What does threadsafe mean?

Asked 14 years, 11 months ago Modified 1 year, 4 months ago

Viewed 113k times

164

Recently I tried to Access a textbox from a thread (other than the UI thread) and an exception was thrown. It said something about the "code not being thread safe" and so I ended up writing a delegate (sample from MSDN helped) and calling it instead.

But even so I didn't quite understand why all the extra code was necessary.

Update: Will I run into any serious problems if I check

```
Controls.CheckForIllegalCrossThread..blah =true
```

multithreading thread-safety definition

Share

Improve this question

Follow

edited Dec 6, 2016 at 22:34

community wiki

7 revs, 5 users 50%

Vivek Bernard

6 Typically, "thread safe" means whatever the person using the term thinks it means, at least to that person. As such, it is not a very useful language construct - you need to be much, much more specific when talking about the behaviour of threaded code. – anon Jan 9, 2010 at 15:54

8 Duplicate?: stackoverflow.com/questions/261683/…
– Dave O. Jan 9, 2010 at 15:54

@dave Sorry I tried searching, but gave up...thanks anyway..
– Vivek Bernard Jan 9, 2010 at 16:25

1 a code that doesn't arises `Race-Condition`
– Muhammad Babar Nov 7, 2014 at 13:36

12 Answers

Sorted by: Highest score (default)

169

Eric Lippert has a nice blog post entitled What is this thing you call "thread safe"? about the definition of thread safety as found of Wikipedia.

3 important things extracted from the links :

> “A piece of code is thread-safe if it functions correctly during simultaneous execution by multiple threads.”
>
> “In particular, it must satisfy the need for multiple threads to access the same shared data, …”
>
> “…and the need for a shared piece of data to be accessed by only one thread at any given time.”

Definitely worth a read!

Share Improve this answer

Follow

edited Sep 19, 2020 at 14:46

Alexis
4,457 ● 1 ● 27 ● 35

answered Jan 9, 2010 at 15:51

Gregory Pakosz
70.2k ● 20 ● 142 ● 165

35 Please avoid link only answers as it may become bad anytime in future. – akhil_mittal Aug 23, 2015 at 5:40

1 updated link: learn.microsoft.com/en-nz/archive/blogs/ericlippert/… – Ryan Buddicom Jan 14, 2020 at 22:45

148

In the simplest of terms threadsafe means that it is safe to be accessed from multiple threads. When you are using multiple threads in a program and they are each attempting to access a common data structure or location in memory several bad things can happen. So, you add some extra code to prevent those bad things. For example, if two people were writing the same document at the same time, the second person to save will overwrite the work of the first person. To make it thread safe then, you have to force person 2 to wait for person 1 to complete their task before allowing person 2 to edit the document.

Share Improve this answer

edited Jun 24, 2018 at 15:31

Follow

answered Jan 9, 2010 at 15:48

Vincent Ramdhanie
103k ● 23 ● 142 ● 194

18 This is called synchronization. Right? – JavaTechnical Feb 7, 2014 at 19:12

3 Yes. Forcing the various threads to wait for access to a shared resource can be accomplished with synchronization. – Vincent Ramdhanie Feb 7, 2014 at 22:29

From Gregory's accepted answer, he is saying ""A piece of code is thread-safe if it functions correctly during simultaneous execution by multiple threads." while you are saying "To make it thread safe then, you have to force person 1 to wait "; isn't he saying simultaneous is acceptable while you are saying it's not? Can you please explain? – mfaani Mar 28, 2016 at 13:54

Its the same thing. I am just suggesting a simple mechanism as an example of what makes code threadsafe. regardless of the mechanism used though multiple threads running the same code should not interfere with each other. – Vincent Ramdhanie Mar 28, 2016 at 15:49

So does this only apply to code utilizing global and static variables then? Using your example of people editing documents, I suppose it doesn't make sense to prevent person 2 from running the document-writing code on another document. – Aaron Franke Jan 6, 2017 at 7:59

Wikipedia has an article on Thread Safety.

24

This definitions page (you have to skip an ad - sorry) defines it thus:

> In computer programming, thread-safe describes a program portion or routine that can be called from multiple programming threads without unwanted interaction between the threads.

A thread is an execution path of a program. A single threaded program will only have one thread and so this problem doesn't arise. Virtually all GUI programs have multiple execution paths and hence threads - there are at least two, one for processing the display of the GUI and handing user input, and at least one other for actually performing the operations of the program.

This is done so that the UI is still responsive while the program is working by offloading any long running process to any non-UI threads. These threads may be created once and exist for the lifetime of the program, or just get created when needed and destroyed when they've finished.

