ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

JASS – Java with ASSertions

Detlef Bartetzko, Clemens Fischer, Michael Möller, Heike Wehrheim

Presented by Florian Froese (froesef)

JASS – Java with ASSertions

- Design by contract extention for Java
- Java precompiler
- Dynamic runtime checks
- Violation of contracts results in a Java Exception

Universal and Existential Quantification

```
forall i : {0 .. seats.length-1} # !seats[i]
exists i : {0 .. seats.length-1} # !seats[i]
```

```
protected final boolean jassCheckExists_1_jass_examples_SimpleCinema() {
try {
          int i;
          for (i = 0; i <= seats.length - 1; i++) {
                    if (!seats[i])
                              return true;
          }
          return false;
} catch (ArrayIndexOutOfBoundsException jassException) {
          throw new jass.runtime.AssertionException(
                              "jass.examples.SimpleCinema", null, 33,
                              "Arraybounds in forall/exists expression violated ! (”
                                        + jassException.toString() + ")");
}
}
```

Preconditions

```
public void add(Object o) {
/** require [valid_object] o != null;
      [buffer_not_full] !full(); **/
```

```
/* precondition */
if (!(o != null))
        throw new jass.runtime.PreconditionException(
                        "jass.examples.Buffer", "add(java.lang.Object)", 29,
                        "valid_object");
if (!(!jassInternal_full()))
        throw new jass.runtime.PreconditionException(
                        "jass.examples.Buffer", "add(java.lang.Object)", 29,
                        "buffer_not_full");
```

Postconditions

- Special constructs: Old, changeonly, Result

```
/** ensure changeonly{out};
       [valid_object] Result != null; **/
```

```
/* postcondition */
if (!(jassResult != null))
          throw new jass.runtime.PostconditionException(
                              "jass.examples.Buffer", "remove()", 40, "valid_object");
if (!(in == jassOld.in && jass.runtime.Tool.arrayEquals(buf,
                    jassOld.buf)))
          throw new jass.runtime.PostconditionException(
                              "jass.examples.Buffer", "remove()", -1,
                              "Method has changed old value.");
```

Class invariant

```
/** invariant [range] 0 <= in - out && in - out <= buf.length;
      [valid_content] buf.length == 0
        || (forall i : {out%buf.length .. in%buf.length-1} #
            buf[i] != null); **/
```

```
private void jassCheckInvariant(String msg) {
        if (!(0 <= in - out && in - out <= buf.length))
                throw new jass.runtime.InvariantException("jass.examples.Buffer",
                                null, 64, "Exception occured " + msg
                                + " (jass.reflect.AssertionLabel@3645ce28)");
        if (!(buf.length == 0 || (jassCheckForAll_0_jass_examples_Buffer())))
                throw new jass.runtime.InvariantException("jass.examples.Buffer",
                                null, 65, "Exception occured " + msg
                                + " (jass.reflect.AssertionLabel@20e1bfee)");
}
```

Loop variant and invariant

```
for (int i = 0; i < seats.length; i++)
        /** invariant 0 <= i && i <= seats.length; **/
        /** variant seats.length-i **/
              if (!seats[i]) return true;
```

- Invariant has to hold at begin and end of every iteration and after loop termination
- Variant decreases with every iteration and cannot be smaller than 0

Checks

- `/** [check_not_zero]check z != 0 **/`
- Same functionality as assert() from Java

Rescue and retry

```
/** rescue catch (PreconditionException e) {
        if (e.label.equals("valid_object")) {
        o = new DefaultObject(); retry;}
        else throw e;
} **/
```

- Ability to rescue an assertion failure and ensure class integrity
- Ability to retry execution with changed parameters
- Translated to a try-catch statement in Java

Refinement checks

Rules for refinement:

- If the abstract method is applicable the concrete one can be invoked too.
- The concrete method is more deterministic then the abstract one.
- Whenever the invariant of the subclass holds the invariant of the superclass must be valid too.

Refinement checks – Implementation

```
public abstract class AbstractBuffer {
          public Object remove() {
                    /** require !empty(); **/
                    ...
                    /** ensure Old.contains(Result); **/
          }
}
```

- Concrete class has to implement the Refinement interface and implement a method jassGetSuperState()

Refinement checks – Implementation

```
public class Buffer extends AbstractBuffer implements jass.runtime.Refinement{
public Object remove() {
          /** require !empty(); **/
          ...
          /** ensure Old.contains(Result);
                              changeonly{count, out}
                              Result.equals(Old.store[Old.out]); **/
}

private AbstractBuffer jassGetSuperState() {
          int capacity = store.length;
          AbstractBuffer aState = new AbstractBuffer(capacity);
          for (int i = capacity + in - count; i < capacity + in; i++)
                    aState.add(store[i % capacity]);
          return aState;
}
}
```

Trace Assertions

- Define a valid order of method invocations
- Constrain method invocations
- Sequence of method calls -> trace
- CSP-like notation

Trace Assertions

- E.g. init().b -> init().e -> start().b -> start().e
- init() = init().b->init().e

```
/** invariant
trace(
        init() -> STOP
);
**/
```

Trace Assertions - Factorial

```
public class Factorial {
public int factorial(int value) {
/** require value > 0; **/
if (value == 1)
        return 1;
else
        return value * factorial(value - 1);
/** ensure Result > 0; **/
}

/** invariant  on next slide... */

}
```

Trace Assertions - Factorial

```
/** invariant [variant] trace (
        MAIN() {
                int value;
                factorial(?value).b -> CALL Decrease(value)
        }
        Decrease(int variant) {
                int nextVariant;
                IF(variant < 0) {
                        EXECUTE(throw new RuntimeException("…");)
                        -> STOP
                } ELSE {
                        ->factorial(?nextVariant).b WHERE(nextVariant<variant)
                        -> CALL Decrease(nextVariant)
                }
        }
        );
**/
```

Trace Assertions - Operators

- Choice:
  - a() -> b() -> STOP <|> a() -> c() -> STOP
  - Valid traces:
    - a()
    - a() b()
    - a() c()
- Parallel:
  - CALL P() |<>| CALL Q()
  - P() { a() -> b() -> STOP}
  - Q() { a() -> c() -> STOP}
  - Valid traces:
    - As above
    - a() b() c()
    - a () c() b()

Trace Assertions – Trace alphabet

- Only those methods are monitored that are in the alphabet
- Implicitly given by trace assertion
- Explicitly specified:

```
trace{init(), start()} (
        init() -> STOP
)
```

Trace Assertions – Trace alphabet

```
/** invariant  trace{public this.*} (
  MAIN() {
   init() -> CALL Initialized()
  }

  Initialized() {
   * EXCEPT init() -> CALL Initialized()
  }
 );
**/
```

Additional Features

- Interference Checks
- JavaDoc Support

Conclusion and future work

- Good concepts
- Trace assertions are nice
- IDE integration would be handy(no more .jass files)

- JASS 3 & Modern Jass (annotations)
- For further information visit:
- http://csd.informatik.uni-oldenburg.de/~jass/index.html