Quiz 3

When: Friday March 9 at 10:00
Where: LAS 1004
What: the material covered in lecture 10–13

Senate policy

According to Senate Policy on the Academic Implications of Disruptions or Cessations of University Business Due to Labour Disputes or Other Causes "Students who do not participate in academic activities because:

- they are unable to do so owing to a Disruption, or
- they choose not to participate in academic activities owing to a strike or lock-out on campus

are entitled to immunity from penalty, to reasonable alternative access to materials covered in their absence, to reasonable extensions of deadlines and to such other remedy as Senate deems necessary and consistent with the principle of academic integrity."

Drop deadline

The drop deadline, originally March 9, will be moved to a later date.

Concurrency

EECS 4315

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

Counter class

```
public class Counter extends Thread {
  private int value;

  public Counter() {
    this.value = 0;
  }

  public void run() {
    this.value++;
  }
}
```

Counter class

```
public void run() {
  this.value++;
}
```

```
0: aload_0
1: dup
2: getfield
5: iconst_1
6: iadd
7: putfield
10: return
```

The app

```
public class Main {
  public static void main(String[] args) {
    Counter one = new Counter();
    Counter two = new Counter();
    one.start();
    two.start();
  }
}
```

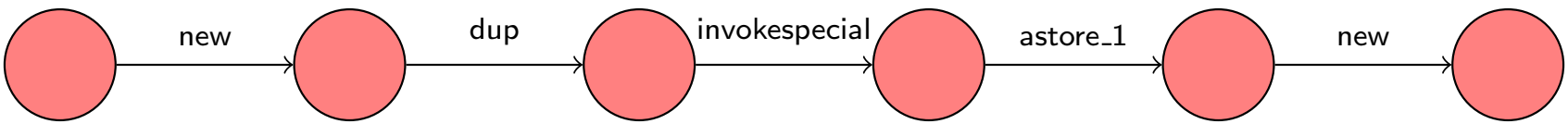
```
0: new           11: dup           20: aload_2
3: dup           12: invokespecial 21: invokevirtual
4: invokespecial 15: astore_2      24: return
7: astore_1      16: aload_1
8: new           17: invokevirtual
```

State-transition diagram

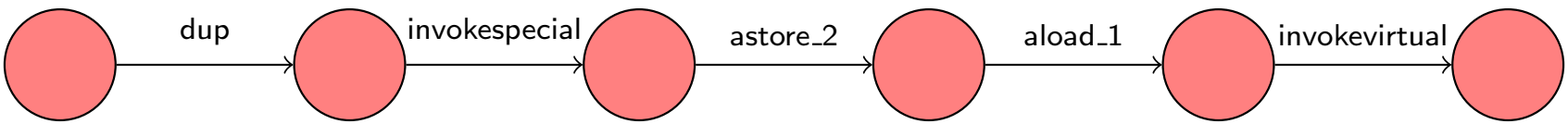
Question

Draw the corresponding state-transition diagram.

State-transition diagram

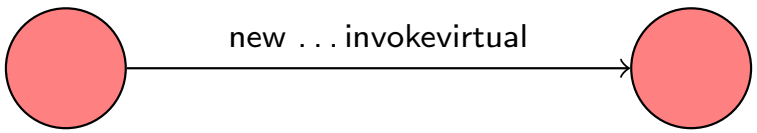


State-transition diagram



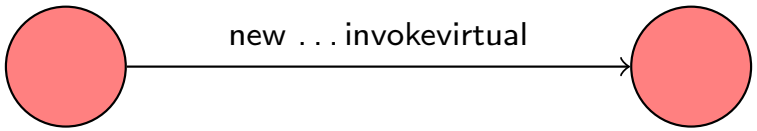
Mini model

Combine the first ten transitions into one.



The actions of the labelled transition system are sequences of bytecode instructions.

State-transition diagram



Next instructions for the main thread:

```
20: aload_2
21: invokevirtual
24: return
```

Next instructions for the thread one:

```
0: aload_0
1: dup
2: getfield
5: iconst_1
6: iadd
7: putfield
```

State-transition diagram

Question

Can the bytecode instructions corresponding to the `run` invocation be modelled as a single transition?

State-transition diagram

Question

Can the bytecode instructions corresponding to the `run` invocation be modelled as a single transition?

Answer

Yes.

State-transition diagram

Question

Can the bytecode instructions corresponding to the `run` invocation be modelled as a single transition?

Answer

Yes.

