UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

tesi di laurea magistrale

A Fault Injection Tool For Java Software Applications

Anno Accademico 2012-2013

Relatore
Ch.mo Prof. Domenico Cotroneo

Relatore
Dr. Roberto Natella

Correlatore
Dr. Santonu Sarkar

Candidato
Antonio Ken Iannillo
matr. M63000242

to the True Love

“A knowledge of the existence of something we cannot penetrate, of the manifestations of the profoundest reason and the most radiant beauty, which are only accessible to our reason in their most elementary forms,it is this knowledge and this emotion that constitute the truly religious attitude„

“I maintain that the cosmic religious feeling is the strongest and noblest motive for scientific research„

Albert Einstein, The World As I See It

Table of Contents

List of Figures

Acknowledgement

I would like to remember all those who have helped me in the writing of the thesis with suggestions, criticisms and observations: they deserve my gratitude, though I shall be responsible for any errors contained in this thesis.

Foremost, I would like to thank my supervisor Prof. Domenico Cotroneo for believing in me with the chance to experience such a worthy conclusion for my university student's years. His patience, motivation, enthusiasm, and immense knowledge have been a great support.

I would like to express my immense thankfulness to the other supervisor of the thesis, Dr. Roberto Natella. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master degree thesis.

I would also like to thank the co-supervisor, Dr. Santonu Sarkar. He made my summer and turn the first work experience of an Italian boy, in perfect fuel for my whole life. He showed me a work environment of innovation, harmony and much more while he gave me a practical support and purpose for this thesis.

Besides my supervisors, I thank Dr. Rajeshwari Ganesan, Arpan Roy and Kajari Ghoshdastidar, for their support in the Infosys Labs in Bangalore, India; Manuel De Luca and Agostino Savignano, my colleagues at university and during the Indian internship, and all my new friends from the internship, for the stimulating discussions, for the endless days we were working together before deadlines; Prof. Stefano Russo, Prof. Marcello Cinque, Dr. Antonio Pecchia and Dr. Flavio Frattini, together with Prof. Cotroneo and Dr. Natella, for helping to make this experience possible.

Special thanks go to my university colleagues and mates, especially to Daniele, who showed me, from the very first moment of my whole university experience, uninterrupted social support; to the companions of our bizarre, funny, unbelievable, off-site lives, for all the times we forget about problems and go on having moments in *Casa Pastore*, *Casa Scaperrotta* and *Casa Pecoraro-Della Ragione*; to my brothers and sisters in Avellino, for making me feel always welcomed and for all the reasons of growth we come up against; to my dearest and oldest friends Fabio, Giacomo and Antonio, for your sincere and gratifying friendship.

In particular, I am grateful to have met a woman so strong and brave to be able to see always, in a way or another, what is right. Elisa makes my life enchanted and she always encouraged me to move forward.

Last but not the least, I would like to thank my family: my relatives, for the feeling that only a genuine Italian family can give; my brother Stefano Kenji, for all the times he could not stand me and I could not stand him but, mainly, for another eye on the world with a point of view complementary to mine; my parents Anna and Gino, for giving birth to me at the first place and supporting me throughout my life. They give me the possibility and the opportunity to pursue my dreams.

Introduction

Nowadays technology is everywhere and software plays a central role. With the advent of new computing methods, and with new business demands, software is being developed and deployed for diverse scenarios. The complexity of the application has increased by an order of magnitude. The applications run in new computing platforms such as virtualized platforms or in hand-held devices. The applications are offered to people as a 24x7 service. With the growing complexity, software is becoming more prone to faults, many of which are extremely hard to reproduce in a traditional testing environment. It is becoming more common that all the software defects do exist but no one knows exactly where they are, when they will reveal themselves and what could be the possible consequences of their activation. It is practically impossible to guarantee that software is defect-free, due to many complex requirements to fulfil and to budget and time constraints that limit testing activities. Moreover, Dijkstra's thesis [1] demonstrates that testing cannot guarantee the absence of errors, but it can only show their presence. Thus, the defect-free software does not exist. What we can do it is to enhance our software with Fault Tolerant Mechanisms and Algorithms (FTMAs).

Fault-tolerance or graceful degradation is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components and the FTMAs enable a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. These algorithms detect that it has made an error, then they take the system state at that time and corrects it, to be able to move forward, or revert the system state back to some earlier, correct version and moves forward from there. Fault-tolerance also impact on availability of a system, i.e. the proportion of time a system is in a functioning condition. FTMAs ensure a high availability of the system. Fault Injection is the process of deliberately introducing faults into a system to assess its behaviour in the presence of faults. The FTMAs should make the system as robust as possible and Fault Injection can validate and improve them.

The objective of this thesis is to show the development of a Fault Injector for Java programs. Java programs run on a virtual processor, called Java Virtual Machine (JVM), which executes a particular intermediate code, called Java bytecode. The tool emulates bugs, i.e. human mistakes occurring during development; and external faults, i.e. exceptional condition of external systems such as a database disconnection. They can be emulated by corrupting a program. Since recently all the systems contain Components Off-The-Shelf (COTS), source code is not always available and, then, the tool works on the Java bytecode. Moreover, it's reasonable to think that a practitioner would like to emulate more bugs related to third-party code than others. The thesis also includes some utilities, designed for a better instrumentation of the Java bytecode, that have been used for the implementation of the tool.

A brief history of the most common injection tools introduces the approach used in this work in chapter 1; the design and some implementation notes for the Java Fault Injection prototype are in chapters 2 and 3; chapter 4 contains some experiments on a demo project from the java enterprise edition tutorial with the aim of showing some capabilities of the tools. Finally, there is a little conclusion that includes some future developments.

Chapter 1

Software Fault Injection

1.1 Basic Concepts

According to Avizienis et al. [2], the dependability of a system is *the ability to avoid service failures that are more frequent and severe than is acceptable.* This definition is based on the concepts of service and failure, so on the following definitions:

service delivered by a system, is its behaviour as it is perceived by its users, which could also be other systems.

behaviour is what the system does to implement its functions and it is described by a sequence of states;

total state is the set of the following states: computation, communication, stored information, interconnection, and physical condition.

Thus, once dependability is defined, threats to dependability can be defined as well:

failure is the event that occurs when the delivered service deviates from correct service;

error is the part of the total state of the system that may lead to its subsequent failure (error could be sees as a deviation from the correct state);

fault is the adjudged or hypothesized cause of an error.

Figure 1.1 explains how error propagates among components, while Figure 1.2 show the fault-error-failure chain from a generic point of view.

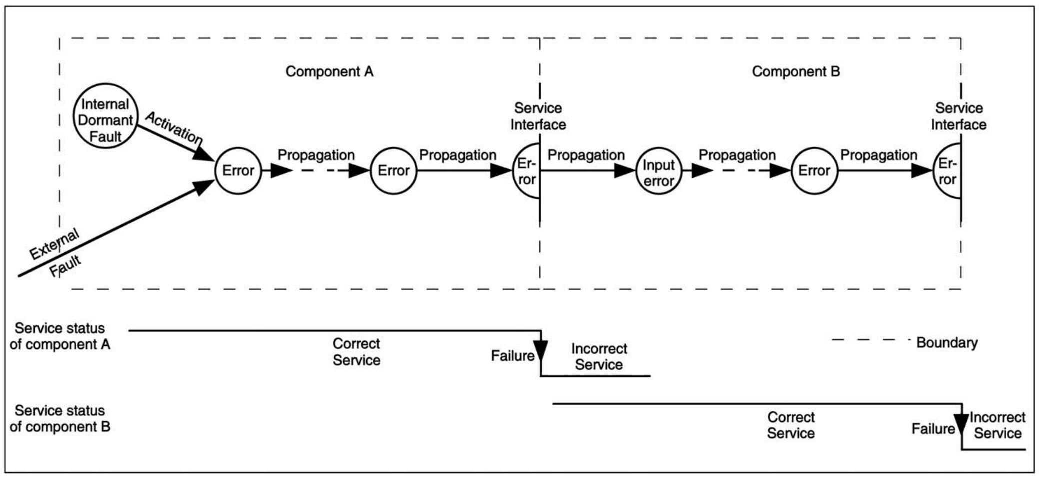


Figure 1.1: Detailed View of Error Propagation

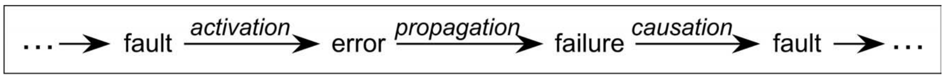


Figure 1.2: The fundamental chain of dependability threats

Figure 1.3 shows the classification of faults according to eight basic viewpoints, i.e. phase of creation (development or operational), system boundaries (internal or external), phenomenological cause (human-made or natural), dimension (hardware or software), objective (malicious or non-malicious), intent (deliberate or non-deliberate), capability (accidental or incompetence) and persistence (persistent or transient). Of particular interest is the difference between internal and external fault. The first originates inside the system boundary, the latter originates outside and propagates errors into the system by interaction or interference. Moreover, a permanent fault is the one of which presence is assumed to be continuous in time and a transient fault has a presence bounded in time.

This thesis considers faults of software components that belong to a system, and they are all covered by the proposed tool. These failures can be caused by:

- *bugs* inside the code of a particular component, sometimes called *software defects* (these are persistent faults activated by particular sequence of inputs and use condition of the component, 1-4 in Figure 1.3);

- *external faults* generated outside the component and that induce the failure of the component (they include the unavailability or exhaustion of physical resources such as memory, connectivity etc., or they include delays or interaction errors with other components such as operating system, database, remote system etc.. They typically are transient, 24-31 in Figure 1.3).

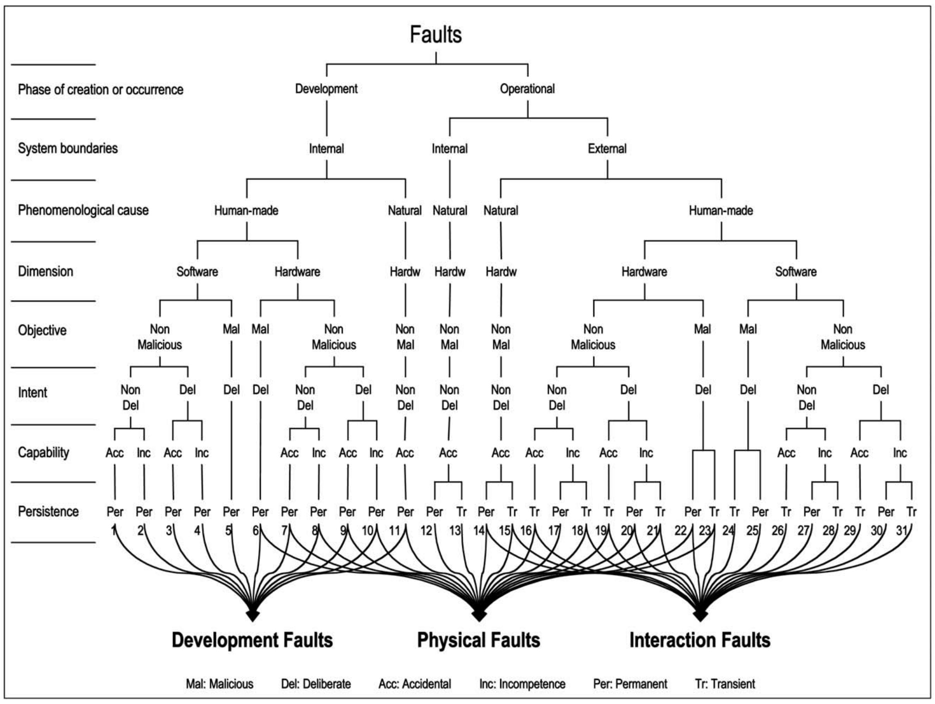


Figura 1.3: Faults Classification

Software faults are unavoidable. No one can prove the correctness of a program[1] and software engineers usually design some mechanisms for the fault tolerance, such as assertions, redundant logics and rollbacks. Those Fault Tolerant Algorithms and Mechanisms (FTAM) give the confidence that the system will be able to deliver a proper service and can be both evaluated and improved by a campaign of fault

[1]Dijkstra says test can't guarantee the absence of errors, but it can only show their presence. The error-free software does not exist.

injections experiments. The software fault injection, thus, reveals and improve the ability of the software to handle faults.

1.2 State of Art

As long as software becomes more complex, the number and the complexity of software functionalities also tend to increase validating the hypothesis that software faults are the most frequent source of system outages. Fault injection is a practical approach for achieving the confidence that software cannot cause serious service failures [2]. Fault injection approaches work in this direction, deliberately introducing faults in a system in order to figure out its behaviour in presence of faults and, thus, to measure and to improve the efficiency of error detection and recovery mechanisms. With the growing complexity, software applications are getting more prone to faults, many of which are extremely hard to reproduce in a traditional testing environment. Thus, the focus of researchers shifted towards the assessment of fault-tolerant systems with respect to software faults. In his thesis, Natella [3] studied and classified all the research works in this field: many techniques and tools have been developed in more than 20 years. Summarizing, three classes can be identified.

Data Error Injection This kind of injection corrupts the memory in a similar way to hardware fault, aiming to reflect the **effects** of software faults. Some tools of this class are: FIAT[4], FERRARI[5], DOCTOR[6], Xception[7].

Interface Error Injection It is a technique that corrupts the input values provided to a target software component, or the output values that the target provides. It aims to emulate the **effects** produced by faults outside the target. Some tools of this class are: Fuzz[8], RIDDLE[9], Ballista[10], MAFALDA[11].

Code Changes Injection The last category of injection tools corrupts directly the program code, changing its semantic. The purpose is to emulate **actual**

software faults in the target component of the system. Some tools of this class are: FAUST[12], FINE[13], DEFINE[14], G-SWFIT[15].

1.3 Objective of the Thesis

The goal of this work is to build a fault injector for Java programs. Java is the foundation for virtually every type of networked application and is the global standard for developing and delivering mobile applications, games, Web-based content, and enterprise software. All of them in several environments, with more than 3 billion devices running Java. It's then obvious how Java software fault injection could be useful, giving the opportunity to improve the dependabily of actual systems. This tool can inject various types of faults into a java software and assist software engineers to analyse the impact of such faults on the runtime behaviour of the application.

The tool gets as input the code of a Java software component (it could be stored as JAR, WAR or EAR, i.e. the archive files for all kind of Java software). The tool analyses the input and extract all the information needed to profile the application and then create a copy of the archive file with a fault inside. As mentioned before, two kind of fault can be emulated: internal and external. Internal faults represent the residual software faults, those that eluded all software test with the unawareness of where they are, when they will reveal, what could be the consequences. On the other hand, external faults are caused by the failure of an external system. Service failure of a system causes a permanent or transient external fault for the other system(s) that receive service from the given system.

The injection of code changes for emulating the effects of real software faults is based on the empirical observation that code changes produce errors and failures that are similar to the ones produced by real software faults [16, 17, 18]. This is the reason why this approach has been chosen for the tool.

The faults are injected at the bytecode level. The Java bytecode is the binary executable on a Java Virtual Machine (JVM) [19]. It is not necessary to have source code in order to inject faults and, thus, they can be injected in third party library or *Component Off The Shelf* (COTS). The program translated in Java bytecode is enclosed in one or more **.class** files, each one that contains instruction codes as well as all the necessary information for the execution of the same in the form of constants' tables. The tool works with the proper parts of this file in order to inject the code changes. These code changes are consistent with the Java language such as it worked with the source code.

What is actually injected is called fault load, i.e. rules and operators for the simulation of the software faults. The fault load is split into two parts according to what to emulate. Software defects are emulated reproducing those kinds of error that programmers usually make, taking this information from some works such as Duraes and Madeira [15] and Basso et al. [20]. The external faults, instead, are reproduced injecting, with the technique of **dead code**[2] to modify the persistence, errors and exceptions of the same kind of those the code could actually throws.

The next sub-sections explain in details these two complementary parts of the fault load.

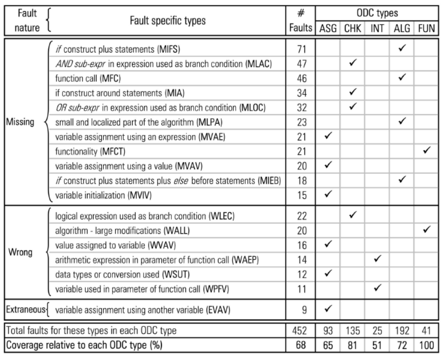
1.3.1 Injection of Software Defects

Residual software faults are faults that eluded all the software tests, with the unawareness of where they are, when they will reveal, what could be the consequences. In order to emulate them, they should be first classified. Duraes and Madeira [15] suggested an extension of the Orthogonal Defect Classification (ODC) [3] for analysing the faults from the point of view of the program context in which they occurs and for relating them with programming language constructs to be changed for fault injection purposes. They also made a field data study on several open source C programs, obtaining that only few kinds of fault are actually representative of

[2]the injection of dead code consists in surrounding the target code with an if-then-else construct; the program will execute faulty code when instructed to do so (the "if" branch), according to a condition specified by the tester; otherwise (the "else" branch), the faulty code will behave as "dead code", and the software component will act normally.

