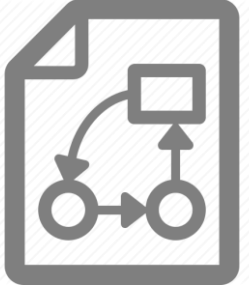
Bachelor thesis Modular Exceptions

Supervised by Prof. Dr. Oscar Nierstrasz

Motivation

- Past researches have shown that many exceptions are handled in similar ways
- However exception handling is often not written in a reusable way
- Goal: Find a way to add exception handling in a modular way

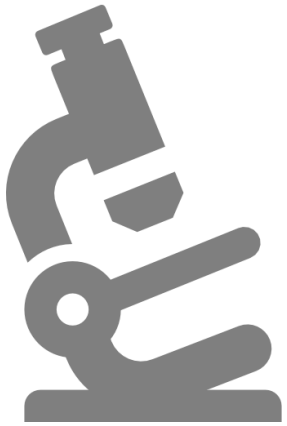
Our approach



- Research exception handling to find patterns
- Create a list of requirements
- Test different approaches in Smalltalk
- Test different approaches in Java
- Pick the best approach
- Create a final implementation in Java

Researching exception handling

- Analyzed two research papers
- Our own Research in Smalltalk
  - Looked at many methods that had a try-catch block



“A Field Study in Java and .NET”

- Analyzed exception handling in Java libraries, servers, server-apps and stand-alone software
- Java exceptions are most commonly handled by…
  - … logging them
  - … re-throwing them
  - … returning to the caller
- Proves that patterns exist

“On the Evolution of Exception Usage in Java Projects”

- Researched the usage of custom exceptions over standard exceptions in Java
- Standard exceptions with description text are used the most
- Proves that our solution must be able to primarily handle standard exceptions

Our own Research

- Analyzed 163 methods in Smalltalk
- Found commonly used handlers
  - Canceling the method
  - Returning a default value
  - Resuming the method execution
- Found that exception handling is often copy-pasted

Our requirements 1/2

- Modular exceptions must be modular
  - Must be easy to add
  - Must be compatible across methods/classes
- Must handle exceptions in the most common ways
  - Logging the exception
  - Re-throwing the exception
  - Returning to the caller (with a default value)
  - Resuming the method

Our requirements 2/2

- Must not be error prone
  - Inserted code must be checked by the compiler
  - Exceptions thrown by our code must be debuggable
  - Should never crash or corrupt the editor

Smalltalk Prototypes

- Tested three approaches
  - Dynamically rewriting method code
    - Smalltalk allows method code to be rewritten and recompiled at runtime
  - Wrapper objects
    - Methods in Smalltalk are saved as objects in the method dictionary of the class
  - MetaLinks
    - MetaLinks dynamically inserts code around method calls
- Wrapper objects were the best approach

Dynamically rewriting method code

- Idea: Write try-catch blocks into source code of method
- Created helper methods that insert code into a method's definition
- Problems:
  - Cannot check inserted code with the compiler
  - Cannot undo mistakes
  - May not be compatible across classes

Wrapper objects

- Idea: Wrap method definition object into our wrapper object
- Created different wrappers for each way to handle exceptions
- Worked very well, no problems

MetaLinks

- Idea: Use MetaLinks to wrap methods into a try-catch block
- Worked like wrapper objects but more complicated
- More Problems:
  - Exceptions thrown from within MetaLinks crashed the editor

Java prototypes

- Tested two approaches
- Byte code transformation
  - Can rewrite the code of methods dynamically
- Aspects with AspectJ
  - Allows us to dynamically insert method calls
- Found aspects to be the best solution

