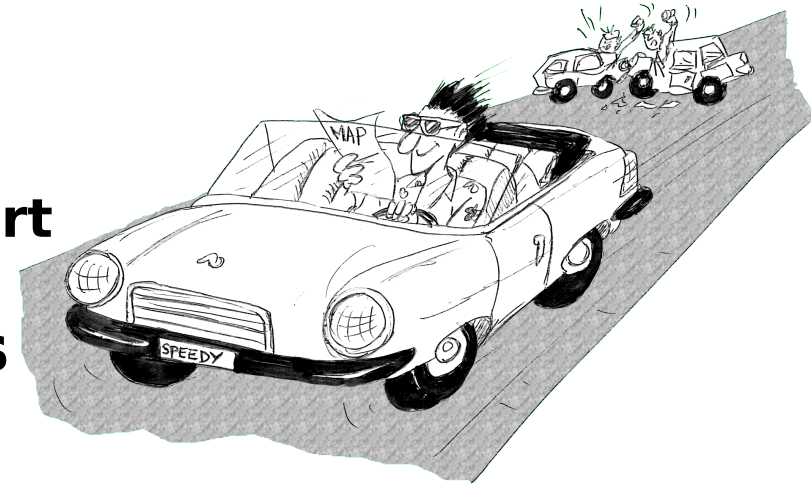
Which IDE ? – Does Not Matter!

- **Whatever your choice, it's *your* choice**
- **Spend 10 hours getting to know keyboard shortcuts**



Fingers Overtaking the Brain

- **You still need to plan**
 - **Stop & think before you start**
- **When shortcuts & fingers are too fast:**
 - **Increase speed of your brain**
 - **Think in higher level concepts, such as Design Patterns**



Design Patterns

- **Mainstream of OO landscape, offering us:**

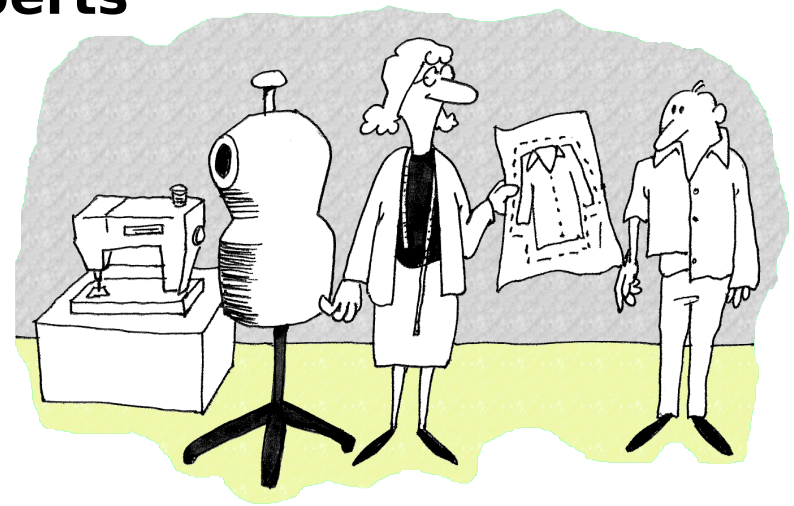
- **View into brains of OO experts**

- **Quicker understanding of existing designs**

- e.g. Visitor pattern used by Annotation Processing Tool

- **Improved communication between developers**

- **Readjusting of “thinking mistakes” by developers**





Vintage Wines



- **Design Patterns are like good red wine**
 - You cannot appreciate them at first
 - As you study them you learn the difference between *plonk* and vintage, or bad and good designs
 - As you become a connoisseur you experience the various textures you didn't notice before
- **Warning: Once you are hooked, you will no longer be satisfied with inferior designs**



“Houston, We Have a Problem”

- **“Our lead developer has left”**

- **Software works most of the time**
- **We have to fix it, and add some features ...**



- **How do you start?**

- **What code is dead?**
 - **Stories of whole teams working on dead code for years**
- **Where are the unit test?**
- **Where could access control be tighter?**
- **What portion of code is commented?**
- **How can I find bad code? Copy & paste code?**

Initial Investigation

- **Check where comments are missing**
 - **Doclet that checks that all elements are documented**
<http://www.javaspecialists.eu/archive/Issue049.html>
- **Find fields that are not private**
 - **Doclet that runs through your code and finds non-private fields**
<http://www.javaspecialists.eu/archive/Issue035.html>
- **Count number of classes, lines of code per class**
 - **Aim is for average of less than 100 lines per class**
 - **One of my customers had one Java class > 30000 LOC**
- **Run code coverage tool against unit tests**

What are Realistic Values?