[3]Orthogonal Defect Classification (ODC) turns semantic information in the software defect stream into a measurement on the process. The ideas were developed in the late '80s and early '90s at IBM Research by Ram Chillarege**(author?)** [21].

residual faults. The most common defect types can be seen in Figure 1.4 and each of these matches with an original ODC class, i.e. *Assignment, Checking, Interface, Timing&Serialization, Algorithm, Function.* A further consideration is that there are no residual faults belonging to the timing&serialization category. Thus they proposed a new software fault injection technique, G-SWFIT, based on emulation operators derived from the field study. This technique consists of finding key programming structures at the machine code-level where high-level software faults can be emulated.



Fault nature	Fault specific types	# Faults	ODC types				
			ASG	CHK	INT	ALG	FUN
Missing	*if* construct plus statements (MIFS)	71				✓	
	AND sub-expr in expression used as branch condition (MLAC)	47		✓			
	function call (MFC)	46				✓	
	if construct around statements (MIA)	34		✓			
	OR sub-expr in expression used as branch condition (MLOC)	32		✓			
	small and localized part of the algorithm (MLPA)	23				✓	
	variable assignment using an expression (MVAE)	21	✓				
	functionality (MFCT)	21					✓
	variable assignment using a value (MVAV)	20	✓				
	if construct plus statements plus *else* before statements (MIEB)	18				✓	
	variable initialization (MVIV)	15	✓				
Wrong	logical expression used as branch condition (WLEC)	22		✓			
	algorithm - large modifications (WALL)	20					✓
	value assigned to variable (WVAV)	16	✓				
	arithmetic expression in parameter of function call (WAEP)	14			✓		
	data types or conversion used (WSUT)	12	✓				
	variable used in parameter of function call (WPFV)	11			✓		
Extraneous	variable assignment using another variable (EVAV)	9	✓				
Total faults for these types in each ODC type		452	93	135	25	192	41
Coverage relative to each ODC type (%)		68	65	81	51	72	100

Figura 1.4: Fault Coverage of the Representive Faults Types

To use these results about java programs, it should be noted that there may be differences with the C language from the point of view of residual errors. Basso et al. [20] performed in their work a similar field study on Java open source programs. Despite some different frequencies, the mistakes made by programmers are common to both C and Java. New fault types were found due to the Java language specific characteristics, but they together account for only 7%, and less than 1% individually, of the total faults.

Thanks to all this information, the software defect injection, used in this work, is based on the G-SWFIT of Duraes and Madeira designing new mutation operators in order to bridge the gap between the procedural paradigm of C programming and the object-oriented paradigm of Java.

1.3.2 Injection of Exceptions

Java, as most of the modern languages, support explicitly the mechanism known as **exception handling**. When a semantic constraint is violated or when some exceptional error condition occurs, an exception is thrown. This causes a local transfer of control from the point where the exception occurred to a point, defined by the programmer, where the exception is caught. If not caught, or wrongly handled, the exception makes the system fail. System usually communicates with the external systems through APIs. When external systems fail, the external faults manifest them as an exception in the API calls. External software faults can, thus, be emulated triggering the exception at the code-level where the related API method is called.

Exceptions in Java are classified in:

- Checked Exception: these are exceptional conditions that a well-written application should anticipate and recover from. Checked exceptions are subject to the Catch or Specify Requirement, i.e. the programmer has no choice but to handle them during the development. Examples are: *java.io.FileNotFoundException*, *java.net.URISyntaxException*, *java.awt.print.PrinterAbortException* and all the subclasses of *java.lang.Exception* but *java.lang.RunTimeException*.

- Error: these are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from. For example, suppose that an application successfully opens a file for input, but is unable to read the file because of a hardware or system malfunction. The unsuccessful read will throw *java.io.IOError*. Other examples are *java.lang.VirtualMachineError* and all the subclasses of *java.lang.Error*.

- Runtime Exception: these are exceptional conditions that are internal to the application, and that the application usually cannot anticipate or recover from. These usually indicate programming bugs, such as logic errors or improper use of an API. For example, consider an application that passes a file name to the constructor for FileReader. If a logic error causes a null to be passed to the constructor, the constructor will throw *java.lang.NullPointerException.* Other examples are *BufferOverflowException*, *NoSuchElementException*, *ConcurrentModificationException* and all the subclasses of *java.lang.RunTimeExcetion.*

Checked Exceptions are defined in the API documentation or could be found inspecting the java class file and their hierarchy of parents. Unchecked exceptions, instead, should follow different criteria in order to decide properly which kind of instruction could throw it and which not, e.g. an *IOError* will only show up when we are doing some kind of IO operation. These considerations show how many possibilities there are and how many external faults can be emulated, but they also force to couple all the external methods of an application with the specific exceptions they can throw.

Chapter 2

Tool Design

2.1 Overview Architecture

The injection tool consists of three different projects designed according an object-oriented design style. An overview architecture is figure 2.1.

The smaller projects are related to the activation mechanism for the injected fault: **JFIRemoteClient** and **JFIRemoteController**.

The **JFIPrototype** project is the main project and cares about the injection process. The designed components of the tools are the following:

Façade allows to use all the other components' functionalities organizing the control flow and providing a simplified interface, in a similar way the façade design pattern provides a higher level view on the whole system;

Injection is interested in the injection mechanism, consisting of all the operators of the fault types;

JarManagement manages the archive files, such as *.jar, .war* and *.ear* files, to give the other components the possibility to work directly with *.class* files;

Profiling uses the technique from [22] to improve the representativeness and the efficiency of the injections selecting a smaller set of locations from the target set.

Util contains all the utility classes used to manipulate in a simpler way the bytecode inside a *.class* file.

All these components, and the remote activation projects, are explained in detail in the following subsections.

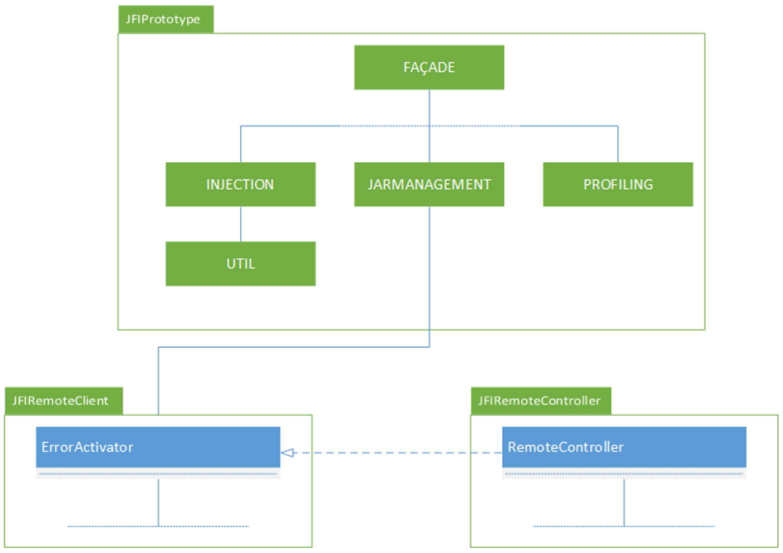


Figure 2.1: Overview Architecture

2.1.1 Façade Package

The façade package is the entry point of the tool and starts the graphic user interface. It is possible to run the tool without the GUI providing some arguments to the tool. So we have two controllers, as shown in figure 2.2, that are instanced according to the arguments of the main. They create the instances of classes of the other packages and use them to execute the work flow of the tool. The **GuiElements** contains a simple class Visualizer that cares about the instantiation of the GUI, represented by the other classes of the same sub-package, linking it to the *Controller.*

The GUI allows the user to:

1. load an archive file, which should be a *.jar/.war* file or a *.ear* file. This distinction is made because the *.jar/.war* files does not contain any other archive files, while a *.ear* file can consist of several *.jar* and *.war* files;

2. have a look at the methods that are candidates to contain a fault-prone location or are external methods, i.e. a method which implementation is not present in the target project (more details later, in subsection 2.1.4);

3. select some of the shown methods and the trigger mode between *remote*, forcing to use later the remote controller, or *limited*, the synchronous trigger mode as explained before but without using the remote controller, or *always*, to inject the fault already activated without a later possibility to deactivate it;

4. inject all the methods, or only the selected methods, with the faults that fit in them according to the fault load saving a faulty copy of the input file in the output folder;

5. select the output folder.

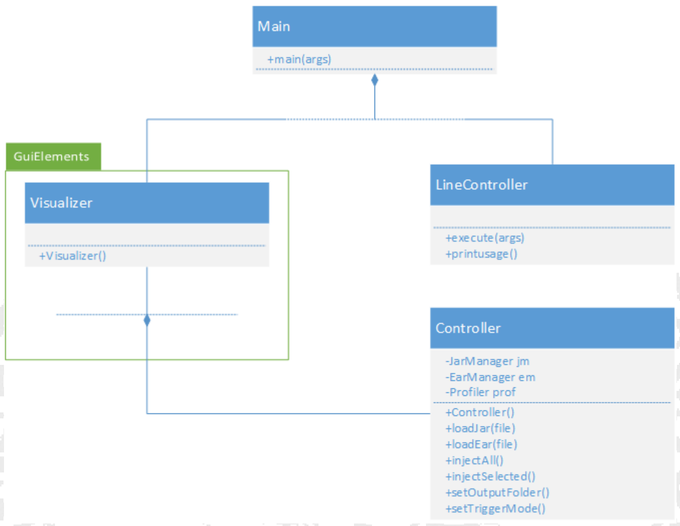


Figure 2.2: Façade Package Design

2.1.2 Injection Package

The process that really cares about getting a *.class* file, looks for matching location and applies the related change, is realized by the injection package. The *Injector* class is the one that is invoked by the controller. It has a *.class* file and a list of methods as inputs, and a list of classes injected with faults as output. According with the type of list the injector gets, it can use the software fault operators or the exception operators. They all extend the *AbstractOperator* class that enclose the common operation, leaving to the extended classes to implement the methods:

searchPattern it looks for those locations that match with the specification of the operator;

changeMethod it deletes, substitutes or adds method instruction, always according to the operator and the locations found by the search;

getName it returns a string with the name of the specific operator.

When the injector gets a methods list, it invokes sequentially all the software fault operators. If the list consists of external methods, the injector can already extract which exception should be injected and where (more details in subsections 2.1.4 and 2.1.5). The location corresponds with *invoke* instruction for the external method and the code to change consists of adding few lines. These lines introduce some dead code, as follows, where *invokeInstruction* is the original line of code:

```
1 if (JFI.RemoteClient.ErrorActivator.isActivated()){
2     throw new Error();
3 }else{
4     invoke();
5 }
```

Line 1 shows the point where the injected *.class* files from the *RemoteClient* project are used, forcing to switch the control to the error activator. The error activator will return true or false according to the activation of the fault. If the fault is activated, the application will throw an exception, as in line 2; if not, the application behaves normally because the only line that affects the application is line 4. This brief example can also show that every exception operator behaves in the same way, it is only needed to specify the exception. Thus, all the exception operators extends the *ErrorOperator* class that extends the *AbstractOperator*, implementing the abstract methods and introducing a new one (**specifyException**).

The package design is in figure 2.3 and the software fault operators detailed design is in section 2.3 (since the software fault operator are designed for java application, they are called *Java Operator* or *JO*).

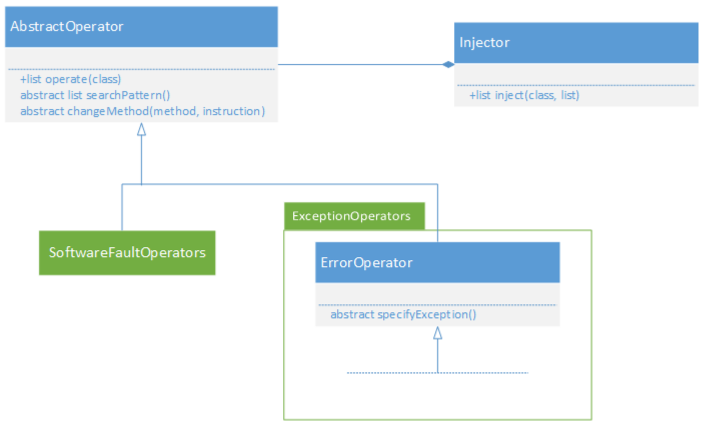

AbstractOperator
+list operate(class)
abstract list searchPattern()
abstract changeMethod(method, instruction)
Injector
+list inject(class, list)
SoftwareFaultOperators
ExceptionOperators
ErrorOperator
abstract specifyException()

Figure 2.3: Injection Package Design

2.1.3 JarManagement Package

The JarManagement package is responsible of managing the files the user provides or receives as results from the tool. It only consists of two class, as shown in figure 2.4: the *JarManager* and the *EarManager*. The last, because a *.ear* file contains *.jar* and *.war* files, can create and use more *JarManagers*. Both can be invoked externally and they have this properties:

loadXar loads the file in the tool;

getXarName returns the name of the file;

getKlassList opens the archive file and scan it returning all the *.class* files as a list;

saveFaultyXar creates a new archive file equal to the original with the only difference of the injected fault and, when necessary, the addiction of the *.class* files of the *RemoteClient* project;

getFaultyJar returns the faulty jar as a file (invoked by the *saveFaultyEar* method of the *EarManager*).

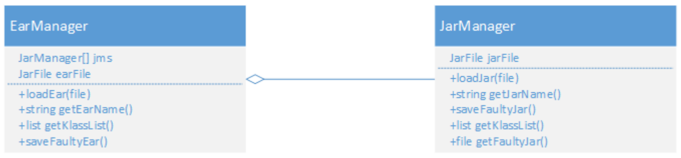


Figure 2.4: JarManagement Package Design

2.1.4 Profiling Package

Figure 2.5 shows the Profiling package. As stated before, there is a technique in **(author?)** [22] that improves:

representativeness is the ability of the fault load to represent the real faults that the system will experience during operation and it can be achieved by defining a realistic fault model;

efficiency refers to the numbers of experiment required to achieve relevant and useful results, considering that a low efficiency occurs when a huge number of experiment is needed in order to discover FTAM deficiencies.

The technique is based on classification algorithms that marks among a list of locations (in this work the methods) where the injection of fault is more representative, also reducing overhead while maintaining high efficiency. The simplest of the classification algorithms is the *K-Mean Clustering* and it uses two software metrics:

Lines Of Code the number of executable lines of code;

Fan Out the count of unique functions that are called by a given function, either directly, or ultimately, via other functions.

After the clustering, the cluster with the lower average *Fan Out* will contain the fault-prone methods. This technique is implemented by *Profiler* class with the aid of the class *Measurer,* which calculates the software metrics of all the methods in the target application. Then it uses the sub-package *clustering* to perform the classification and saves the fault prone methods list.

Another task of the *Profiler* is to scan the application for external method invocations and organize an external methods list. When it finds one, it try to create an *ExternalMethodItem*. The constructor of this class first checks if the external method is present in the *InclinationTable* (so it can be injected with an exception, details in subsection2.1.5) and then it will keep trace of all the invocation occurrences of the external method.

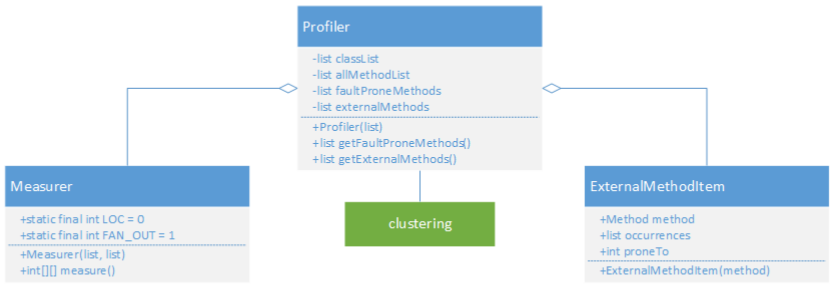


Figure 2.5: Profiling Package Design

2.1.5 Util Package

The Util package is represented by figure 2.6 and all its classes are used by the Injection package. These classes are utilities, they abstract some concepts to make the understanding and the manipulation of the bytecode easier. These concepts are peculiarity and they are not included in any third-party library, so they have to be included in the design of the tool.

The only exception is the inclination table. This is a static table that encodes some information useful for the exception injection. Every entry of the table consists of:

method name of an external method in dotted notation (e.g. *javax.persistence.Query.getResultList*);

inclination an integer that encodes the particular exception that the method can throw and for which exception it exists in the fault load a related operator.

