LASIGE reliable software systems

LiquidJava:

Extending Java with Liquid Types

Catarina Gamboa

Paulo Canelas

Christopher Timperley

Jonathan Aldrich

Alcides Fonseca

Carnegie Mellon Portugal

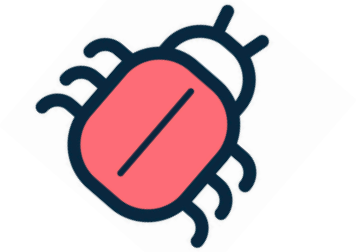
S3D

U LISBOA

Ciências ULisboa

FCT Fundação para a Ciência e a Tecnologia

Introduction



Software Quality

Software Verification

Strongly-typed programming languages

```
int x = "hello world!";
```

LASIGE reliable software systems

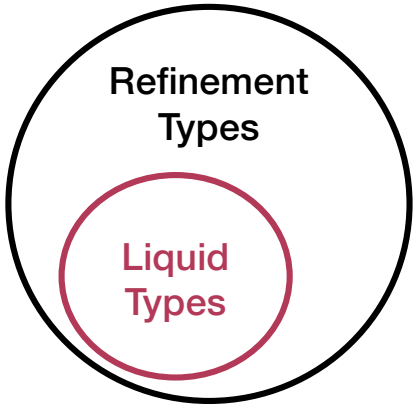
Refinement Types

```
@Refinement("0 <= red && red <= 255")
int red;
red = 70;
red = -1;
```

Logical Predicate

Verified at compile-time

Detect:
- Division by Zero
- Out-of-Bounds Array Access
- Security Issues
- Protocol violations



Refinement Types

LASIGE reliable software systems

Strongly typed functional languages


```
{-@ head :: [a] -> a @-}
head (x:_) = x
```

Refinement Types

LASIGE reliable software systems

Mainstream languages

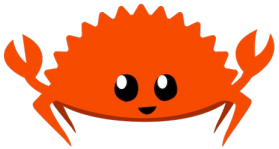




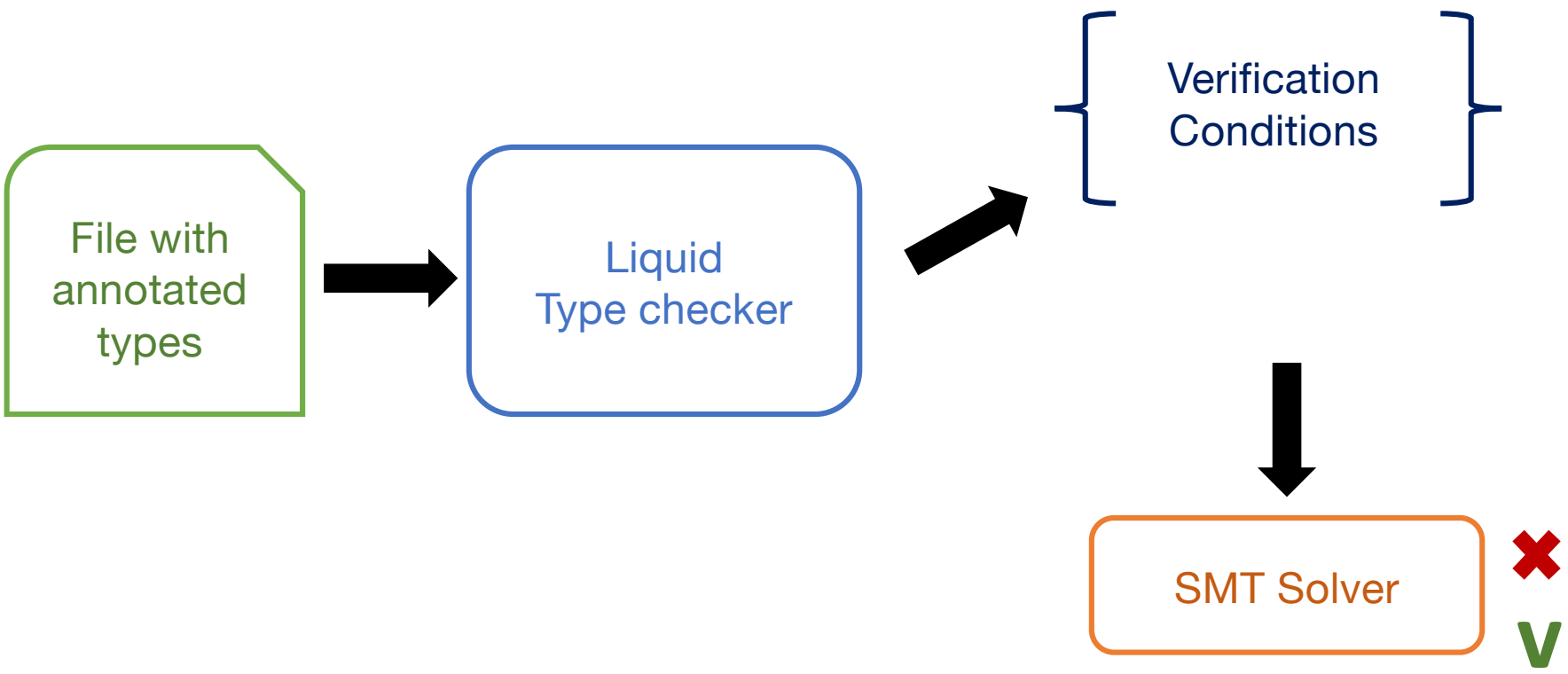
Streams

Object-Oriented aspects

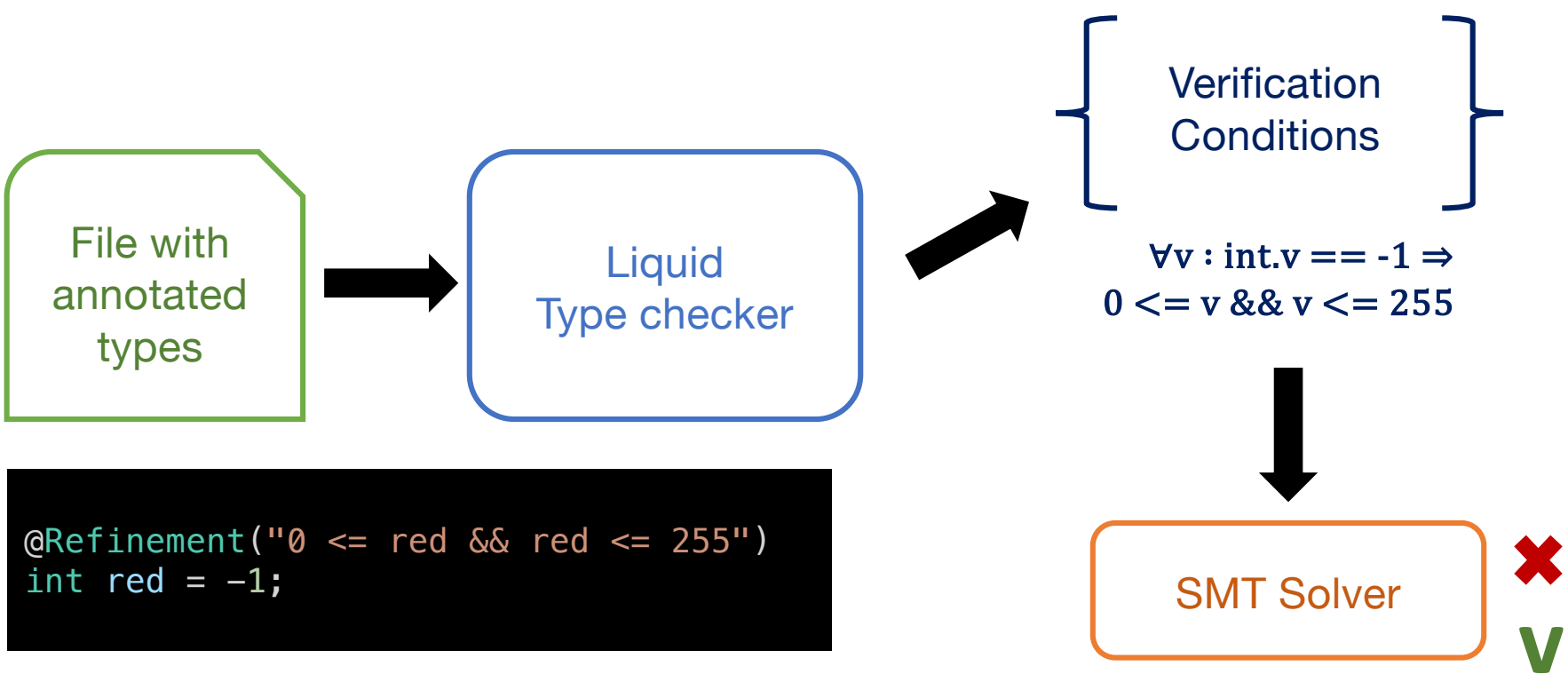
Popular languages



Liquid Type Checking - example



Liquid Type Checking - example




```
@Refinement("0 <= red && red <= 255")
int red = -1;
```

Liquid Types can add a verification layer on top of a language's type checker so users can verify more properties of their code

LASIGE reliable software systems

PROBLEM

Carnegie Mellon Portugal

S3D U LISBOA

Ciências ULisboa

FCT Fundação para a Ciência e a Tecnologia

Problem

Refinement Types are not widely adopted yet

Lack of Awareness

Lack of Experience

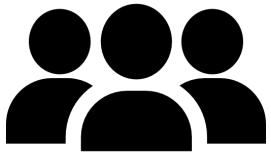
Not popular languages

Goal

LASIGE reliable software systems

Improve the Usability of Liquid Types

Exploring Barriers to Using Liquid Types



Current
Former
New

Users

User-Centered Implementation of Liquid Types in Java

Guide the design and evaluation of Liquid Java

Goal

LASIGE reliable software systems

Improve the Usability of Liquid Types

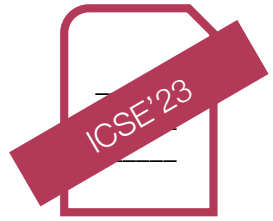
Exploring Barriers to Using Liquid Types

Current
Former
New

Users

User-Centered Implementation of Liquid Types in Java

Guide the design and evaluation of Liquid Java



LASIGE reliable software systems

Carnegie Mellon Portugal

S3D

U LISBOA

Ciências ULisboa

FCT Fundação para a Ciência e a Tecnologia

Verification in Java

LASIGE reliable software systems

Model Checking

JPF

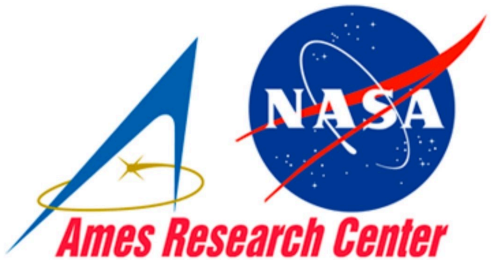
Design by Contract

OpenJML

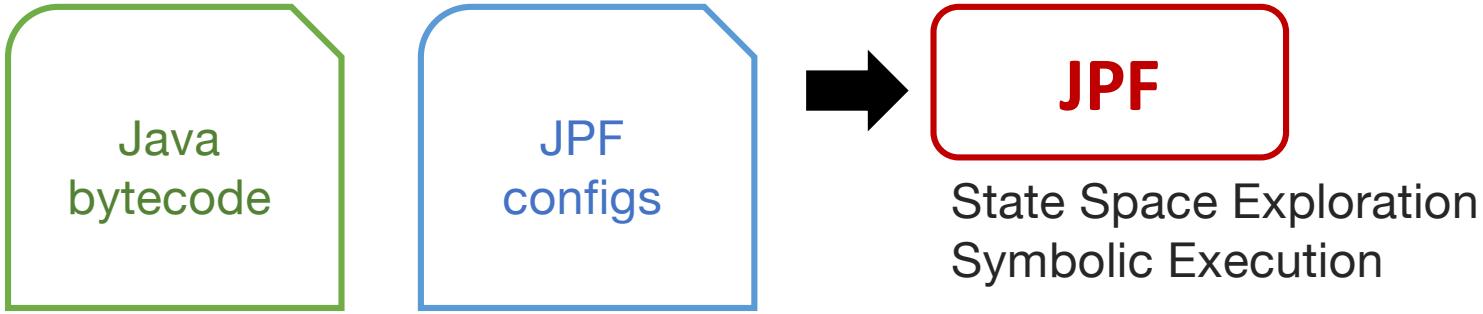
Java Path Finder – Model Checking

LASIGE reliable software systems

Open-source tool for model checking Java bytecode



Finds **deadlocks, data races, and assertion violations** by systematically exploring different execution paths



State Space Exploration
Symbolic Execution
…

Java Path Finder – Model Checking

LASIGE reliable software systems

```
import java.util.Random;

public class Rand {
  public static void main (String[] args) {
    System.out.println("computing c = a/(b+a - 2)..");
    Random random = new Random(42);       // (1)


    int a = random.nextInt(2);            // (2)
    System.out.printf("a=%d\n", a);

    //... lots of code here

    int b = random.nextInt(3);            // (3)
    System.out.printf("  b=%d       ,a=%d\n", b, a);

    int c = a/(b+a -2);                   // (4)
    System.out.printf("=>  c=%d     , b=%d, a=%d\n", c, b, a);
    }
}
```

From
https://blog.devgenius.io/introduction-to-java-pathfinder-80828b53309a

OpenJML – Design by Contract

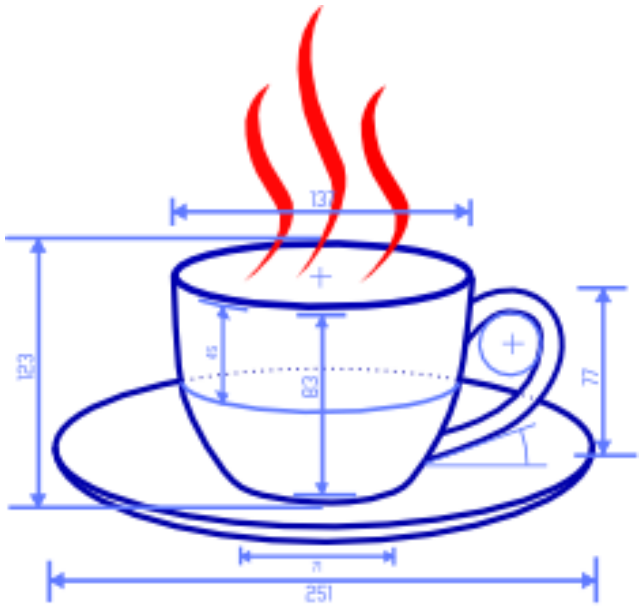
LASIGE reliable software systems

Java Modeling Language (JML) is a language for writing specifications

- @requires
- @ensures
- @invariant
- @ghost

OpenJML checks the implementation against the specification using:

