

Quiz 1

- When: Friday January 17 during the lab
- Topic: testing

Descriptive variable names

```
Short s1 = new Short(0);  
Short s2 = new Short(1);
```

Question

Are these variable names descriptive?

Descriptive variable names

```
Short s1 = new Short(0);  
Short s2 = new Short(1);
```

Question

Are these variable names descriptive?

Answer

No. They are cryptic variable names. As a result, their meaning might not be clear to others. They are also very similar, which makes it easy to mix them up.

Instead of

```
@Test  
public void testMaxValue() {  
    Assertions.assertEquals(32767, Short.MAX_VALUE);  
}
```

Instead of

```
@Test
public void testMaxValue() {
    Assertions.assertEquals(32767, Short.MAX_VALUE);
}
```

use

```
@Test
public void testMaxValue() {
    short MAX_VALUE = 32767;
    Assertions.assertEquals(MAX_VALUE, Short.MAX_VALUE);
}
```

Definition

A magic number is any number different from 0, 1, -1, 2, -2.

Instead of using magic numbers in your code (in arithmetic expressions, as arguments of methods calls, etc), introduce them as constants (see previous slide).

Question

What is the scope of the attribute in the following code snippet?

```
public class ShortTest {  
    private Short s;  
  
    @Test  
    public void testConstructor() { ... }  
  
    @Test  
    public void testCompareTo() { ... }  
  
    ...  
}
```

Question

What is the scope of the attribute in the following code snippet?

```
public class ShortTest {  
    private Short s;  
  
    @Test  
    public void testConstructor() { ... }  
  
    @Test  
    public void testCompareTo() { ... }  
  
    ...  
}
```

Answer

The whole class. If possible, try to limit the scope.


```
/**  
 * Tests that the constructor throws an exception  
 * for negative arguments.  
 */  
@Test  
public void testConstructorException() {  
    ..  
}
```

```
import static org.junit.jupiter.api.Assertions.*;  
import org.junit.jupiter.api.Assertions;  
import java.lang.IllegalArgumentException;
```

```
import static org.junit.jupiter.api.Assertions.*;  
import org.junit.jupiter.api.Assertions;  
import java.lang.IllegalArgumentException;
```

All classes in the package `java.lang` are automatically imported. Hence, the class `java.lang.IllegalArgumentException` need not be imported.

The first two imports are almost the same. Either one suffices.

Question

How many numbers are printed by the following loop?

```
for (short s = Short.MIN_VALUE; s <= Short.MAX_VALUE; s++)  
    System.out.println(s);  
}
```

Question

How many numbers are printed by the following loop?

```
for (short s = Short.MIN_VALUE; s <= Short.MAX_VALUE; s++)  
    System.out.println(s);  
}
```

Answer

Infinitely many.

Question

How many numbers are printed by the following loop?

```
for (int i = Short.MIN_VALUE; i <= Short.MAX_VALUE; i++) {  
    System.out.println(i);  
}
```

Question

How many numbers are printed by the following loop?

```
for (int i = Short.MIN_VALUE; i <= Short.MAX_VALUE; i++) {  
    System.out.println(i);  
}
```

Answer

2^{15} (only the non-negative short values since `Short.MIN_VALUE` of `lab.Short` is zero).

Pseudorandom number generator

An algorithm for generating a sequence of numbers properties of which approximate those of sequences of random numbers.

```
private long seed;

private static long MULTIPLIER = 25214903917L;
private static long INCREMENT = 11;
private static long MODULUS = (long) Math.pow(2, 48);

public Random(long seed) {
    this.seed = seed;
}

public long nextLong() {
    this.seed =
        (MULTIPLIER * this.seed + INCREMENT) % MODULUS;
    return this.seed;
}
```


Check models

EECS 4315

`wiki.eecs.yorku.ca/course/4315/`

The state space explosion problem

Problem

The size of the state space, that is, the number of states, may become very large.

The state space explosion problem

Problem

The size of the state space, that is, the number of states, may become very large.

