Gestählte Anwendungen

durch Verifikation

Andy Walter
COO, aicas GmbH
July 2010

Die aicas Gruppe

- Embedded und Echtzeit Java seit 2001
- Analyse und Debugging Werkzeuge für Java
- Hauptsitz: Karlsruhe, Deutschland
- aicas inc.: New Haven, Connecticut
- aicas SARL: Paris, Frankreich

Kunden Anwendungen

SIEMENS

petaFuel

EADS DEFENCE & SECURITY

Qualitätssicherheit

- Statische Erkennung von
  - Deadlocks
  - Race Conditions
  - möglichen Laufzeit Exceptions

Class cast
Array Store
Index out of bounds
Negative array size
Divide by zero
Scope cycle

Illegal assignment
Null pointer
Sync needed
Illegal sync use
Illegal wait use

mit VeriFlux

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;


  int value = s.reading();

  ...
}
...
```

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException

ClassCastException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException

ClassCastException

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException

ClassCastException

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException

ClassCastException

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException ✔

ClassCastException

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException

values(MyDevice.sensor)
contains only MySensor

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException

values(MyDevice.sensor)
contains only MySensor

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

values(MyDevice.sensor)
contains only MySensor

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException

Datenflussanalyse

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException ✔

Datenflussanalyse

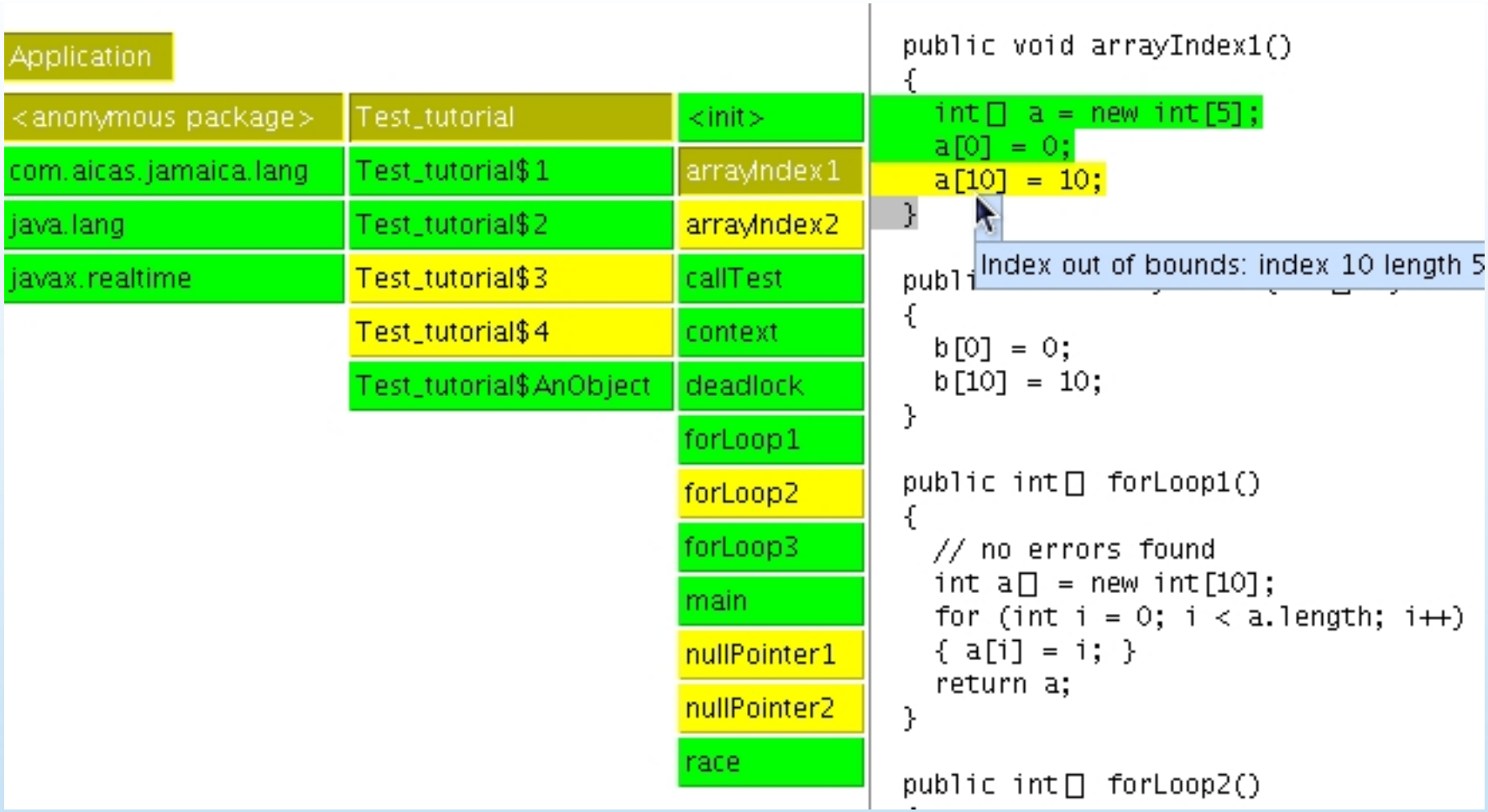
```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