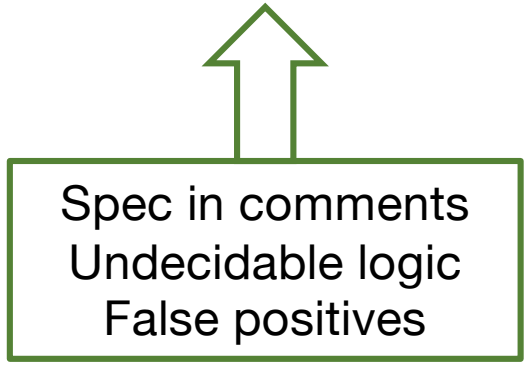
- Deductive Verification or
- ESC (extended static checking)



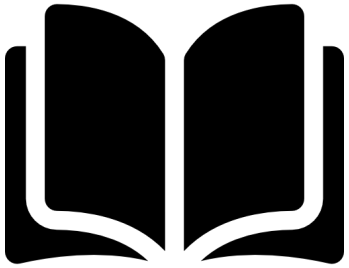
OpenJML – Design by Contract

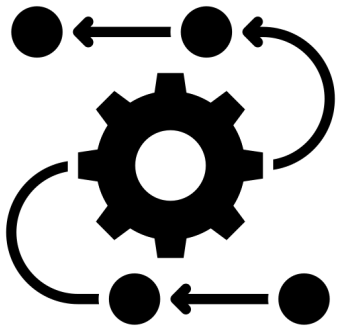
LASIGE reliable software systems

```
public class BinarySearchGood {
      //@ requires sortedArray != null && 0 < sortedArray.length < Integer.MAX_VALUE;
      //@ requires \forall int i; 0 <= i < sortedArray.length; \forall int j; i < j < sortedArray.length;
      sortedArray[i] <= sortedArray[j];
      //@ old boolean containsValue = (\exists int i; 0 <= i < sortedArray.length; sortedArray[i] == value);
      //@ ensures containsValue <==> 0 <= \result < sortedArray.length;
      //@ ensures !containsValue <==> \result == -1;
      //@ pure
      public static int search(int[] sortedArray, int value) {
            //@ ghost boolean containsValue = (\exists int i; 0 <= i < sortedArray.length; sortedArray[i] == value);
            if (value < sortedArray[0]) return -1;
            if (value > sortedArray[sortedArray.length-1]) return -1;
            int lo = 0;
            int hi = sortedArray.length-1;
            //@ loop_invariant 0 <= lo < sortedArray.length && 0 <= hi < sortedArray.length;
            //@ loop_invariant containsValue ==> sortedArray[lo] <= value <= sortedArray[hi];
            //@ loop_invariant \forall int i; 0 <= i < lo; sortedArray[i] < value;
            //@ loop_invariant \forall int i; hi < i < sortedArray.length; value < sortedArray[i];
            //@ loop_decreases hi - lo;
            while (lo <= hi) {
                  int mid = lo + (hi-lo)/2;
                  if (sortedArray[mid] == value) {
                        return mid;
                  } else if (sortedArray[mid] < value) {
                        lo = mid+1;
                  } else {
                        hi = mid-1;
                  }
            }
            return -1;
      }}
```



LiquidJava Design





LiquidJava Design Requirements

LASIGE reliable software systems

1. Refinements are optional;

```
@Refinement("x > 0")
```

2. Idiomatic Refinements

Syntax based on input of Java developers

3. Refinement type-checking should be decidable

Limit the language to Liquid Types (P. Rondon, M. Kawaguchi, R. Jhala, 2008)

4. Refinement type-checking works on top of existing Java type-checker

Uses the Eclipse Compiler Kit, via Spoon

LASIGE reliable software systems

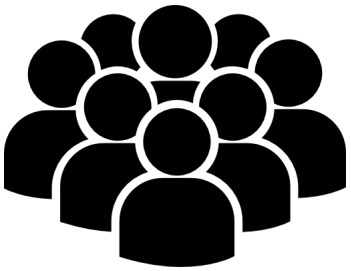
DESIGN WITH DEVELOPERS' FEEDBACK

Carnegie Mellon Portugal · S3D · U LISBOA · Ciências ULisboa · FCT Fundação para a Ciência e a Tecnologia

Language Design

LASIGE reliable software systems

```
@Refinement("x > 0")
```



Refinements in Methods

In this section we present syntax examples for refinements in methods, which includes refinements for the parameters and the return value.
These refinements express the following conditions:
- grade, the first parameter, is an int greater than or equal to 0;
- scale, the second parameter, is a positive int;
- the return value must be an int between 0 and 100.

Analyse each of the examples below.

A

```
@Refinement("\\v >= 0 && \\v <= 100")
public static int percentageFromGrade (@Refinement("grade >= 0") int grade,
                                       @Refinement("scale > 0")  int scale)
```

B

```
@Refinement("{grade >= 0} -> {scale > 0} -> {\\v >= 0 && \\v <= 100}")
public static int percentageFromGrade (int grade, int scale)
```

Evaluate your preference on each of the above syntaxes.

	Not acceptable	Acceptable	Preferable
A	○	○	○
B	○	○	○

Which one would you choose?

LASIGE reliable software systems

```
int percentageFromGrade(int grade, int scale)
```

A

```
@Refinement("return >= 0 && return <= 100")
int percentageFromGrade(@Refinement("grade >= 0")int grade,
                        @Refinement("scale > 0") int scale)
```

B

```
@Refinement("{grade >= 0}->{scale > 0}->{\\v >= 0 && \\v <= 100 }")
int percentageFromGrade(int grade, int scale)
```

LiquidJava Syntax Survey

LASIGE reliable software systems

50

Accepted Answers

52%

Very Familiar with **Java**

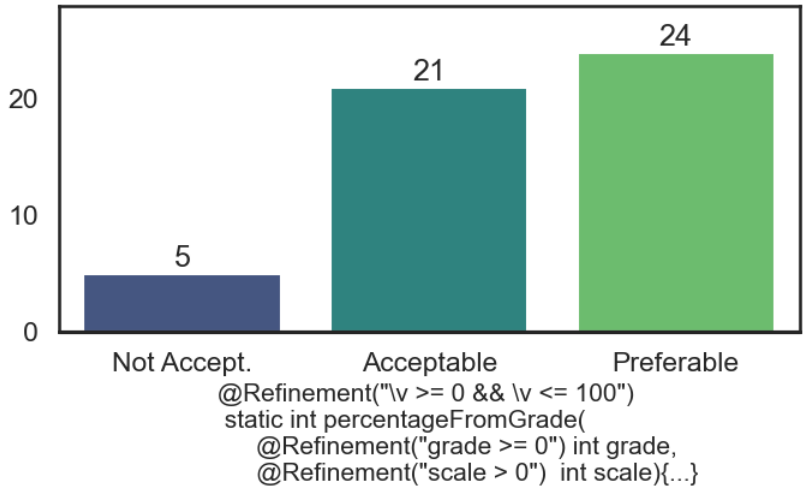
56%

Not Familiar **Refinement Types**

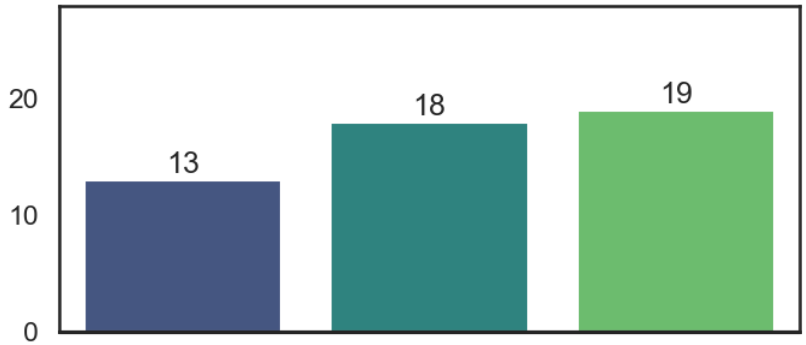
Features

- Variables
- Methods
- Predicate Aliases
- Anonymous Variables

Refinements in Methods




```
@Refinement("\v >= 0 && \v <= 100")
static int percentageFromGrade(
    @Refinement("grade >= 0") int grade,
    @Refinement("scale > 0")  int scale){...}
```




```
@Refinement("{grade >= 0} -> {scale > 0} -> {\v >= 0 && \v <= 100}")
static int percentageFromGrade(int grade, int scale){...}
```

LiquidJava Specification

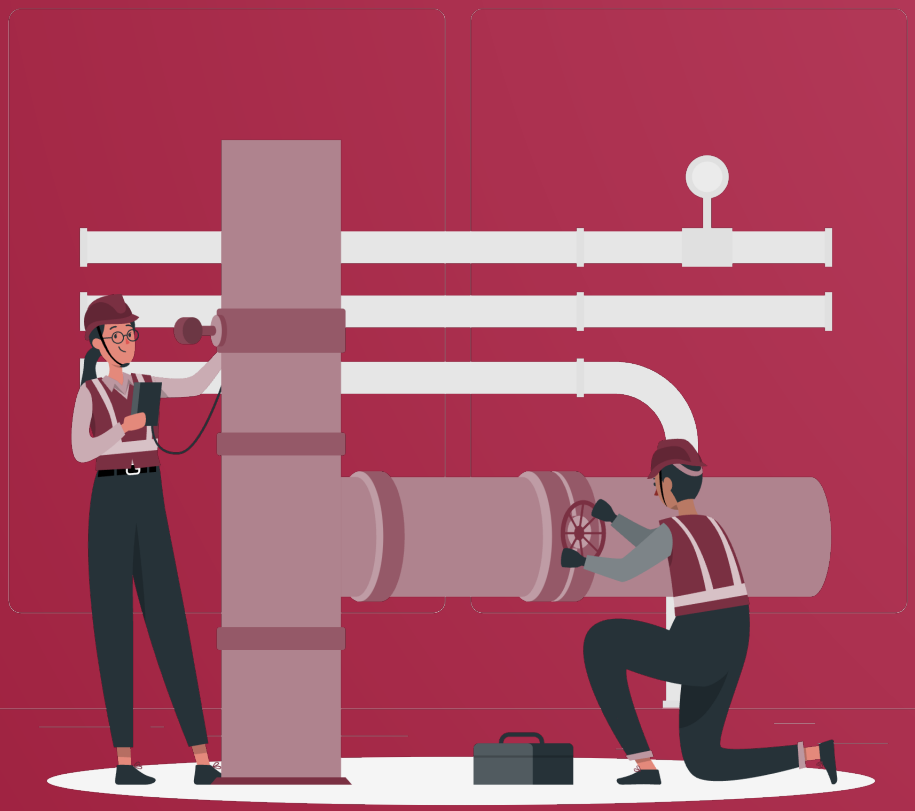
LASIGE reliable software systems

```
@Refinement("r >= 0 && r <= 255")
int r;
```

```
@RefinementAlias,
@Ghost,
@StateSet,
@StateRefinement
```

start	S	::=	$Pred \mid AliasDecl \mid GhostDecl;$
predicate	$Pred$	::=	Exp
		|	$Pred\ LOP\ Pred$
		|	$!\ Pred$
		|	$Pred\ ?\ Pred : Pred;$
expression	Exp	::=	$Exp\ BOP\ Exp$
		|	$Oper;$
operand	$Oper$	::=	$LitExp$
		|	$Oper\ AOP\ Oper$
		|	$UOP\ Oper;$
literal expression	$LitExp$	::=	c
		|	x
		|	_
		|	$x\ (\ Arg\ );$
argument	Arg	::=	$Pred$
		|	$Pred,\ Arg;$
ghost decl.	$GhostDecl$	::=	**ghost** $GhostDecl'$
		|	$GhostDecl'$
	$GhostDecl'$	::=	$T\ x(\ ArgDecl\ )$
		|	$T\ x;$
alias decl.	$AliasDecl$	::=	**type** $AliasDecl'$
		|	$AliasDecl';$
	$AliasDecl'$	::=	$x\ (\ ArgDecl\ )\ \{\ Pred\ \};$
argument decl.	$ArgDecl$	::=	$T\ x;$
logical operator	LOP	::=	&& | || ;
boolean operator	BOP	::=	> | >= | < | <= | == | != ;
unary operator	UOP	::=	! | + | − ;
arithmetic operator	AOP	::=	+ | − ;
Types	T	::=	**int** | **boolean**;

LASIGE reliable software systems



LIQUID JAVA VERIFICATION

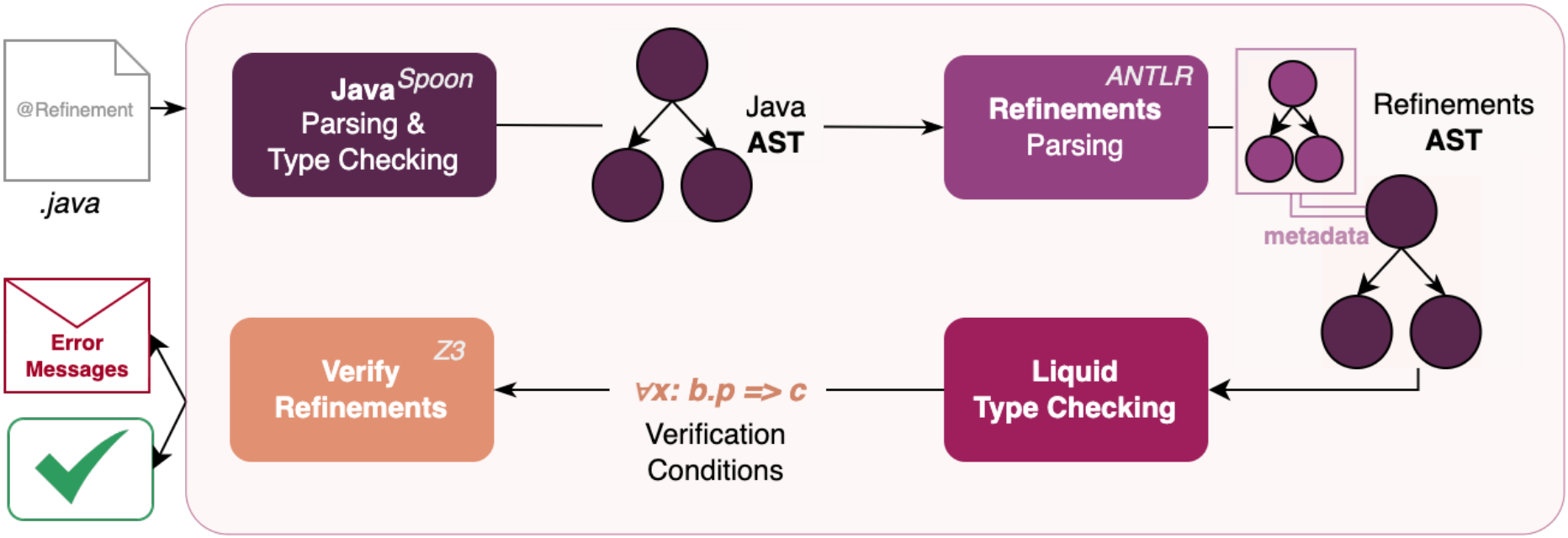
Carnegie Mellon Portugal

S3D U LISBOA

Ciências ULisboa

FCT Fundação para a Ciência e a Tecnologia

LiquidJava Verification



Liquid Type Checking - Variables

Refinement in Variables

```
@Refinement("a > 0")
int a = -10; //Error
```

Verification Condition

$\forall a : int.a == -10 \Rightarrow a > 0$

- Strong Updates

```
@Refinement("a > 0") int a = 10;
@Refinement("b > 15") int b = a + a;
a -= b; //Error
```