This table has to be populated manually and, most likely, according to a previous analysis of the target application and to the exceptional situation.

The detailed design of the other utilities is placed in the next section.

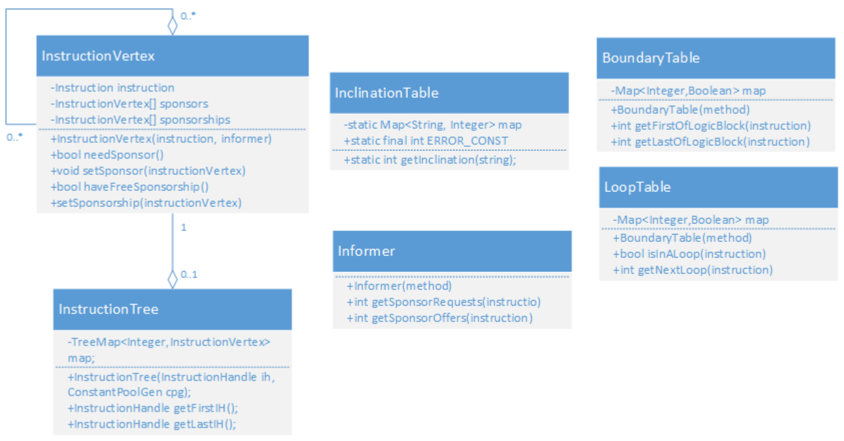

0..*
InstructionVertex
-Instruction instruction
-InstructionVertex[] sponsors
-InstructionVertex[] sponsorships
+InstructionVertex(instruction, informer)
+bool needSponsor()
+void setSponsor(instructionVertex)
+bool haveFreeSponsorship()
+setSponsorship(instructionVertex)
0..*
1
0..1
InstructionTree
-TreeMap<Integer, InstructionVertex> map;
+InstructionTree(InstructionHandle ih, ConstantPoolGen cpg);
+InstructionHandle getFirstIH();
+InstructionHandle getLastIH();
InclinationTable
-static Map<String, Integer> map
+static final int ERROR_CONST
+static int getInclination(string);
Informer
+Informer(method)
+int getSponsorRequests(instructio)
+int getSponsorOffers(instruction)
BoundaryTable
-Map<Integer,Boolean> map
+BoundaryTable(method)
+int getFirstOfLogicBlock(instruction)
+int getLastOfLogicBlock(instruction)
LoopTable
-Map<Integer,Boolean> map
+BoundaryTable(method)
+bool isInALoop(instruction)
+int getNextLoop(instruction)

Figure 2.6: Util Package Design

2.1.6 Remote Activator

As mentioned before, both permanent and transient fault can be emulated with the tool. The transient fault injection needs the remote activator, designed for the activation of an injected fault with a remote controller, external to the target. It consists of two projects.

1. The **JFIRemoteClient** contains the classes that will be included into the faulty version of the application and it activates or deactivates the fault according to the commands acquired by the remote controller;

2. The **JFIRemoteController** is an external application with a graphical user interface and the ability to communicate with the application.

The two projects reproduce the client-server paradigm, where the first acts as a client and the second as a server. Thus, *JFIRemoteClient* contacts first *JFIRemoteController* that answers according to the user input. The GUI of the controller gives the users the possibility to adopt two different fault activation modes:

Event-Triggered the user can choose how many times the target code should be executed before the fault is activated;

User-Triggered the user has the possibility to activate or deactivate the fault at any time using a switch.

The communication between these two first components is based on messages (in figure 2.1 this is represented by the dotted arrow), because they will run on different processes and should be considered as a distributed system. Instead, the relationship between the JFIRemoteClient and the JFIPrototype indicates that the latter uses the *.class* files produced by the compilation of the first and it will injects them into the target.

2.2 Utilities

2.2.1 Concepts and Definitions

Bytecode Instruction Every bytecode instruction works on a operand stack, allocated only for the method that contains it. Every bytecode instruction pushes and pops a compile-time well-defined number of values into and from the stack. Every Java instruction is translated into more than one bytecode instructions. They are sometimes called opcodes.

Basic Block Portion of the code within a program. A basic block has only one entry point and it has only one exit point. Every time the first instruction is executed, the rest of the instructions within the basic block are necessarily executed, in order, exactly once.

Functional Block A functional block is a set of consecutive bytecode instructions, within a basic block, that accomplishes the function required by a Java instruction. A Java instruction is translated into a functional block.

Instruction Tree An instruction tree is a data structure that abstract the concept of functional block. It is not a tree, but it is a direct acyclic graph *(DAG)* $G = (V, E)$, where:

- V is the set of instruction vertices, matching the single bytecode instruction;
- E is the set of the edges between the vertices.

There is an edge $(v, w) \in E$ if the instruction v pushes a value into the stack that the instruction w pops. We say that v is a sponsor for w or that w is sponsored by v. Further, w accepts a sponsor offer from v or v accomplish a sponsor request from w.

2.2.2 Boundary Table

The boundary table actualizes the abstraction of the basic blocks within a method. The entries of this table consists of:

key the bytecode instruction number, *i.e.* the position of the instruction in the method's bytecode;

value a boolean value that indicates if the bytecode instruction delineates the beginning or the end of a basic block.

In order to create the boundary table, the initialization algorithm scans the instruction list of the method and it adds:

- with value *false* the method's first instruction;
- with value *true* every jump-kind instruction, with value *false* the subsequent, with value *false* its target and with value *true* the previous of its target;
- with value *true* the method's last instruction.

Thus, once the boundary table is initialized, given an instruction of the method, it is possible to obtain the positions of the first and last instructions (boundaries) of the basic block where the instruction belongs to.

2.2.3 Loop Table

The loop table is a utility that tracks all the loops inside a method. A loop can be either a *for* construct or a *while/do-while* construct. These constructs are translated in Java Bytecode with *IfInstruction* opcodes that have a negative offset as argument. The entry of this table consists of:

key the bytecode instruction number (i.e., the position) of the first instruction that starts a loop;

value the bytecode instruction number (i.e., the position) of the last instruction that ends the same loop.

Thus, once the boundary table is initialized, given an instruction of the method, it is possible to know if that instruction belongs to a loop or not.

2.2.4 Informer

The Informer is a utility that gives information about what a bytecode instruction does. Actually, given a bytecode instruction, it can return both the number of values that instruction pushes and pops.

The only exceptions are the opcodes with the mnemonics *DUP2, DUP2_X1* and *DUP_X2* that care about words not values, i.e. the number of values pushed or popped depends if they represent integer, long or doubles and this information are available only with a further analysis of the bytecode instruction list.

2.2.5 Instruction Tree

During the injection process, the tool locates the bytecode instruction that matches with the search pattern of an operator. Most of the operators work directly on those bytecode instructions, while others work on the corresponding higher level instruction. A Java instruction is usually translated in more than one bytecode instruction, each of them for a particular atomic action of the high level statement. Thus, an operator designed to affect a Java instruction should be able to merge the bytecode instructions of the same functional block, *i.e.* the translated Java instruction in bytecode instructions. In order to enrich the tool with this ability, the following property has been considered:

> *P1.* When executing, the functional block starts with the operand stack in a certain state and ends with the operand stack in the same state.

The **operand stack** is a LIFO stack used to store arguments and return values of many of the virtual machine instructions[1]. A Java instruction, once executed, never leaves the operand stack with not consumed value and never expect some value in the operand stack from previous operations. It loads the operands onto the stack from the local variables or the constant pool and it saves its results (if any) into local variables or simply discard them. Thus, the functional block, that represents the whole Java instruction, acts the same. This is a run-time property but it holds also for static analysis with some considerations about the jump instructions. The *IfInstructions* are always located at the end of a functional blocks and create no problem at all because it consumes all the pending values on the operand stack. The instruction *GOTO*, instead, alters the control flow independently and can break the hypothesis of contiguous blocks in a static context. A possible workaround is to consider as next instruction of the *GOTO* the target of the same, also because the next instruction certainly belongs to another functional block. A generic example of functional block and its property is given in Figure 2.7.

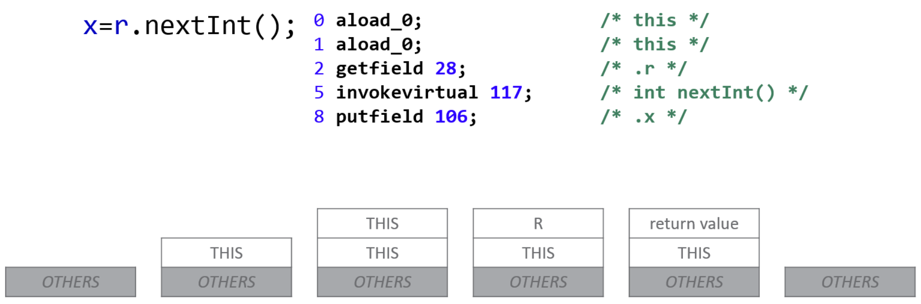

```
x=r.nextInt(); 0 aload_0;               /* this */
               1 aload_0;               /* this */
               2 getfield 28;           /* .r */
               5 invokevirtual 117;     /* int nextInt() */
               8 putfield 106;          /* .x */
```

Figure 2.7: Example of Functional Block and Its Property

The **instruction tree** abstracts the idea of functional block with a graph and the **swinging algorithm** uses the property *P1* to build the instruction tree from

[1]The Java virtual machine works on frames. A frame is used to store data and partial results. It is created each time a method is invoked and destroyed when its method invocation completes. Frames are allocated from the Java Virtual Machine stack of the thread creating the frame. Each frame has its own array of local variables, its own operand stack, and a reference to the run-time constant pool of the class of the current method. The Java Virtual Machine supplies instructions to load constants or values from local variables or fields onto the operand stack. Other Java Virtual Machine instructions take operands from the operand stack, operate on them, and push the result back onto the operand stack. The operand stack is also used to prepare parameters to be passed to methods and to receive method results. [19]

any of its bytecode instruction. This algorithm consists of three section:

Initialization it's executed once and it is the entry point of the algorithm. It:

1. creates two LIFO structures, one for the pending sponsor requests (*PSR*) and one for the pending sponsor offers (*PSO*);
2. considers instruction *i;*
3. adds *i* to the graph;
4. adds *i* to *PSR* as many times as it requires a sponsor and to *PSO* as many times as its sponsors' offers;
5. if *PSR* is not empty, go to backward step; else if *PSO* is not empty, go to Backward Step;

Backward Step

1. considers *i* as the first (lowest key) instruction among the instructions already considered;
2. adds the previous instruction (*i.prev*) to the graph;
3. for each sponsor offer of *i.prev* until PSR is not empty, creates an edge (*i.prev*, *PSR.pop()*);
4. for each sponsor offer left, adds *i.prev* to PSO;
5. for each sponsor request, adds *i.prev* to PSR;
6. considers *i.prev* as *i*;
7. if *PSR* is not empty, goes to Backward Step.2;
8. if PSO is not empty, goes to Forward Step;

Forward Step

1. considers *i* as the last (highest key) instruction among the instructions already considered;
2. adds the next instruction (*i*.next) to the graph;

3. for each sponsor request of *i.next* until PSO is not empty, creates an edge (*i*.next, *PSO.pop()*);
4. for each sponsor request left, adds *i.next* to PSR;
5. for each sponsor offer, adds *i.next* to PSO;
6. considers *i.next* as *i*;
7. if *PSO* is not empty, goes to Forward Step.2;
8. if PSR is not empty, goes to Backward Step;

An example of how the algorithm works is given in Appendix A .

The correctness of the algorithm is given by the property *P1.* The swinging algorithm's steps aim to merge instructions until there are no more pending requests and offers, thus that the operand stack before those instructions is in a state, all the pushed values are consumed by the same instructions and after the last instruction the operand stack is in the same state. The termination of the algorithm depends on the simple consideration that the analysed code derive from syntactically and semantically verified code. There will never be instructions that require the presence of certain values on the stack without these have actually been previously entered, or *vice versa.* Thus the algorithm converges but, with this simplified version, *GOTO* instructions are dangerous (see above) and makes the algorithm not to terminate. Adding a check on the instruction under consideration will stop the algorithm, returning an error of fail while guaranteeing termination.

This algorithm should not be considered as a *data flow analysis* algorithm because it works with the number of values on the operand stack. While, the *data flow analysis* works with the values of all the variables in the program and its framework is completely different. For further information, see chapter 9 in [23].

2.3 Software Fault Operators

Among the most common software defects as in Figure 1.4, we detected nine different faults from different *ODC* classes and designed the related operators. These operators are:

Java Operator-Missing Method Call (JoMmc) to remove a method call from its context;

Java Operator-Missing If Around Statement (JoMias) to remove an *if* construct surrounding a set of statements;

Java Operator-Missing If Plus Statement (JoMips) to remove an if construct and the surrounded set of statement;

Java Operator-Missing variable Initialization with a Value (JoMiv) to reproduce the omission of the initialization (first assignment) of a given local variable with a constant value;

Java Operator-Missing variable Assignment with a Value (JoMav) to reproduce the omission of the assignment (not the first) of a given local variable with a constant value;

Java Operator-Missing Localized Part of an Algorithm (JoMlpa) to reproduce the omission of a small localized part of the algorithm;

Java Operator-Missing Logical Sub-expression in a Branch (JoMlsb) to emulate the omission of part of a logical expression used in a branch condition;

Java Operator-Wrong Arithmetic Expression in a method's Parameter (JoWaep) to emulate a wrong arithmetic expression used as parameter in a method call;

Java Operator-Wrong Value Assigned to a Variable (JoWvav) to emulate a wrong assignment of a given loval variable with a different constant value.

Each operator is described according to the rules that define the search pattern and the code change to apply to the locations identified by the search pattern. The search is bound by constraints that help to avoid locations where the code change would not emulate a realistic fault. An example is given in the next subsection.

2.3.1 Java Operator - Missing Method Call Design

This injection hypothesizes the case in which a programmer forgets to insert a call to a method. This could happen when the program is very large and complex, and the programmer must take into account several aspects.

The search pattern consists of all those opcodes for the method invocations, such as *INVOKEINTERFACE, INVOKESPECIAL, INVOKESTATIC and INVOKEVIRTUAL.*

The code change to apply is the deletion of all the bytecode instruction related to the method invocation, including the parameters preparation.

The constrains are:

C1 value of the function must not be used;

C2 call must not be the only statement in the block;

C3 call must not be referred to a constructor.

The algorithm used by the operator consists of the following steps:

1. Consider each *InvokeInstruction (INVOKEINTERFACE, INVOKESPECIAL, INVOKESTATIC and INVOKEVIRTUAL)*, let's say **ii**;
2. Check constrain **C1**:
   (a) get the return type of the invoked method. If void, go to 3.;
   (b) get the next instruction of **ii**, until the current instruction **ii**$! = NOP$;
   (c) if **ii**$== POP$, go to 3.;
   (d) if **ii**$instanceof$ *StoreInstruction*, search after **ii** for a *LoadInstruction* on the same variable. If there is none, go to 3.;
   (e) in any other case, **C1** is violated, consider the next *InvokeInstruction* and go to 2.;
3. Check constrain **C2**:

   (a) calculate the *BoundaryTable* **bt**, if not already calculated for the method under consideration;

   (b) calculate the *InstructionTree* **it** from **ii**;

   (c) if the first instruction of the **it** is different from the first instruction of the **bt** *OR* if the last instruction of the **it** is different from the last instruction of the **bt**, go to 4.;

   (d) in any other case, **C2** is violated, consider the next *InvokeInstruction* and go to 2.;

4. Check constrain **C3**:

   (a) if the name of the invoked method is *<init>* (the name given to the constructor in the bytecode), **C3** is violated, consider the next *InvokeInstruction* and go to 2.;

5. Delete the instructions of the **it**;

6. Consider the next *InvokeInstruction* and go to 2..

Chapter 3

Tool Implementation

3.1 Development Environment

The tool is completely implemented with the Java language for the execution environment *JavaSE-1.7.* The development environment is *Eclipse Java EE IDE for Web Developers*, version: *Juno Service Release 2.*

In Eclipse, three project have been developed according to the design: *JFIPrototype, JFIRemoteClient* and *JFIRemoteController*.

The injection package of the *JFIPrototype* project uses a third-party library called *BCEL [24].* The used version is the 5.2. This library parses the Bytecode and give the possibility to instantiate objects that abstract basic concepts as instruction, target etc..

All the other abstractions and algorithms are translated in high level language as classes organized in packages.

Another package has been added to track the control flow of the tool: the *logger* package. It implements a logger, with both a text formatter and an xml formatter, which logs all the useful information assigning an appropriate level.