NullPointerException ✔

VeriFlux


Application
<anonymous package>
com.aicas.jamaica.lang
java.lang
javax.realtime
Test_tutorial
Test_tutorial$1
Test_tutorial$2
Test_tutorial$3
Test_tutorial$4
Test_tutorial$AnObject
<init>
arrayIndex1
arrayIndex2
callTest
context
deadlock
forLoop1
forLoop2
forLoop3
main
nullPointer1
nullPointer2
race
public void arrayIndex1()
{
int[] a = new int[5];
a[0] = 0;
a[10] = 10;
}
Index out of bounds: index 10 length 5
{
b[0] = 0;
b[10] = 10;
}
public int[] forLoop1()
{
// no errors found
int a[] = new int[10];
for (int i = 0; i < a.length; i++)
{ a[i] = i; }
return a;
}
public int[] forLoop2()

Typcasts

```
class Apfel {
public: int nr;
private: static int count;

public:  Apfel() {  // Konstruktor
    nr = ++count;
  };

  void eat() {
    cout << "Apfel " <<  nr << "
gegessen" << endl;
  };
};

int Apfel::count = 0;
```

```
class Birne {
public:  double feld1;
         int nr;
private: static int count;

public:   Birne() {  // Konstruktor
    nr = ++count;
  };

  void eat() {
    cout << "Birne " << nr << "
gegessen" << endl;
  };
};

int Birne::count = 0;
```

Typcast (2)

```
main()
{
  Apfel* apfel;
  apfel = new Apfel();
  apfel->eat();

  Birne* birne;
  birne = (Birne*)apfel; // Kein Fehler in C++ !
  birne->eat();
}
```

Typcast (3)

- In C++: Programm läuft weiter, Fehler bleibt unentdeckt
- In Java:
  - Hier: Compiler Fehler
  - Schlimmstenfalls: Laufzeit Exception
- Java + Veriflux:
  - In jedem Fall statisch erkannt.

Array Überlauf

```
class Apfel {
public:  int meinFeld(4);
         int nr;
private: static int count;

public:  Apfel() {  // Konstruktor
    nr = ++count;
  };

  void eat() {
    cout << "Apfel " <<  nr << "
gegessen" << endl;
  };
};
int Apfel::count = 0;
```

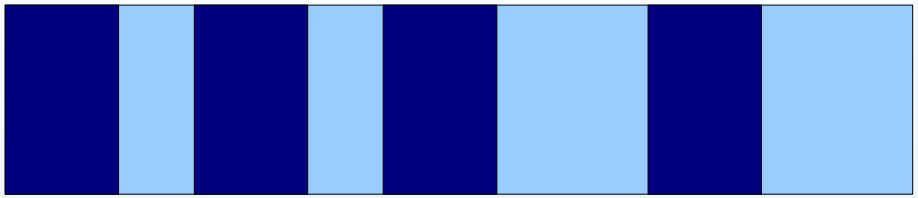
```
main()
{
  Apfel* apfel;
  apfel = new Apfel();
  apfel->meinFeld(4) = 88;
  apfel->eat();
}
```