As these threads will often need to perform common actions - disk i/o, outputting results to the screen etc. - these parts of the code will need to be written in such a way that they can handle being called from multiple threads, often at the same time. This will involve things like:

- Working on copies of data

- Adding locks around the critical code
- Opening files in the appropriate mode - so if reading, don't open the file for write as well.
- Coping with not having access to resources because they're locked by other threads/processes.

Share Improve this answer Follow

edited Jun 9, 2021 at 15:45

answered Jan 9, 2010 at 15:46

ChrisF ♦
137k ● 31 ● 262 ● 333

18

Simply, thread-safe means that a method or class instance can be used by multiple threads at the same time without any problems occurring.

Consider the following method:

```
private int myInt = 0;
public int AddOne()
{
    int tmp = myInt;
    tmp = tmp + 1;
    myInt = tmp;
    return tmp;
}
```

Now thread A and thread B both would like to execute `AddOne()`. but A starts first and reads the value of `myInt (0)` into `tmp`. Now for some reason, the scheduler decides to halt thread A and defer execution to thread B. Thread B now also reads the value of `myInt` (still 0) into it's own variable tmp. Thread B finishes the entire method so in the end `myInt = 1`. And 1 is returned. Now it's Thread A's turn again. Thread A continues. And adds 1 to tmp (tmp was 0 for thread A). And then saves this value in `myInt`. `myInt` is again 1.

So in this case the method `AddOne()` was called two times, but because the method was not implemented in a thread-safe way the value of `myInt` is not 2, as expected, but 1 because the second thread read the variable `myInt` before the first thread finished updating it.

Creating thread-safe methods is very hard in non-trivial cases. And there are quite a few techniques. In Java you can mark a method as `synchronized`, this means that only one thread can execute that method at a given time. The other threads wait in line. This makes a method thread-safe, but if there is a lot of work to be done in a method, then this wastes a lot of space. Another technique is to *'mark only a small part of a method as synchronized'* by creating a lock or semaphore, and locking this small part (usually called the critical section). There are even some methods that are implemented as lock-less thread-safe, which means that they are built in such a way that multiple threads can race through them at the same time without ever causing problems, this can be the case when a method only executes one atomic call. Atomic calls are calls that can't be interrupted and can only be done by one thread at a time.

Share Improve this answer Follow

edited Feb 2, 2021 at 0:29

community wiki
2 revs, 2 users 89%
Sujith PS

if method AddOne was called two times – Sujith PS Sep 19, 2014 at 5:03

13

Let's give real world example for a laymen explanation.

Let's suppose you have a bank account, and your account only has $100. You performed a $50 transfer balance to your brother's account, and at the very same time, simultaneously, your spouse was shopping using the same bank account and paid $80. If this bank account is not thread-safe, then the bank fumbled pretty bad, allowing you and your spouse to perform two transactions at the same time, and then the bank will become bankrupt!

The bank account is *the shared state*, and you and your spouse are *two different threads* trying to perform a write operation to the shared state.

Thread-safe means that multiple threads *cannot* simultaneously access an object's state, only one thread will perform the write/read operation first, and then the next thread will have its access, in an orderly manner.

So it's either you or your spouse will perform the first transaction successfully, but not both at the same time, *the bank account is thread safe*.

Share Improve this answer Follow

edited Aug 22, 2023 at 8:28

community wiki
4 revs, 3 users 50%
Yasir Shabbir Choudhary

7

You can get more explanation from the book "Java Concurrency in Practice":

> A class is thread-safe if it behaves correctly when accessed from multiple threads, regardless of the scheduling or interleaving of the execution of those threads by the runtime environment, and with no additional synchronization or other coordination on the part of the calling code.

Share Improve this answer Follow

answered Jul 15, 2014 at 3:04

community wiki
Jacky

5

A module is thread-safe if it guarantees it can maintain its invariants in the face of multi-threaded and concurrence use.

Here, a module can be a data-structure, class, object, method/procedure or function. Basically scoped piece of code and related data.

The guarantee can potentially be limited to certain environments such as a specific CPU architecture, but must hold for those environments. If there is no explicit delimitation of environments, then it is usually taken to imply that it holds for all environments that the code can be compiled and executed.

Thread-unsafe modules *may* function correctly under mutli-threaded and concurrent use, but this is often more down to luck and coincidence, than careful design. Even if some module does not break for you under, it may break when moved to other environments.

Multi-threading bugs are often hard to debug. Some of them only happen occasionally, while others manifest aggressively - this too, can be environment specific. They can manifest as subtly wrong results, or deadlocks. They can mess up data-structures in unpredictable ways, and cause other seemingly impossible bugs to appear in other remote parts of the code. It can be very application specific, so it is hard to give a general description.