This is one of the major challenges in **model checking**.

Develop a model (states connected by transitions) of the code and check properties of the model.

Model checking was developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980s.

Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In, Dexter Kozen, editor, *Proceedings of Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52-71. Yorktown Heights, NY, USA, May 1981. Springer-Verlag.

Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In, Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337-351. Torino, Italy, April 1982. Springer-Verlag.

- Recipient of the Turing Award (2007)
- Recipient of the ACM Paris Kanellakis Award (1999)
- Member of the National Academy of Engineering (2005)
- Member of the American Academy of Arts and Sciences (2011)



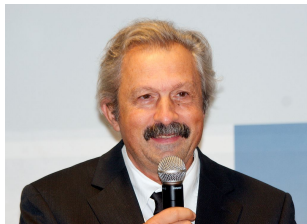
source: Dennis Hamilton

- Recipient of the Turing Award (2007)
- Recipient of the ACM Paris Kanellakis Award (1999)
- Recipient of the CMU Newell Medal (1999)



source: Marsha Miller

- Recipient of the Turing Award (2007)
- Grand officer of France's national order of merit (2008)
- Commander in France's legion of honour (2011)



source: David Monniaux



source: unknown

Model of a system

A **model** of a system is an **abstraction** of the system.

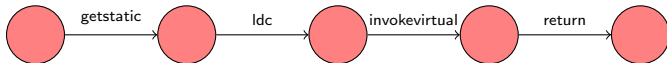


Source: Toronto Star

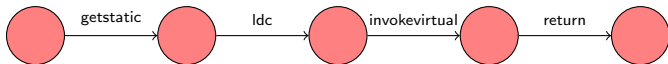
There are many levels of abstraction and, hence, a system can be modelled in many different ways.

Model of a system

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```



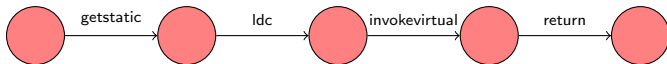
Model of a system



Question

What are the three entities that make up the above model?



Model of a system



Question

What are the three entities that make up the above model?

Answer

- 1 
- 2 
- 3 `getstatic, ldc, ...`

Model of a system

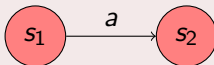


Question

Such a model consists of

- a set S of *states*,
- a set A of *actions*, and
- a set of *transitions*.

Which three ingredients specify the following transition?



Model of a system

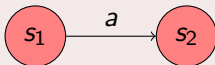


Question

Such a model consists of

- a set S of *states*,
- a set A of *actions*, and
- a set of *transitions*.

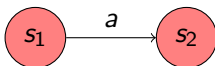
Which three ingredients specify the following transition?



Answer

- 1 the source state s_1 ,
- 2 the label a , and
- 3 the target state s_2 .

The transition



is specified by

- 1 the source state s_1 ,
- 2 the label a , and
- 3 the target state s_2 .

Hence, the transition can be captured by the triple (s_1, a, s_2) . This is an *element* of $S \times A \times S$, that is,

$$(s_1, a, s_2) \in S \times A \times S.$$

Therefore, the set of transitions is a *subset* of $S \times A \times S$.

Model of a system



Question

How can we model **all** the labelled transitions?

Model of a system



Question

How can we model **all** the labelled transitions?

Answer

$\{ (s_1, \text{getstatic}, s_2), (s_2, \text{ldc}, s_3), (s_3, \text{invokevirtual}, s_4), (s_4, \text{return}, s_5) \}$.

$\{ (s_1, \text{getstatic}, s_2), (s_2, \text{ldc}, s_3), (s_3, \text{invokevirtual}, s_4), (s_4, \text{return}, s_5) \}$ is a subset of $S \times A \times S$.

Question

$\{ (s_1, \text{getstatic}, s_2), (s_2, \text{ldc}, s_3), (s_3, \text{invokevirtual}, s_4), (s_4, \text{return}, s_5) \}$ is a r... over the set S , A and S .

