Deadlocks and Race Conditions

EECS 4315

www.cse.yorku.ca/course/4315/

Deadlock

Deadlock: two or more threads are each waiting for the other.

Deadlock

Deadlock: two or more threads are each waiting for the other.

Question

In Java, what causes a thread to wait for another thread?

Deadlock

Deadlock: two or more threads are each waiting for the other.

Question

In Java, what causes a thread to wait for another thread?

Answer

One thread waits for a lock held by another thread.

Deadlock

Deadlock: two or more threads are each waiting for the other.

Question

In Java, what causes a thread to wait for another thread?

Answer

One thread waits for a lock held by another thread.

Question

Give a Java app in which one thread waits for the other and vice versa.

Deadlock

```
public class Locker extends Thread {
  private Locker other;

  public void setOther(Locker other) {
    this.other = other;
  }

  public void run() {
    synchronized(this) {
      synchronized(this.other) {
        // do nothing
      }
    }
    System.out.println("done");
  }
}
```

Deadlock

```
public class TwoLocks {
  public static void main(String[] args) {
    Locker one = new Locker();
    Locker another = new Locker();
    one.setOther(another);
    another.setOther(one);
    one.start();
    another.start();
  }
}
```

Deadlock

JPF checks by default for deadlocks.

```
target=TwoLocks
classpath=.
```

Deadlock

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

```
target=Twolocks
classpath=.
listener=gov.nasa.jpf.listener.StateSpaceDot
```

Deadlock

An example of the Java tutorial.

```
public class Friend {
  private final String name;

  public Friend(String name) {
    this.name = name;
  }

  public String getName() {
    return this.name;
  }
```

Deadlock

```
  public synchronized void bow(Friend bower) {
    System.out.format("%s: %s has bowed to me!%n",
      this.getName(), bower.getName());
    bower.bowBack(this);
  }

  public synchronized void bowBack(Friend bower) {
    System.out.format("%s: %s has bowed back to me!
      this.getName(), bower.getName());
  }
}
```

Deadlock

```
public class TwoFriends {
  public static void main(String[] args) {
    final Friend alphonse = new Friend("Alphonse");
    final Friend gaston = new Friend("Gaston");
    new Thread(new Runnable() {
      public void run() { alphonse.bow(gaston); }
    }).start();
    new Thread(new Runnable() {
      public void run() { gaston.bow(alphonse); }
    }).start();
  }
}
```

Anonymous Class

```
new Runnable() {
  public void run() { alphonse.bow(gaston); }
}
```

is an anonymous class expression.

Anonymous Class

The anonymous class expression consists of

- the new operator,
- the name of an interface to implement or a class to extend,
- parentheses that contain the arguments to a constructor, just like a normal class instance creation expression,[1]
- a body, which is a class declaration body.

[1] When you implement an interface, there is no constructor, so you use an empty pair of parentheses.

Deadlock

JPF checks by default for deadlocks.

```
target=TwoFriends
classpath=.
```

Deadlock

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

```
target=Friends
classpath=.
listener=gov.nasa.jpf.listener.StateSpaceDot
```

Deadlock

An example that comes with JPF: `DiningPhil`

The Java source code of the JPF examples can be found in jpf/jpf-core/src/examples/

The Java source code of the JPF examples can be found in jpf/jpf-core/build/examples/

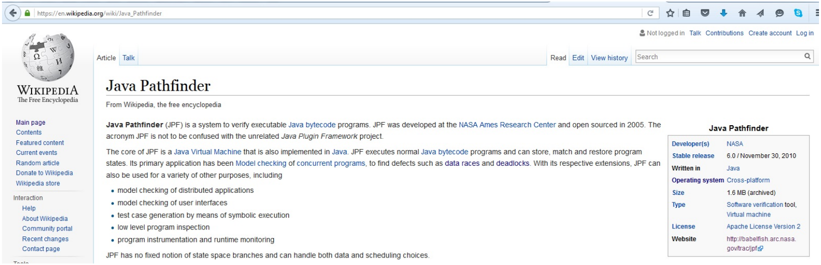
Race Condition

Race condition: two threads access the same shared data at the same time and at least one of the two threads writes.

Race conditions are also known as data races.

Race conditions are not bugs per se, but often indicate potential trouble spots in the code.

JPF’s Wikipedia Page



https://en.wikipedia.org/wiki/Java_Pathfinder

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Java Pathfinder

From Wikipedia, the free encyclopedia

Java Pathfinder (JPF) is a system to verify executable Java bytecode programs. JPF was developed at the NASA Ames Research Center and open sourced in 2005. The acronym JPF is not to be confused with the unrelated *Java Plugin Framework* project.

The core of JPF is a Java Virtual Machine that is also implemented in Java. JPF executes normal Java bytecode programs and can store, match and restore program states. Its primary application has been Model checking of concurrent programs, to find defects such as data races and deadlocks. With its respective extensions, JPF can also be used for a variety of other purposes, including

- model checking of distributed applications
- model checking of user interfaces
- test case generation by means of symbolic execution
- low level program inspection
- program instrumentation and runtime monitoring

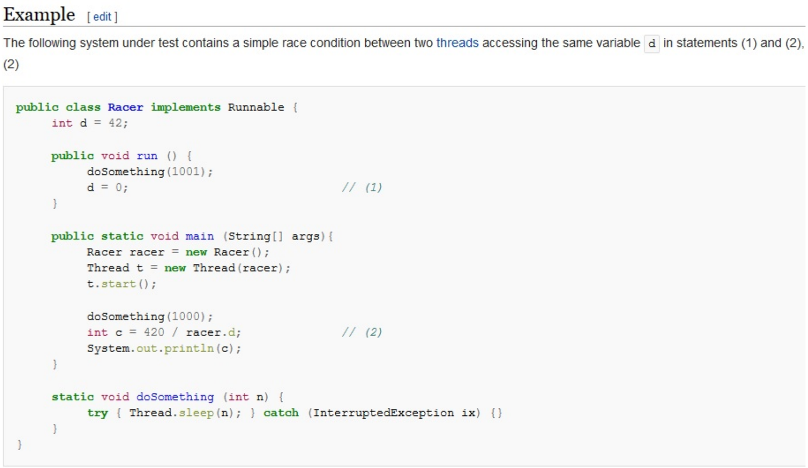
JPF has no fixed notion of state space branches and can handle both data and scheduling choices.

Java Pathfinder

Developer(s)	NASA
Stable release	6.0 / November 30, 2010
Written in	Java
Operating system	Cross-platform
Size	1.6 MB (archived)
Type	Software verification tool, Virtual machine
License	Apache License Version 2
Website	http://babelfish.arc.nasa.gov/trac/jpf

Its primary application has been model checking of concurrent programs, to find defects such as data races and deadlocks.

JPF’s Wikipedia Page



Example [edit]

The following system under test contains a simple race condition between two threads accessing the same variable `d` in statements (1) and (2), (2)

```
public class Racer implements Runnable {
     int d = 42;

     public void run () {
          doSomething(1001);
          d = 0;                              // (1)
     }

     public static void main (String[] args){
          Racer racer = new Racer();
          Thread t = new Thread(racer);
          t.start();

          doSomething(1000);
          int c = 420 / racer.d;              // (2)
          System.out.println(c);
     }

     static void doSomething (int n) {
          try { Thread.sleep(n); } catch (InterruptedException ix) {}
     }
}
```