Refinement in Fields

```
@RefinementAlias("RGB(int x) {0 <= x && x <= 255}")
public class Color {
  @Refinement("RGB( _ )") private int r;
  @Refinement("RGB( _ )") private int g;
  @Refinement("RGB( _ )") private int b;
}
```

Liquid Type Checking - Methods

LASIGE reliable software systems

Refinement in Methods

```
@Refinement("_ >= a && _ <= b")
public static int inRange(int a, @Refinement("b > a") int b){
    return a + 1;
}
…
inRange(10, 9);
```

$\forall a : int.true \Rightarrow$
$\forall b : int.\ b > a \Rightarrow$
$\forall return : int.return == a + 1 \Rightarrow$
$return \geq a \wedge return \leq b$

- Recursion
- Inside the methods: **Path conditions**

```
@Refinement("_ >= 0 && _ >= n")
public static int sum(int n) {
  if(n < 1)
     return 0;
  else
     return n + sum(n-1);
}
```

$\forall n : int.true \Rightarrow$
$\forall fresh : int.\neg(n < 1) \Rightarrow$
$\forall return : int.return == n + sum1 \Rightarrow$
$\forall sum1 : int.sum1 \geq 0 \wedge sum1 \geq (n - 1) \Rightarrow$
$return \geq 0 \wedge return \geq n$

EXERCISE

Add a refinement to the variable *currentMonth* and to the function *Fibonacci*

```
/* A month needs to have a value between 1 and 12*/
int currentMonth;
currentMonth = 13; //Error
currentMonth = 5; //Correct
```

```
/**
* Computes the fibonacci of index n
* @param n The index of the required fibonacci number (greater or equal to 0)
* @return The fibonacci nth number. The fibonacci sequence follows the formula
Fn = Fn-1 + Fn-2 and has the starting values of F0 = 1 and F1 = 1
*/
public static int fibonacci(int n){
    if(n <= 1)
        return 0;
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

Liquid Type Checking - Classes

LASIGE reliable software systems

Refinement in Classes

Object Type State

- **Properties** – Ghost attributes

```
@Ghost("int size")
```

- **State transitions** – Protocol definition using State Machines

```
@StateSet({"open", "closed"})
```

```
@StateRefinement(from = state, to = state)
type class_method();
```

Liquid Type Checking - Classes

LASIGE reliable software systems

Refinement in Classes

Properties

```
@ExternalRefinementsFor("java.util.ArrayDeque")
@Ghost("int size")
public interface ArrayDequeRefinements<E> {

}
```

```
ArrayDeque<Integer> a =
     new ArrayDeque<Integer>();
a.remove();
```

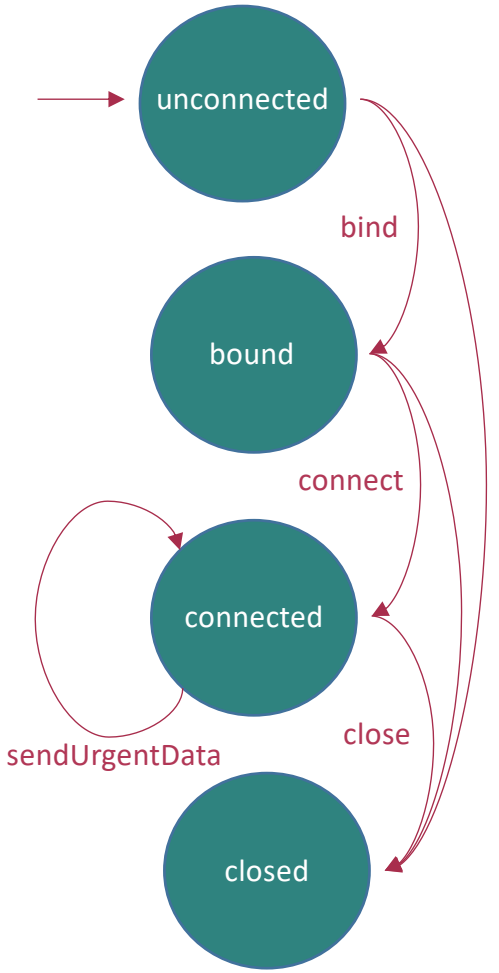
Liquid Type Checking - Classes

LASIGE reliable software systems

Refinement in Classes

State Transitions

```
@ExternalRefinementsFor("java.net.Socket")
@StateSet({"unconnected", "bound", "connected", "closed"})
public interface SocketRefinements {
```




```
}
```

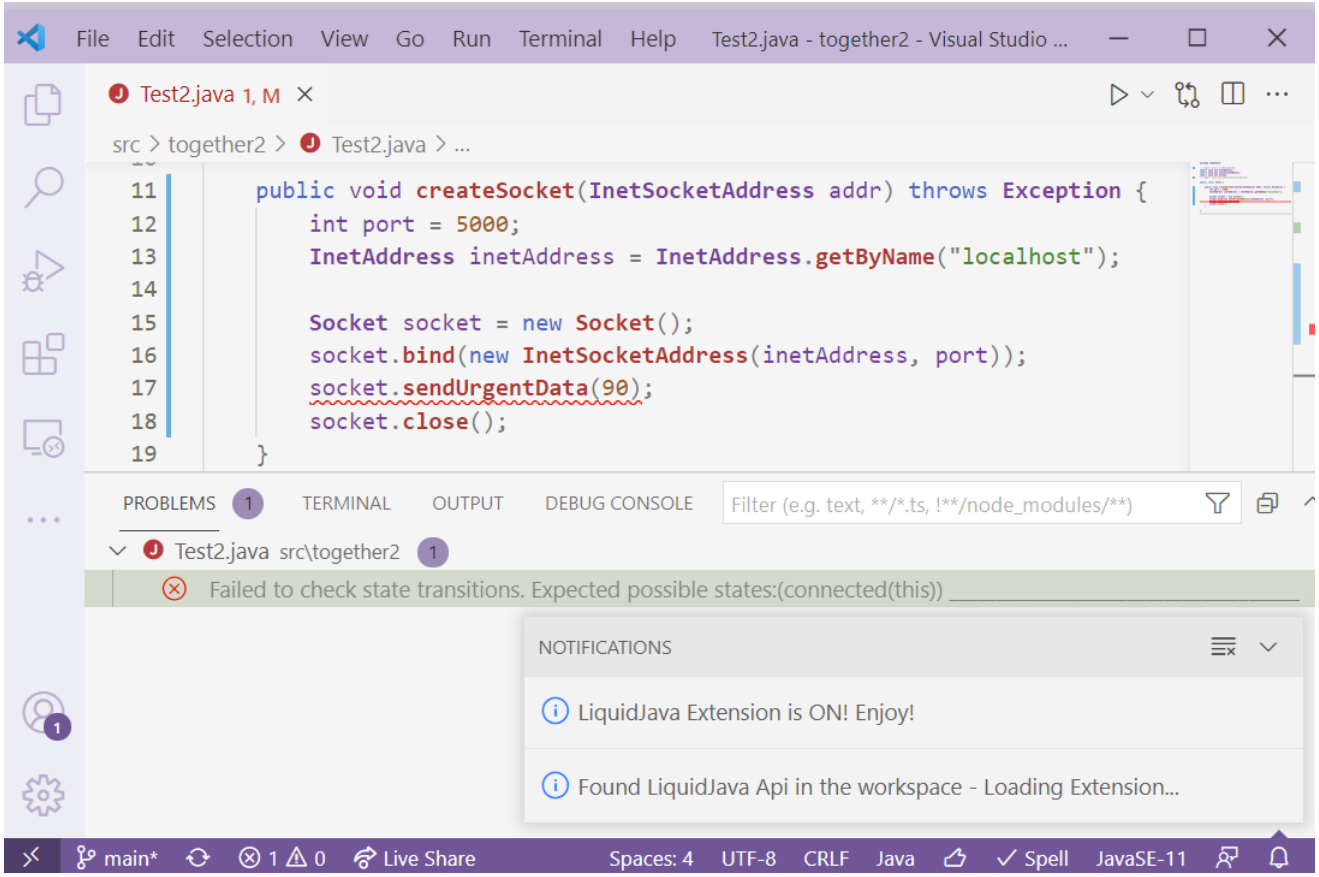
LiquidJava Implementation and Plugin

LASIGE reliable software systems

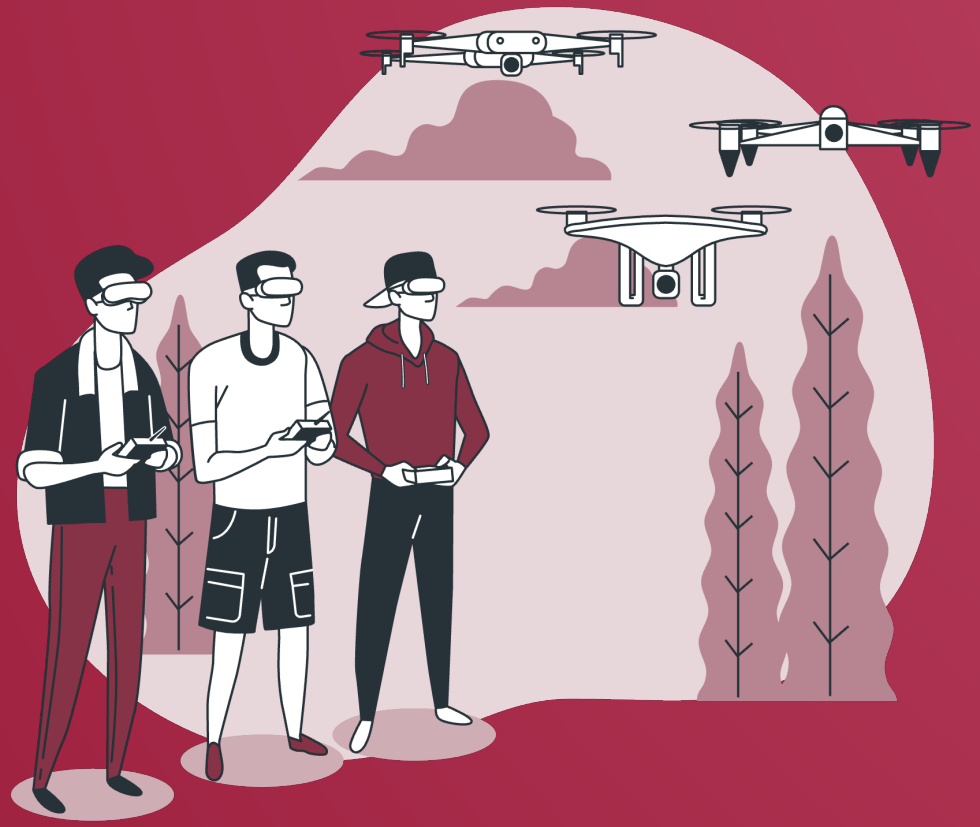
Visual Studio Code Plugin

LSP - Language Server Protocol

- Error reporting
- Error information



LASIGE reliable software systems



CASE STUDY

Project by
Catarina Gamboa and Simon Shu

Carnegie Mellon Portugal

S3D U LISBOA

Ciências ULisboa

FCT Fundação para a Ciência e a Tecnologia

Applied case study

Project by Catarina Gamboa and Simon Shu

LASIGE reliable software systems

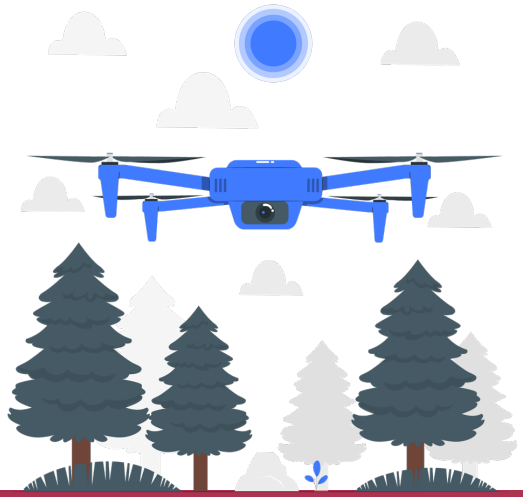
Testing techniques are still limited and expensive:

- Deployment in real-world hardware
- Simulation-based testing has expensive

1. Extracted state machine from documentation

2. Annotated flight controller API with LiquidJava

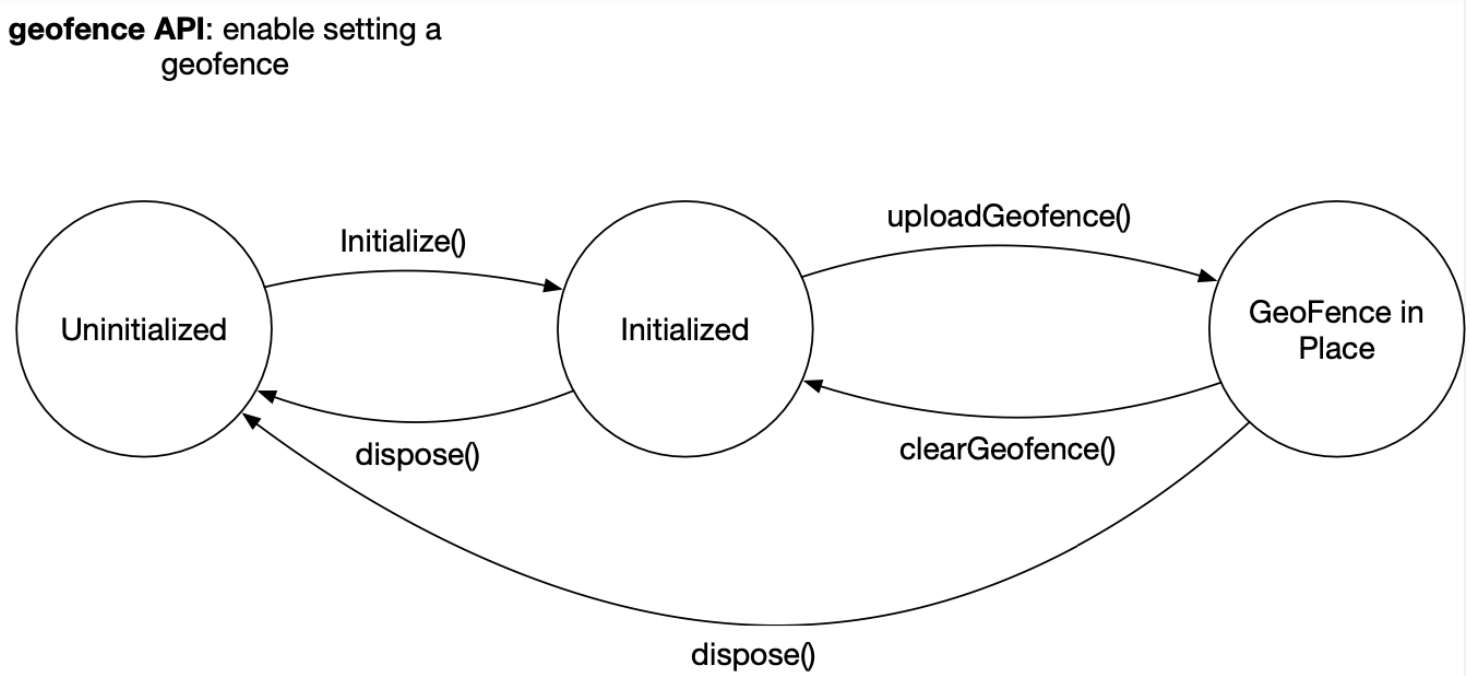
3. Statically analyzed controller code



Applied case study - Exercise

Project by Catarina Gamboa and Simon Shu

io.mavsdk.geofence.Geofence



Answer

