Concurrency

EECS 4315

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

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

```java
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 {
  private boolean writing;
  private boolean reading;

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

Write method

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

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

Read method

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

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

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.

Let us use JPF to try to find bugs in the `Database` class.

Configuration file

```
target=concurrency.ReadersAndWriters
classpath=/courses/4315/workspace/concurrency/bin/
```

JPF report

```
JavaPathfinder core system v8.0 (rev 2+) - (C) 2005-2014 Un
====================================== syst
concurrency.ReadersAndWriters.main()
====================================== sear
====================================== resu
no errors detected
====================================== stat
elapsed time:       00:00:11
states:             new=28983,visited=64764,backtracked=93747
search:             maxDepth=49,constraints=0
choice generators: thread=28983 (signal=2517,lock=8221,shar
heap:               new=400,released=157142,maxLive=386,gcCyc
instructions:       470903
max memory:         372MB
loaded code:        classes=61,methods=1381
====================================== sear
```

No writer

Question

How can we use JPF to check that there is no writer writing when a reader is reading?

No writer

Question

How can we use JPF to check that there is no writer writing when a reader is reading?

Answer

Add `assert !this.writing` in the `read` method where the database is read. If the assertion fails, an exception is thrown. JPF detects exceptions that are thrown and not caught.

JPF report

```
JavaPathfinder core system v8.0 (rev 2+) - (C) 2005-2014 Un
==================================================== syst
concurrency.ReadersAndWriters.main()
==================================================== sear
==================================================== erro
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError
at concurrency.Database.read(concurrency/Database.java:28)
at concurrency.Reader.run(concurrency/Reader.java:25)
==================================================== snap
thread concurrency.Reader:{id:2,name:Thread-2,status:RUNNIN
call stack:
at concurrency.Database.read(Database.java:28)
at concurrency.Reader.run(Reader.java:25)

thread concurrency.Writer:{id:4,name:Thread-4,status:RUNNIN
call stack:
```

Smallest instance

Try to find the smallest instance for which the error occurs.

Smallest instance

Try to find the smallest instance for which the error occurs.

```
READER = 1
WRITERS = 1
no errors detected

READER = 2
WRITERS = 1
no errors detected

READER = 1
WRITERS = 2
error
```

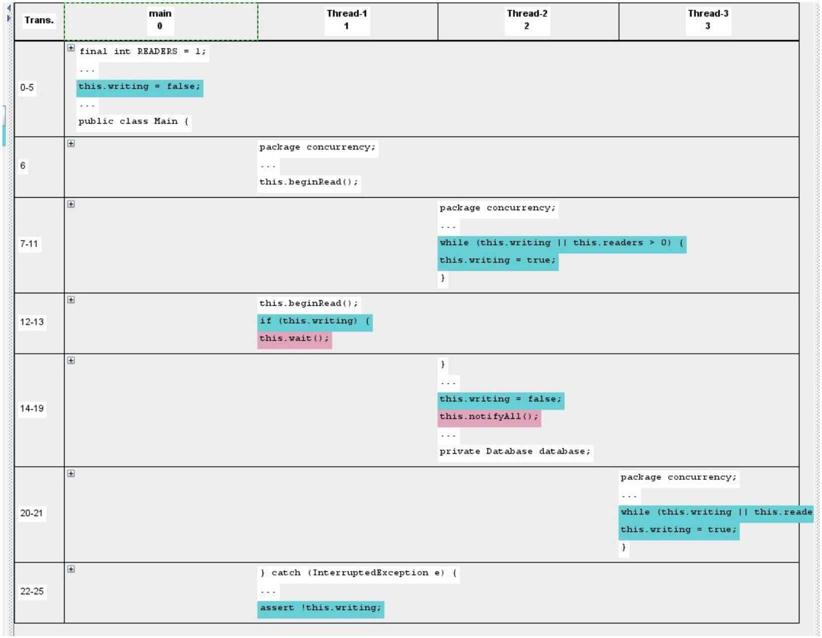
Locating the error

```
==================================================== erro
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError
at concurrency.Database.read(concurrency/Database.java:28)
at concurrency.Reader.run(concurrency/Reader.java:25)
```

Line 28 of the `Database` class.