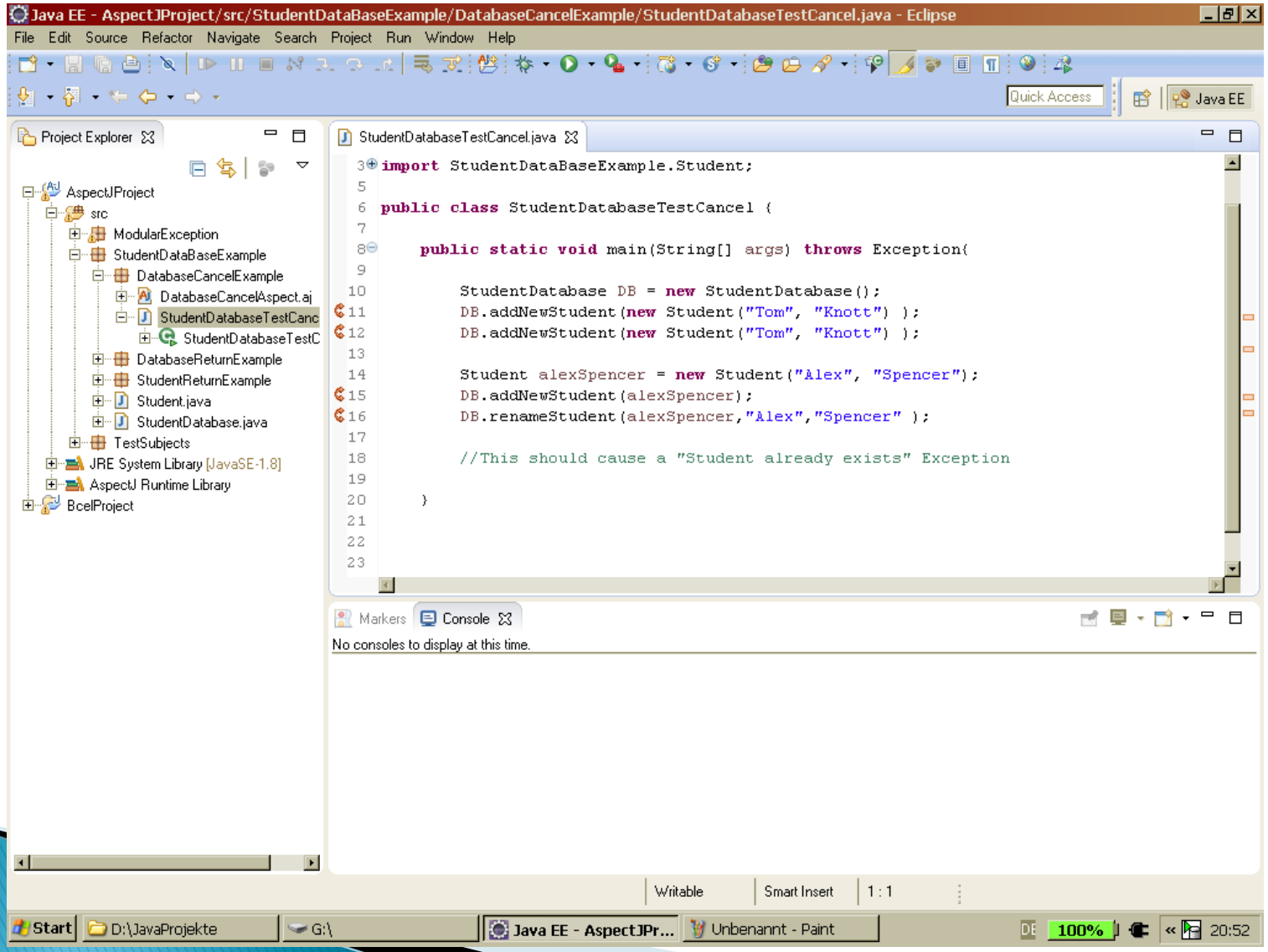
Byte code transformation

- Idea: Dynamically rewrite byte code to insert try-catch blocks
- Used BCEL library from Apache
- Rewrote example projects to test approach
- Same problem as dynamically rewriting source code

AspectJ

- Idea: Use aspects to wrap methods into try-catch blocks
- Created an example project and used aspects to handle its exceptions
- Worked very well
  - Flexible
  - Stable
  - Easy to understand

AspectJ


Java EE - AspectJProject/src/StudentDataBaseExample/DatabaseCancelExample/StudentDatabaseTestCancel.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Project Explorer
AspectJProject
src
ModularException
StudentDataBaseExample
DatabaseCancelExample
DatabaseCancelAspect.aj
StudentDatabaseTestCanc
StudentDatabaseTestC
DatabaseReturnExample
StudentReturnExample
Student.java
StudentDatabase.java
TestSubjects
JRE System Library [JavaSE-1.8]
AspectJ Runtime Library
BcelProject
StudentDatabaseTestCancel.java
3 import StudentDataBaseExample.Student;
5
6 public class StudentDatabaseTestCancel {
7
8 public static void main(String[] args) throws Exception{
9
10 StudentDatabase DB = new StudentDatabase();
11 DB.addNewStudent(new Student("Tom", "Knott"));
12 DB.addNewStudent(new Student("Tom", "Knott"));
13
14 Student alexSpencer = new Student("Alex", "Spencer");
15 DB.addNewStudent(alexSpencer);
16 DB.renameStudent(alexSpencer,"Alex","Spencer");
17
18 //This should cause a "Student already exists" Exception
19
20 }
21
22
23
Markers
Console
No consoles to display at this time.
Writable
Smart Insert
1 : 1
Start
D:\JavaProjekte
G:\
Java EE - AspectJPr...
Unbenannt - Paint
100%
20:52

Final Implementation

- Has an example project that showcases how it works
- Can dynamically wrap methods into try-catch block
- Can dynamically insert handler code into existing catch blocks
- Has templates that can be easily copy pasted across projects
- All dynamic changes are signalized to the user

The End