```
@ExternalRefinementsFor("io.mavsdk.geofence.Geofence")
@StateSet({"geoUninitialized", "geoInitialized", "geoInPlace"})
public interface GeofenceRefinements {

    @StateRefinement(to="geoUninitialized(this)")
    void Geofence();

    @StateRefinement(from="geoUninitialized(this)", to="geoInitialized(this)")
    void initialize();

    @StateRefinement(from="!geoUninitialized(this)", to="geoUninitialized(this)")
    void dispose();

    @StateRefinement(from="geoInitialized(this)", to="geoInPlace(this)")
    io.reactivex.Completable uploadGeofence(java.util.List<Geofence.Polygon>
    polygons);


    @StateRefinement(from="geoInPlace(this)", to="geoInitialized(this)")
    io.reactivex.Completable clearGeofence();

}
```

LASIGE reliable software systems

Evaluation: User Study

Carnegie Mellon Portugal · S3D · U Lisboa · Ciências ULisboa · FCT Fundação para a Ciência e a Tecnologia

Research Questions

RQ1 Are refinements easy to understand?

RQ2 Is it easier and faster to find implementation errors using LiquidJava than with plain Java?

RQ3 How hard is it to annotate a program with refinements?

RQ4 Are developers open to using LiquidJava in their projects?

30

Familiar with Java

56% *Very Familiar* with Java

80% *Vaguely* or *Not Familiar* with Refinement Types

Study Configuration

Task 1 - Find the error in plain Java RQ2

Sum Socket — *Exercises* — Fibonacci ArrayDeque

Task 2 RQ1

Understand Refinements without prior explanation

Overview of LiquidJava

Task 3 - Find the error in LiquidJava RQ2

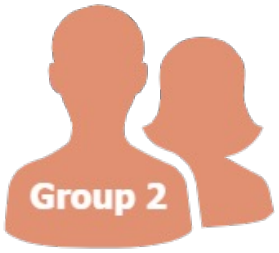
Fibonacci ArrayDeque — *Exercises* — Sum Socket

Task 4 RQ1+3

Annotate Java programs with LiquidJava

Comments RQ4

Describe what they liked and disliked about LiquidJava



Analysed Data

- Answers *(Correct, Incorrect, Compiler Correct, Unanswered)*
- Time spent
- Long Answers

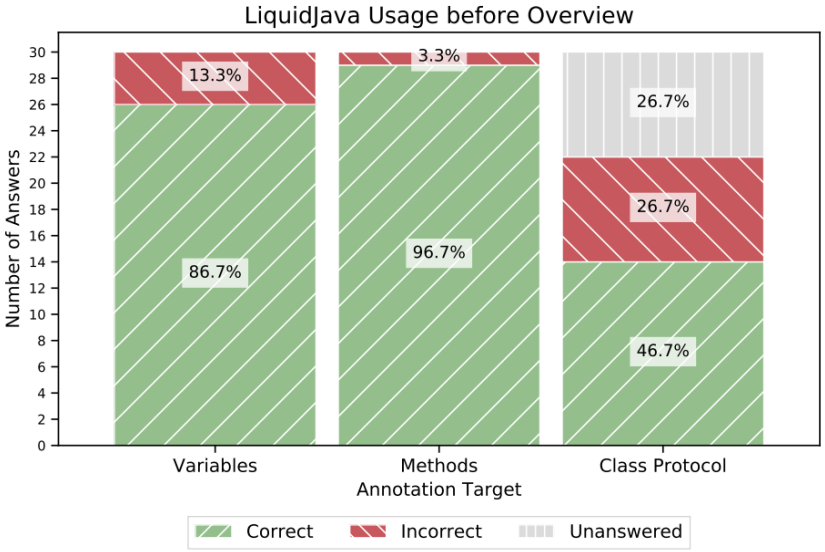
Evaluation

LASIGE reliable software systems

2 Understand the Refinements

RQ1

```
@Refinement("-25 <= x && x <= 45")
int x;
//Correct:
x = 0;
//Incorrect:
x = 46;
```

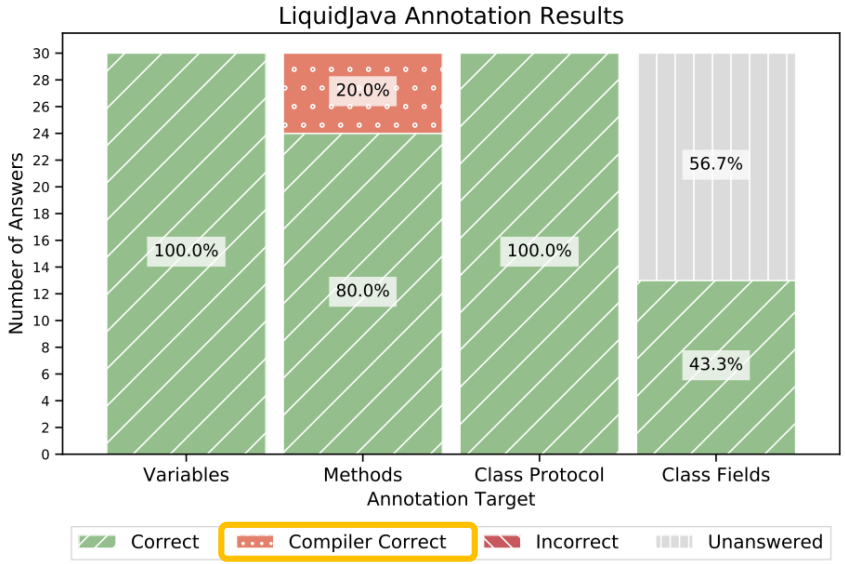


Overview

Video and website

4 Annotate with LiquidJava

RQ3



Evaluation

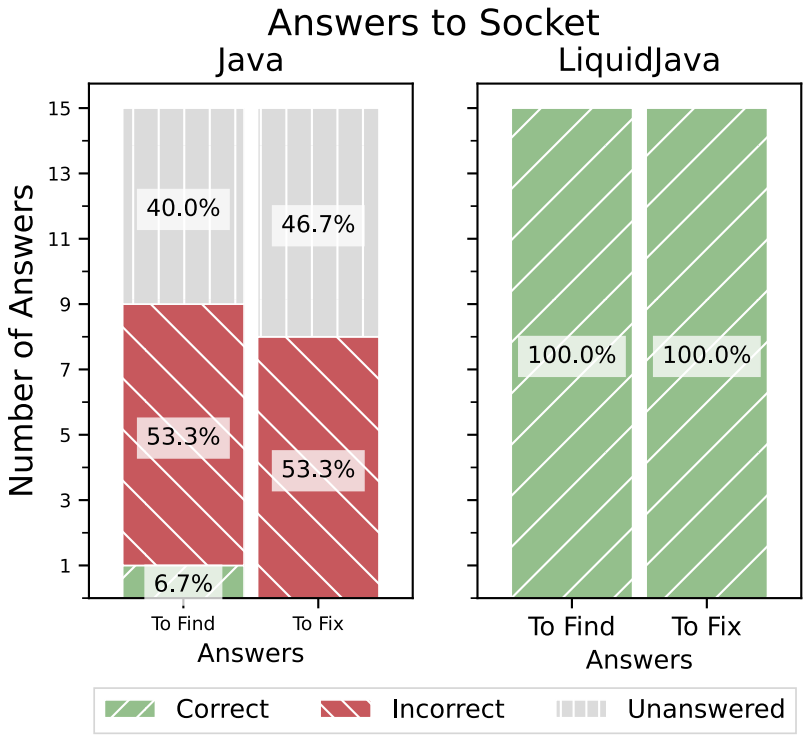
RQ2

1 Find Bug in plain Java

3 Find Bug with LiquidJava

```
public void createSocket (InetSocketAddress addr)
                                  throws IOException{
  int port = 5000;
  InetAddress inetAddress =
            InetAddress.getByName("localhost");