```
assert !this.writing;
```

jpf-visual



Trans.	main 0	Thread-1 1	Thread-2 2	Thread-3 3

0-5	`final int READERS = 1;`
	`...`
	`this.writing = false;`
	`...`
	`public class Main {`			

6		`package concurrency;`
		`...`
		`this.beginRead();`		

7-11			`package concurrency;`
			`...`
			`while (this.writing || this.readers > 0) {`
			`this.writing = true;`
			`}`	

12-13		`this.beginRead();`
		`if (this.writing) {`
		`this.wait();`		

14-19			`}`
			`...`
			`this.writing = false;`
			`this.notifyAll();`
			`...`
			`private Database database;`	

20-21				`package concurrency;`
				`...`
				`while (this.writing || this.reade`
				`this.writing = true;`
				`}`

22-25		`} catch (InterruptedException e) {`
		`...`
		`assert !this.writing;`		

Trace

main: running

```
final int READERS = 1;
final int WRITERS = 2;
Database database = new Database();
for (int r = 0; r < READERS; r++) {
  (new Reader(database)).start();
}
```

main: running, Reader: runnable

Trace

`main`: running, `Reader`: runnable

```
for (int w = 0; w < WRITERS; w++) {
  (new Writer(database)).start();
}
```

`main`: running, `Reader`: runnable, `Writer`: runnable, `Writer`: runnable

Trace

`main`: runnable, `Reader`: running, `Writer`: runnable, `Writer`: runnable

```
this.database.read();
```

`main`: runnable, `Reader`: running, `Writer`: runnable, `Writer`: runnable

Trace

`main`: runnable, `Reader`: running, `Writer`: runnable, `Writer`: runnable

```
this.beginRead();
```

`main`: runnable, `Reader`: running, `Writer`: runnable, `Writer`: runnable

Trace

main: runnable, Reader: runnable, Writer: running, Writer: runnable

```
this.database.write();
```

main: runnable, Reader: runnable, Writer: running, Writer: runnable

Trace

`main`: runnable, `Reader`: runnable, `Writer`: running,
`Writer`: runnable

```
this.beginWrite();
```

`main`: runnable, `Reader`: runnable, `Writer`: running,
`Writer`: runnable

Trace

main: runnable, Reader: runnable, Writer: running,
Writer: runnable

```
if (this.writing || this.readers > 0) {
  try {
    this.wait();
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
this.writing = true;
```

main: runnable, Reader: runnable, Writer: running,
Writer: runnable

Trace

main: runnable, Reader: running, Writer: runnable, Writer: runnable

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

main: runnable, Reader: blocked, Writer: runnable, Writer: runnable

Trace

`main`: runnable, `Reader`: blocked, `Writer`: running, `Writer`: runnable

```
assert this.readers == 0;
this.endWrite();
```

`main`: runnable, `Reader`: blocked, `Writer`: running, `Writer`: runnable

Trace

`main`: runnable, `Reader`: blocked, `Writer`: running,
`Writer`: runnable

```
this.writing = false;
this.notifyAll();
```

`main`: runnable, `Reader`: runnable, `Writer`: running,
`Writer`: runnable

Trace

`main`: runnable, `Reader`: runnable, `Writer`: runnable,
`Writer`: running

```
this.database.write();
```

`main`: runnable, `Reader`: runnable, `Writer`: runnable,
`Writer`: running

Trace

`main`: runnable, `Reader`: runnable, `Writer`: runnable,
`Writer`: running

```
this.beginWrite();
```

`main`: runnable, `Reader`: runnable, `Writer`: runnable,
`Writer`: running

Trace

main: runnable, Reader: runnable, Writer: runnable,
Writer: running

```
if (this.writing || this.readers > 0) {
  try {
    this.wait();
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
this.writing = true;
```

main: runnable, Reader: runnable, Writer: runnable,
Writer: running

Trace

`main`: runnable, `Reader`: running, `Writer`: runnable, `Writer`: runnable