	# Classes	Total LOC AVG/STDEV	Uncommented Elements
Project 1 South Africa	1359	263790 194 / 337	24291 18 per class
Project 2 Germany	442	62393 141 / 149	7298 17 per class
Ideal	1000	80260 80 / 61	1000 max 1 per class

- **Beware, LOC is only a *rough* measurement**

Comments Should Explain “Why”

- **Should not just be: *Method getName returns the name.***
- **Switch off automatic comment generation**
- **Either fill in comments properly, or leave them out**
- **Method names and parameters should be descriptive**
- **“Why I don’t read your code comments ...”**
 - **Most misunderstood newsletter**
 - <http://www.javaspecialists.eu/archive/Issue039.html>
 - **I do write my own comments, but about “why” not “what”**
 - **But, I seldom find projects with well-written comments**

Comments: java.awt.color.ColorSpace

- **Rather insightful comment in JDK 1.3:**

```
/**  
 * Returns the name of the component given the  
 * component index  
 */  
public String getName (int idx) {  
    /* REMIND - handle common cases here */  
    return new String(  
        "Unnamed color component("+idx+")");  
}
```

- **What is “REMIND” supposed to tell us?**

Comments: java.awt.color.ColorSpace

- In JDK 1.4, more comments, but still the question

```
/**  
 * Returns the name of the component given the  
 * component index.  
 * @param idx The component index.  
 * @return The name of the component at the  
 * specified index.  
 */  
public String getName (int idx) {  
    /* REMIND - handle common cases here */  
    return new String(  
        "Unnamed color component("+idx+")");  
}
```

Comments: java.awt.color.ColorSpace

- **Java 5**

```
/** Returns the name of the component given the  
 * component index.  
 * @param idx The component index.  
 * @return The name of the component at the  
 * specified index.  
 * @throws IllegalArgumentException if idx is less  
 * than 0 or greater than numComponents - 1 */  
public String getName (int idx) {  
    /* REMIND - handle common cases here */  
    if ((idx < 0) || (idx > numComponents - 1)) {  
        throw new IllegalArgumentException(  
            "Component index out of range: " + idx);  
    }  
    return new String(  
        "Unnamed color component("+idx+")");  
}
```

Comments: java.awt.color.ColorSpace

- **Java 6**

```
/** Returns the name of the component given the  
 * component index.  
 * @param idx The component index.  
 * @return The name of the component at the  
 * specified index.  
 * @throws IllegalArgumentException if idx is less  
 * than 0 or greater than numComponents - 1 */
```

```
public String getName (int idx) {  
    /* REMIND - handle common cases here */  
    if ((idx < 0) || (idx > numComponents - 1)) {  
        throw new IllegalArgumentException(  
            "Component index out of range: " + idx);  
    }  
    if (compName == null) {  
        switch (type) {  
            case ColorSpace.TYPE_XYZ:  
                compName = new String[] {"X", "Y", "Z"}; break;
```

Commenting Out Code

- **Source Control Systems**
 - Have been around for decades
- **Don't duplicate work done by source control**
- **If code is dead, delete it, don't comment it out**



Funny Comments

Shouldn't that be
ObjectInputStream?

- **JDK 1.3: java.io.ObjectStreamClass**

```
private final static Class[] NULL_ARGS = {};  
//WORKAROUND compiler bug with following code.  
//static final Class[] OIS_ARGS={ObjectInpuStream.class};  
//static final Class[] OOS_ARGS={ObjectOutpuStream.class};  
private static Class[] OIS_ARGS = null;  
private static Class[] OOS_ARGS = null;  
private static void initStaticMethodArgs() {  
    OOS_ARGS = new Class[1];  
    OOS_ARGS[0] = ObjectOutputStream.class;  
    OIS_ARGS = new Class[1];  
    OIS_ARGS[0] = ObjectInputStream.class;  
}
```

- **“The compiler team is writing useless code again ...”**

- <http://www.javaspecialists.eu/archive/Issue046.html>

“Wonderfully Disgusting Hack”

- **JDK 1.4: java.awt.Toolkit**

```
static boolean enabledOnToolkit(long eventMask) {  
    // Wonderfully disgusting hack for Solaris 9
```

- **This made me think:**

- 1. All software contains hacks.**

- 2. I would prefer to know about them.**

- 3. Only a real developer would write "hack" into his comments.**

- 4. Rather use Java than black-box proprietary solution with hundreds of undocumented hacks**

- **“Wonderfully Disgusting Hack”**

- <http://www.javaspecialists.eu/archive/Issue077.html>

Before You Change Any Code...