3.2 Java Operators

The java operators' implementation follow the injection package design. The *AbstractOperator* class is extended by all the other operators that inherit:

public void setProfList(List<?> list) this method set the profiler list for this operator. Only the methods contained in the list will be considered for injection. If the list is null, all the methods of the class will be considered.

public List<FaultyKlassItem> operate(JavaClass klass) this method is the entry point of every operator. It gets as input a *BCEL*'s *JavaClass* object and returns a list of *FaultyClassItem*s. This method:

1. creates an empty list;
2. gets the methods and the constant pool[1] of the class;
3. scans all the methods in the class and considers only those one contained in the profiler list (if any);
4. for every methods, invokes the method *searchPattern* for that method;
5. for every pattern found, invokes the method *changeCode* and adds to the list its result;
6. returns the list;

protected FaultyKlassItem changeCode(InstructionHandle ih) this method creates and returns the item that represents the faulty class injected by the operator's software defect. It takes as input an *InstructionHandle*, i.e. an object from the *BCEL* library that manages a bytecode instruction in the class file. This method:

1. creates a *JavaClass* object as a deep copy of the original class file;

[1]A constant pool is an ordered set of constants present in the class file and used by the type, including literals (string, integer, and floating point constants) and symbolic references to types, fields, and methods.

2. looks for the method to change in the new *JavaClass* object and gets the method generator[2] of that method;
3. invokes the method *changeMethod* passing as input the instruction handler and the method generator and returning the modifies (injected) method;
4. sets the new constant pool for the modified *JavaClass*;
5. creates and returns a *FaultyClassItem* for the modified *JavaClass;*

protected InstructionHandle getNewIH(MethodGen mg, int pos) This method bypasses a problem with the instruction handler. When there is a deep copy of a *JavaClass* object the instruction handler of the original object cannot be linked directly to the new object. This method uses the position of the instruction in order to find the new instruction handler for the deep copy of the *JavaClass* object.

The abstract methods are implemented according to the single operator design. The following subsection give an example.

3.2.1 Java Operator - Missing Method Call Implementation

The JO-MMC provides an implementation, for the three abstract methods of the superclass *AbstractOperator,* that follows the design in Subsection 2.3.1. Moreover it has also an hash table used by these methods to keep a cache for the boundary tables. This avoids long time waiting for the calculation of a boundary table of an already considered method.

The method *searchPattern* instantiates a particular class of the *BCEL* library called *InstructionFinder.* This object is a tool to search for given instructions patterns, in this case for all the invoke instructions. Code patterns found are checked using an additional user-defined constraint object whether they really

[2]The method generator is a *BCEL* template class for building up a method. This is done by defining exception handlers, adding thrown exceptions, local variables and attributes, whereas the 'LocalVariableTable' and 'LineNumberTable' attributes will be set automatically for the code

match the needed criterion. The *CodeConstrain* object's method *checkCode* checks sequentially the three operator constrains implemented with as many predicates.

3.3 Remote Fault Activator

The remote communication of the *JFIRemoteClient* and *JFIRemoteController* is realized thanks to the *RMI* mechanisms provided by the Java language. It abstracts the communication interface to the level of a method call. Instead of working directly with sockets, the programmer has the illusion of using a local object, when in fact the arguments of the call are packaged up and shipped off to the remote target of the call.

The messages between the components are defined by the *RemoteControllerInterface*, which consists of:

public int getTriggerMode() gets an integer which encodes the trigger mode chosen by the remote controller and returns :

- 0 if *event-triggered*, i.e. when the fault is activated after a chosen number of the instruction executions;
- 1 if *user-triggered*, i.e. when the fault is activated according to the real-time state of the remote controller;

public int getLimit() works only if the trigger mode is *event-triggered* and returns the number of times the instruction has to run before activating the fault;

public int isApproved() works only if the trigger mode is *user-triggered* and returns:

- true if the fault should be activated;
- false if it should be deactivated;

public int connect() implements a handshaking between the server and the client and has a string as parameter for the name of the software where is located the activator.

JFIRemoteController implements the *RemoteControllerInterface* and creates and exports a registry instance on the local host that accepts requests on the port 30000. Thus load on the registry an instance of the *RemoteController*, which change its variables' value according to the user interface. *JFIRemoteClient*, instead, has a class singleton *ErrorActivator* with a private constructor and a static method *isActivated*. The target program is injected with a transient fault and remote trigger mode. When it executes the method *isActivated* (see Subsection2.1.2), the remote methods are queried. Moreover, *isActivated* calls the constructor, if not yet invoked once, and establishes a connection with the remote controller importing the remote object from the registry. The method *isActivated* returns:

true if the conditions to activate the fault are verified, according to the remote commands;

false otherwise.

Chapter 4

Case Study: Duke's Forest

4.1 Overview of the Case Study

This chapter shows the tool in the context of a real project, how it is used and which results and feedback it can return to improve the robustness and integrity of data. In order to understand the capability of the tool, it is first necessary that the case study is a project representative of the type of system which the tool has been developed for. It's also important that the application is well known so that the experiments are immediately clear. From the very beginning, Java always gives practical examples in its tutorial useful to show the language possibilities thanks to small but complete applications (e.g., the well-known *petstore* application). The last version of the java enterprise edition, the 7th, has a tutorial as well and the **Duke's Forest** case study example [25] is used here for evaluating the Java Fault Injection tool.

Duke's Forest is a simple e-commerce application that consists of three subsystems, as shown in Figure 4.1:

Duke's Store: a web application that provides a product catalog, customer self-registration, and a shopping cart. It also has an administration interface for product, category and user management;

Duke's Shipment: a web application that provides an interface for order shipment management;

Duke's Payment: a web service application that has a RESTful web service for order payment.

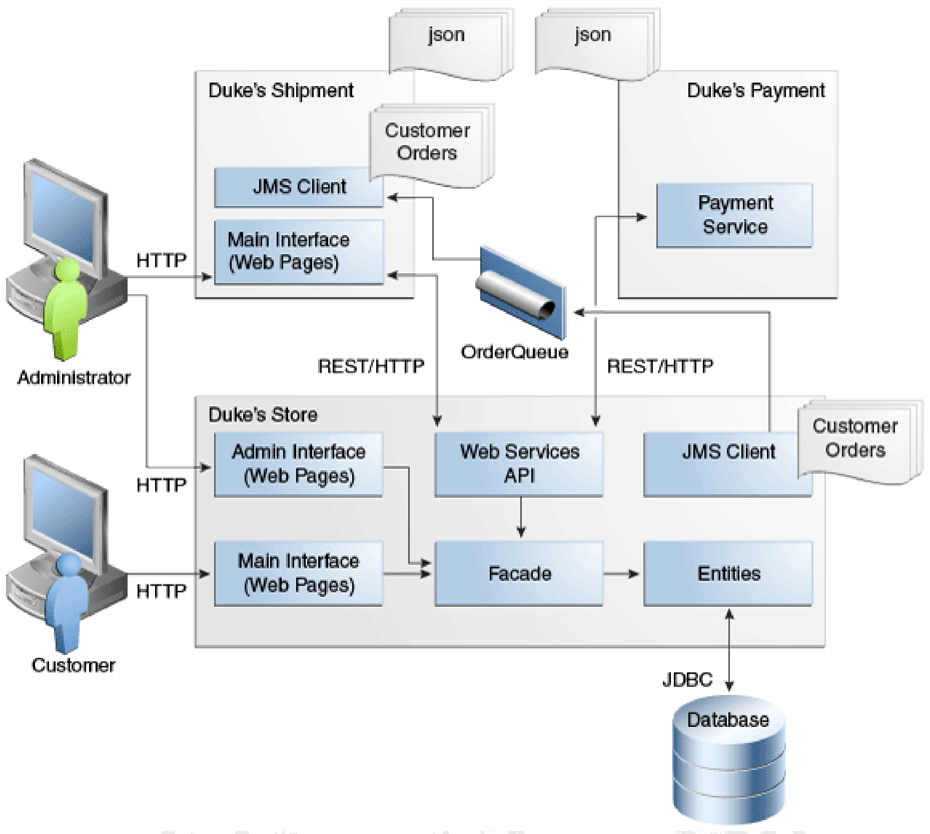


Figure 4.1: Architecture of the Duke's Forest Application

Duke's Store, a web application, is the core application of Duke's Forest. It is responsible for the main store interface for customers as well as the administration interface. The main interface of Duke's Store allows the user to perform the following tasks:

- Browsing the product catalog;
- Signing up as a new customer;
- Adding products to the shopping cart;
- Checking out;

- Viewing order status.

The administration interface of Duke's Store allows administrators to perform the following tasks:

- Product maintenance (create, edit, update, delete);
- Category maintenance (create, edit, update, delete);
- Customer maintenance (create, edit, update, delete);
- Group maintenance (create, edit, update, delete).

Duke's Shipment is a web application with a login page, a main Facelets page, and some other objects. This application, which is accessible only to administrators, consumes orders from a JMS queue and calls the RESTful web service exposed by Duke's Store to update the order status. The main page of Duke's Shipment shows a list of orders pending shipping approval and a list of shipped orders. The administrator can approve or deny orders for shipping. If approved, the order is shipped, and it appears under the Shipped heading. If denied, the order disappears from the page, and on the customer's Orders list it appears as cancelled.

The dukes-payment project is a web project that holds a simple Payment web service. Since this is an example application, it does not obtain any real credit information or even customer status to validate the payment. For now, the only rule imposed by the payment system is to deny all orders above $1,000. This application illustrates a common scenario where a third-party payment service is used to validate credit cards or bank payments.

4.2 Experimental Design

We injected the components with faults according to the nature of the component itself. The core application, duke's store, has been injected with exceptions in order to emulate a database connection loss; the duke's payment and shipments projects have been injected with software defects because they represent third-party or minor applications not adequately tested. The duke's payment application is a really small application, so the profiling filter was de-activated during the injection process to have more faults for the experiments.

To deploy and run the application on a server, we used a virtual machine with **Ubuntu 13.11** on **Oracle VM VirtualBox 4.2.18**. On the virtual machine *host-only networking* is used as networking mode, i.e. the virtual machine can talk to the host as if they were connected through a physical Ethernet switch. After the installation of such machine, projects were downloaded and deployed, server was started, all according to the tutorial [25]. Because of the nature of the bytecode instrumentation, the JVM in the GlassFish Server should be executed with the *SplitVerifier* option.

To evaluate the behaviour of the duke's forest, tests have been written to cover most of the possible use cases. The test environment used is **Selenium IDE 4.1.0**, a plug-in of the browser **Mozilla Firefox 25.0.1** [26]. Selenium allows to write in a simple way a test case, by indicating the buttons to push and the characters to type inside the browser. Every test case has a mnemonic name and contains several assert statements in order to check the integrity of the execution in every point. The test cases are:

RobertBuys User Robert logs in */dukes-store/*, browses through the products and buys for a total of $495. He checks his order and sees the order details, then logs out. Then he logs in */dukes-store/* and click on *Approve Shipment.* He's redirected to */dukes-shipment/* where he finds Robert's pending order. He confirms the order and tags it as confirmed order. The administrator returns to */dukes-store/* and sees the list of order where he deletes Robert's confirmed order from the list. He logs out.

AdminWantsToBuy The administrator logs in */dukes-store/*, browses through the products and buys for a total of $20. At the moment of the checkout from the cart, he gets an error saying *Administrators are not allowed to buy.* He clears the cart, checks that his order is not in the list of orders and logs out.

JackOverspends User Jack logs in */dukes-store/*, browses through the products and buys for a total of $1,550. He checks his order and finds it *Cancelled* because *$1000 order limit exceeded.* He logs out. The administrator logs in */dukes-store/* and click on *Approve Shipment.* He is redirected to */dukes-shipment/* where he does not find Jack's pending order. The administrator returns to */dukes-store/* and sees the list of order where he finds Robert's cancelled order and deletes it from the list. He logs out.

KenSignedAndBanned New user Ken visits */dukes-store/* and starts the signing procedure clicking on *Sign In.* He fills the forms with password *1234.* He clicks on *Save* but an error occurs because *Password must be between 7 and 100 characters and not empty.* He changes the password in *123456789* and saves. Ken logs in, browses the products and logs out. The administrator logs in */dukes-store/* and checks the list of customers. He finds Ken and changes his address. He checks the changes and, then, he deletes Ken. He logs out.

AdminCreatesNewItems The administrator logs in */dukes-store/*, browses through the categories and create a new one without description. An error occurs, he adds the description and saves the new category. He check the new category in the list of categories and changes his description again before deleting it. He browses through the products and create a new one without description. An error occurred, he adds the description and saves the new product. He check the new product in the list of products and deletes it. He logs out.

4.3 Experimental Results

This section consists of the results of some selected experiments we performed. In all, 30 experiments plus a golden run were performed. For the purpose of exposure, six experiments have been selected, each representative of a class of failures observed during the entire experimental campaign. For each of the selected experiments, the performed injection is reported with the diagnosis of the observed failure.

4.3.1 Experiment A

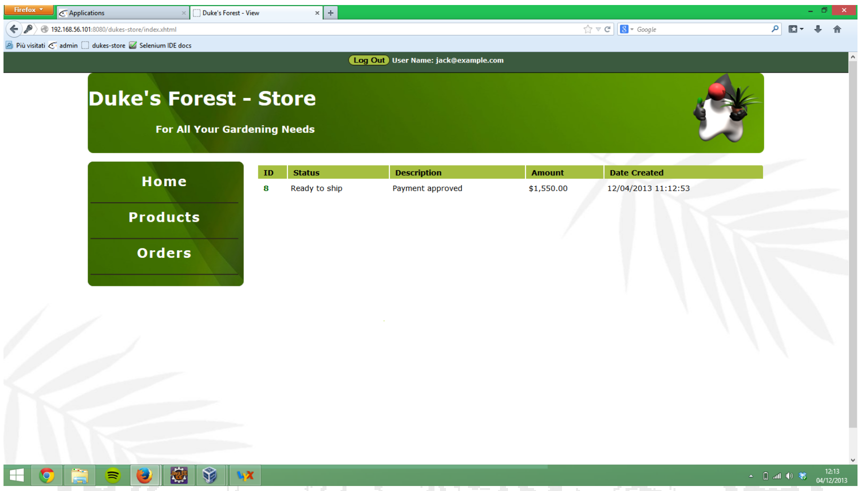
faulty component dukes-payment.war

changed method com.forest.payment.services.PaymentService.processPayment

line 36

operator Java Operator Missing If Around Statement

During the test *JackOverspends*, Jack logs in and checks out an order for $1,550. At this point dukes-payment should refuse the order and cancel it because is greater than $1,000[1]. Instead, because of the injected fault, the payment is approved and the administrator can find the order in the Duke's Shipment application. See Figure 4.2 for the screenshot.



ID	Status	Description	Amount	Date Created
8	Ready to ship	Payment approved	$1,550.00	12/04/2013 11:12:53

Figure 4.2: Screenshot Experiment A - Payment Over $1000 Not Denied

4.3.2 Experiment B

faulty component dukes-payment.war

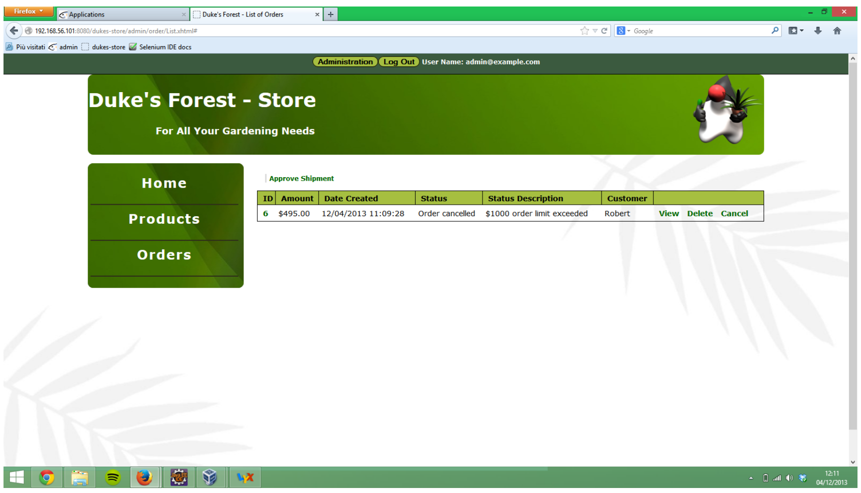
changed method com.forest.payment.services.PaymentService.processPayment

line 36

[1] recalling that $1,000 is a fake limit, but it can be considered as the total amount on a credit card or bank account that should deny payment greater than actual balance.

operator Java Operator Missing If Plus Statement

This failure is the dual of 4.3.1, but it shows up in the *RobertBuys* test case. Robert logs in and wants to spend $445. There should be no problem while the payment is not accepted, cancelling the order as shown in Figure 4.3.