Question

Why?

State-transition diagram

Question

Can the bytecode instructions corresponding to the `run` invocation be modelled as a single transition?

Answer

Yes.

Question

Why?

Answer

Because the execution of this method does not impact the other threads.

Combining bytecode instructions

- We combine the first ten bytecode instructions since there is only one thread.
- We combine the bytecode instructions corresponding to the `run` invocation because those do not impact the other threads.

Combining bytecode instructions

- We combine the first ten bytecode instructions since there is only one thread.
- We combine the bytecode instructions corresponding to the `run` invocation because those do not impact the other threads.

General idea

Combine those bytecode instructions that do not impact other threads.

Combining bytecode instructions

Problem

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads.

Combining bytecode instructions

Problem

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads.

Question

Give an algorithm that solves the problem.

Combining bytecode instructions

Problem

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads.

Question

Give an algorithm that solves the problem.

Answer

Impossible!

Proving impossibility

Question

Which other problems cannot be solved?

Proving impossibility

Question

Which other problems cannot be solved?

Answer

The halting problem: given code and input for that code, determine whether the code terminates.

Proving impossibility

Problem

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads.

Question

Explain (informally) why the problem cannot be solved.

Writer

```
public class Writer extends Thread {
  public static boolean shared = false;

  public void run() {
    Writer.shared = true;
  }
}
```

Writer

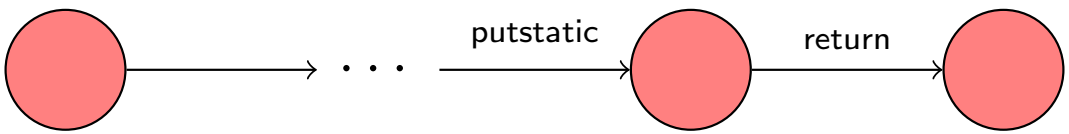
```
public class Reader extends Thread {
  public void run() {
    this.code();
    if (Writer.shared) {
      ...
    }
  }

  public void code() {
    ...
  }
}
```

Main

```
public class Main {
  public static void main(String[] args) {
    Reader reader = new Reader();
    Writer writer = new Writer();
    reader.start();
    writer.start();
  }
}
```

Writer

Transitions of the `Writer` thread:



Assume that the `code` method does not use the attribute `Writer.shared`. Then the bytecode instruction `putstatic` of the `Writer` thread impacts the `Reader` thread if and only if the method call to `code` terminates.

Combining bytecode instructions

General idea

Combine those bytecode instructions for which we can prove that they do not impact other threads.

Combining bytecode instructions

General idea

Combine those bytecode instructions for which we can prove that they do not impact other threads.

The idea of combining consecutive transitions labelled with invisible (outside the current thread) actions into a single transition is due to Patrice Godefroid.

Combining bytecode instructions

General idea

Combine those bytecode instructions for which we can prove that they do not impact other threads.

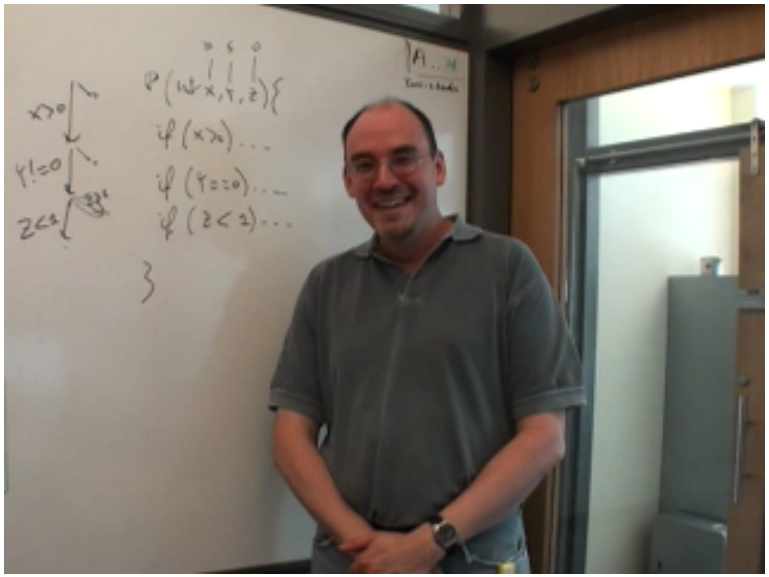
The idea of combining consecutive transitions labelled with invisible (outside the current thread) actions into a single transition is due to Patrice Godefroid.