- **Refactoring is dangerous!**
- **You must have good unit tests**
 - **And great skill if you don't have unit tests...**
- **Also system tests**
- **In troubled projects, unit tests often absent**

Real-Life Case Study

- **Customer has kindly agreed for you to see his code**
- **Domains, names, etc. have been altered**
- **This is not the *worst* I have had to work with**

Real-Life Example

- **Company someone.com has Java application**
- **Single programmer has left**
- **Features must be added and bugs fixed**
- **Initial stats:**

	# Classes	Total LOC AVG / STDEV	Uncommented Elements
Someone.com	97	19478 201 / 181	2461 25 per class

Better Metrics

● **Fanout (FO)**

- **Number of other classes used in**
 - **Fields**
 - **Parameters**
 - **Local variables**
 - **Return**
 - **Throws**
- **Primitives and supertypes not counted**
- **Recommended maximum of 15**
- **Warning sign: Large number of “import” statements**

Better Metrics

- **Halstead Program Length (HPLen)**
 - **Halstead Software Science metric**
 - **Calculated per class**
 - **'Number of Operators' + 'Number of Operands'**
 - **Maximum of 2000**
 - **Average should be much less**

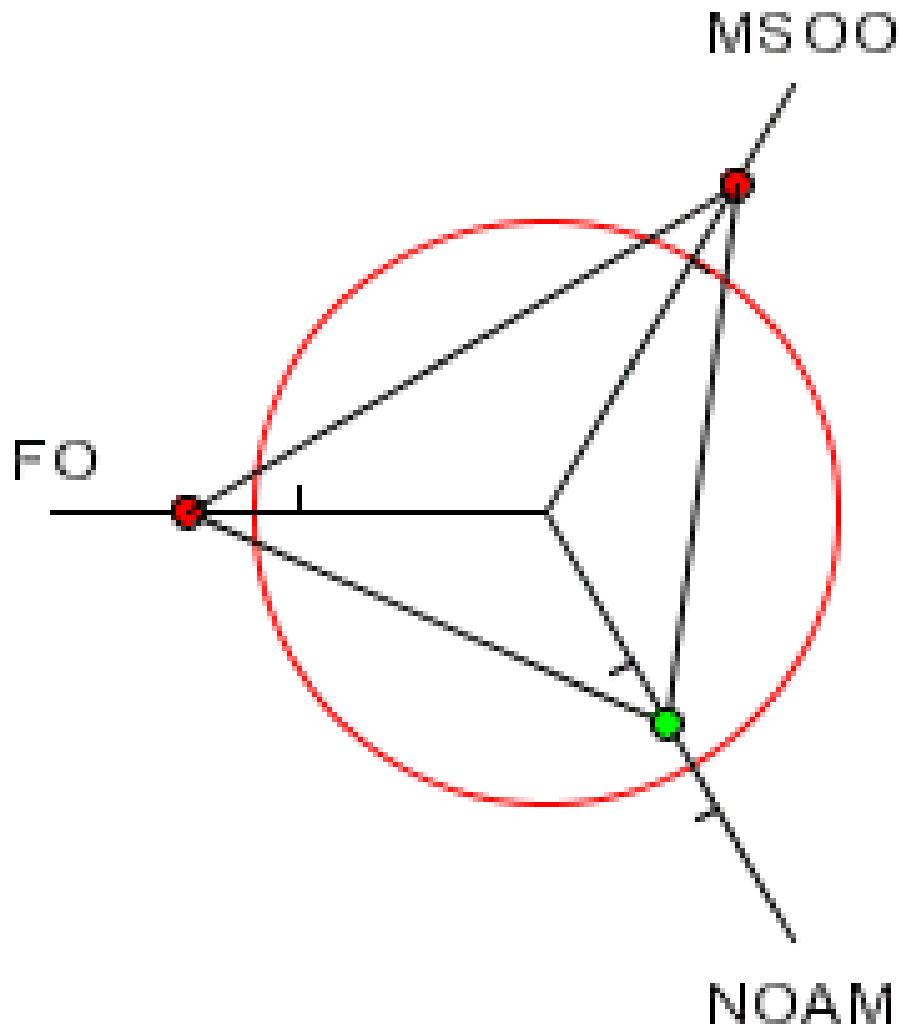
Better Metrics

- **Maximum Size Of Operation (MSOO)**
 - **Counts maximum size of operations for a class**
 - **Method size determined by cyclomatic complexity**
 - **number of if, for and while statements**
 - **Finds overly complex, badly factored methods**