ID	Amount	Date Created	Status	Status Description	Customer	
6	$495.00	12/04/2013 11:09:28	Order cancelled	$1000 order limit exceeded	Robert	View Delete Cancel

Figure 4.3: Screenshot Experiment B - Payment Under $1000 Denied

4.3.3 Experiment C

faulty component dukes-shipment.war

changed method com.forest.shipment.web.ShippingBean.getPendingOrders

line 9

operator Java Operator Missing If Around Statement

With this injection we are touching the duke's shipment application that also manages the paid order. Running the test *RobertBuys* it seems everything goes without any problem at client-side, but when the administrator checks the pending orders he can't find the one payed by Robert. Anyway he can find it in the list of orders, realizing that it has been paid. In fact, analysing the server log in section B.1, the order has been processed correctly and the payment has be done. Moreover the order has been updated and the OrderBrowser doesn't give any error even if it doesn't show the pending orders.

4.3.4 Experiment D

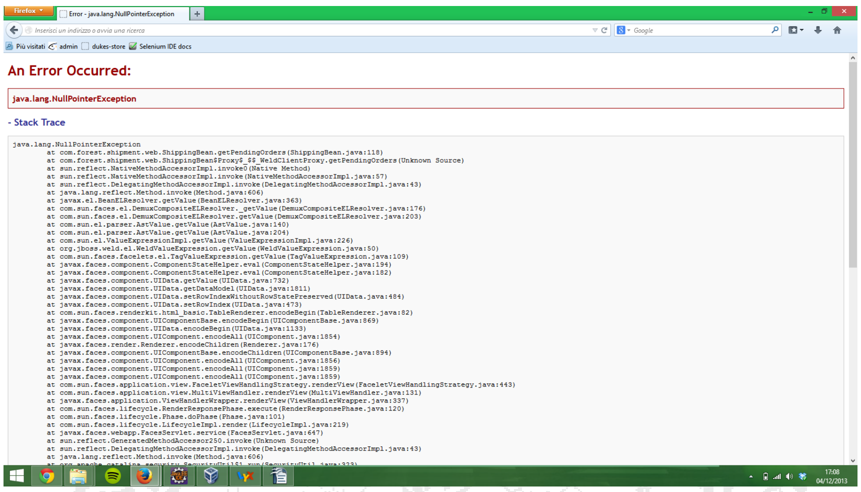
faulty component dukes-shipment.war

changed method com.forest.shipment.web.ShippingBean.getPendingOrders

line 16

operator Java Operator Missing Method Call

The scenario is similar to 4.3.3, but in this case we have an error page as in Figure 4.4 with an incomprehensible stack trace for the duke's shipment user, i.e. the administrator. The *NullPointerException* has not been caught by the application and propagates to the interface of the application, showing up in this page.


Firefox
Error - java.lang.NullPointerException
Inserisci un indirizzo o avvia una ricerca
Più visitati
admin
dukes-store
Selenium IDE docs

An Error Occurred:

java.lang.NullPointerException

- Stack Trace

```
java.lang.NullPointerException
	at com.forest.shipment.web.ShippingBean.getPendingOrders(ShippingBean.java:118)
	at com.forest.shipment.web.ShippingBean$Proxy$_$$_WeldClientProxy.getPendingOrders(Unknown Source)
	at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
	at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	at java.lang.reflect.Method.invoke(Method.java:606)
	at javax.el.BeanELResolver.getValue(BeanELResolver.java:363)
	at com.sun.faces.el.DemuxCompositeELResolver._getValue(DemuxCompositeELResolver.java:176)
	at com.sun.faces.el.DemuxCompositeELResolver.getValue(DemuxCompositeELResolver.java:203)
	at com.sun.el.parser.AstValue.getValue(AstValue.java:140)
	at com.sun.el.parser.AstValue.getValue(AstValue.java:204)
	at com.sun.el.ValueExpressionImpl.getValue(ValueExpressionImpl.java:226)
	at org.jboss.weld.el.WeldValueExpression.getValue(WeldValueExpression.java:50)
	at com.sun.faces.facelets.el.TagValueExpression.getValue(TagValueExpression.java:109)
	at javax.faces.component.ComponentStateHelper.eval(ComponentStateHelper.java:194)
	at javax.faces.component.ComponentStateHelper.eval(ComponentStateHelper.java:182)
	at javax.faces.component.UIData.getValue(UIData.java:732)
	at javax.faces.component.UIData.getDataModel(UIData.java:1811)
	at javax.faces.component.UIData.setRowIndexWithoutRowStatePreserved(UIData.java:484)
	at javax.faces.component.UIData.setRowIndex(UIData.java:473)
	at com.sun.faces.renderkit.html_basic.TableRenderer.encodeBegin(TableRenderer.java:82)
	at javax.faces.component.UIComponentBase.encodeBegin(UIComponentBase.java:869)
	at javax.faces.component.UIData.encodeBegin(UIData.java:1133)
	at javax.faces.component.UIComponent.encodeAll(UIComponent.java:1854)
	at javax.faces.render.Renderer.encodeChildren(Renderer.java:176)
	at javax.faces.component.UIComponentBase.encodeChildren(UIComponentBase.java:894)
	at javax.faces.component.UIComponent.encodeAll(UIComponent.java:1856)
	at javax.faces.component.UIComponent.encodeAll(UIComponent.java:1859)
	at javax.faces.component.UIComponent.encodeAll(UIComponent.java:1859)
	at com.sun.faces.application.view.FaceletViewHandlingStrategy.renderView(FaceletViewHandlingStrategy.java:443)
	at com.sun.faces.application.view.MultiViewHandler.renderView(MultiViewHandler.java:131)
	at javax.faces.application.ViewHandlerWrapper.renderView(ViewHandlerWrapper.java:337)
	at com.sun.faces.lifecycle.RenderResponsePhase.execute(RenderResponsePhase.java:120)
	at com.sun.faces.lifecycle.Phase.doPhase(Phase.java:101)
	at com.sun.faces.lifecycle.LifecycleImpl.render(LifecycleImpl.java:219)
	at javax.faces.webapp.FacesServlet.service(FacesServlet.java:647)
	at sun.reflect.GeneratedMethodAccessor250.invoke(Unknown Source)
	at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	at java.lang.reflect.Method.invoke(Method.java:606)
	at org.apache.catalina.security.SecurityUtil$1.run(SecurityUtil.java:323)
```

Figure 4.4: Screenshot Experiment D - Error Page

4.3.5 Experiment E

faulty component dukes-store.war

changed method com.forest.ejb.AbstractFacade.count

line 60

operator Java Operator No Result Exception

As mentioned before, the Duke's Store is injected with exceptions and, more precisely, with exception related to database connections. Database failures, then, can be simulated in order to understand how the application react when such an important service is suddenly off. In this experiment, all the test cases that browse a list (*RobertBuys, KenSignedAndBanned, JackOverspends* and *AdminCreatesNewItems*) crash at the moment of rendering the result page with the list of items. The failure consists in hanging on a blank page with not working navigation buttons at the top, see Figure 4.5. The only way to go on the website is to interact directly with the browser typing the homepage address. Anyway, each time the application need to query the database it hangs on this kind of situation until the database connection is restored.

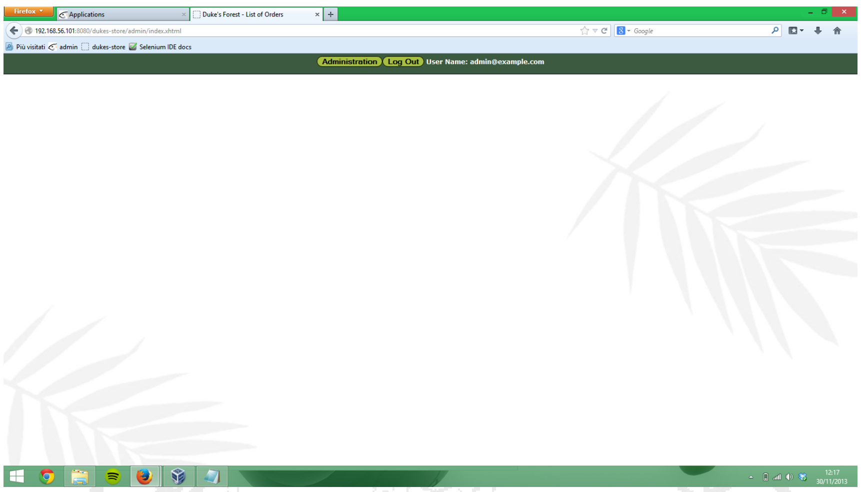

Firefox
Applications
Duke's Forest - List of Orders
192.168.56.101:8080/dukes-store/admin/index.xhtml
Google
Più visitati
admin
dukes-store
Selenium IDE docs
Administration
Log Out
User Name: admin@example.com
12:17
30/11/2013

Figure 4.5: Screenshot Experiment E - Blank Page with Not Working Navigation Buttons

Focusing on the detection, the server log can help. Section B.2 shows how the database error causes two unhandled exceptions:

- An **EJBException**, that usually report that the invoked business method or callback method could not be completed because of an unexpected error, e.g. the instance failed to open a database connection.
- An **ELException**, a subclass of runtime exception, representing any of the exception conditions that can arise during expression evaluation; less significant but indication of the page rendering problem.

4.3.6 Experiment F

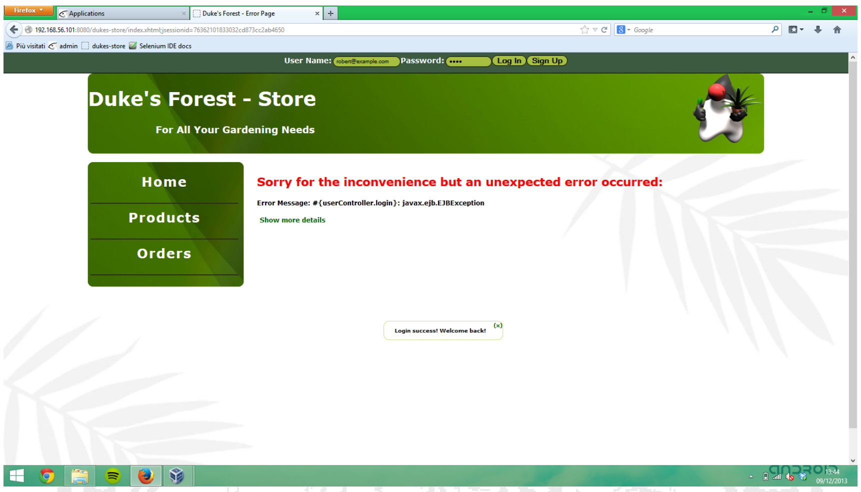
faulty component dukes-store.war

changed method com.forest.ejb.UserBean.getUserByEmail

line 23

operator Java Operator No Result Exception

This injection affects the whole test suite because the consequence is that neither the users nor the administrator can log in the application. A further anomaly could be the visualized page when we first try to log in, shown in Figure 4.6. Here there is both an error message with the java exception written in bold red, and the little pop up message that ensure the user to be logged in.


Sorry for the inconvenience but an unexpected error occurred:

Error Message: #{userController.login}: javax.ejb.EJBException

Show more details

Login success! Welcome back!
Duke's Forest - Error Page
192.168.56.101:8080/dukes-store/index.xhtml;jsessionid=76362101833032cd873cc2ab4650
User Name: robert@example.com
Password:
Log In
Sign Up
Duke's Forest - Store
For All Your Gardening Needs
Home
Products
Orders

Figure 4.6: Screenshot Experiment F - Page with Discordant Message

In order to understand the transient nature of this failure, the component has been injected again with the same operator in the same location but with the remote option. The trigger mode chosen is *user-triggered* in order to activate and deactivate the fault as needed. A new test case has been executed, with this flow:

1. User Robert logs in;
2. The Remote Controller is activated, the trigger mode chosen is synchronous and the fault is deactivated;

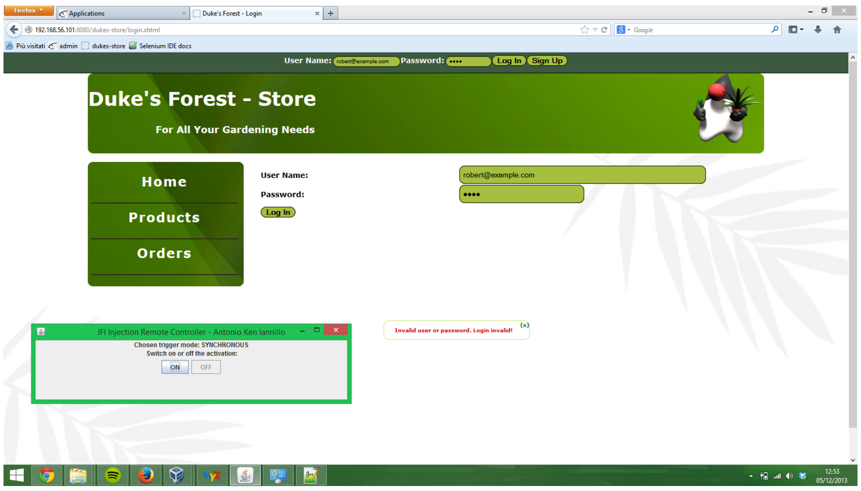
3. User Robert makes some shopping and logs out;

4. The fault is activated in the Remote Controller;

5. User Robert tries and logs in but gets the error as before.

6. The fault is deactivated in the Remote Controller.

7. User Robert tries and logs in but gets another error, as shown in Figure 4.7.


192.168.56.101:8080/dukes-store/login.xhtml

Duke's Forest - Store

For All Your Gardening Needs

User Name: robert@example.com

Password: ••••

JFI Injection Remote Controller - Antonio Ken Iannillo

Chosen trigger mode: SYNCHRONOUS

Switch on or off the activation:

Invalid user or password. Login invalid!

Figure 4.7: Screenshot Experiment F - Login Invalid after Deactivation of the Fault

The reason of this new failure mode can be extracted by the server log, reported in B.3.2. The log starts with the records of the remote controller activation. Using the RMI mechanism, the application gets the registry and the remote object reference. Then the error activator is initialized and gets the activation parameters. As written in the log, the trigger mode is *0*, *i.e.* *user-triggered*, and the boolean variable for the activation of the fault is set to false. Subsequently there are the records for a purchase with payment and shipment. When the user logs out and logs in again the fault is switched on and, as expected, the exceptions are recorded in the similar way to the previous case. Since the user did not succeed in logging in, remotely the fault is deactivated and the user tries to log in again. Although the fault is off, the session keeps memory of the failed trial and does not even try to query the database and logs the user. The unhandled exception in this case is a **ServletException**, caused by the inconsistent state of the session.

4.4 Discussion of Results

A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails [27]. Recovery from errors in fault-tolerant systems can be characterised as either **roll-forward** or **roll-back**. When the system detects that it has made an error, roll-forward recovery takes the system state at that time and corrects it, to be able to move forward. Roll-back recovery reverts the system state back to some earlier, correct version, for example using **checkpointing**, and moves forward from there.

The experimental results can give us a way to discuss about the fault-tolerant mechanisms and how they can be improved in our case study.

A first primitive property, derived from *experiment B,* can be a kind of caution during the fault-tolerant mechanisms implementation. The failure is not really dangerous because the e-shop does not lose any money and the user can see the mismatch inside his list of orders. There is no immediate need to improve the fault tolerance but a maintenance operation should be done to correct the defect when it shows up. Here the application denies the transaction, keeps its state safe and consistent and the user gets an error message, even if misleading.

A basic characteristic of fault tolerance requires **redundancy**, *i.e.* the duplication of critical components or functions of a system with the intention of increasing reliability of the system. Software redundancy comes in two flavours:

asymmetric where a part of the application performs the computation and another part is in charge of detecting errors and performing some kind of error processing and recovery; this part makes use of redundant logic to check the consistency of the return values of the services;

symmetric where all the parts have the same role; for example multiple functionally equivalent programs are independently generated from the same initial specifications, *i.e. N-Version Programming.*

In *experiment A*, the failure shows a defect inside a third-party application and its user (in this case the administrator) can lose even money when some condition are not verified correctly. In this case some redundancy should be added in the *dukes-shipment*, but it is a service that the application uses. A different solution is possible, operating directly on the application: a **log cross validation**. With the hypothesis to have the logs of the services, they can be analysed with the logs of the applications in order to detect any inconsistency between what the service tells to the application and logs. Logs are useful even in case where no third-party component is involved. In fact, *experiment C*'s failure doesn't affect the status of the orders but prevent the administrator from seeing them. The time of the transaction can increase drastically if the administrator does not catch the failure manually. In order to detect promptly this kind of problems, it would be appropriate to introduce a fault detection mechanism that analyses periodically the log and matching the pending orders with the list visualized by the administrator. Then it can add the pending order and notify the administrator.