Examples of invisible actions

- Reading or writing an attribute that can be proved to be not shared.
- Reading or writing a local variable.
- ...

Patrice Godefroid

- Ph.D. degree in Computer Science from the University of Liege, Belgium.
- Worked at Bell Laboratories.
- Currently at Microsoft Research.



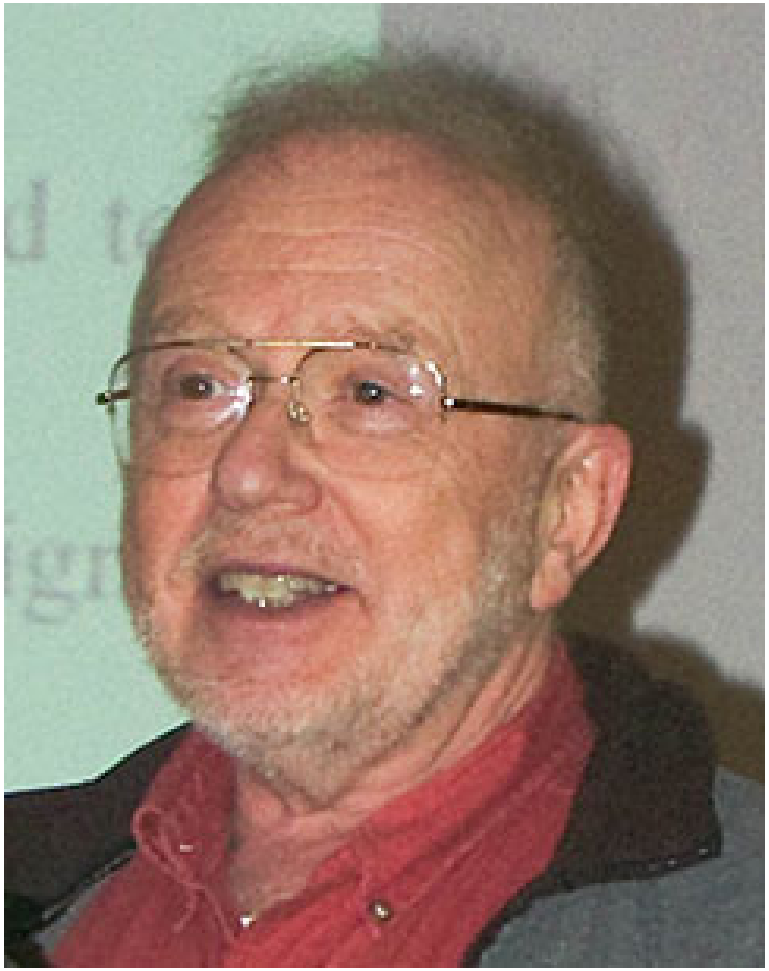
Source: Patrice Godefroid

The readers-writers problem

The readers and writers problem, due to Courtois, Heymans and Parnas, is a classical concurrency problem. It models access to a database. There are many competing threads wishing to read from and write to the database. It is acceptable to have multiple threads reading at the same time, but if one thread is writing then no other thread may either read or write. A thread can only write if no thread is reading.

David Parnas

- Canadian early pioneer of software engineering.
- Ph.D. from Carnegie Mellon University.
- Taught at the University of North Carolina at Chapel Hill, the Technische Universität Darmstadt, the University of Victoria, Queen's University, McMaster University, and University of Limerick.
- Won numerous awards including ACM SIGSOFT's "Outstanding Research" award.



Source: Hubert Baumeister

Pierre-Jacques Courtois

Professor emeritus at the Catholic University of Leuven.

Source:

`https://www.info.ucl.ac.be/~courtois/`

Reader class

```
public class Reader extends Thread {
  private Database database;

  public Reader(Database database) {
    this.database = database;
  }

  public void run() {
    this.database.read();
  }
}
```

Writer class

```
public class Writer extends Thread {
  private Database database;

  public Writer(Database database) {
    this.database = database;
  }

  public void run() {
    this.database.write();
  }
}
```

Database class

```
public class Database {
  ...
  public Database() { ... }
  public void read() { ... }
  public void write() { ... }
}
```

Main class

```
final int READERS = 5;
final int WRITERS = 2;
Database database = new Database();
for (int r = 0; r < READERS; r++) {
  (new Reader(database)).start();
}
for (int w = 0; w < WRITERS; w++) {
  (new Writer(database)).start();
}
```

The readers-writers problem

Question

If we make both methods synchronized, does that solve the problem?