```
this.readers++;
assert !this.writing;
```

`main`: runnable, `Reader`: running, `Writer`: runnable, `Writer`: runnable

Bug

```
if (this.writing) {
  try {
    this.wait();
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
this.readers++;
```

Although the attribute `waiting` was `false` when the state of the `Reader` thread changed from blocked to runnable, it was not any more when the state of the `Reader` thread changed from runnable to running.

Bug

```
if (this.writing) {
  try {
    this.wait();
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
this.readers++;
```

Question

How do we modify the above code so that we check that `waiting` is `false` when the state of the `Reader` thread changed from runnable to running?

Bug

```
if (this.writing) {
  try {
    this.wait();
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
this.readers++;
```

Bug

```
if (this.writing) {
  try {
    this.wait();
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
this.readers++;
```

Answer

Replace `if` with `while`.

No reader

Question

How can we use JPF to check that there is no reader reading when a writer is writing?

No reader

Question

How can we use JPF to check that there is no reader reading when a writer is writing?

Answer

Add `assert this.readers == 0` in the `write` method where the database is written.

No other writer

Question

How can we use JPF to check that there is no other writer writing when a writer is writing?

No other writer

Question

How can we use JPF to check that there is no other writer writing when a writer is writing?

Answer

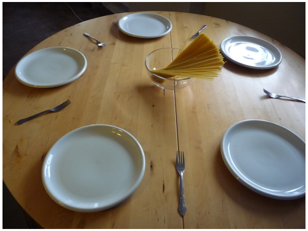
- Add attribute `writers`.
- Initialize `writers` to zero.
- Increment and decrement `writers` in the `write` method.
- Add `assert this.writers == 1` in the `write` method where the database is written.

Synchronized blocks

```
public void read() {
  synchronized(this) {
    while (this.writing) {
      this.wait();
    }
    this.readers++;
  }
  // read
  assert !this.writing;
  synchronized (this) {
    this.readers--;
    if (this.readers == 0) {
      this.notifyAll();
    }
  }
}
```

The dining philosophers problem

In the dining philosophers problem, due to Dijkstra, five philosophers are seated around a round table. Each philosopher has a plate of spaghetti. A philosopher needs two forks to eat it. The layout of the table is as follows.



The life of a philosopher consists of alternative periods of eating and thinking. When philosophers get hungry, they try to pick up their left and right fork, one at a time, in either order. If successful in picking up both forks, the philosopher eats for a while, then puts down the forks and continues to think.

The dining philosophers problem

```
public class Philosopher extends Thread {
  private int id;
  private Table table;

  public Philosopher(int id, Table table) {
    this.id = id;
    this.table = table;
  }

  public void run() {
    while (true) {
      this.table.pickUp(id);
      this.table.pickUp((id + 1) % 5);
      // eat
      this.table.putDown(id);
      this.table.putDown((id + 1) % 5);
    }
```

The dining philosophers problem

```
public class Table {
  public Table() { ... }
  public void pickUp(int id) { ... }
  public void putDown(int id) { ... }
}
```

The dining philosophers problem

```
public class Philosophers {
  public static void main(String[] args) {
    Table table = new Table();
    for (int p = 0; p < 5; p++) {
      (new Philosopher(p, table)).start();
    }
  }
}
```

Table

Question

Of what information about table and its forks should we keep track?

Table

Question

Of what information about table and its forks should we keep track?

Answer

Which forks have been picked up.

Table

Question

Of what information about table and its forks should we keep track?

Answer

Which forks have been picked up.

Question

How do we represent this information?

Table

Question

Of what information about table and its forks should we keep track?

Answer

Which forks have been picked up.

Question

How do we represent this information?

Answer

As an attribute of type `boolean[]`.

Table

Question

Where and how do we initialize the attribute?

Table

Question

Where and how do we initialize the attribute?

Answer

```
private boolean[] pickedUp;

public Table() {
  this.pickedUp = new boolean[5]; // all false
}
```

Table

Question

Implement the method `pickUp(int id)`.

- When does a `Philosopher` have to wait?
- How does the array `pickedUp` need to be updated?

Table

Question

Implement the method `pickUp(int id)`.

- When does a `Philosopher` have to wait?
- How does the array `pickedUp` need to be updated?

Answer