Array Überlauf (2)

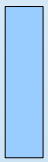
- In C++: Programm läuft weiter, Fehler bleibt unentdeckt
- In Java:
  - Laufzeit Exception
- Java + Veriflux:
  - In jedem Fall statisch erkannt.

Fragmentierter Speicher

Speicherzustand:



Neues Objekt soll erzeugt werden:



- freier Speicher

- belegter Speicher

Fragmentierter Speicher (2)

- In C:
  - Allokation nicht immer möglich
- In C++:
  - Allokation nicht immer möglich,
  - insbesondere nicht in Echtzeit
- in Java:
  - Kein Problem, mit JamaicaVM auch in Echtzeit

Deadlocks

```
// Thread 1:
public void deadlock1() {
  synchronized (a) {
    // …
    synchronized (b) {
      // …
    }
}
```

```
// Thread 2:
public void deadlock2() {
  synchronized (b) {
    // …
    synchronized (a) {
      // …
    }
}
```

- C, C++, Java: Blockiert u.U. zur Laufzeit vollständig
- JamaicaVM: Wirft Exception anstatt zu blockieren
- Veriflux: Erkennt Problem statisch

Division durch Null

- int b = a / 0;
- C / C++: Wirft Posix Signal SIGFPE
- Java: Exception, kann komfortabel behandelt werden
- Mit Veriflux: Fehler wird statisch erkannt.

Null Zeiger Dereferenzierung

```
char *nix = NULL;
strcpy (nix, “x”);
```

- C / C++: Wirft Posix Signal SIGSEGV
- Java: Exception, kann komfortabel behandelt werden
- Mit Veriflux: Fehler wird statisch erkannt.

Race Conditions

```
boolean wertGesetzt = false;
String inhalt = null;

// Thread 1:
public void race1() {
  wertGesetzt = true;
  inhalt = "meinInhalt";
}
```

```
// Thread 2:
public void race2() {
  if (wertGesetzt) {
    c = inhalt.charAt(2);
  }
}
```

- C, C++, Java: fehlende Synchronisierung führt zu inkonsistenten Daten (hier: Exception)
- Veriflux: Erkennt Problem statisch

Sichere Entwicklung

	C	C++	Java	JamaicaVM	JamaicaVM with Veriflux
Faulty Type Conversion	✘	✘	o	o	✔
Deadlocks	✘	✘	✘	o	✔
Dangling Pointers	✘	✘	✔	✔	✔
Fragmented Memory	✘	✘	✔	✔	✔
Array Range Violation	✘	✘	o	o	✔
Race Conditions	✘	✘	✘	✘	✔
Uninitialised Variables	✘	✘	✔	✔	✔
Dereferencing Null Pointer	(o)	(o)	o	o	✔
Division by Zero	(o)	(o)	o	o	✔
Exceptions	✘	o	o	o	✔

✘ danger , **o** runtime error, ✔ detected during development

Eigenschaften Veriflux

- Findet alle Fehler der genannten Fehlerarten
- Zahl der falschen Warnungen ist minimal
- Kurze Analysezeiten durch geschicktes Unifizieren von Wertemengen
- Untersuchen von Bibliotheken möglich
- Analysiert wird Bytecode
- Quellcode zur Darstellung möglich
- Schnittstelle zum Einbinden in eigene QA

Robuste Anwendungen mit Java

- Java bringt
  - Höhere Produktivität durch Fehlervermeidung
  - Robustere Anwendungen
- Mit Veriflux kann die Code Qualität noch gesteigert werden

JamaicaVM
We make Java realtime capable!

VeriFlux