Model of a system

$\{ (s_1, \text{getstatic}, s_2), (s_2, \text{ldc}, s_3), (s_3, \text{invokevirtual}, s_4), (s_4, \text{return}, s_5) \}$ is a subset of $S \times A \times S$.

Question

$\{ (s_1, \text{getstatic}, s_2), (s_2, \text{ldc}, s_3), (s_3, \text{invokevirtual}, s_4), (s_4, \text{return}, s_5) \}$ is a r... over the set S , A and S .

Answer

relation.

Model of a system

$\{ (s_1, \text{getstatic}, s_2), (s_2, \text{ldc}, s_3), (s_3, \text{invokevirtual}, s_4), (s_4, \text{return}, s_5) \}$ is a subset of $S \times A \times S$.

Question

$\{ (s_1, \text{getstatic}, s_2), (s_2, \text{ldc}, s_3), (s_3, \text{invokevirtual}, s_4), (s_4, \text{return}, s_5) \}$ is a r... over the set S , A and S .

Answer

relation.

The relation is usually denoted by \rightarrow and called the **transition relation**.

Systems can be modelled by means of labelled transition systems.

Definition

A labelled transition system is a tuple $\langle S, A, \rightarrow, s \rangle$ consisting of

- a set S of states,
- a set A of actions,
- a set of transitions $\rightarrow \subseteq S \times A \times S$, and
- a start state $s \in S$.

Systems can be modelled by means of labelled transition systems.

Definition

A labelled transition system is a tuple $\langle S, A, \rightarrow, s \rangle$ consisting of

- a set S of states,
- a set A of actions,
- a set of transitions $\rightarrow \subseteq S \times A \times S$, and
- a start state $s \in S$.

Instead of $(s_1, a, s_2) \in \rightarrow$, we usually write $s_1 \xrightarrow{a} s_2$.

Model of a system



Question

Give the corresponding labelled transition system.

Model of a system



Question

Give the corresponding labelled transition system.

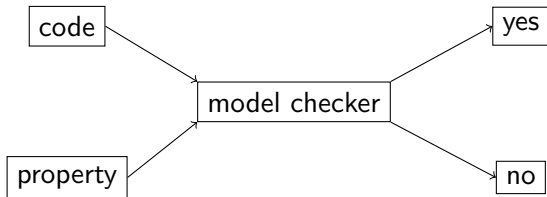
Answer

$\langle \{s_1, s_2, s_3, s_4, s_5\},$
 $\{\text{getstatic}, \text{ldc}, \text{invokevirtual}, \text{return}\},$
 $\{(s_1, \text{getstatic}, s_2), (s_2, \text{ldc}, s_3), (s_3, \text{invokevirtual}, s_4), (s_4, \text{return}, s_5)\},$
 $s_1 \rangle.$

Introduction to Java PathFinder

EECS 4315

www.eecs.yorku.ca/course/4315/



Clarke and Emerson used the term *model checking* because they wanted to determine if the property (expressed as a temporal formula) f was true in the Kripke structure M , that is, whether the structure M was a *model* for the formula f .

Some people believe erroneously that the use of the term *model* refers to the dictionary meaning of this word and indicates that an abstraction of the actual system is considered.

In 1999, Klaus Havelund introduced Java PathFinder (JPF).

Klaus Havelund. Java PathFinder – A Translator from Java to Promela. In, Dennis Dams, Rob Gerth, Stefan Leue and Mieke Massink, editors, *Proceedings of the 5th and 6th International SPIN Workshops*, volume 1680 of *Lecture Notes in Computer Science*, page 152. Springer-Verlag.