Experiment D shows the problem of unhandled exceptions, common for languages of new generation as Java, that introduces a mechanism called **error masking**. When an exception is raised, the application should have a handler for it in order to give the user an appropriate error message instead of the stack trace of the application. Locate where to add and implement carefully the *try-catch* blocks, guided by fault injection experiment like this, is a useful maintenance operation. Furthermore, this situation can simply crashing the application without any message, as *experiment E*, and this detection mechanism should also be associated at a specific recovery mechanism. This should establish again a connection with the database in a consistent way, better if the database is duplicated for a higher reliability.

Finally, the need of a more complex recovery mechanism is discovered by *Experiment F*. The detection could be simply done catching the related exceptions of the failure for the inconsistent state of the application. Thus, the application should be able to reset the session, taking in count also the database entries for that session. Thus, the fault tolerance should mask the failure to the user, revert the system state back to some earlier correct version and try the queries again. Once again, implemented this mechanism, suitable test cases can be performed using the Faut Injection Tool for Java software applications.

Conclusion and Future Developments

In this thesis we started with the notion of availability and how it is important for nowadays software. Then we went through the software fault injection field study, understanding why it is so important for software dependability. From chapter 2 on, we focused about a tool able to inject fault inside java bytecode. These fault injections follow specific criteria and their design should have been so accurate. There were also other features of the tool, from the management of java archive file to the profiling, to the utility needed for the bytecode manipulation. Finally we wanted to show how the tool works and which kind of results can be obtained. We also briefly discuss about these results, making some consideration on the possible fault-tolerant algorithms and mechanisms a software engineer should take in these situations and more others.

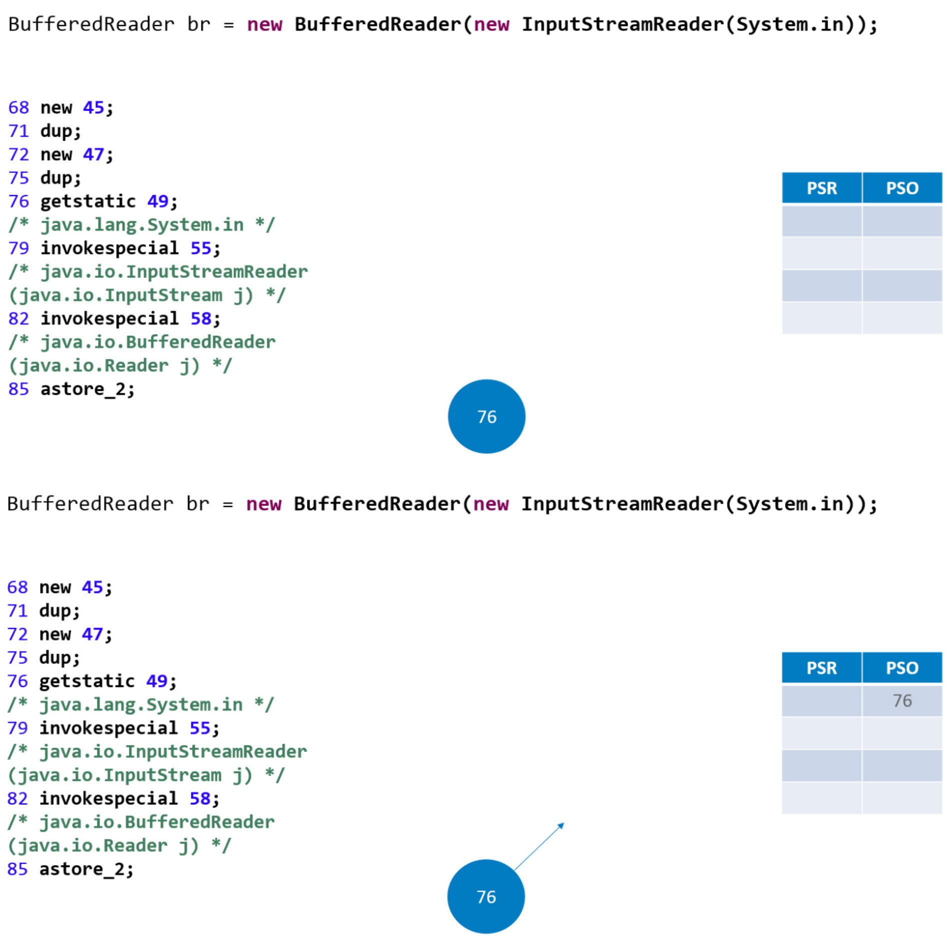
A possible improvement of the fault injection tool could be to extend it with more fault types. In fact, the fault types encompassed by the tool cover the most frequent types of faults found in the field according to previous studies, but more types could be introduced to increase the coverage of field faults, and to customize the faultload to the types of faults experienced by the users on their specific systems

The mechanism of the inclination table can be substituted. The tool uses that for the exception injections, because it does not know what kind of exception a method can throw. Instead, the tool can develop the ability to watch inside the java .class files, of the application and its libraries, in order to find this information. The inheritance and polymorphism of java language can create some obstacles.

Finally, the remote activation can include another mode. We can call it a stack-triggered mode. The tool, at the moment of the activation, can check inside the stack trace of the thread. This way allows the user to activate the fault only if a declared method is part of the stack. This mode want to emulate situation where the fault is activated only if a specific path has been taken during execution.

Appendix A

Swinging Algorithm Example



```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));


68 new 45;
71 dup;
72 new 47;
75 dup;
76 getstatic 49;
/* java.lang.System.in */
79 invokespecial 55;
/* java.io.InputStreamReader
(java.io.InputStream j) */
82 invokespecial 58;
/* java.io.BufferedReader
(java.io.Reader j) */
85 astore_2;
```

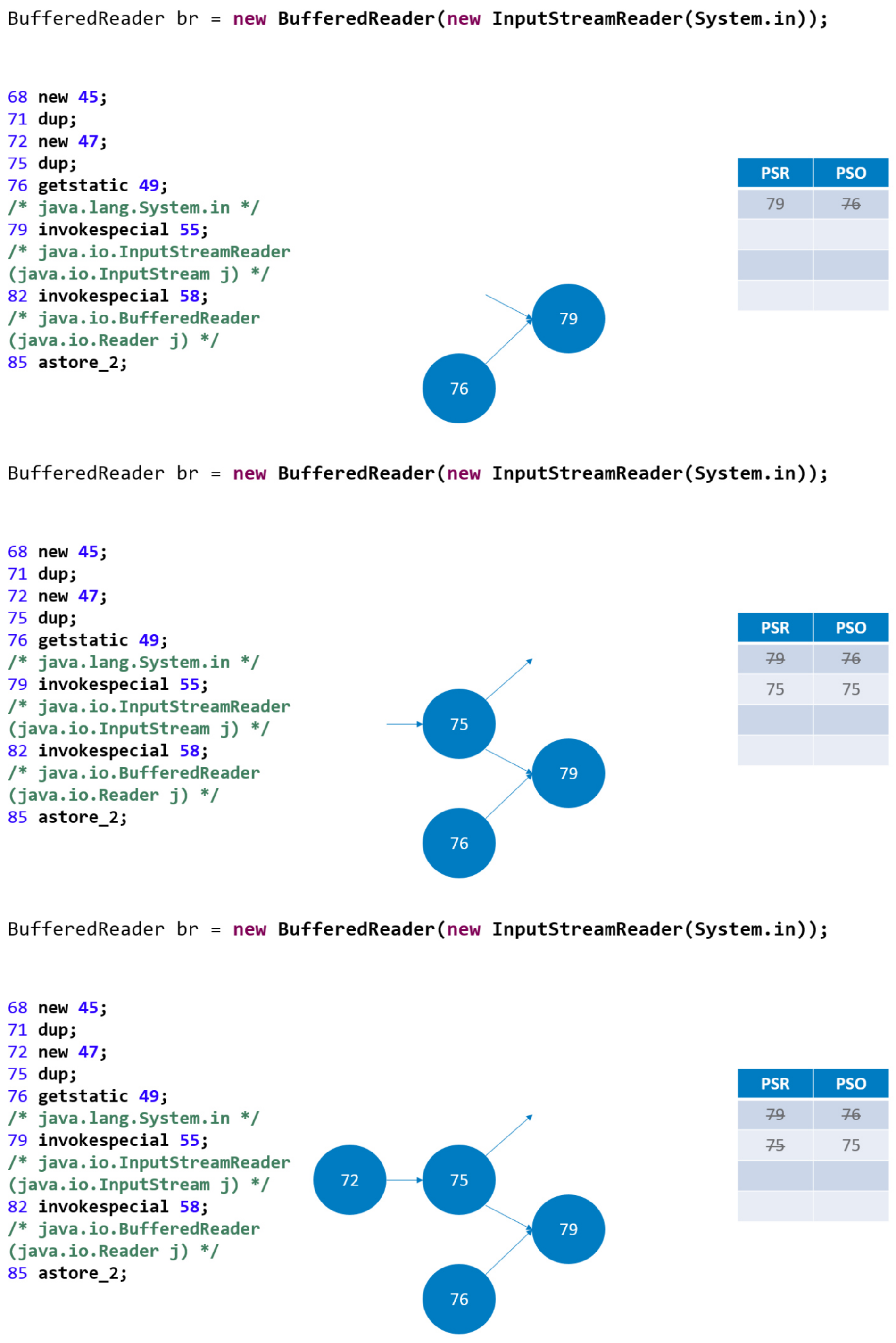
PSR	PSO

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));


68 new 45;
71 dup;
72 new 47;
75 dup;
76 getstatic 49;
/* java.lang.System.in */
79 invokespecial 55;
/* java.io.InputStreamReader
(java.io.InputStream j) */
82 invokespecial 58;
/* java.io.BufferedReader
(java.io.Reader j) */
85 astore_2;
```

PSR	PSO
	76

Figure A.1: Swinging Algorithm Example - (A) (B)



```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

68 new 45;
71 dup;
72 new 47;
75 dup;
76 getstatic 49;
/* java.lang.System.in */
79 invokespecial 55;
/* java.io.InputStreamReader
(java.io.InputStream j) */
82 invokespecial 58;
/* java.io.BufferedReader
(java.io.Reader j) */
85 astore_2;
```

PSR	PSO
79	~~76~~

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

68 new 45;
71 dup;
72 new 47;
75 dup;
76 getstatic 49;
/* java.lang.System.in */
79 invokespecial 55;
/* java.io.InputStreamReader
(java.io.InputStream j) */
82 invokespecial 58;
/* java.io.BufferedReader
(java.io.Reader j) */
85 astore_2;
```

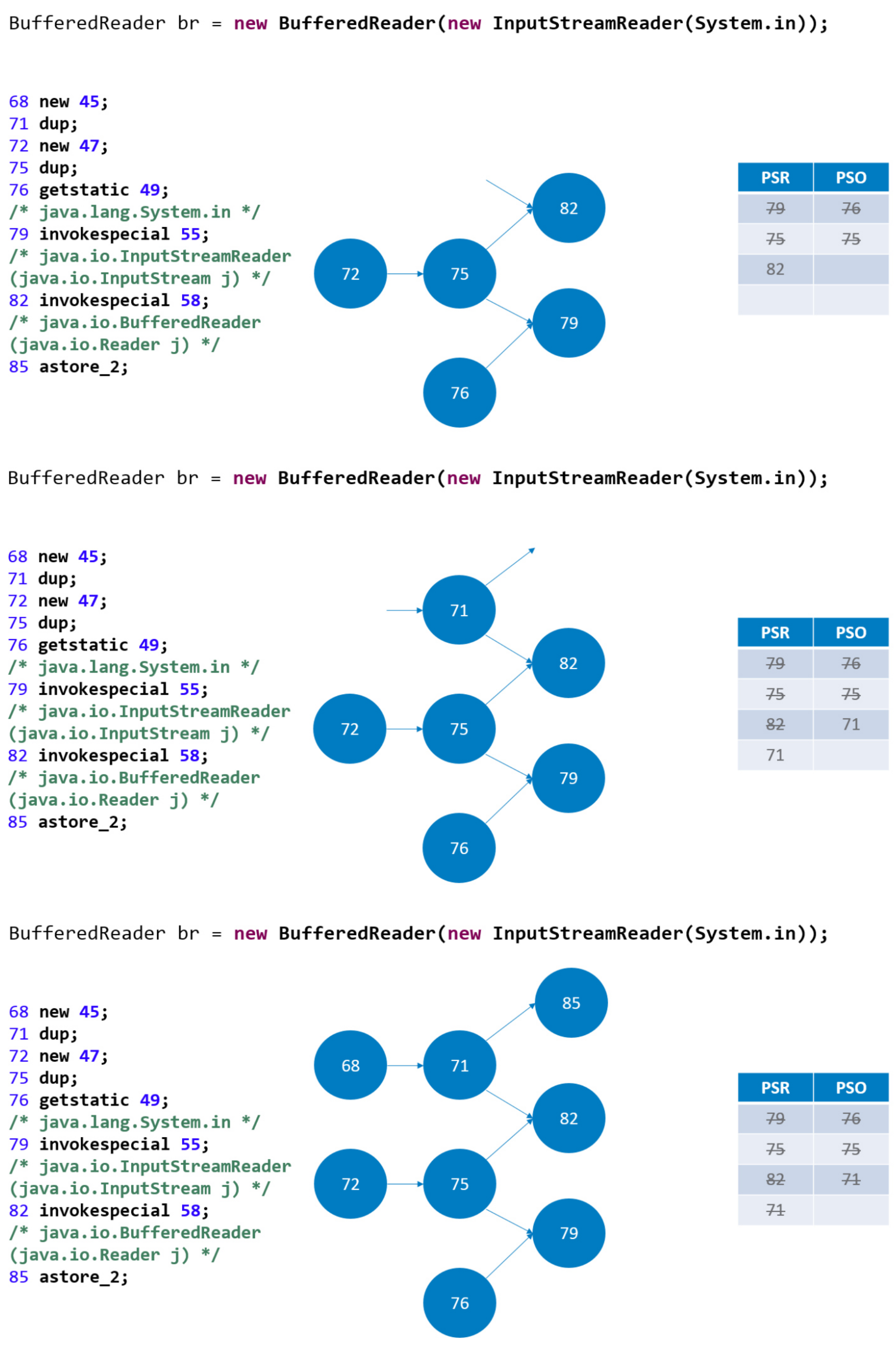
PSR	PSO
~~79~~	~~76~~
75	75

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

68 new 45;
71 dup;
72 new 47;
75 dup;
76 getstatic 49;
/* java.lang.System.in */
79 invokespecial 55;
/* java.io.InputStreamReader
(java.io.InputStream j) */
82 invokespecial 58;
/* java.io.BufferedReader
(java.io.Reader j) */
85 astore_2;
```

PSR	PSO
~~79~~	~~76~~
~~75~~	75

Figure A.2: Swinging Algorithm Example - (C) (D) (E)



```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

68 new 45;
71 dup;
72 new 47;
75 dup;
76 getstatic 49;
/* java.lang.System.in */
79 invokespecial 55;
/* java.io.InputStreamReader
(java.io.InputStream j) */
82 invokespecial 58;
/* java.io.BufferedReader
(java.io.Reader j) */
85 astore_2;
```

PSR	PSO
~~79~~	~~76~~
~~75~~	~~75~~
82	

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

68 new 45;
71 dup;
72 new 47;
75 dup;
76 getstatic 49;
/* java.lang.System.in */
79 invokespecial 55;
/* java.io.InputStreamReader
(java.io.InputStream j) */
82 invokespecial 58;
/* java.io.BufferedReader
(java.io.Reader j) */
85 astore_2;
```

PSR	PSO
~~79~~	~~76~~
~~75~~	~~75~~
~~82~~	71
71	

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

68 new 45;
71 dup;
72 new 47;
75 dup;
76 getstatic 49;
/* java.lang.System.in */
79 invokespecial 55;
/* java.io.InputStreamReader
(java.io.InputStream j) */
82 invokespecial 58;
/* java.io.BufferedReader
(java.io.Reader j) */
85 astore_2;
```

PSR	PSO
~~79~~	~~76~~
~~75~~	~~75~~
~~82~~	~~71~~
~~71~~	

Figure A.3: Swinging Algorithm Example - (F) (G) (H)

Appendix B

Server Logs

B.1 Experiment C

```
[2013-11-29T17:00:07.181+0100] [glassfish 4.0] [INFO]
[] [PaymentService] [tid: _ThreadID=113 _ThreadName=http-listener-1(1)] [↩
    timeMillis: 1385740807181] [levelValue: 800]
[[ Amount: 495.0]]

[2013-11-29T17:00:07.183+0100] [glassfish 4.0] [INFO]
[] [com.forest.handlers.PaymentHandler] [tid: _ThreadID=147 _ThreadName=__ejb-↩
    thread-pool16] [timeMillis: 1385740807183] [levelValue: 800] [
[[ PaymentHandler] Response status 200]]

[2013-11-29T17:00:07.186+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=147 _ThreadName=__ejb-thread-↩
    pool16] [timeMillis: 1385740807186] [levelValue: 800]
[[ Order id:1 - Status:2]]

[2013-11-29T17:00:07.193+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=147 _ThreadName=__ejb-thread-↩
    pool16] [timeMillis: 1385740807193] [levelValue: 800]
[[ Order Updated!]]