Better Metrics

- **Number Of Added Methods (NOAM)**
 - **Counts the number of operations added by a class**
 - **Inherited and overridden operations are not counted**
 - **Absolute maximum is 50**
 - **Maybe too generous?**
 - **Large value means subclass is too different to superclass**

What do the Metrics Say?



Encapsulation

- **Encapsulation is more than private fields**
 - Though all non-constant fields should be private
- **Getters and Setters often break encapsulation**
- **What is the difference between public getName() and setName() methods and making field public?**
 - Yes, you can check input values
 - Fundamental problem – we are not delegating responsibility
 - E.g. Entity Beans History

Non-private Fields

- **Search with Doclet**

- <http://www.javaspecialists.eu/archive/Issue035.html>

- **Output:**

Non-private data members:

public com.someone.gui.InfoFrame:

 java.lang.StringBuffer buffer

public com.someone.gui.TableMap:

 protected javax.swing.table.TableModel model

public com.someone.io.DelimitedFileDataSource:

 protected java.lang.String[][] data

 protected int index

public com.someone.io.DelimitedFileReader:

 protected java.lang.String fileName

 protected java.lang.String[] headers

 protected int numberOfFields

Found 203
non-private
non-
constant
fields

Fixing the Code

- **Either make them all private and see what breaks**
 - Kind of tedious
- **Or use a tool to tighten field access control**
 - Made me into an IntelliJ convert
 - Short demonstration ...
- **Rerun the doclet: 104 non-private fields**
 - An improvement from 203!
- **Now real work begins - why are they not private?**

Immutable Fields

- **Fields should be marked “final” where possible**
 - <http://www.javaspecialists.eu/archive/Issue025.html>
- **Immutable objects are easier to work with**
 - **Helps discover bugs**
 - **Synchronization is easier**
- **Garbage collector copes well with short-lived objects**
- **A class with descriptive long names**

```
public class SessionConnectorWithRetryAtLeastThreeTimes {  
    private String connectionNameReceivedFromInternet;  
    private int numberOfTimesThatWeShouldRetryAtLeast;  
}
```

Add a Constructor

```
public class SessionConnectorWithRetryAtLeastThreeTimes {  
    private String connectionNameReceivedFromInternet;  
    private int numberOfTimesThatWeShouldRetryAtLeast;  
    public SessionConnectorWithRetryAtLeastThreeTimes(  
        String c, int n) {  
        connectionNameReceivedFromInternet = c;  
        numberOfTimesThatWeShouldRetryAtLeast = n;  
    }  
}
```

- **Problem – we need to read the comments to know what c and n are**

Use the Classic “this.” Assignment

- **It compiles and runs, but one field is not initialised**

```
public class SessionConnectorWithRetryAtLeastThreeTimes
{ private String connectionNameReceivedFromInternet;
  private int numberOfTimesThatWeShouldRetryAtLeast;
  public SessionConnectorWithRetryAtLeastThreeTimes(
    String connectionNameReceivedFromInternet,
    int numberOfTimesThatWeShouldRetryAtLeast) {
    this.connectionNameReceivedFromInternet =
      connectionNameReceivedFromInternet;
    this.numberOfTimesThatWeShouldRetryAtLeast =
      numberOfTimesThatWeShouldRetryAtLeast;
  }
}
```

Make Fields Final

- **Making them final shows the problem:**
 - Parameter `connectionNameReoeivedFromInternet`
- **So, make all fields as private and final as possible**
- **Search for non-final fields using a Doclet**
 - Not published, but easy to write
 - In our example, 644 fields were non-final
- **Again, fix either one class at a time, or use a tool**
 - Quick demonstration with IntelliJ – by hand takes longer
 - We now have 380 non-final fields left

How Final is “final”?