The readers-writers problem

Question

If we make both methods synchronized, does that solve the problem?

Answer

Yes.

The readers-writers problem

Question

If we make both methods synchronized, does that solve the problem?

Answer

Yes.

Question

Is it a satisfactory solution?

The readers-writers problem

Question

If we make both methods synchronized, does that solve the problem?

Answer

Yes.

Question

Is it a satisfactory solution?

Answer

No.

The readers-writers problem

Question

Why is it not satisfactory?

The readers-writers problem

Question

Why is it not satisfactory?

Answer

It does not allow multiple readers to read at the same time.

The readers-writers problem

Question

When does a reader have to wait until it can start reading?

The readers-writers problem

Question

When does a reader have to wait until it can start reading?

Answer

When a writer is writing.

The readers-writers problem

Question

When does a reader have to wait until it can start reading?

Answer

When a writer is writing.

Question

When does a writer have to wait until it can start writing?

The readers-writers problem

Question

When does a reader have to wait until it can start reading?

Answer

When a writer is writing.

Question

When does a writer have to wait until it can start writing?

Answer

When another writer is writing or a reader is reading.

The attributes

Question

Of which type of information do we need to keep track so that we can determine

- whether a writer is writing, and
- whether a writer is writing or a reader is reading.

The attributes

Question

Of which type of information do we need to keep track so that we can determine

- whether a writer is writing, and
- whether a writer is writing or a reader is reading.

Answer

Two booleans.

The attributes

Question

Of which type of information do we need to keep track so that we can determine

- whether a writer is writing, and
- whether a writer is writing or a reader is reading.

Answer

Two booleans.

Question

What are appropriate names for these two attributes?

The attributes

Question

Of which type of information do we need to keep track so that we can determine

- whether a writer is writing, and
- whether a writer is writing or a reader is reading.

Answer

Two booleans.

Question

What are appropriate names for these two attributes?

Answer

`writing` and `reading`.

Initializing the attributes

Question

```
public class Database {
  private boolean writing;
  private boolean reading;

  ...
}
```

Where and how are the attributes `writing` and `reading` initialized?

Initializing the attributes

Question

```
public class Database {
  private boolean writing;
  private boolean reading;

  ...
}
```

Where and how are the attributes `writing` and `reading` initialized?

Answer

```
  public Database() {
    this.writing = false;
    this.reading = false;
  }
```

Waiting when a writer is writing

Question

In

```
public void read() {
  ...
  \\ read
  ...
}
```

how do we express that a thread has to wait if a writer is writing?

Waiting when a writer is writing

Question

In

```
public void read() {
  ...
  \\ read
  ...
}
```

how do we express that a thread has to wait if a writer is writing?

Answer

```
if (this.writing) {
  this.wait();
}
```

The wait method

The `wait` method throws an `InterruptedException` if any thread interrupted the current thread before or while the current thread was waiting for a notification.

Since an `InterruptedException` is a checked exception, it needs to be specified or caught.

Catch the InterruptedException

```
public void read() {
  if (this.writing) {
    try {
      this.wait();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
  \\ read
  ...
}
```

Specify the InterruptedException

```
public void read() throws InterruptedException {
  if (this.writing) {
    this.wait();
  }
  \\ read
  ...
}
```

The wait method

When invoking `object.wait()`, the current thread must own the lock (or monitor) of `object`. If that is not the case, a `IllegalMonitorStateException` is thrown.

Question

How can we ensure that the current thread owns the lock of the database when executing `wait` within the `read` method?

Acquiring the lock of the database

```
private synchronized void beginRead() {
  if (this.writing) {
    try {
      this.wait();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}

public void read() {
  beginRead();
  \\ read
  ...
}
```

The writing attribute

Question

Where and how do we modify the value of the attribute `writing`?

The writing attribute

Question

Where and how do we modify the value of the attribute `writing`?

Answer

```
public void write() {
  ...
  this.writing = true;
  // write
  this.writing = false;
  ...
}
```

Waiting when a reader is reading

Question

In

```
public void write() {
  ...
  \\ write
  ...
}
```

how do we express that a thread has to wait if a writer is writing or a reader is reading?

Waiting when a reader is reading

Question

In

```
public void write() {
  ...
  \\ write
  ...
}
```

how do we express that a thread has to wait if a writer is writing or a reader is reading?