```
while (this.pickedUp[id]) {
  this.wait();
}
this.pickedUp[id] = true;
```

Table

Question

Implement the method `putDown(int id)`.

- How does the array `pickedUp` need to be updated?
- Do `Philosopher`s need to be notified?

Table

Question

Implement the method `putDown(int id)`.

- How does the array `pickedUp` need to be updated?
- Do `Philosopher`s need to be notified?

Answer

```
this.pickedUp[id] = false;
this.notifyAll();
```

The dining philosophers problem

Question

Does this solve the problem?

The dining philosophers problem

Question

Does this solve the problem?

Answer

No.

The dining philosophers problem

Question

Does this solve the problem?

Answer

No.

Question

Why not?

The dining philosophers problem

Question

Does this solve the problem?

Answer

No.

Question

Why not?

Answer

Deadlock.

JPF

```
JavaPathfinder core system v8.0 (rev 32+) - (C) 2005-2014 U

===================================== syst
concurrency.Philosophers.main()

===================================== sear

===================================== erro
gov.nasa.jpf.vm.NotDeadlockedProperty
deadlock encountered:
thread concurrency.Philosopher:{id:1,name:Thread-1,status:W
thread concurrency.Philosopher:{id:2,name:Thread-2,status:W
thread concurrency.Philosopher:{id:3,name:Thread-3,status:W
thread concurrency.Philosopher:{id:4,name:Thread-4,status:W
thread concurrency.Philosopher:{id:5,name:Thread-5,status:W
...
```

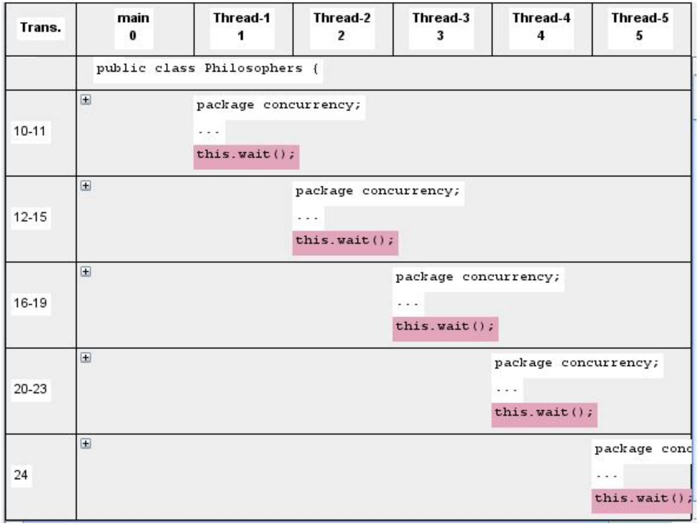
jpf-visual

```
target=Philosophers
classpath=<path to Philosophers.class>
sourcepath=<path to Philosophers.java>

@using=jpf-visual

report.errorTracePrinter.property_violation=trace
report.publisher+=,errorTracePrinter
report.errorTracePrinter.class=ErrorTracePrinter
shell=gov.nasa.jpf.shell.basicshell.BasicShell
shell.panels+=,errorTrace
shell.panels.errorTrace=ErrorTracePanel
```

jpf-visual



Trans.	main 0	Thread-1 1	Thread-2 2	Thread-3 3	Thread-4 4	Thread-5 5
	public class Philosophers {					
10-11	⊞	package concurrency; ... this.wait();				
12-15	⊞		package concurrency; ... this.wait();			
16-19	⊞			package concurrency; ... this.wait();		
20-23	⊞				package concurrency; ... this.wait();	
24	⊞					package conc ... this.wait();

Bug

All five philosophers pick up their left fork first and then all wait for their right fork.

Solutions:

- One left handed philosophers (picks up left fork first) and four right handed philosophers (pick up right forks first)
- Only allow at most four philosophers to enter the dining room
- Keep track of each philosopher (thinking, hungry, eating)