- **Sun Microsystems ambivalent:**
 - **JDK 1.1:**
 - Access control (private, etc.) not checked at runtime
 - Final fields cannot be rebound at runtime
 - **JDK 1.2:**
 - Access control checked at runtime, `setAccessible(true)` overrides
 - Final fields could be rebound at runtime with reflection
 - **JDK 1.3 + 1.4:**
 - Final fields cannot be rebound at runtime
 - **JDK 1.5 + 1.6:**
 - Final fields can be rebound at runtime with reflection
 - Except when primitive or String fields are set at declaration time

Java Versions: When “final” Was Final

- **Java versions and lifespans**

Version	Code Name	Release Date	Lifespan (months)	Final is final
JDK 1.1.4	Sparkler	1997-09-12	15	Yes
J2SE 1.2	Playground	1998-12-04	18	No
J2SE 1.3	Kestrel	2000-05-08	21	Yes
J2SE 1.4	Merlin	2002-02-13	31	Yes
J2SE 5.0	Tiger	2004-09-29	18	No

- **Suggestion: Treat final as if it really was ...**

- <http://www.javaspecialists.eu/archive/Issue096.html>

Dead Code

- **Many times I have fixed bugs in dead code**
- **Dead code should be pruned**
 - 1. Make elements as private as possible**
 - 2. Make fields final**
 - 3. Search for dead code and delete**
 - 4. GOTO 1**

After Pruning Dead Code

- **Rerun the doclets:**
 - **89 classes (down by 8)**
 - **16879 LOC (down by 2599)**
 - **79 non-private fields (down by 25)**
 - **324 non-final fields (down by 56)**

Back to Comments

- **Strip out useless comments and commented-out code**
 - **Source Control System is doing source control**
 - **Don't duplicate effort!**
 - **Root of problem is fear**
- **If commented code looks useful, leave a note**
 - **E.g. `// CodeComment removed`**
 - **Coder can look in source control system for CodeComment**
- **Our system now has 14505 LOC**
 - **Originally 19478 – reduced by over 25%**

Depth of Inheritance Hierarchy

- **Complexity of code can be related to hierarchy depth**
- **Overly deep hierarchies should be avoided**
- **You can check the depth with this simple tool**
 - <http://www.javaspecialists.eu/archive/Issue121.html>
- **Try beat our record:**
 - **Proprietary code: hierarchy depth of 10**
 - **Open Source: Hierarchy depth of 12**
 - **Rob Mulcahey, Current Inc, Colorado Springs**
 - **`org.apache.batik.dom.svg.SVGOMAltGlyphElement`**

Exception Handling

- **Quick manual inspection for bad exception handling**
- **Methods should not throw “Exception”**

```
private void initGui() throws Exception {  
    initNorth();  
    tabbedPane = new JTabbedPane();  
    getContentPane().add(tabbedPane,  
        BorderLayout.CENTER);  
}
```
- **And the catch blocks should not be empty**

Never Catch RuntimeException

- **Code should not catch RuntimeException**

```
try {  
    data = FruitspecTableModel.getColumnData(i);  
} catch (RuntimeException e) {  
}
```

- **Replace that with a check on the value of “i”**

- **Implies not catching Exception**

```
try {  
    data = FruitspecTableModel.getColumnData(i);  
} catch (Exception e) {  
}
```

Sloppy Exception Handling

- **Can cause parts of system to stop working**
 - Gives user false sense of security
- **All exceptions need to be noted**
 - Either logged to a file or the help desk
- **With Java 5 you can specify global exception handler**
 - <http://www.javaspecialists.eu/archive/Issue089.html>
 - **Nice, but does not solve the “poor coding” of empty catch blocks**

Global Exception Handling

```
public class DefaultExceptionHandler implements
    Thread.UncaughtExceptionHandler {
    public void uncaughtException(Thread t, Throwable e) {
        // You need more robust, permanent record of problems
        JOptionPane.showMessageDialog(findActiveFrame(),
            e.toString(), "Exception Occurred",
            JOptionPane.OK_OPTION);
        e.printStackTrace();
    }
    private Frame findActiveFrame() {
        for (Frame frame : JFrame.getFrames()) {
            if (frame.isVisible()) return frame;
        }
        return null;
    }
}
```


Register with Class Thread

- **Thread.setDefaultUncaughtExceptionHandler()**

```
public class EvenBetterGui {  
    public static void main(String[] args) {  
        Thread.setDefaultUncaughtExceptionHandler(  
            new DefaultExceptionHandler());  
        Gui gui = new Gui();  
        gui.pack();  
        gui.setDefaultCloseOperation(  
            JFrame.EXIT_ON_CLOSE);  
        gui.setVisible(true);  
    }  
}
```