  Socket socket = new Socket();
  socket.bind(
            new InetSocketAddress(inetAddress, port));
  //missing socket.connect(addr);
  socket.sendUrgentData(90);
  socket.close();
}
```



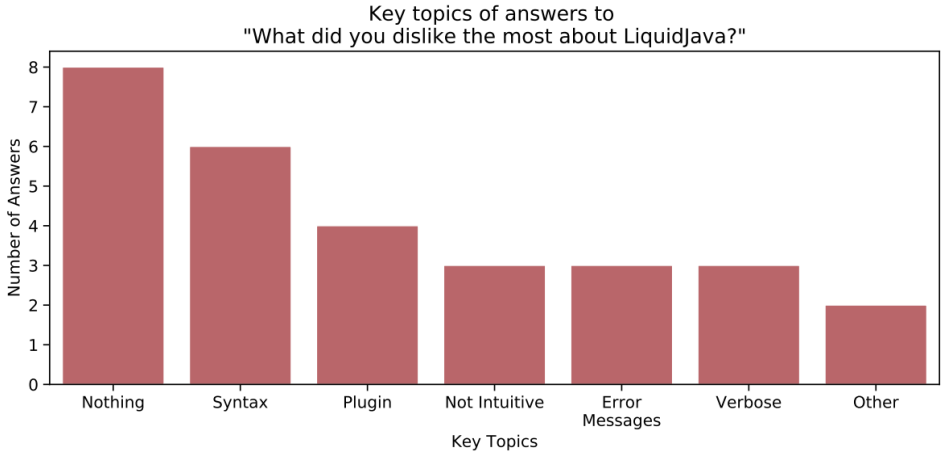
5 min. 35 sec.

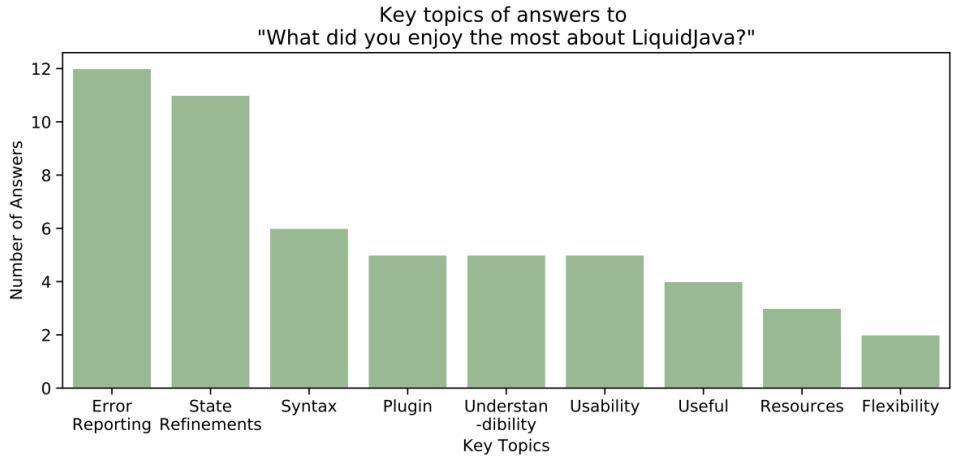
4 min. 42 sec.

Evaluation

LASIGE reliable software systems

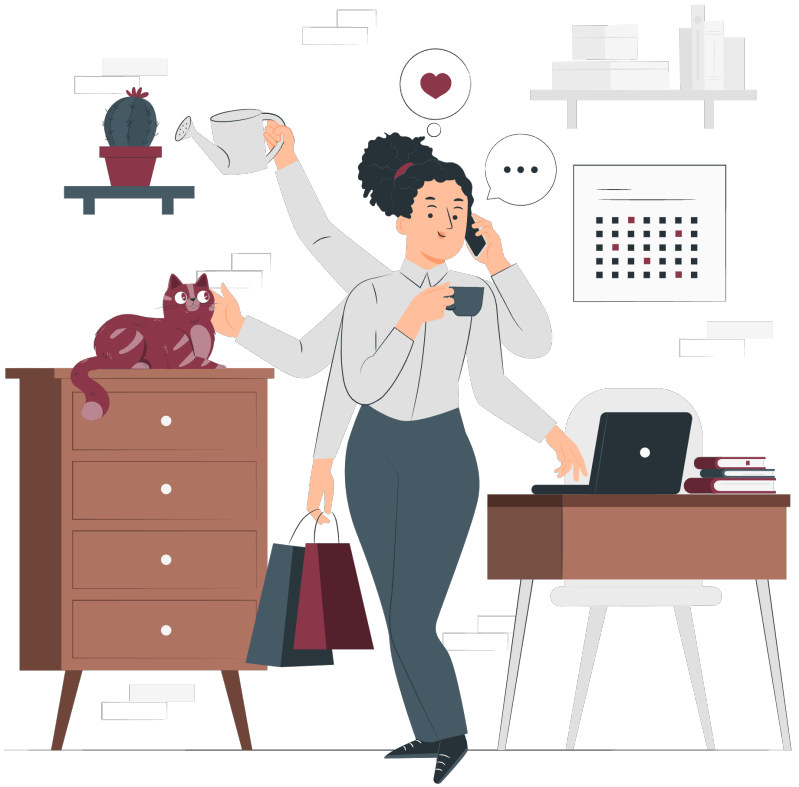
5 Final Overview





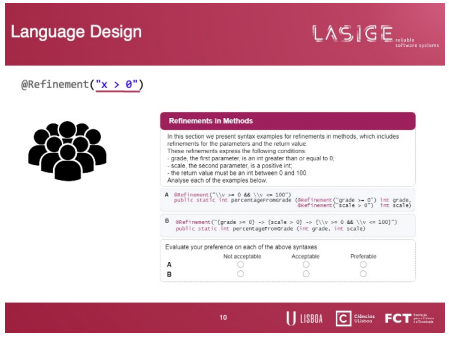
Current Work

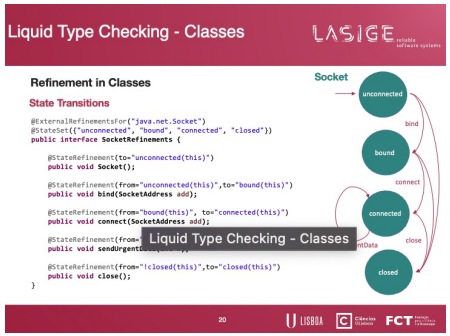
- Barriers across different Liquid Types Implementations
- Formalization of Liquid Types in Java in the presence of aliasing and mutability
- Improve Error messages and IDE Integration



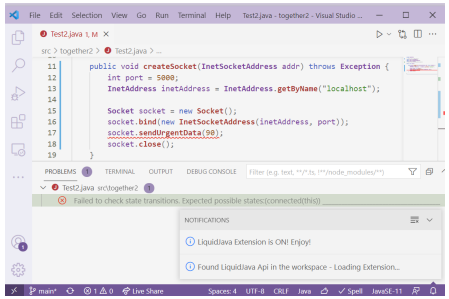
Conclusion

LASIGE reliable software systems

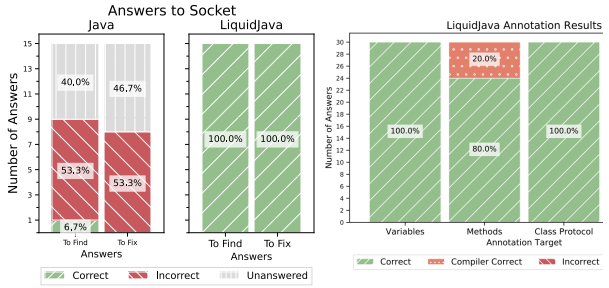


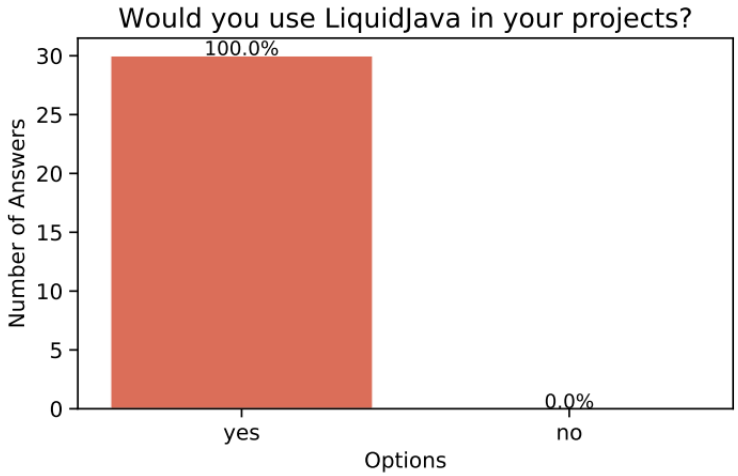


Implementation and IDE integration



Evaluation with 30 participants





LASIGE reliable software systems

LiquidJava:
Extending Java with Liquid Types

Contact me:
cgamboa@andrew.cmu.edu

Catarina Gamboa

Paulo Canelas

Christopher Timperley

Jonathan Aldrich

Alcides Fonseca

Carnegie Mellon Portugal · INESC-ID · U LISBOA · Ciências ULisboa · FCT Fundação para a Ciência e a Tecnologia

Thank you!

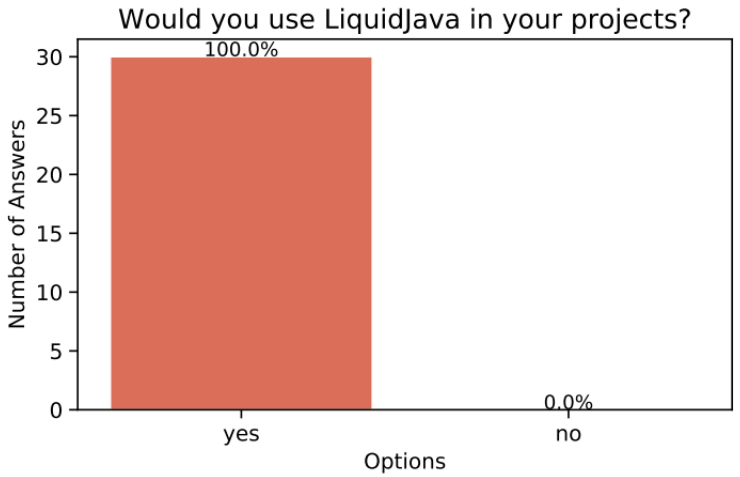
Contact me:
cvgamboa@fc.ul.pt

Project Website:
https://catarinagamboa.github.io/liquidjava.html
YouTube presentation:
https://www.youtube.com/watch?v=MOZESsOtUAU

Icons from Noun Project

Conclusion

- Extension of Java with Liquid Types, focusing on usability
  - Syntax of refinements guided by a survey of 50 Java developers
  - State transitions were added to model Java's OOP nature
- Implementation of a LiquidJava type-checker and its integration into an IDE Plugin
- Evaluation of LiquidJava with 30 participants
  - It was easier and faster to find and correct bugs in LiquidJava
  - Developers focused on silencing errors



Thank you!

Publications – Papers and Posters

LASIGE reliable software systems

Extending Java with Refinements
Catarina Gamboa, Paulo Santos, and Alcides Fonseca. 2020. Student Paper. In Program Semantics, Specification and Verification: Theory and Applications (PSSV-2020), November 2020

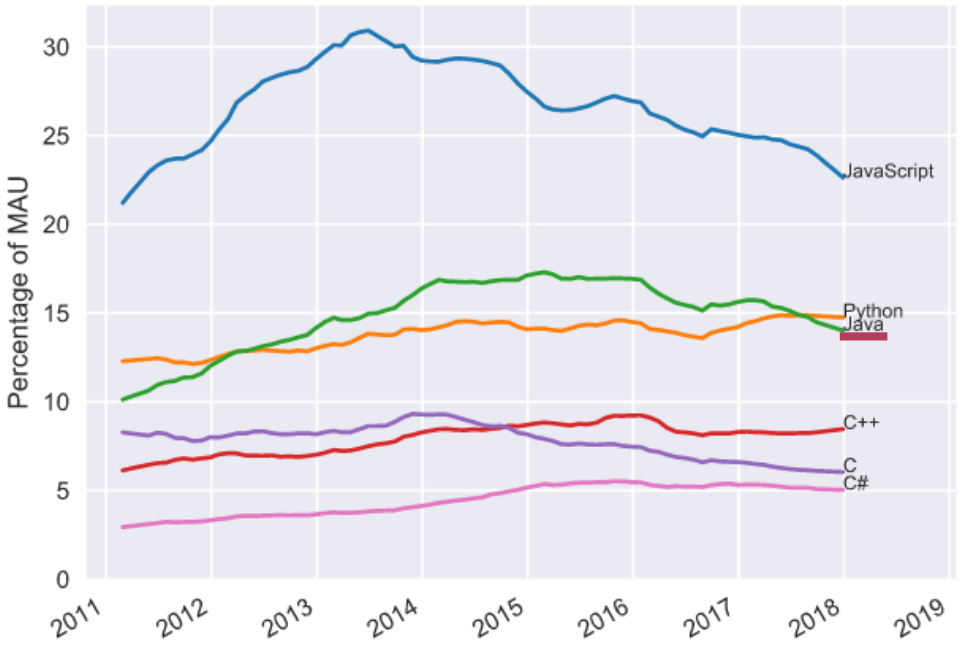
Poster - LiquidJava: Extending Java with Refinement Types
Catarina Gamboa, Paulo Santos, Christopher Timperley, and Alcides Fonseca. 2021. Poster. In LASIGE Workshop 2021, May 2021

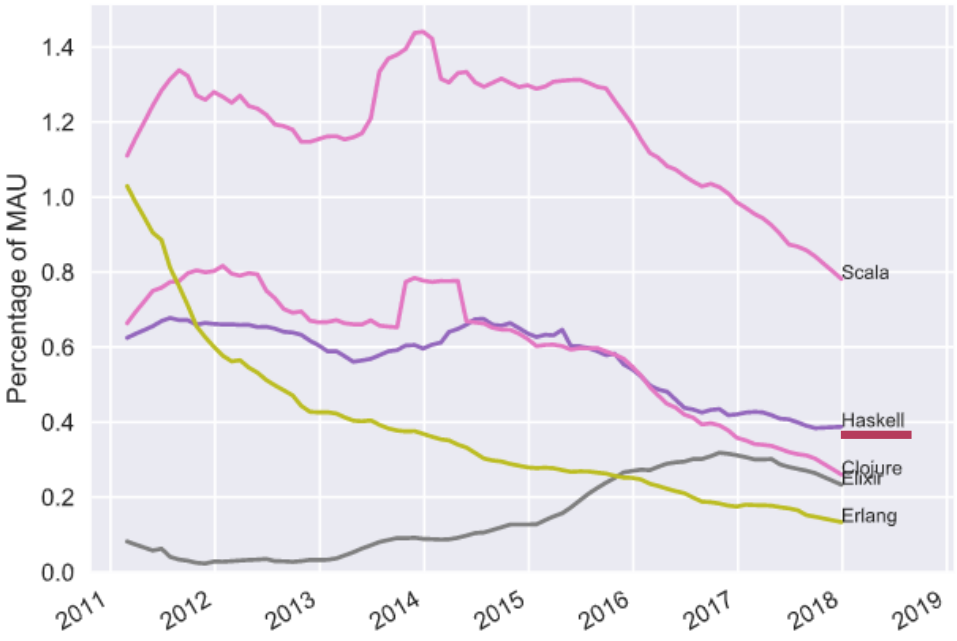
User-driven design and evaluation of Liquid Types in Java
Catarina Gamboa, Paulo Santos, Christopher S. Timperley, and Alcides Fonseca. 2021. Presentation in Human Aspects of Types and Reasoning Assistants Workshop (HATRA), co-located with SPLASH 2021, September 2021

LiquidJava: Adding Lightweight Verification to Java
Catarina Gamboa, Paulo Santos, Christopher S. Timperley, and Alcides Fonseca. 2021. Presented Paper. In INForum 2021 - Informatics Symposium, September 2021

Understandable and Useful Error Messages for Liquid Types
Alcides Fonseca, Catarina Gamboa, João David, Guilherme Espada, Paulo Canelas. 2021. Accepted talk at WITS - Workshop on the Implementation of Type Systems 2022, colocated with POPL 2022, January 2022

Java Popularity





https://www.benfrederickson.com/ranking-programming-languages-by-github-users/

Java Popularity

LASIGE reliable software systems

Rank	Change	Language	Share
1		Python	30.8 %
2		Java	16.79 %
3		JavaScript	8.37 %
4		C#	6.42 %
5		PHP	5.92 %
6		C/C++	5.78 %
7		R	4.16 %
8		Objective-C	3.57 %
9		Swift	2.29 %
10		TypeScript	1.84 %
11		Matlab	1.65 %
12		Kotlin	1.64 %
13	↑↑	Go	1.43 %
14	↓	Ruby	1.2 %
15	↓	VBA	1.11 %
16	↑↑	Rust	0.97 %
17	↓	Scala	0.87 %
18	↓	Visual Basic	0.78 %
19	↑↑↑↑	Ada	0.62 %
20	↑↑↑↑	Lua	0.58 %
21	↑	Dart	0.57 %
22	↓↓↓	Perl	0.47 %
23	↓↓↓	Abap	0.45 %
24	↓↓↓	Groovy	0.43 %
25	↑↑	Julia	0.41 %
26	↓	Cobol	0.32 %
27	↓	Haskell	0.29 %
28		Delphi	0.27 %

http://pypl.github.io/PYPL.html

Class Refinements
SCALA & TYPESCRIPT

LASIGE reliable software systems

```
1  type NonNeg = { v: Int => v >= 0 }
2  def range(lo: Int, x: Int, hi: Int) = lo <= x && x <= hi
3
4  abstract class IntArray(val length: NonNeg, init: Int) {
5   def access(i: {v: Int => range(0, v, this.length-1)}):
        Int }
```

Example from SMT-Based Checking of Predicate-Qualified Types for Scala
[Georg Stefan Schmid and Viktor Kuncak, 2016]

```
type pos      = {v:number | 0 < v}
type ArrayN<T,n> = {v:T[] | len(v) = n}
type grid<w,h>   = ArrayN<number,(w+2)*(h+2)>
type okW         = natLE<this.w>
type okH         = natLE<this.h>

class Field {
  immutable w : pos;
  immutable h : pos;
  dens        : grid<this.w, this.h>;

  constructor(w:pos,h:pos,d:grid<w,h>){
    this.h = h; this.w = w; this.dens = d;
  }
  setDensity(x:okW, y:okH, d:number) {
    var rowS = this.w + 2;
    var i    = x+1 + (y+1) * rowS;
    this.dens[i] = d;
  }
```

Example from
Refinement types for Typescript
[P. Vekris, B. Cosman, R. Jhala 2016]

LiquidJava and JML

LASIGE reliable software systems

- Refinements attached to the types
- Decidable Logic

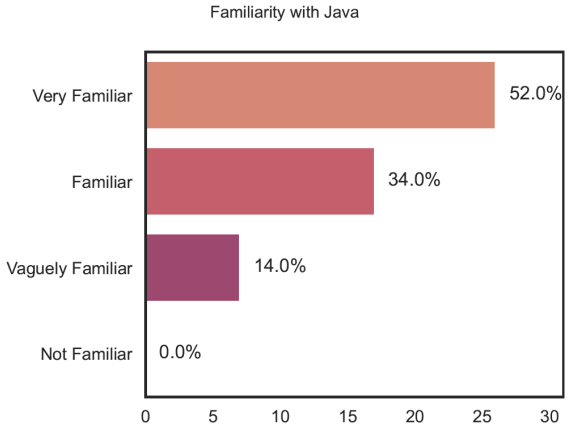
LiquidJava

```
@Refinement(" n < (_ + 1) * (_ + 1)")
int intSqrt ( @Refinement("n >= 0") int n ){...}
```

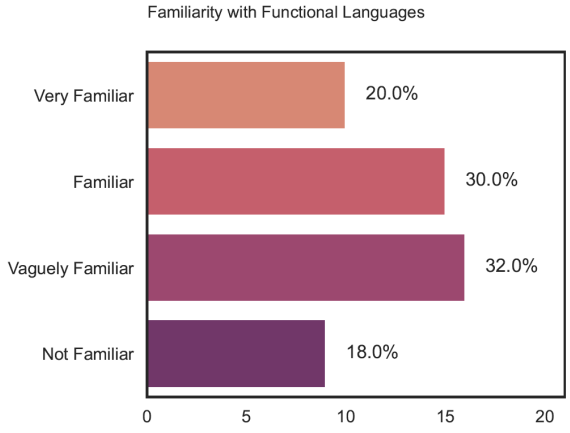
JML