[2013-11-29T17:00:07.195+0100] [glassfish 4.0] [INFO]
[] [com.forest.handlers.PaymentHandler] [tid: _ThreadID=147 _ThreadName=__ejb-↩
    thread-pool16] [timeMillis: 1385740807195] [levelValue: 800]
[[ Payment Approved]]

[2013-11-29T17:00:07.198+0100] [glassfish 4.0] [INFO]
```

```
[] [com.forest.handlers.DeliveryHandler] [tid: _ThreadID=129 _ThreadName=__ejb-↩
    thread-pool2] [timeMillis: 1385740807198] [levelValue: 800]
[[ Order #1 has been paid in the amount of 495. Order is now ready for delivery↩
    !]]

[2013-11-29T17:00:07.199+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=129 _ThreadName=__ejb-thread-↩
    pool2] [timeMillis: 1385740807199] [levelValue: 800]
[[ Order id:1 - Status:3]]

[2013-11-29T17:00:07.200+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=129 _ThreadName=__ejb-thread-↩
    pool2] [timeMillis: 1385740807200] [levelValue: 800]
[[ Order Updated!]]

[2013-11-29T17:00:08.936+0100] [glassfish 4.0] [INFO]
[] [com.forest.shipment.ejb.OrderBrowser] [tid: _ThreadID=116 _ThreadName=http-↩
    listener-1(4)] [timeMillis: 1385740808936] [levelValue: 800]
[[ Message ID: ID:10-127.0.1.1(e8:67:26:9:44:d5)-1-1385740807282]]

[2013-11-29T17:00:08.973+0100]  [glassfish 4.0] [INFO]
[] [com.forest.shipment.ejb.OrderBrowser] [tid: _ThreadID=116 _ThreadName=http-↩
    listener-1(4)] [timeMillis: 1385740808936] [levelValue: 800]
[[ Message ID: ID:10-127.0.1.1(e8:67:26:9:44:d5)-1-1385740807282]]

[2013-11-29T17:00:08.980+0100] [glassfish 4.0] [INFO]
[] [com.forest.shipment.ejb.OrderBrowser] [tid: _ThreadID=116 _ThreadName=http-↩
    listener-1(4)] [timeMillis: 1385740808936] [levelValue: 800]
[[ Message ID: ID:10-127.0.1.1(e8:67:26:9:44:d5)-1-1385740807282]]
```

B.2 Experiment E

```
[2013-11-30T12:12:36.759+0100] [glassfish 4.0] [INFO]
[] [PaymentService] [tid: _ThreadID=91 _ThreadName=http-listener-1(4)] [↩
    timeMillis: 1385809956759] [levelValue: 800]
[[Amount: 495.0]]

[2013-11-30T12:12:36.771+0100] [glassfish 4.0] [INFO]
[] [com.forest.handlers.PaymentHandler] [tid: _ThreadID=988 _ThreadName=__ejb-↩
    thread-pool10] [timeMillis: 1385809956771] [levelValue: 800]
[[[PaymentHandler] Response status 200]]

[2013-11-30T12:12:36.773+0100] [glassfish 4.0] [INFO]
```

```
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=988 _ThreadName=__ejb-thread-↩
    pool10] [timeMillis: 1385809956773] [levelValue: 800]
[[Order id:2 - Status:2]]

[2013-11-30T12:12:36.778+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=988 _ThreadName=__ejb-thread-↩
    pool10] [timeMillis: 1385809956778] [levelValue: 800]
[[Order Updated!]]

[2013-11-30T12:12:36.779+0100] [glassfish 4.0] [INFO]
[] [com.forest.handlers.PaymentHandler] [tid: _ThreadID=988 _ThreadName=__ejb-↩
    thread-pool10] [timeMillis: 1385809956779] [levelValue: 800]
[[Payment Approved]]

[2013-11-30T12:12:36.792+0100] [glassfish 4.0] [INFO]
[] [com.forest.handlers.DeliveryHandler] [tid: _ThreadID=990 _ThreadName=__ejb-↩
    thread-pool11] [timeMillis: 1385809956792] [levelValue: 800]
[[Order #2 has been paid in the amount of 495. Order is now ready for delivery!]]

[2013-11-30T12:12:36.796+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=990 _ThreadName=__ejb-thread-↩
    pool11] [timeMillis: 1385809956796] [levelValue: 800]
[[Order id:2 - Status:3]]

[2013-11-30T12:12:36.816+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=990 _ThreadName=__ejb-thread-↩
    pool11] [timeMillis: 1385809956816] [levelValue: 800]
[[Order Updated!]]

[2013-11-30T12:12:38.943+0100] [glassfish 4.0] [INFO]
[] [com.forest.shipment.ejb.OrderBrowser] [tid: _ThreadID=92 _ThreadName=http-↩
    listener-1(5)] [timeMillis: 1385809958943] [levelValue: 800]
[[Message ID: ID:3-127.0.1.1(a8:2:15:c5:1f:d3)-1-1385809957076]]

[2013-11-30T12:12:40.968+0100] [glassfish 4.0] [INFO]
[] [com.forest.shipment.ejb.OrderBrowser] [tid: _ThreadID=90 _ThreadName=http-↩
    listener-1(3)] [timeMillis: 1385809960968] [levelValue: 800]
[[Processing Order ID:3-127.0.1.1(a8:2:15:c5:1f:d3)-1-1385809957076]]

[2013-11-30T12:12:41.011+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=92 _ThreadName=http-listener↩
    -1(5)] [timeMillis: 1385809961011] [levelValue: 800]
[[Order id:2 - Status:4]]

[2013-11-30T12:12:41.023+0100] [glassfish 4.0] [INFO]
```

```
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=92 _ThreadName=http-listener↩
    -1(5)] [timeMillis: 1385809961023] [levelValue: 800]
[[Order Updated!]]

[2013-11-30T12:12:41.080+0100] [glassfish 4.0] [INFO]
[] [com.forest.shipment.ejb.OrderBrowser] [tid: _ThreadID=90 _ThreadName=http-↩
    listener-1(3)] [timeMillis: 1385809961080] [levelValue: 800]
[[No messages on the queue!]]

[2013-11-30T12:12:42.261+0100] [glassfish 4.0] [WARNING]
[ejb.system_exception] [javax.enterprise.system.container.ejb.com.sun.ejb.↩
    containers]
[tid: _ThreadID=90 _ThreadName=http-listener-1(3)] [timeMillis: 1385809962261] [↩
    levelValue: 900]
[[EJB5184:A system exception occurred during an invocation on EJB OrderBean, ↩
    method: public int com.forest.ejb.AbstractFacade.count()]]

[2013-11-30T12:12:42.262+0100] [glassfish 4.0] [WARNING]
[] [javax.enterprise.system.container.ejb.com.sun.ejb.containers] [tid: _ThreadID↩
    =90 _ThreadName=http-listener-1(3)]
[timeMillis: 1385809962262] [levelValue: 900]
[[
javax.ejb.EJBException
    at com.sun.ejb.containers.EJBContainerTransactionManager.↩
        processSystemException(EJBContainerTransactionManager.java:748)
    at com.sun.ejb.containers.EJBContainerTransactionManager.completeNewTx(↩
        EJBContainerTransactionManager.java:698)
    at com.sun.ejb.containers.EJBContainerTransactionManager.postInvokeTx(↩
        EJBContainerTransactionManager.java:503)
    at com.sun.ejb.containers.BaseContainer.postInvokeTx(BaseContainer.java:4475)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:2009)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:1979)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(↩
        EJBLocalObjectInvocationHandler.java:220)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandlerDelegate.invoke(↩
        EJBLocalObjectInvocationHandlerDelegate.java:88)
    at com.sun.proxy.$Proxy336.count(Unknown Source)
    ... (hidden for reason of space)
    at java.lang.Thread.run(Thread.java:744)
Caused by: javax.persistence.NoResultException
    at com.forest.ejb.AbstractFacade.count(AbstractFacade.java:72)
    ... (hidden for reason of space)
    ... 73 more
]]

[2013-11-30T12:12:42.265+0100] [glassfish 4.0] [SEVERE]
```

```
[] [javax.enterprise.resource.webcontainer.jsf.application] [tid: _ThreadID=90 ↩
    _ThreadName=http−listener−1(3)]
[timeMillis: 1385809962265] [levelValue: 1000]
[
[Error Rendering View[/admin/order/List.xhtml]
javax.el.ELException: /admin/order/List.xhtml @19,217 rendered="#{↩
    customerOrderController.pagination.hasNextPage}": javax.ejb.EJBException
    at com.sun.faces.facelets.el.TagValueExpression.getValue(TagValueExpression.↩
        java:114)
    at javax.faces.component.ComponentStateHelper.eval(ComponentStateHelper.java↩
        :194)
    at javax.faces.component.UIComponentBase.isRendered(UIComponentBase.java:462)
    at com.sun.faces.renderkit.html_basic.HtmlBasicRenderer.encodeRecursive(↩
        HtmlBasicRenderer.java:297)
    at com.sun.faces.renderkit.html_basic.GroupRenderer.encodeChildren(↩
        GroupRenderer.java:115)
    ... (hidden for reason of space)
    at java.lang.Thread.run(Thread.java:744)
Caused by: javax.el.ELException: javax.ejb.EJBException
    at javax.el.BeanELResolver.getValue(BeanELResolver.java:368)
    at com.sun.faces.el.DemuxCompositeELResolver._getValue(↩
        DemuxCompositeELResolver.java:176)
    at com.sun.faces.el.DemuxCompositeELResolver.getValue(↩
        DemuxCompositeELResolver.java:203)
    at com.sun.el.parser.AstValue.getValue(AstValue.java:140)
    at com.sun.el.parser.AstValue.getValue(AstValue.java:204)
    at com.sun.el.ValueExpressionImpl.getValue(ValueExpressionImpl.java:226)
    at org.jboss.weld.el.WeldValueExpression.getValue(WeldValueExpression.java↩
        :50)
    at com.sun.faces.facelets.el.TagValueExpression.getValue(TagValueExpression.↩
        java:109)
    ... 56 more
Caused by: javax.ejb.EJBException
    at com.sun.ejb.containers.EJBContainerTransactionManager.↩
        processSystemException(EJBContainerTransactionManager.java:748)
    at com.sun.ejb.containers.EJBContainerTransactionManager.completeNewTx(↩
        EJBContainerTransactionManager.java:698)
    at com.sun.ejb.containers.EJBContainerTransactionManager.postInvokeTx(↩
        EJBContainerTransactionManager.java:503)
    at com.sun.ejb.containers.BaseContainer.postInvokeTx(BaseContainer.java:4475)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:2009)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:1979)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(↩
        EJBLocalObjectInvocationHandler.java:220)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandlerDelegate.invoke(↩
        EJBLocalObjectInvocationHandlerDelegate.java:88)
    at com.sun.proxy.$Proxy336.count(Unknown Source)
    ... (hidden for reason of space)
    ... 63 more
Caused by: javax.persistence.NoResultException
    at com.forest.ejb.AbstractFacade.count(AbstractFacade.java:72)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java↩
        :57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(↩
        DelegatingMethodAccessorImpl.java:43)
    ... (hidden for reason of space)
    ... 73 more
]]
```

B.3 Experiment F

B.3.1 First Log

```
[2013−12−05T12:23:55.044+0100] [glassfish 4.0] [WARNING]
[ejb.system_exception] [javax.enterprise.system.container.ejb.com.sun.ejb.↩
    containers]
[tid: _ThreadID=102 _ThreadName=http−listener−1(5)] [timeMillis: 1386242635044] [↩
    levelValue: 900]
[[
EJB5184:A system exception occurred during an invocation on EJB UserBean, method:↩
     public com.forest.entity.Person com.forest.ejb.UserBean.getUserByEmail(java.↩
    lang.String)]]

[2013−12−05T12:23:55.044+0100] [glassfish 4.0] [WARNING]
[] [javax.enterprise.system.container.ejb.com.sun.ejb.containers]
[tid: _ThreadID=102 _ThreadName=http−listener−1(5)] [timeMillis: 1386242635044] [↩
    levelValue: 900]
[[
javax.ejb.EJBException
    at com.sun.ejb.containers.EJBContainerTransactionManager.↩
        processSystemException(EJBContainerTransactionManager.java:748)
    at com.sun.ejb.containers.EJBContainerTransactionManager.completeNewTx(↩
        EJBContainerTransactionManager.java:698)
    at com.sun.ejb.containers.EJBContainerTransactionManager.postInvokeTx(↩
        EJBContainerTransactionManager.java:503)
    at com.sun.ejb.containers.BaseContainer.postInvokeTx(BaseContainer.java:4475)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:2009)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:1979)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(↩
        EJBLocalObjectInvocationHandler.java:220)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandlerDelegate.invoke(↩
        EJBLocalObjectInvocationHandlerDelegate.java:88)
    at com.sun.proxy.$Proxy286.getUserByEmail(Unknown Source)
    at com.forest.ejb.__EJB31_Generated__UserBean__Intf____Bean__.getUserByEmail(↩
        Unknown Source)
    at com.forest.web.UserController.login(UserController.java:59)
    at com.forest.web.UserController$Proxy$_$$_WeldClientProxy.login(Unknown ↩
        Source)
    ... (hidden for reason of space)
    at java.lang.Thread.run(Thread.java:744)
Caused by: javax.persistence.NoResultException
    at com.forest.ejb.UserBean.getUserByEmail(UserBean.java:46)
    ... (hidden for reason of space)
    ... 62 more

]]



[2013-12-05T12:23:55.047+0100] [glassfish 4.0] [WARNING]
[] [javax.enterprise.resource.webcontainer.jsf.lifecycle]
[tid: _ThreadID=102 _ThreadName=http-listener-1(5)] [timeMillis: 1386242635047] [↩
    levelValue: 900]
[[#{userController.login}: javax.ejb.EJBException
javax.faces.FacesException: #{userController.login}: javax.ejb.EJBException
    at com.sun.faces.application.ActionListenerImpl.processAction(↩
        ActionListenerImpl.java:118)
    at javax.faces.component.UICommand.broadcast(UICommand.java:315)
    at javax.faces.component.UIViewRoot.broadcastEvents(UIViewRoot.java:790)
    at javax.faces.component.UIViewRoot.processApplication(UIViewRoot.java:1282)
    at com.sun.faces.lifecycle.InvokeApplicationPhase.execute(↩
        InvokeApplicationPhase.java:81)
    at com.sun.faces.lifecycle.Phase.doPhase(Phase.java:101)
    ... (hidden for reason of space)
    at java.lang.Thread.run(Thread.java:744)
Caused by: javax.faces.el.EvaluationException: javax.ejb.EJBException
    at javax.faces.component.MethodBindingMethodExpressionAdapter.invoke(↩
        MethodBindingMethodExpressionAdapter.java:101)
    at com.sun.faces.application.ActionListenerImpl.processAction(↩
        ActionListenerImpl.java:102)
    ... 46 more
Caused by: javax.ejb.EJBException
    at com.sun.ejb.containers.EJBContainerTransactionManager.↩
        processSystemException(EJBContainerTransactionManager.java:748)
    at com.sun.ejb.containers.EJBContainerTransactionManager.completeNewTx(↩
        EJBContainerTransactionManager.java:698)
    at com.sun.ejb.containers.EJBContainerTransactionManager.postInvokeTx(↩
        EJBContainerTransactionManager.java:503)
    at com.sun.ejb.containers.BaseContainer.postInvokeTx(BaseContainer.java:4475)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:2009)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:1979)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(↩
        EJBLocalObjectInvocationHandler.java:220)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandlerDelegate.invoke(↩
        EJBLocalObjectInvocationHandlerDelegate.java:88)
    at com.sun.proxy.$Proxy286.getUserByEmail(Unknown Source)
    at com.forest.ejb.__EJB31_Generated__UserBean__Intf____Bean__.getUserByEmail(↩
        Unknown Source)
    at com.forest.web.UserController.login(UserController.java:59)
    at com.forest.web.UserController$Proxy$_$$_WeldClientProxy.login(Unknown ↩
        Source)
    ... (hidden for reason of space)
    ... 47 more
Caused by: javax.persistence.NoResultException
    at com.forest.ejb.UserBean.getUserByEmail(UserBean.java:46)
    ... (hidden for reason of space)
    ... 62 more

]]
```

B.3.2 Second Log

```
[2013-12-05T12:49:44.791+0100] [glassfish 4.0] [INFO]
[] [] [tid: _ThreadID=99 _ThreadName=http-listener-1(2)] [timeMillis: ↩
    1386244184791] [levelValue: 800]
[[it already exists a security manager, be careful of the permission!]]

