Concurrency

EECS 4315

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

Counter class

Problem

Implement the class `Counter` with attribute `value`, initialized to zero, and the methods `increment` and `decrement`.

Counter class

```
public class Counter {
  private int value;

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

  public void increment() {
    this.value++;
  }

  public void decrement() {
    this.value--;
  }
}
```

Counter class

Question

Can multiple threads share a `Counter` object and use methods such as `increment` and `decrement` concurrently?

Counter class

Question

Can multiple threads share a `Counter` object and use methods such as `increment` and `decrement` concurrently?

Answer

Yes, but, as before, if two threads invoke `increment` concurrently, the counter may only be incremented by one (rather than two).

Synchronized methods

Methods such as `increment` should be executed atomically. This can be accomplished by declaring the method to be `synchronized`.

A lock is associated with every object. For threads to execute a `synchronized` method on an object, first its lock needs to be acquired.

Synchronized methods

Methods such as `increment` should be executed atomically. This can be accomplished by declaring the method to be `synchronized`.

A lock is associated with every object. For threads to execute a `synchronized` method on an object, first its lock needs to be acquired.

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

public synchronized void decrement() {
  this.value--;
}
```

Resource class

Problem

Implement the class `Resource` with attribute `available`, initialized to true, and the methods `acquire` and `release`.

Resource class

```
public class Resource {
  private boolean available;

  public Resource() {
    this.available = true;
  }

  public void acquire() {
    this.available = false;
  }

  public void release() {
    this.available = true;
  }
}
```

User class

Problem

Implement the class `User` that extends the `Thread` class. The class contains a static attribute of type `Resource`, a resource shared among users. In its `run` method acquires and subsequently releases that resource.

User class

```
public class User extends Thread {
  private static Resource resource = new Resource();

  public void run() {
    resource.acquire();
    resource.release();
  }
}
```

Multiple users

Problem

Implement a class whose `main` method runs multiple `User`s concurrently.

Main method

```
final int USERS = Integer.parseInt(args[0]);
final User[] users = new User[USERS];
for (int i = 0; i < USERS; i++) {
  users[i] = new User();
}
for (int i = 0; i < USERS; i++) {
  users[i].start();
}
```

Multiple users

Problem

Verify, using JPF, if at most one `User` has the resource at any time.

Multiple users

Question

How can we keep track how many `User`s have acquired the resource?

Multiple users

Question

How can we keep track how many `User`s have acquired the resource?

Answer

Introduce a static attribute capturing number of invocations of acquire - number of invocations of release.

Multiple users

Question

How can we keep track how many `User`s have acquired the resource?

Answer

Introduce a static attribute capturing number of invocations of acquire - number of invocations of release.

Such attributes are helpful in the verification process, but are not needed for the code to execute. They are sometimes called "ghost variables."

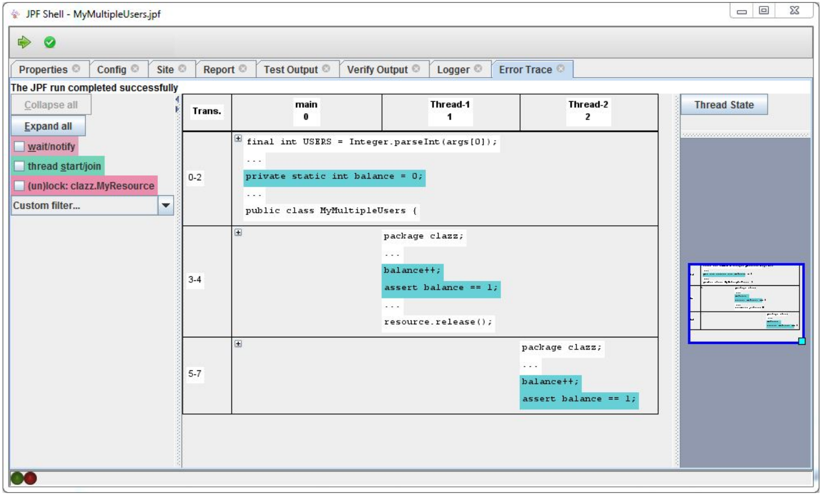
Multiple users

```
private static int balance = 0;

public synchronized void acquire() {
  balance++;
  assert balance == 1;
  this.available = false;
}

public synchronized void release() {
  this.available = true;
  balance--;
}
```

Multiple users