```
/*@
@requires n >= 0;
@ensures \result * \result <= n ;
@ensures n < (\result + 1 ) * (\result + 1) ;
@*/
int intSqrt ( int n ){...}
```

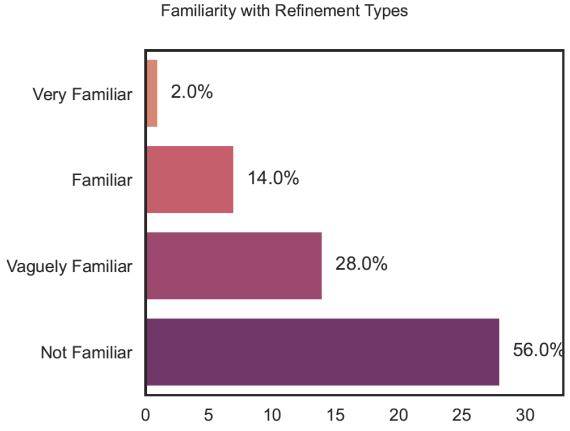
SYNTAX SURVEY



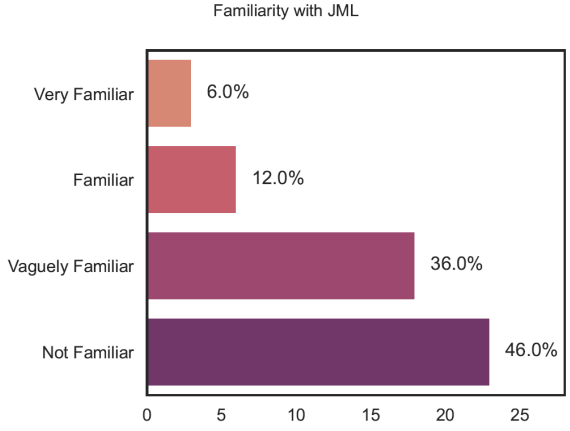
(a)



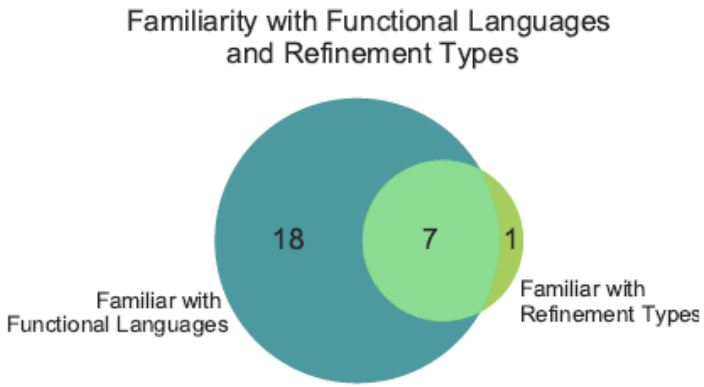
(b)



(c)



(d)



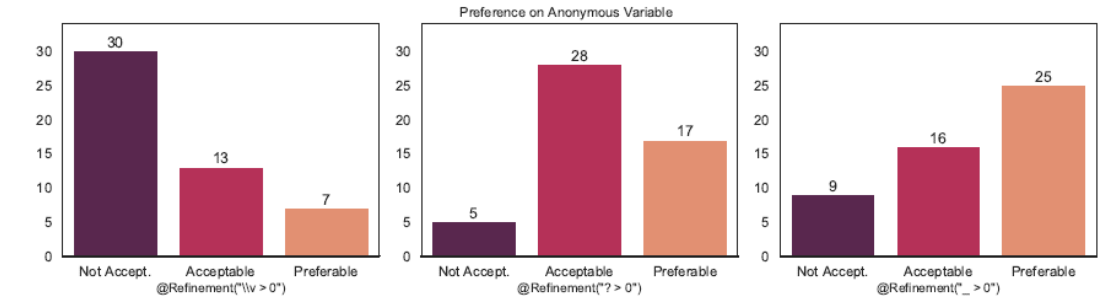
(a)

(b)

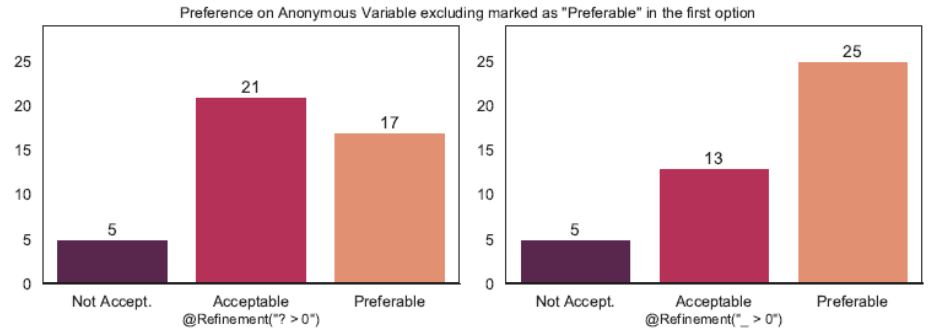
Syntax survey – Anonymous Variable

LASIGE reliable software systems

	(a)	(b)	(c)
1	@Refinement("\\v > 0")	@Refinement("? > 0")	@Refinement("_ > 0")
2	int biggerThanZero = 10;	int biggerThanZero = 10;	int biggerThanZero = 10;



(a)



(b)

Syntax survey – Variables

```
@Refinement("negativeGrade < 10")
int negativeGrade = 8;
@Refinement("excellentGrade == 19 || excellentGrade == 20")
int excellentGrade = 19;
@Refinement("goodGrade > negativeGrade && goodGrade < excellentGrade")
int goodGrade = 17;
```

(a)

```
@Refinement("{ x | x < 10}")
int negativeGrade = 8;
@Refinement("{ y | y == 19 || y == 20}")
int excellentGrade = 19;
@Refinement("{ x | x > negativeGrade && x < excellentGrade}"
int goodGrade = 17;
```

(b)

Figure 4.6: Syntax options for the refinement

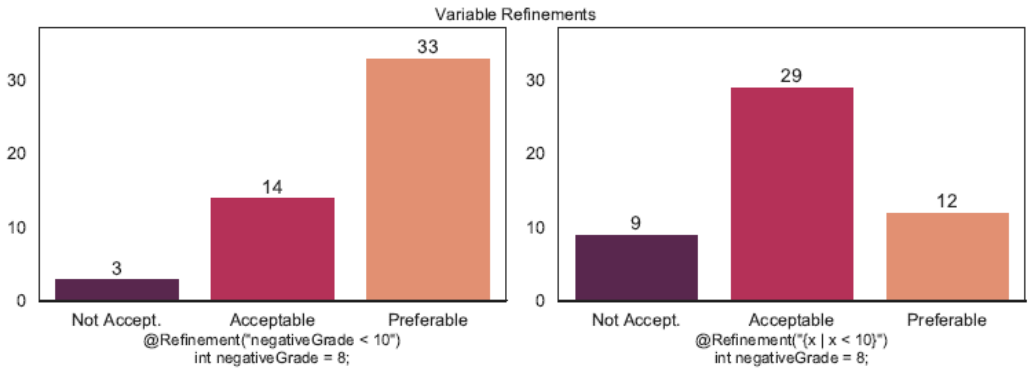


Figure 4.7: Answers on the syntax for refinements in variables.

Syntax survey – Alias

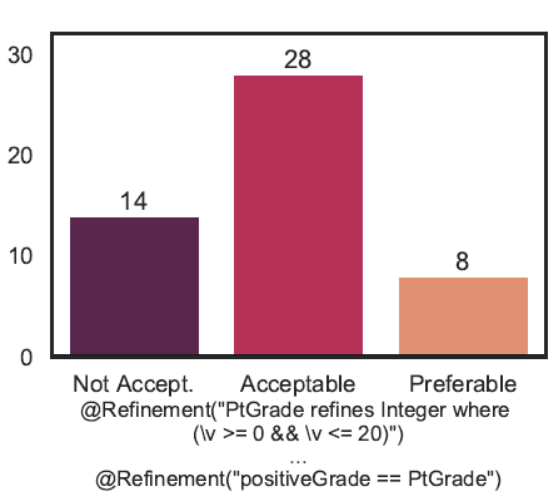
```
@Refinement("PtGrade refines Integer where (_ >= 0 && _ <= 20")
class MyClass{
   ...
   @Refinement("positiveGrade == PtGrade && positiveGrade >= 10 )")
   int positiveGrade = 12;
}
```

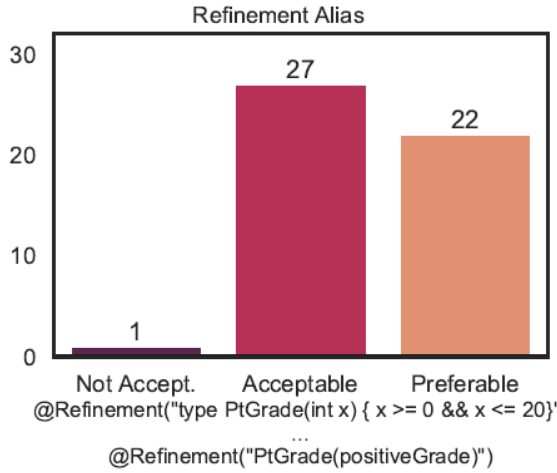
(a)

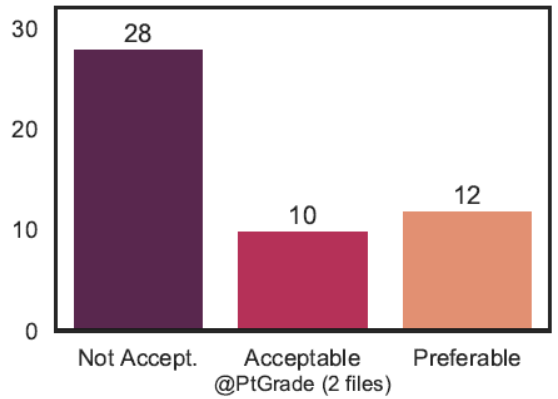
```
@Refinement("type PtGrade(int x) { x >= 0 && x <= 20}")
class MyClass{
   ...
   @Refinement("PtGrade(positiveGrade) && positiveGrade >= 10")
   int positiveGrade = 12;
}
```

```
//File PtGrade.java
@Refinement("{int x | x >= 0 && x <= 20}")
@Retention(RetentionPolicy.CLASS)
@Target({ElementType.METHOD, ElementType.FIELD,
                ElementType.LOCAL_VARIABLE,
                ElementType.PARAMETER, ElementType.TYPE})
public @interface PtGrade{}

//File MyClass.java
class MyClass{
   ...
   @PtGrade @Refinement("positiveGrade >= 10")
   int positiveGrade = 12;
}
```







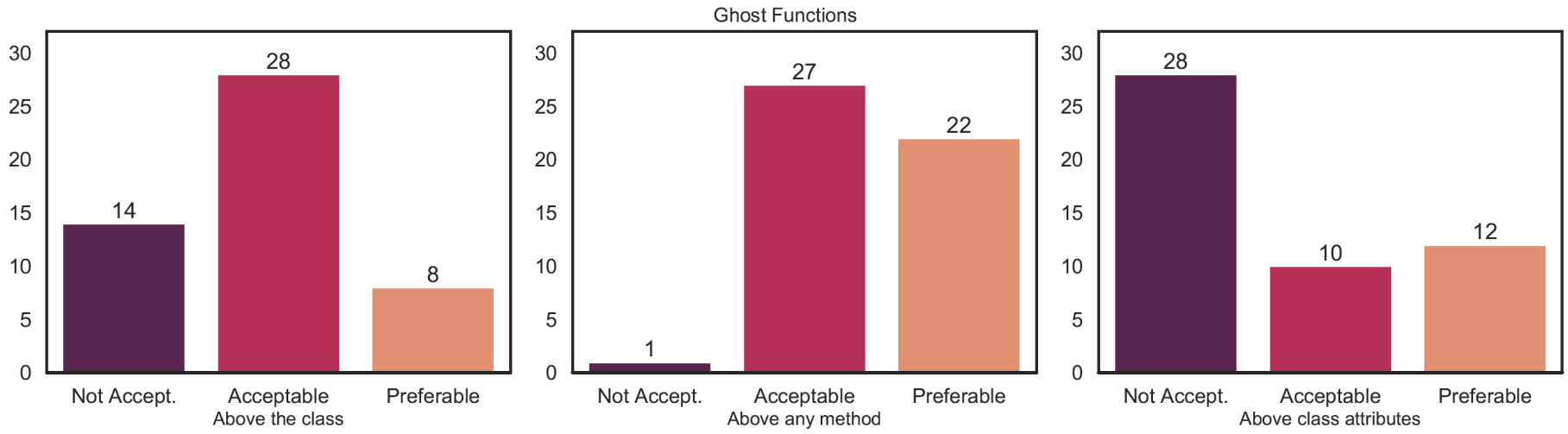
Syntax survey – Ghost Functions

LASIGE reliable software systems

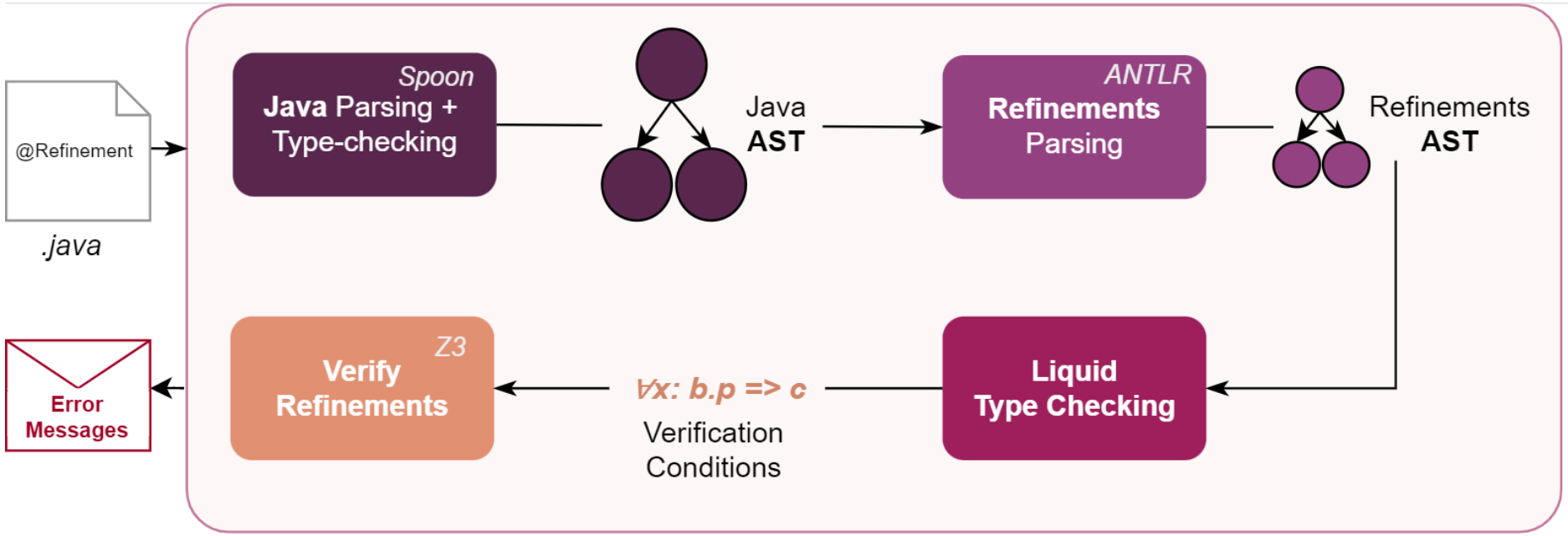
```
1  //Declaration above the class
2  @Refinement("ghost int len(List xs")
3  class MyList{
4     static final int MAX_VALUE = 50;
5
6     @Refinement("len(_) == 0")
7     public List createList() {...}
8
9     @Refinement("len(_) == len(xs) + 1")
10    public List append(List xs, int k) {...}
11 }
```

```
1  //Declaration above the class
2  @Refinement("ghost int len(List xs")
3  class MyList{
4     static final int MAX_VALUE = 50;
5
6     @Refinement("len(_) == 0")
7     public List createList() {...}
8
9     @Refinement("len(_) == len(xs) + 1")
10    public List append(List xs, int k) {...}
11 }
```