[2013-12-05T12:49:44.793+0100] [glassfish 4.0] [SEVERE]
[] [] [tid: _ThreadID=99 _ThreadName=Thread-4] [timeMillis: 1386244184793] [↩
    levelValue: 1000]
[[Dec 05, 2013 12:49:44 PM it.truestoryfactory.ken.JFI.RemoteClient.Client <init>
INFO: GOT: registry RegistryImpl_Stub[UnicastRef [liveRef: [endpoint↩
    :[192.168.56.1:30000](remote),objID:[0:0:0, 0]]]]]]

[2013-12-05T12:49:59.828+0100] [glassfish 4.0] [SEVERE]
[] [] [tid: _ThreadID=99 _ThreadName=Thread-4] [timeMillis: 1386244199828] [↩
    levelValue: 1000]
[[Dec 05, 2013 12:49:59 PM it.truestoryfactory.ken.JFI.RemoteClient.Client <init>
```

```
INFO: GOT: remote Proxy[RemoteControllerInterface,RemoteObjectInvocationHandler[↩
    UnicastRef
    [liveRef: [endpoint:[192.168.56.1:50826](remote),objID:[27325302:142c2976014↩
        :-7fff, -6459316828993268569]]]]]]

[2013-12-05T12:49:59.865+0100] [glassfish 4.0] [INFO]
[] [] [tid: _ThreadID=99 _ThreadName=http-listener-1(2)] [timeMillis: ↩
    1386244199865] [levelValue: 800]
[[GOT: remote connected!]]

[2013-12-05T12:49:59.865+0100] [glassfish 4.0] [INFO]
[] [] [tid: _ThreadID=99 _ThreadName=http-listener-1(2)] [timeMillis: ↩
    1386244199865] [levelValue: 800]
[[Error Activator inizialized with client for 192.168.56.1 with installed tm=0]]

[2013-12-05T12:50:07.975+0100] [glassfish 4.0] [SEVERE]
[] [] [tid: _ThreadID=99 _ThreadName=Thread-4] [timeMillis: 1386244207975] [↩
    levelValue: 1000]
[[Dec 05, 2013 12:50:07 PM it.truestoryfactory.ken.JFI.RemoteClient.↩
    ErrorActivator isActivatedRemotly
INFO: trigger mode is SYNCHRONOUS (approved = false]]

[2013-12-05T12:51:12.979+0100] [glassfish 4.0] [INFO]
[] [PaymentService] [tid: _ThreadID=101 _ThreadName=http-listener-1(4)] [↩
    timeMillis: 1386244272979] [levelValue: 800]
[[Amount: 5.0]]

[2013-12-05T12:51:12.982+0100] [glassfish 4.0] [INFO]
[] [com.forest.handlers.PaymentHandler] [tid: _ThreadID=252 _ThreadName=__ejb-↩
    thread-pool8] [timeMillis: 1386244272982] [levelValue: 800]
[[[PaymentHandler] Response status 200]]

[2013-12-05T12:51:12.984+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=252 _ThreadName=__ejb-thread-↩
    pool8] [timeMillis: 1386244272984] [levelValue: 800]
[[Order id:1 - Status:2]]

[2013-12-05T12:51:12.995+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=252 _ThreadName=__ejb-thread-↩
    pool8] [timeMillis: 1386244272995] [levelValue: 800]
[[Order Updated!]]

[2013-12-05T12:51:12.996+0100] [glassfish 4.0] [INFO]
[] [com.forest.handlers.PaymentHandler] [tid: _ThreadID=252 _ThreadName=__ejb-↩
    thread-pool8] [timeMillis: 1386244272996] [levelValue: 800]
[[Payment Approved]]
```

```
[2013-12-05T12:51:13.000+0100] [glassfish 4.0] [INFO]
[] [com.forest.handlers.DeliveryHandler] [tid: _ThreadID=254 _ThreadName=__ejb-↩
    thread-pool10] [timeMillis: 1386244273000] [levelValue: 800]
[[Order #1 has been paid in the amount of 5. Order is now ready for delivery!]]

[2013-12-05T12:51:13.001+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=254 _ThreadName=__ejb-thread-↩
    pool10] [timeMillis: 1386244273001] [levelValue: 800]
[[Order id:1 - Status:3]]

[2013-12-05T12:51:13.011+0100] [glassfish 4.0] [INFO]
[] [com.forest.ejb.ShoppingCart] [tid: _ThreadID=254 _ThreadName=__ejb-thread-↩
    pool10] [timeMillis: 1386244273011] [levelValue: 800]
[[Order Updated!]]

[2013-12-05T12:51:44.915+0100] [glassfish 4.0] [INFO]
[] [com.forest.shipment.ejb.OrderBrowser] [tid: _ThreadID=100 _ThreadName=http-↩
    listener-1(3)] [timeMillis: 1386244304915] [levelValue: 800]
[[Message ID: ID:2-127.0.1.1(db:38:c2:f7:fc:35)-1-1386244273073]]

[2013-12-05T12:51:44.920+0100] [glassfish 4.0] [INFO]
[] [com.forest.shipment.ejb.OrderBrowser] [tid: _ThreadID=100 _ThreadName=http-↩
    listener-1(3)] [timeMillis: 1386244304920] [levelValue: 800]
[[Processing Order ID:2-127.0.1.1(db:38:c2:f7:fc:35)-1-1386244273073]]

[2013-12-05T12:51:57.959+0100] [glassfish 4.0] [SEVERE]
[] [] [tid: _ThreadID=102 _ThreadName=Thread-4] [timeMillis: 1386244317959] [↩
    levelValue: 1000]
[[Dec 05, 2013 12:51:57 PM it.truestoryfactory.ken.JFI.RemoteClient.↩
    ErrorActivator isActivatedRemotly
INFO: trigger mode is SYNCHRONOUS (approved = true]]

[2013-12-05T12:51:57.960+0100] [glassfish 4.0] [WARNING]
[ejb.system_exception] [javax.enterprise.system.container.ejb.com.sun.ejb.↩
    containers]
[tid: _ThreadID=102 _ThreadName=http-listener-1(5)] [timeMillis: 1386244317960] [↩
    levelValue: 900]
[[EJB5184:A system exception occurred during an invocation on EJB UserBean, ↩
    method: public com.forest.entity.Person com.forest.ejb.UserBean.↩
    getUserByEmail(java.lang.String)]]

[2013-12-05T12:51:57.960+0100] [glassfish 4.0] [WARNING]
[] [javax.enterprise.system.container.ejb.com.sun.ejb.containers]
```

```
[tid: _ThreadID=102 _ThreadName=http-listener-1(5)] [timeMillis: 1386244317960] [↩
    levelValue: 900]
[[javax.ejb.EJBException
    at com.sun.ejb.containers.EJBContainerTransactionManager.↩
        processSystemException(EJBContainerTransactionManager.java:748)
    at com.sun.ejb.containers.EJBContainerTransactionManager.completeNewTx(↩
        EJBContainerTransactionManager.java:698)
    at com.sun.ejb.containers.EJBContainerTransactionManager.postInvokeTx(↩
        EJBContainerTransactionManager.java:503)
    at com.sun.ejb.containers.BaseContainer.postInvokeTx(BaseContainer.java:4475)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:2009)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:1979)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(↩
        EJBLocalObjectInvocationHandler.java:220)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandlerDelegate.invoke(↩
        EJBLocalObjectInvocationHandlerDelegate.java:88)
    at com.sun.proxy.$Proxy349.getUserByEmail(Unknown Source)
    at com.forest.ejb.__EJB31_Generated__UserBean__Intf____Bean__.getUserByEmail(↩
        Unknown Source)
    at com.forest.web.UserController.login(UserController.java:59)
    at com.forest.web.UserController$Proxy$_$$_WeldClientProxy.login(Unknown ↩
        Source)
    ... (hidden for reason of space)
    at java.lang.Thread.run(Thread.java:744)
Caused by: javax.persistence.NoResultException
    at com.forest.ejb.UserBean.getUserByEmail(UserBean.java:46)
    ... (hidden for reason of space)
    ... 61 more

]]

[2013-12-05T12:51:57.961+0100] [glassfish 4.0] [WARNING]
[] [javax.enterprise.resource.webcontainer.jsf.lifecycle]
[tid: _ThreadID=102 _ThreadName=http-listener-1(5)] [timeMillis: 1386244317961] [↩
    levelValue: 900]
[[#{userController.login}: javax.ejb.EJBException
javax.faces.FacesException: #{userController.login}: javax.ejb.EJBException
    at com.sun.faces.application.ActionListenerImpl.processAction(↩
        ActionListenerImpl.java:118)
    at javax.faces.component.UICommand.broadcast(UICommand.java:315)
    at javax.faces.component.UIViewRoot.broadcastEvents(UIViewRoot.java:790)
    at javax.faces.component.UIViewRoot.processApplication(UIViewRoot.java:1282)
    at com.sun.faces.lifecycle.InvokeApplicationPhase.execute(↩
        InvokeApplicationPhase.java:81)
    at com.sun.faces.lifecycle.Phase.doPhase(Phase.java:101)
    ... (hidden for reason of space)
    at java.lang.Thread.run(Thread.java:744)
Caused by: javax.faces.el.EvaluationException: javax.ejb.EJBException
    at javax.faces.component.MethodBindingMethodExpressionAdapter.invoke(↩
        MethodBindingMethodExpressionAdapter.java:101)
    at com.sun.faces.application.ActionListenerImpl.processAction(↩
        ActionListenerImpl.java:102)
    ... 45 more
Caused by: javax.ejb.EJBException
    at com.sun.ejb.containers.EJBContainerTransactionManager.↩
        processSystemException(EJBContainerTransactionManager.java:748)
    at com.sun.ejb.containers.EJBContainerTransactionManager.completeNewTx(↩
        EJBContainerTransactionManager.java:698)
    at com.sun.ejb.containers.EJBContainerTransactionManager.postInvokeTx(↩
        EJBContainerTransactionManager.java:503)
    at com.sun.ejb.containers.BaseContainer.postInvokeTx(BaseContainer.java:4475)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:2009)
    at com.sun.ejb.containers.BaseContainer.postInvoke(BaseContainer.java:1979)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(↩
        EJBLocalObjectInvocationHandler.java:220)
    at com.sun.ejb.containers.EJBLocalObjectInvocationHandlerDelegate.invoke(↩
        EJBLocalObjectInvocationHandlerDelegate.java:88)
    at com.sun.proxy.$Proxy349.getUserByEmail(Unknown Source)
    at com.forest.ejb.__EJB31_Generated__UserBean__Intf____Bean__.getUserByEmail(↩
        Unknown Source)
    at com.forest.web.UserController.login(UserController.java:59)
    at com.forest.web.UserController$Proxy$_$$_WeldClientProxy.login(Unknown ↩
        Source)
    ... (hidden for reason of space)
    ... 46 more
Caused by: javax.persistence.NoResultException
    at com.forest.ejb.UserBean.getUserByEmail(UserBean.java:46)
    ... (hidden for reason of space)
    ... 61 more

]]


[2013-12-05T12:52:03.201+0100] [glassfish 4.0] [SEVERE]
[] [com.forest.web.UserController] [tid: _ThreadID=100 _ThreadName=http-listener↩
    -1(3)]
[timeMillis: 1386244323201] [levelValue: 1000]
[[javax.servlet.ServletException: This is request has already been authenticated
    at org.apache.catalina.connector.Request.login(Request.java:2237)
    at org.apache.catalina.connector.Request.login(Request.java:2224)
    at org.apache.catalina.connector.RequestFacade.login(RequestFacade.java:1113)
    at com.forest.web.UserController.login(UserController.java:55)
    at com.forest.web.UserController$Proxy$_$$_WeldClientProxy.login(Unknown ↩
        Source)
    ... (hidden for reason of space)
    at java.lang.Thread.run(Thread.java:744)
]]
```

Bibliography

[1] E. W. Dijkstra, E. W. Dijkstra, and E. W. Dijkstra, *Notes on structured programming.* Technological University Eindhoven Netherlands, 1970. (document)

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, 2004. 1.1, 1.2

[3] R. Natella, "Achieving representative faultloads in software fault injection," Ph.D. dissertation, Università di Napoli Federico II, 2011. 1.2

[4] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, R. Dancey, A. Robinson, and T. Lin, "Fiat-fault injection based automated testing environment," in *Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on.* IEEE, 1988, pp. 102–107. 1.2

[5] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "Ferrari: A flexible software-based fault and error injection system," *Computers, IEEE Transactions on*, vol. 44, no. 2, pp. 248–260, 1995. 1.2

[6] S. Han, K. G. Shin, and H. A. Rosenberg, "Doctor: An integrated software fault injection environment for distributed real-time systems," in *Computer Performance and Dependability Symposium, 1995. Proceedings., International.* IEEE, 1995, pp. 204–213. 1.2

[7] J. Carreira, H. Madeira, and J. G. Silva, "Xception: A technique for the experimental evaluation of dependability in modern computers," *Software Engineering, IEEE Transactions on*, vol. 24, no. 2, pp. 125–136, 1998. 1.2

[8] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990. 1.2

[9] A. K. Ghosh, M. Schmid, and V. Shah, "Testing the robustness of windows nt software," in *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on.* IEEE, 1998, pp. 231–235. 1.2

[10] P. Koopman and J. DeVale, "The exception handling effectiveness of posix operating systems," *Software Engineering, IEEE Transactions on*, vol. 26, no. 9, pp. 837–848, 2000. 1.2

[11] J. Arlat, J.-C. Fabre, and M. Rodríguez, "Dependability of cots microkernel-based systems," *Computers, IEEE Transactions on*, vol. 51, no. 2, pp. 138–163, 2002. 1.2

[12] J. Hudak, B.-H. Suh, D. Siewiorek, and Z. Segall, "Evaluation and comparison of fault-tolerant software techniques," *Reliability, IEEE Transactions on*, vol. 42, no. 2, pp. 190–204, 1993. 1.2

[13] W.-I. Kao, R. K. Iyer, and D. Tang, "Fine: A fault injection and monitoring environment for tracing the unix system behavior under faults," *Software Engineering, IEEE Transactions on*, vol. 19, no. 11, pp. 1105–1118, 1993. 1.2

[14] W.-L. Kao and R. K. Iyer, "Define: A distributed fault injection and monitoring environment," in *Fault-Tolerant Parallel and Distributed Systems, 1994., Proceedings of IEEE Workshop on.* IEEE, 1994, pp. 252–259. 1.2

[15] J. A. Duraes and H. S. Madeira, "Emulation of software faults: a field data study and a practical approach," *Software Engineering, IEEE Transactions on*, vol. 32, no. 11, pp. 849–867, 2006. 1.2, 1.3, 1.3.1

[16] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978. 1.3

[17] M. Daran and P. Thévenod-Fosse, "Software error analysis: a real case study involving real faults and mutations," in *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 3. ACM, 1996, pp. 158–171. 1.3

[18] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?[software testing]," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on.* IEEE, 2005, pp. 402–411. 1.3

[19] T. Lindholm, F. Yellin, B. G., and A. Buckley, "The java virtual machine specification, java se 7 edition," http://docs.oracle.com/javase/specs/jvms/se7/jvms7.pdf, 2013. 1.3, 1

[20] T. Basso, R. Moraes, B. P. Sanches, and M. Jino, "An investigation of java faults operators derived from a field data study on java software faults," in *Workshop de Testes e Tolerância a Falhas*, 2009, pp. 1–13. 1.3, 1.3.1

[21] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *Software Engineering, IEEE Transactions on*, vol. 18, no. 11, pp. 943–956, 1992. 3

[22] R. Natella, D. Cotroneo, J. Duraes, and H. Madeira, "On fault representativeness of software fault injection," *Software Engineering, IEEE Transactions on*, vol. 39, no. 1, pp. 80–96, 2013. 2.1, 2.1.4

[23] A. V. Aho *et al.*, *Compilers: principles, techniques, & tools.* Pearson Education India, 2007. 2.2.5

[24] A. Common, "The byte code engineering library," http://commons.apache.org/proper/commons-bcel/, 2013. 3.1

[25] E. Jendrock, R. Cervera-Navarro, I. Evans, K. Haase, W. Markito, and C. Srivathsa, "The java ee 7 tutorial," http://docs.oracle.com/javaee/7/tutorial/doc/, 2013. 4.1, 4.2

[26] B. A. SeleniumHQ, "Selenium ide," http://docs.seleniumhq.org/projects/ide/, 2013. 4.2

[27] B. W. Johnson, "Fault-tolerant microprocessor-based systems." *IEEE Micro*, vol. 4, no. 6, pp. 6–21, 1984. 4.4

[28] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation," in *Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, 1978, pp. 3–9.