Now Code is More Manageable

- **Now the real work starts:**
 - Find and eliminate duplicate code
 - Encapsulate fields that are still non-private
 - Set up test environment
- **From here, you must tread carefully**
 - Make sure you can roll back easily
 - Check frequently that code still works

Automatic Tools and Reflection

- **Java tools rely on static compilation of classes**
- **Be careful when using Reflection and Dynamic Proxies**

Check your code

- **Regularly check your own work:**
 - **Elements are properly commented**
 - **Exceptions are handled correctly**
 - **Fields are private**
 - **Fields are final where possible**
 - **Unit tests cover your code base**
 - **Look for copy & paste code**
 - **Sometimes difficult to eliminate**

Develop with Pleasure!

- **Make your code a pleasure to work with**
- **And don't be scared of fixing messy code**

Productive Coder

Dr Heinz M. Kabutz

The Java Specialists' Newsletter

<http://www.javaspecialists.eu>



JavaSpecialists.EU

Some Keystroke Hints

- **The appendix contains some hints on keyboard shortcuts in Eclipse and IntelliJ**

Eclipse



- **Create new class: Alt+Shift+N, C**
- **Autocompletion on keywords?**
 - Type “in” followed by CTRL+Space ...
 - Reaching Esc is awkward on the keyboard
 - My fingers have to leave the safety of the dimples
- **Error or unknown symbols in Eclipse – press Ctrl+1**
- **How do I get back to the previous file without using the mouse?**
 - Alt+left and Alt+right

Autogenerating Java Code

- **Make constructor: Alt+S, A**
 - **Parameters not in same order as fields**
 - **Though this may be a setting somewhere**
 - **Enter does not work**
 - **Names of parameters not the same as the fields**
- **Getters / Setters: Alt+S, R**
 - **Again, not in same order as fields & enter does not work**
- **Main method: main Ctrl+Space**
- **Ctrl + Shift + Space shows parameters**

Eclipse Magic Keys

- **Ctrl+Space autocompletes**
 - “syso” generates: `System.out.println();`
 - “for” generates: `for (int i = 0; i < args.length; i++) { }`
 - Problem is that Ctrl+Space is awkward to type
- **Ctrl+1 autofixes code**
 - But cursor jumps all over the place ☹
- **An IDE needs to be like a chef’s knife, sharp and true**

IntelliJ **IDEA** 5.0

Develop with pleasure!

IntelliJ IDEA

- **Create new class: In project window (Alt+1)
Alt+Insert**
- **Autocompletion on keywords?**
 - Works a bit better...
 - Type “in” followed by CTRL+Space ...
- **Error or unknown symbols in IntelliJ – press
Ctrl+Enter**
 - F2 finds the next problem
- **How do I get back to the previous file
without using the mouse?**
 - Alt+Ctrl+left and Alt+Ctrl+right

Autogenerating Java Code

- **Make constructor: Alt+Insert**
 - Parameters same order as fields
 - Names of parameters same as the fields
- **Getters / Setters: Alt+Insert**
 - It does what I expect
- **equals() & hashCode(): Alt+Insert**
 - Enter does not work that well
- **Ctrl+plus and Ctrl+minus folds & unfolds methods**
- **Main method: psvm Tab**

IntelliJ Magic Keys

- **Ctrl + Shift + Space is intelligent autocomplete**
 - Extremely useful
- **Tab fills in Live Templates**
 - “sout” generates: `System.out.println();`
 - “itar” generates: `for (int i = 0; i < args.length; i++) { }`
 - Problem is that **Ctrl+Space** is awkward to type
- **Alt+Enter autofixes code**
 - Cursor stays in the same place 😊
- **Ctrl+W selects wider and wider scope**

Style and Metrics Tools

- **MetricsReloaded (IntelliJ IDEA Plugin)**
 - <http://www.sixthandredriver.com/metricsreloaded.html>
- **Together Control Center**
- **CheckStyle**
 - <http://checkstyle.sourceforge.net>
- **FindBugs**
 - <http://findbugs.sourceforge.net>
- **Java PathFinder (from NASA)**
 - <http://javapathfinder.sourceforge.net>
- **Project Mess Detector (PMD)**
 - <http://pmd.sourceforge.net/>

Questions?

Heinz Kabutz

heinz@javaspecialists.eu

The Java Specialists' Newsletter

<http://www.javaspecialists.eu>