Answer

```
if (this.writing || this.reading) {
  this.wait();
}
```

The reading attribute

Question

Where and how do we modify the value of the attribute `reading`?

The reading attribute

Question

Where and how do we modify the value of the attribute `reading`?

Answer

```
public void read() {
  ...
  this.reading = true;
  // read
  this.reading = false;
  ...
}
```

The reading attribute

Question

Where and how do we modify the value of the attribute `reading`?

Answer

```
public void read() {
  ...
  this.reading = true;
  // read
  this.reading = false;
  ...
}
```

Since multiple readers can read at the same time, we cannot set the attribute `reading` to false after `// read`.

Waiting when a reader is reading

We need more fine-grained information than a boolean that captures whether readers are reading. From this more fine-grained information we should be able to derive whether readers are reading.

Waiting when a reader is reading

We need more fine-grained information than a boolean that captures whether readers are reading. From this more fine-grained information we should be able to derive whether readers are reading.

Question

What type of more fine-grained information is needed?

Waiting when a reader is reading

We need more fine-grained information than a boolean that captures whether readers are reading. From this more fine-grained information we should be able to derive whether readers are reading.

Question

What type of more fine-grained information is needed?

Answer

`int` to keep track of the number of active readers.

Waiting when a reader is reading

We need more fine-grained information than a boolean that captures whether readers are reading. From this more fine-grained information we should be able to derive whether readers are reading.

Question

What type of more fine-grained information is needed?

Answer

`int` to keep track of the number of active readers.

Question

What is an appropriate name for this attribute?

Waiting when a reader is reading

We need more fine-grained information than a boolean that captures whether readers are reading. From this more fine-grained information we should be able to derive whether readers are reading.

Question

What `type` of more fine-grained information is needed?

Answer

`int` to keep track of the number of active readers.

Question

What is an appropriate name for this attribute?

Answer

`readers`.

Initializing the attributes

Question

```
public class Database {
  private boolean writing;
  private int readers;

  ...
}
```

Where and how are the attributes `writing` and `readers` initialized?

Initializing the attributes

Question

```
public class Database {
  private boolean writing;
  private int readers;

  ...
}
```

Where and how are the attributes `writing` and `readers` initialized?

Answer

```
public Database() {
  this.writing = false;
  this.readers = 0;
}
```

Waiting when a reader is reading

Question

In

```
public void write() {
  this.beginWrite();
  ...
}
```

how do we express that a thread has to wait if a writer is writing or a reader is reading?

Waiting when a reader is reading

Question

In

```
public void write() {
  this.beginWrite();
  ...
}
```

how do we express that a thread has to wait if a writer is writing or a reader is reading?

Answer

```
private synchronized void beginWrite() {
  if (this.writing || this.readers > 0) {
    try {
      this.wait();
    } catch (InterruptedException e) {
      e.printStackTrace();
```

The reading attribute

Question

Where and how do we modify the value of the attribute `readers`?

The reading attribute

Question

Where and how do we modify the value of the attribute `readers`?

Answer

```
private synchronized void beginRead() {
  ...
  this.readers++;
}

private synchronized void endRead() {
  this.readers--;
}
```

Waking up waiting readers

Question

Readers may be waiting because a writer is writing. Where and how do we "wake up" these waiting readers?

Waking up waiting readers

Question

Readers may be waiting because a writer is writing. Where and how do we "wake up" these waiting readers?

Answer

Use the `notifyAll` once the writer is done with writing.

Waking up waiting readers

```
private synchronized void endWrite() {
  this.writing = false;
  this.notifyAll(); // notify all threads that are
                    // waiting on this database
}
```

Waking up waiting writers

Question

Writers may be waiting because a writer is writing or readers are reading. Where and how do we "wake up" a waiting writer?

Waking up waiting writers

Question

Writers may be waiting because a writer is writing or readers are reading. Where and how do we "wake up" a waiting writer?

Answer

Use the `notifyAll` once the last reader is done with reading.

Waking up waiting writers

```
private synchronized void endRead() {
  this.readers--:
  if (this.readers == 0) {
    this.notifyAll(); // notify all threads that are
                      // waiting on this database
  }
}
```

Correct?

Question

Is the developed class `Database` correct?

Correct?

Question

Is the developed class `Database` correct?

Answer

Maybe.

Correct?

Question

Is the developed class `Database` correct?

Answer

Maybe.

In the next lecture, we will use JPF to hunt for bugs in the `Database` class.