```
1  //Declaration above any class attribute
2  class MyList{
3     @Refinement("ghost int len(List xs")
4     static final int MAX_VALUE = 50;
5
6     @Refinement("len(_) == 0")
7     public List createList() {...}
8
9     @Refinement("len(_) == len(xs) + 1")
10    public List append(List xs, int k) {...}
11 }
```



SYSTEM


@Refinement
.java
Spoon
Java Parsing +
Type-checking
Java
AST
ANTLR
Refinements
Parsing
Refinements
AST
Liquid
Type Checking
∀x: b.p => c
Verification
Conditions
Z3
Verify
Refinements
Error
Messages

Strong Updates - Variables

LASIGE reliable software systems

Weak Updates

Γ
a: a > 0

$\forall b : int.\ b == a \Rightarrow$
$\forall a0 : int.\ a > 0 \Rightarrow$
$b > 5$

```
@Refinement("a > 0")
int a = 10;
@Refinement("b > 5")
int b = a;
```

Strong Updates

Δ
a0: a0 == 10

$\forall b : int.\ b == a0 \Rightarrow$
$\forall a0 : int.\ a0 == 10 \Rightarrow$
$b > 5$

```
a = 50; // create a1  == 50
@Refinement("_ > 55")
int c = a + b; // a1 + b1 && b1 == a0
```

Use of Global Context in Dependent refinements

LASIGE reliable software systems

```
1 @Refinement("a > 0")
2 int a = 10;
3 @Refinement("d < a")
4 int d = 7;
5 a = 5;
```

Under Strong uses would be correct $\forall d : int.\ d == 7 \Rightarrow \forall a0 : int.\ a0 == 10 \Rightarrow d < a0$

d no longer respects the refinement but there was no modification on it

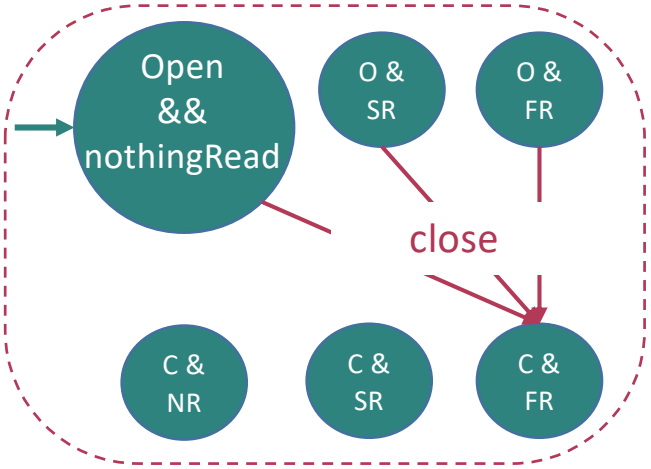
$\forall d : int.\ d == 7 \Rightarrow \forall a : int.\ a > 0 \Rightarrow d < a$ *Type Error*

```
6 d = -1;
```

$\forall d : int.\ d == -1 \Rightarrow \forall a : int.\ a > 0 \Rightarrow d < a$ *Correct*

MULTIPLE STATE SETS

LASIGE reliable software systems




```
1 @StateSet({"open", "close"})
2 @StateSet({"nothingRead", "startedReading", "finishedReading"})
3 public class FileReader{
4             @StateRefinement(to="open(this) && nothingRead(this)")
5             public FileReader(String filename){}
6
7             @StateRefinement(from="open(this)", to="startedReading(this)")
8             public void read(){}
9
10            @StateRefinement(from="finishedReading(this)")
11            @StateRefinement(from="startedReading(this)")
12            public void getText(){}
13
14            @StateRefinement(from="open(this)", to="close(this) && finishedReading(this)")
15            public void close(){}
16 }
```

Join states, ghosts and method refinement

LASIGE reliable software systems

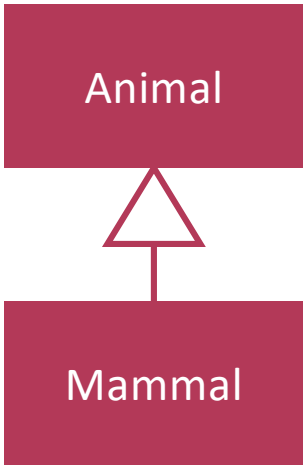
```
1  @StateSet({"open", "close","prizeSent"})
2  @Ghost("int currentValue")
3  @Ghost("int clientNumber")
4  public class Auction{
5          @StateRefinement(to = "open(this) && currentValue(this) == start && clientNumber(this)
           ,→ == −1")
6          public Auction(@Refinement("_ >= 0") int start){...}
7
8          @StateRefinement(from = "open(this) && currentValue(this) < value && clientNumber(this) != client",
9                           to = "currentValue(this) == value && clientNumber(this) == client")
10         public void bid(@Refinement("_ > 0") int client, @Refinement("_ > 0") int value){...}
11
12         @StateRefinement( to = "close(this)")
13         public void endAuction(){...}
14
           @StateRefinement(from = "close(this) && clientNumber(this) != −1 && currentValue(this)> 0",
15                          to = "prizeSent(this)")
           public void sendPrize(){...}
16
17         public static void main(String[] args){
18                 Auction auction = new Auction(300);
19                 auction.bid(1, 350);
20                 auction.bid(2, 500);
21                 auction.bid(1, 400);//Error, due to the value bid
22                 auction.bid(2, 600);//Error, due to same client
23                 auction.sendPrize();//Error, sendPrize before end of auction
24                 auction.endAuction();
25         }
26 }
```

- There are **three sequential states** for the auction: open, close and prizeSent;
- The clients that can bid have an **id number (positive)**, and the same client **cannot bid twice sequentially** (raising their own bid);
- The **bid values need to be positive** and always **greater than the last** placed bid;
- After the auction is closed, the prize must only be sent **if a client placed a bid.**

Hierarchy Example

Subclass		Superclass
Pre-condition	:>	Pre-condition
Post-condition	<:	Post-condition

```
1 public class Animal{
2             @Refinement("_ < 11000")
3             public untilMaximumLifespan(@Refinement("_ > 0") int age){...}
4 }
5
6 public class Mammal extends Animal{
7             @Refinement("_ < 250")
8             public untilMaximumLifespan(@Refinement("_ >= 0") int age){...}
9 }
```



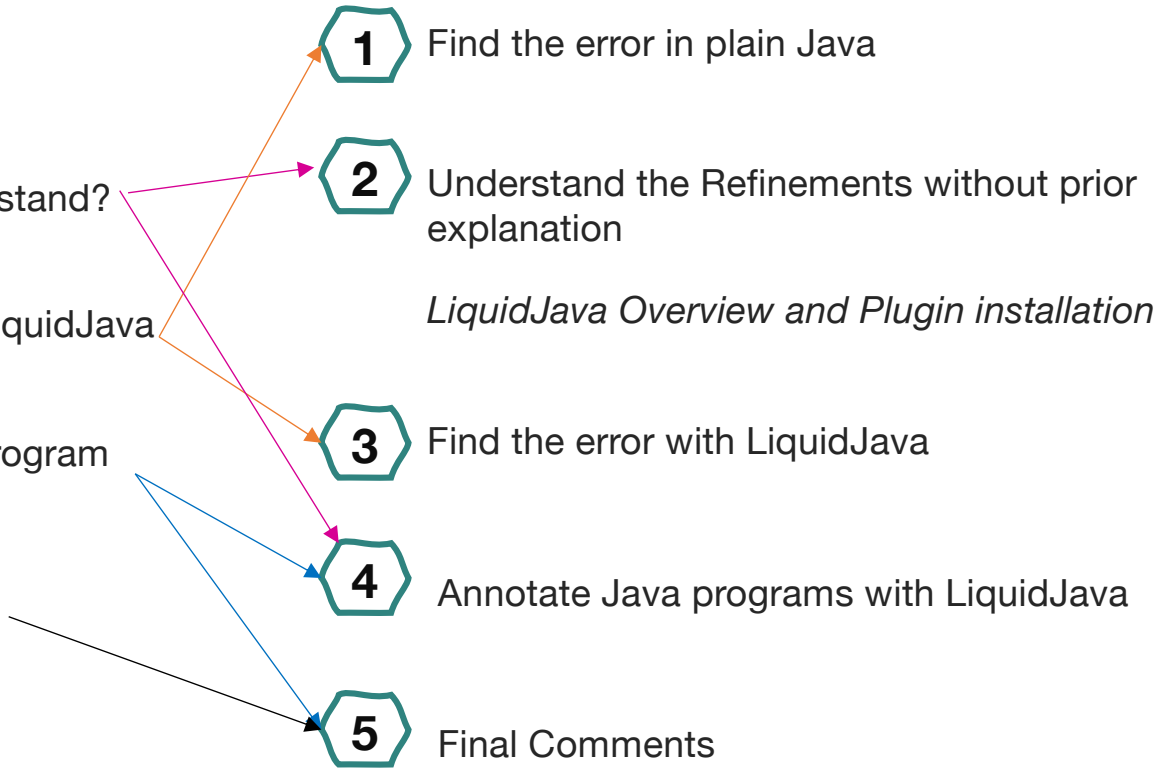
Evaluation

RQ1 Are refinements easy to understand?

RQ2 Is it easier and faster to find implementation errors using LiquidJava than with plain Java?

RQ3 How hard is it to annotate a program with refinements?

RQ4 Are developers open to using LiquidJava in their projects?



Evaluation - Participants

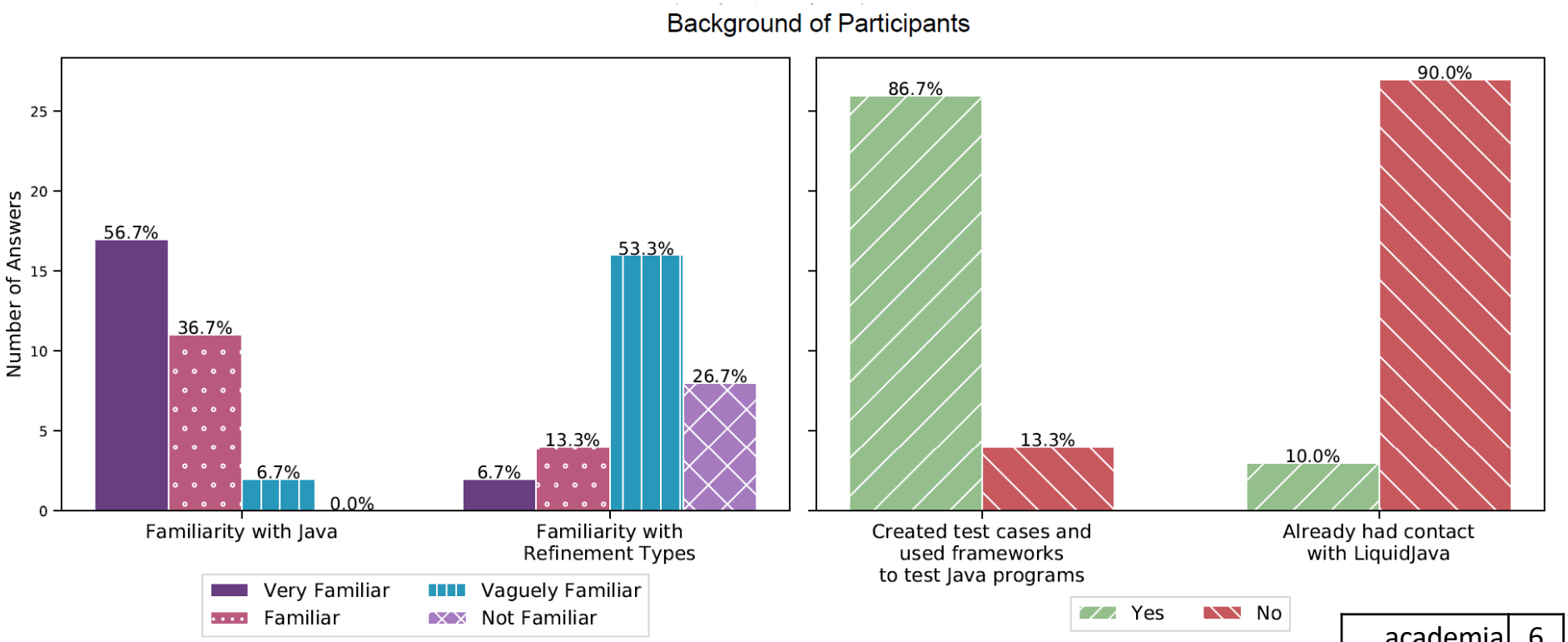


Figure B.1: Background information of the 30 participants selected to participate in the study.

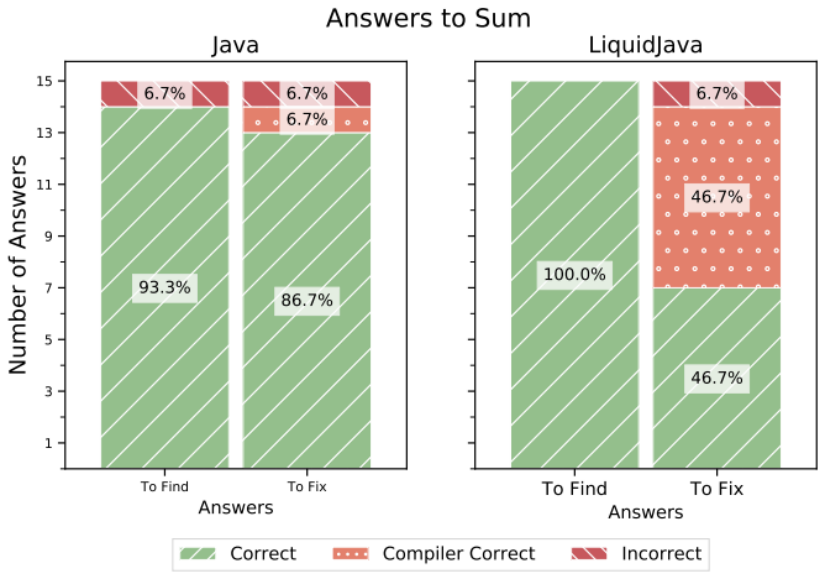
academia	6
business	8
phd	5
msc	7
bsc	4

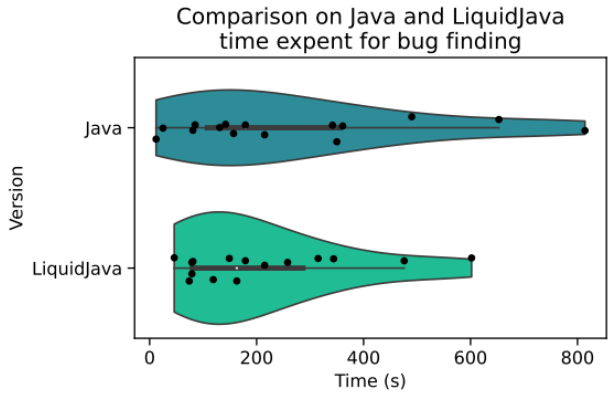
Evaluation Study
Sum

LASIGE reliable software systems

```java
@RefinementAlias("Nat(int x) {x >= 0}")
public class Test2 {
    /**
    * The sum of all numbers between 0 and n
    * @param n
    * @return a positive value that represents the sum of
 all numbers between 0 and n, or 0 if n is negative
    */
    @Refinement("Nat(_) && _ >= n")
    public static int sum(int n) {
        if(n <= 1)//correct: (n < 1) or (n <= 0)
            return 0;
        else {
            int t1 = sum(n-1);
            return n + t1;
        }
    }