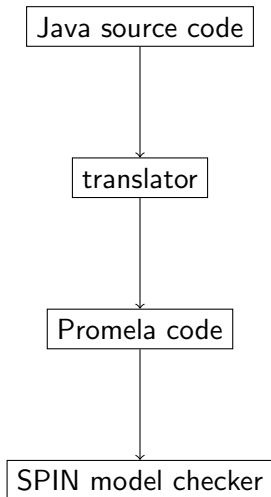
- PhD in Computer Science from the University of Copenhagen.
- Senior Research Scientist at NASA's Jet Propulsion Laboratory.
- ASE 2014 most influential paper award.



source: Klaus Havelund

Others who initially worked on JPF:

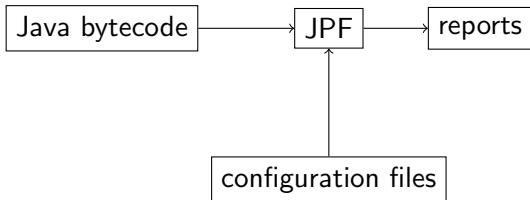
- Michael Lowry (NASA)
- John Penix (NASA, now Google)
- Thomas Pressburger (NASA)
- Jens Ulrik Skakkebaek (Stanford, now Google)
- Willem Visser (NASA, now Stellenbosch University)



Major limitations:

- Representing all features of Java in Promela is impossible;
- Mapping bugs found by SPIN in the Promela code back to the Java code is challenging.

Second version of JPF



The second version of JPF is a Java virtual machine (JVM).

Willem Visser, Klaus Havelund, Guillaume Brat, Seungjoon Park.
Model Checking Programs. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering*, pages 3–12, Grenoble, France, September 2000. IEEE

The Automated Software Engineering conference series has a rich history of good contributions to the area of research and development. The ASE most influential paper award is an effort to identify the most influential ASE paper 14 years after being published. In 2014, the above paper won this award.

A simple example

```
import java.util.Random;

public class PrintRandom {
    public static void main(String[] args) {
        Random random = new Random();
        final int MAX = 4;
        System.out.println(random.nextInt(MAX + 1));
    }
}
```

A simple example

```
target=PrintRandom
```

```
classpath=<path to directory that contains PrintRandom.class>
```

A simple example

```
JavaPathfinder core system v8.0 (rev 2+) - (C) 2005-2014 Un
===== syst
PrintRandom.main()
===== sea
0
===== resu
no errors detected
===== stat
elapsed time:      00:00:00
states:           new=1,visited=0,backtracked=1,end=1
search:           maxDepth=1,constraints=0
choice generators: thread=1 (signal=0,lock=1,sharedRef=0,th
heap:             new=350,released=12,maxLive=0,gcCycles=1
instructions:     3176
max memory:       61MB
loaded code:      classes=57,methods=1232
```

A simple example

Question

To how many different executions may the Java code give rise?

A simple example

Question

To how many different executions may the Java code give rise?

Answer

5.

A simple example

Question

To how many different executions may the Java code give rise?

Answer

5.

Question

How many different executions does JPF check?

A simple example

Question

To how many different executions may the Java code give rise?

Answer

5.

Question

How many different executions does JPF check?

Answer

1.

A simple example

Let's have a look at the state space diagram.

```
target=PrintRandom  
classpath=<path to directory that contains PrintRandom.class>  
listener=gov.nasa.jpf.listener.StateSpaceDot
```

A simple example



A simple example

Configure JPF so that it explores all random choices.

```
target=PrintRandom  
classpath=<path to directory that contains PrintRandom.class>  
cg.enumerate_random=true
```

A simple example

```
JavaPathfinder core system v8.0 (rev 2+) - (C) 2005-2014 Uri  
===== syst  
PrintRandom.main()  
===== sear  
0  
1  
2  
3  
4  
===== resu  
no errors detected  
===== stat  
elapsed time:      00:00:00  
states:           new=2,visited=9,backtracked=11,end=10  
search:           maxDepth=2,constraints=0  
choice generators: thread=1 (signal=0,lock=1,sharedRef=0,th  
heap:             new=350,released=102,maxLive=348,gcCyc
```

A simple example

Let's have a look at the state space diagram.

```
target=PrintRandom  
classpath=<path to directory that contains PrintRandom.class>  
cg.enumerate_random=true  
listener=gov.nasa.jpf.listener.StateSpaceDot
```

A simple example