```
target=MultipleUsers
target.args=2
classpath=<path to MultipleUsers.class>
sourcepath=<path to MultipleUsers.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
```

Multiple users



The JPF run completed successfully

Trans.	main 0	Thread-1 1	Thread-2 2
0-2	`final int USERS = Integer.parseInt(args[0]);` `...` `private static int balance = 0;` `...` `public class MyMultipleUsers {`		
3-4		`package clazz;` `...` `balance++;` `assert balance == 1;` `...` `resource.release();`	
5-7			`package clazz;` `...` `balance++;` `assert balance == 1;`

Wait and notify

Multiple users can acquire the resource at the same time. To avoid that, we exploit the following methods.

The `Object` class contains the following three methods:

- `wait`: causes the current thread to wait for this object's lock until another thread wakes it up.
- `notify`: wakes up a single thread waiting on this object's lock; if there is more than one waiting, an arbitrary one is chosen; if there are none, nothing is done.
- `notifyAll`: wakes up all threads waiting on this objects lock.

Wait and notify

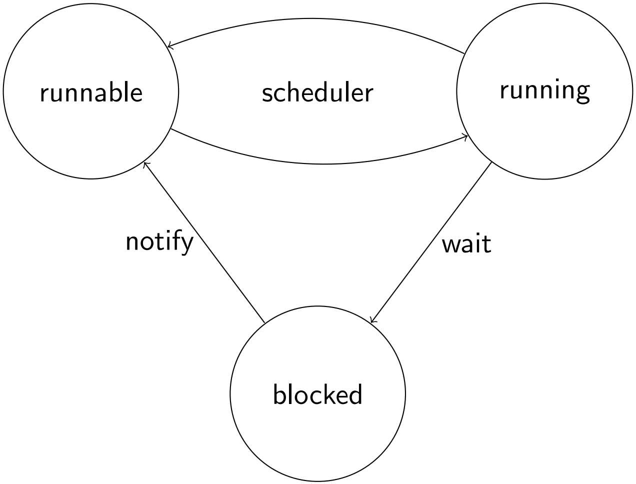
Multiple users can acquire the resource at the same time. To avoid that, we exploit the following methods.

The `Object` class contains the following three methods:

- `wait`: causes the current thread to wait for this object's lock until another thread wakes it up.
- `notify`: wakes up a single thread waiting on this object's lock; if there is more than one waiting, an arbitrary one is chosen; if there are none, nothing is done.
- `notifyAll`: wakes up all threads waiting on this objects lock.

Since every class extends the class `Object`, these methods are available to every object.

States of a thread



Wait and notify

General questions to ask:

- When does a thread have to wait?
- When can a waiting thread potentially continue?

Wait and notify

Question

When does a `User` have to wait?

Wait and notify

Question

When does a `User` have to wait?

Answer

When the resource is not available.

Wait and notify

Question

When does a `User` have to wait?

Answer

When the resource is not available.

Question

When can a waiting `User` potentially continue?

Wait and notify

Question

When does a `User` have to wait?

Answer

When the resource is not available.

Question

When can a waiting `User` potentially continue?

Answer

When another `User` releases the resource.

Wait

A `User` has to wait when the resource is not available.

Question

In which method does a `User` have to wait: `acquire` or `release`?

Wait

A `User` has to wait when the resource is not available.

Question

In which method does a `User` have to wait: `acquire` or `release`?

Answer

`acquire`

Wait

A User has to wait in the acquire method when the resource is not available.

```
public synchronized void acquire() {
  this.available = false;
}
```

Question

Add a call to wait to the acquire method.

Wait

A User has to wait in the acquire method when the resource is not available.

```
public synchronized void acquire() {
  this.available = false;
}
```

Question

Add a call to wait to the acquire method.

Answer

```
public synchronized void acquire() {
  if (!this.available) {
    this.wait();
  }
  this.available = false;
}
```

Wait

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

Wait

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

Since this is a checked exception, it has to be caught or specified.

Wait

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

Since this is a checked exception, it has to be caught or specified.

Question

Catch the `InterruptedException`.

Wait

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

Since this is a checked exception, it has to be caught or specified.

Question

Catch the `InterruptedException`.

Answer

```
public synchronized void acquire() {
  if (!this.available) {
    try {
      this.wait();
    } catch (InterruptedException e) {}
  }
  this.available = false;
}
```

Notify

A `User` can potentially continue when another `User` releases the resource.

Question

In which method does the other `User` signal that waiting `User` can potentially continue: `acquire` or `release`?

Notify

A `User` can potentially continue when another `User` releases the resource.

Question

In which method does the other `User` signal that waiting `User` can potentially continue: `acquire` or `release`?

Answer

`release`

Wait

Another User signals in the release method that waiting User can potentially continue.

```
public synchronized void release() {
  this.available = true;
}
```

Question

Add a call to notify to the release method.

Wait

Another User signals in the release method that waiting User can potentially continue.

```
public synchronized void release() {
  this.available = true;
}
```

Question

Add a call to notify to the release method.

Answer

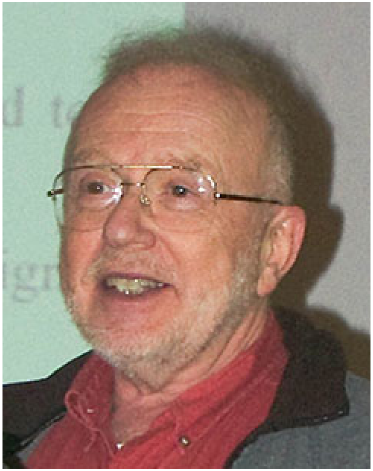
```
public synchronized void release() {
  this.available = true;
  this.notify();
}
```

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

In which method does a reader have to wait: `read` or `write`?

The readers-writers problem

Question

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

Answer

When a writer is writing.

Question

In which method does a reader have to wait: `read` or `write`?

Answer

`read`.

The readers-writers problem

Question

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

The readers-writers problem

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 readers-writers problem

Question

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

Answer

When another writer is writing or a reader is reading.

Question

In which method does a writer have to wait: `read` or `write`?

The readers-writers problem

Question

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

Answer

When another writer is writing or a reader is reading.

Question

In which method does a writer have to wait: `read` or `write`?

Answer

`write`.

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

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) {}
  }
}

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

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) {}
```

The readers attribute

Question

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

The readers 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--;
}
```

Drop deadline

The last date to drop the course without receiving a grade for it is Friday March 13.