}
```



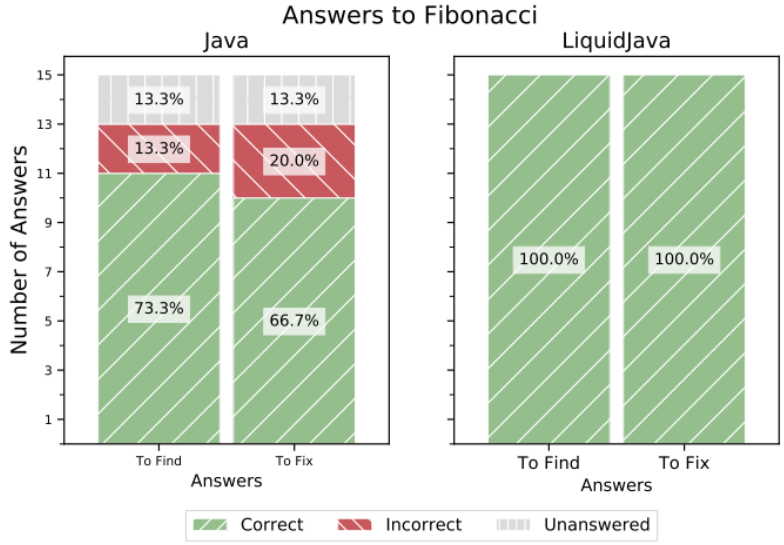


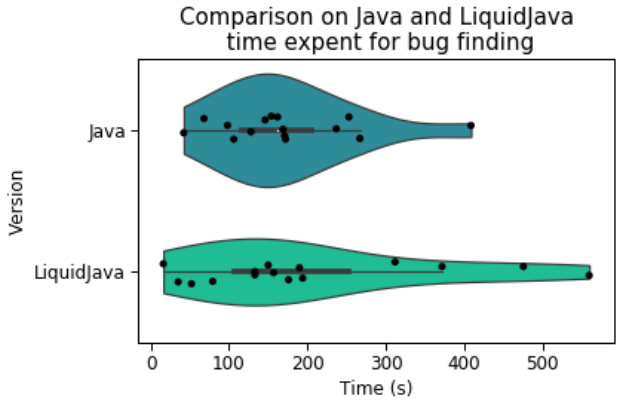
LiquidJava was **57,2s** faster in average

Evaluation Study
Fibonacci

LASIGE reliable software systems

```java
@RefinementAlias("GreaterEqualThan(int x, int y) {x >= y}")
public class Test2 {
    /**
    * Computes the fibonacci of index n
    * @param n The index of the required fibonnaci number
    * @return The fibonacci nth number. The fibonacci seque
nce follows the formula Fn = Fn-1 + Fn-
2 and has the starting values of F0 = 1 and F1 = 1
    */
    @Refinement( "_ >= 1 && GreaterEqualThan(_, n)")
    public static int fibonacci(@Refinement("Nat(n)") int n
){
        if(n <= 1)
            return 0;//correct: change to 1
        else
            return fibonacci(n-1) + fibonacci(n-2);
    }
}
```





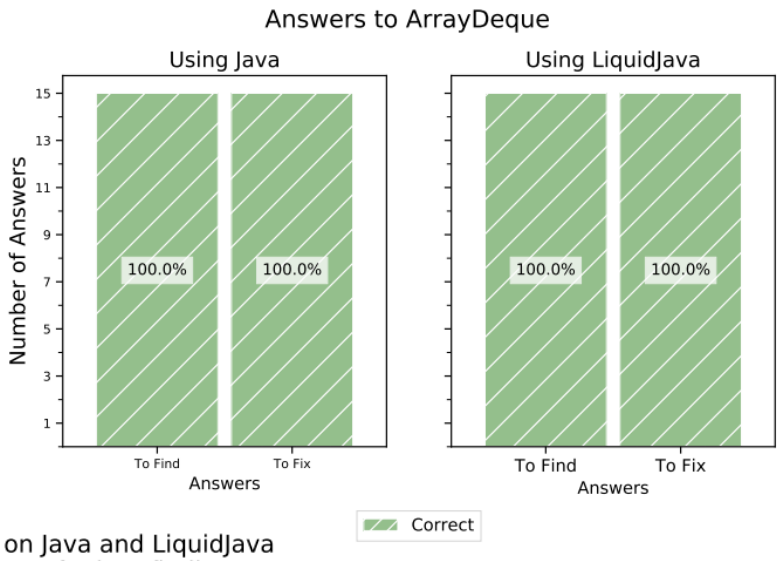
LiquidJava was **30,1s** SLOWER in average

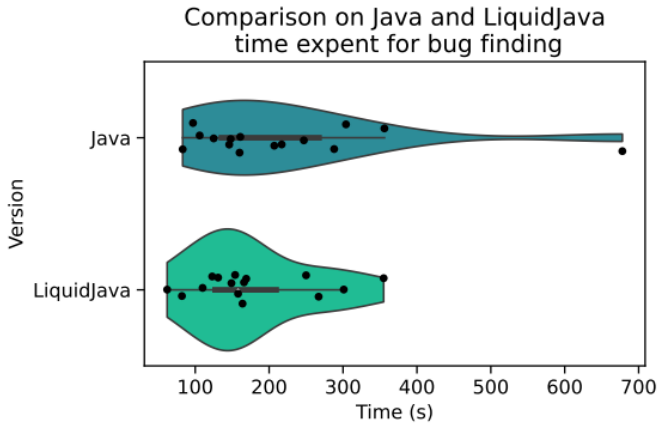
Evaluation Study
Array Deque

LASIGE reliable software systems

```
public class Test2 {

    public static void main(String[] args) throws IOException{
        ArrayDeque<Integer> p = new ArrayDeque<>();
        p.add(2);
        p.remove();
        p.offerFirst(6);
        p.getLast();
        p.remove();
        p.getLast();
        p.add(78);
        p.add(8);
        p.getFirst();
    }
}
```





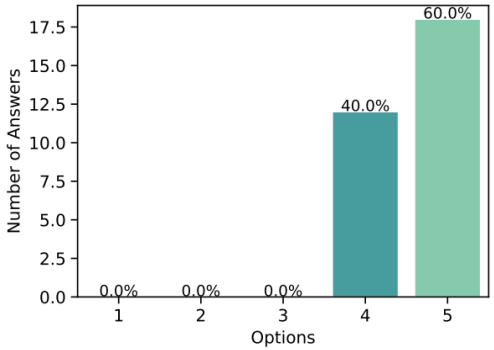
LiquidJava was **45,5s** faster in average

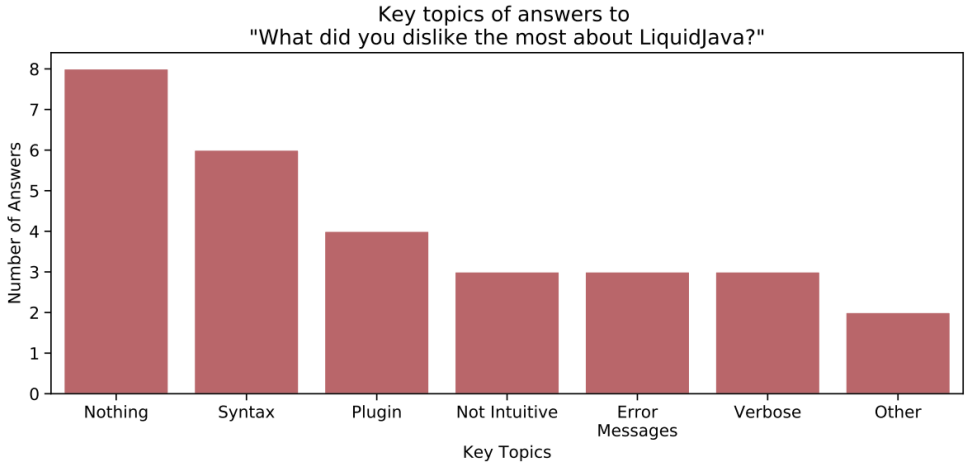
Evaluation

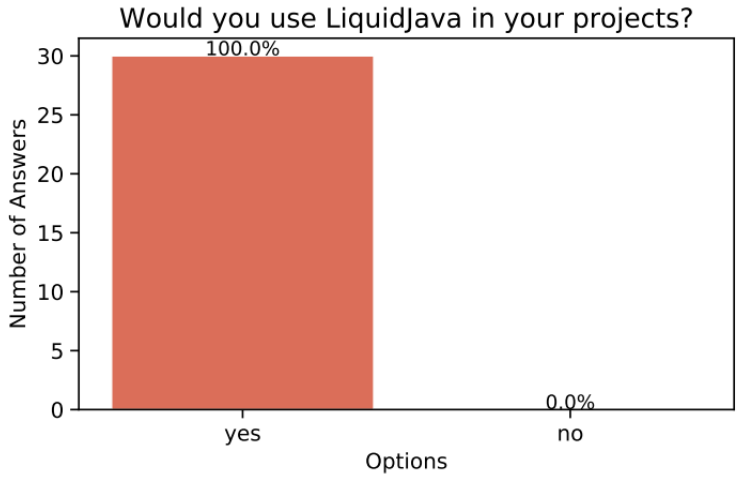
LASIGE reliable software systems

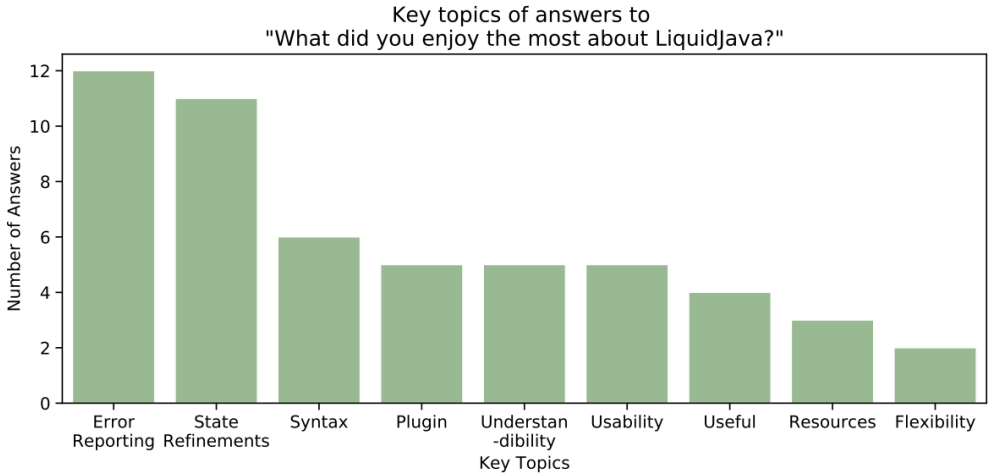
5 Final Overview

How easy was it to add Refinement Types annotations?









Evaluation – Annotate Class

```
1  public class TrafficLight {
2
3     private int r;
4     private int g;
5     private int b;
6
7     public TrafficLight() {
8        r = 76; g = 187; b = 23;
9     }
10
11    public void transitionToGreen() {
12       r = 76; g = 187; b = 23;
13    }
14
15    public void transitionToAmber() {
16       r = 255; g = 120; b = 0;
17    }
18
19    public void transitionToRed() {
20       r = 230; g = 0; b = –1;
21    }
22 }
```

```
23 //Correct Test – different file
24 TrafficLight tl = new TrafficLight();
25 tl.transitionToAmber();
26 tl.transitionToRed();
27 tl.transitionToGreen();
28 tl.transitionToAmber();
29
30 //Incorrect Test – different file
31 TrafficLight tl = new TrafficLight();
32 tl.transitionToAmber();
33 tl.transitionToRed();
34 tl.transitionToAmber();
35 tl.transitionToGreen();
```

Path condition example

```
double vat = 0.23;
...
public double mailFee(int weight) {
    @Refinement("fee > 0")
    int fee = 2;

    if(weight > 50)
        fee += 20;

    @Refinement("_ == 2 || _ == 22")
    double result = fee;
    return result * vat;
}
```

Out-of-Bound Array Access

LASIGE reliable software systems

```
@ExternalRefinementsFor("java.util.ArrayList")
@Ghost("int size")
public interface ArrayListRef<E>{
    @StateRefinement(to="size(this) == 0")
    public void ArrayList();

    @StateRefinement(to="size(this) == (size(old(this)) + 1)")
    public boolean add(E elem);

    @StateRefinement(from="index >= 0 && index <= size(this)",
                     to="size(this) == (size(old(this)) + 1)")
    public boolean add(int index, E elem);

    @StateRefinement(from="size(this) > 0 && index >= 0 && index < size(this)",
                     to="size(this) == size(old(this))")
    public void get(int index);

    @StateRefinement(to="size(this) == 0")
    public void clear();

    @StateRefinement(from="size(this) > 0 && index >= 0 && index < size(this)",
                    to="size(this) == (size(old(this)) - 1)")
    public void remove( int index);


    @Refinement("_ == size(this)")
    public int size();

    @Refinement("_ == (size(this) <= 0)")
    public boolean isEmpty();

}
```

```
ArrayList<Integer> list = new ArrayList();
list.add(1);
list.remove(0);
list.add(27);
list.add(30);
list.clear();
int x = 5;
list.get(x);
```

References

Ranjit Jhala and Niki Vazou. **Refinement types: A tutorial**.
https://arxiv.org/abs/2010.07763, 2020

Vazou N., Seidel E., Jhala R., Vytiniotis D., Peyton-Jones S.,. **Refinement Types For Haskell**.
http://goto.ucsd.edu/~rjhala/papers/refinement_types_for_haskell.pdf

Freeman T., Pfenning F.,. Refinement Types for ML. https://www.cs.cmu.edu/~fp/papers/pldi91.pdf

Kazerounian M. , Vazou N. , Bourgerie A. , Foster J., , Torlak E. **Refinement Types for Ruby**.
https://nikivazou.github.io/static/VMCAI18/paper.pdf

Rondon P., Bakst A., Kawaguchi M., Jhala R.,. **CSolve: Verifying C With Liquid Types**.
http://goto.ucsd.edu/~rjhala/papers/csolve_verifying_c_with_liquid_types.pdf

Benno Stein, Lazaro Clapp, Manu Sridharan, and Bor-Yuh Evan Chang. **Safe stream-based programming with refinement types.** In Marianne Huchard, Christian Kästner, and Gordon Fraser, editors, Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018, pages 565–576. ACM, 2018.

Chugh R., Herman D., Jhala R. **Dependent Types for JavaScript**. http://goto.ucsd.edu/~ravi/research/oopsla12-djs.pdf

References

de Moura L., Bjørner N.,. **Z3: An Efficient SMT Solver**.
https://www.researchgate.net/publication/225142568_Z3_an_efficient_SMT_solver

Pawlak R., Monperrus M., Petitprez N., Noguera C., Seinturier L.,. **Spoon: A Library for Implementing Analyses and Transformations of Java Source Code**. https://hal.inria.fr/hal-01078532v2/document

Vazou, N., Rondon P., Jhala R., **Abstract Refinement Types**,
http://goto.ucsd.edu/~rjhala/liquid/abstract_refinement_types.pdf