Share Improve this answer answered Jan 9, 2010 at 16:15

Follow

Chris Vest
8,672 ● 3 ● 38 ● 43

4

Thread safety: A thread safe program protects it's data from memory consistency errors. In a highly multi-threaded program, a thread safe program does not cause any side effects with multiple read/write operations from multiple threads on same objects. Different threads can share and modify object data without consistency errors.

You can achieve thread safety by using advanced concurrency API. This documentation page provides good programming constructs to achieve thread safety.

Lock Objects support locking idioms that simplify many concurrent applications.

Executors define a high-level API for launching and managing threads. Executor implementations provided by java.util.concurrent provide thread pool management suitable for large-scale applications.

Concurrent Collections make it easier to manage large collections of data, and can greatly reduce the need for synchronization.

Atomic Variables have features that minimize synchronization and help avoid memory consistency errors.

ThreadLocalRandom (in JDK 7) provides efficient generation of pseudorandom numbers from multiple threads.

Refer to java.util.concurrent and java.util.concurrent.atomic packages too for other programming constructs.

Share Improve this answer Follow

edited May 20, 2016 at 17:26

community wiki
2 revs
Ravindra babu

2

Producing Thread-safe code is all about managing access to shared mutable states. When mutable states are published or shared between threads, they need to be synchronized to avoid bugs like race conditions and memory consistency errors.

I recently wrote a blog about thread safety. You can read it for more information.

Share Improve this answer Follow

answered Aug 16, 2021 at 13:47

community wiki
Hamza Belmellouki

You are clearly working in a WinForms environment. WinForms controls exhibit thread affinity, which means

1

that the thread in which they are created is the only thread that can be used to access and update them. That is why you will find examples on MSDN and elsewhere demonstrating how to marshall the call back onto the main thread.

Normal WinForms practice is to have a single thread that is dedicated to all your UI work.

Share Improve this answer Follow

answered Jan 9, 2010 at 15:48

David M
72.8k ● 13 ● 163 ● 187

1

I find the concept of http://en.wikipedia.org/wiki/Reentrancy_%28computing%29 to be what I usually think of as unsafe threading which is when a method has and relies on a side effect such as a global variable.

For example I have seen code that formatted floating point numbers to string, if two of these are run in different threads the global value of decimalSeparator can be permanently changed to '.'

```
//built in global set to locale specific value
(here a comma)
decimalSeparator = ','

function FormatDot(value : real):
    //save the current decimal character
    temp = decimalSeparator

    //set the global value to be
    decimalSeparator = '.'

    //format() uses decimalSeparator behind the
scenes
    result = format(value)

    //Put the original value back
    decimalSeparator = temp
```

Share Improve this answer Follow

answered Nov 1, 2011 at 19:07

community wiki
Aaron Robson

-4

To understand thread safety, read below sections:

> **4.3.1. Example: Vehicle Tracker Using Delegation**
>
> As a more substantial example of delegation, let's construct a version of the vehicle tracker that delegates to a thread-safe class. We store the locations in a Map, so we start with a thread-safe Map implementation, `ConcurrentHashMap`. We also store the location using an immutable Point class instead of `MutablePoint`, shown in Listing 4.6.
>
> **Listing 4.6. Immutable Point class used by DelegatingVehicleTracker.**

```
 class Point{
  public final int x, y;

  public Point() {
        this.x=0; this.y=0;
    }

  public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

}
```

`Point` is thread-safe because it is immutable. Immutable values can be freely shared and published, so we no longer need to copy the locations when returning them.

`DelegatingVehicleTracker` in Listing 4.7 does not use any explicit synchronization; all access to state is managed by `ConcurrentHashMap`, and all the keys and values of the Map are immutable.

Listing 4.7. Delegating Thread Safety to a ConcurrentHashMap.

```
  public class DelegatingVehicleTracker {

  private final ConcurrentMap<String,
Point> locations;
    private final Map<String, Point>
unmodifiableMap;

  public
DelegatingVehicleTracker(Map<String,
Point> points) {
        this.locations = new
ConcurrentHashMap<String, Point>(points);
        this.unmodifiableMap =
Collections.unmodifiableMap(locations);
    }

  public Map<String, Point> getLocations()
{
        return this.unmodifiableMap; //
User cannot update point(x,y) as Point is
immutable
    }

  public Point getLocation(String id) {
        return locations.get(id);
    }

  public void setLocation(String id, int
x, int y) {
        if(locations.replace(id, new
Point(x, y)) == null) {
             throw new
IllegalArgumentException("invalid vehicle
name: " + id);
        }
    }
```

}

If we had used the original `MutablePoint` class instead of Point, we would be breaking encapsulation by letting `getLocations` publish a reference to mutable state that is not thread-safe. Notice that we've changed the behavior of the vehicle tracker class slightly; while the monitor version returned a snapshot of the locations, the delegating version returns an unmodifiable but “live” view of the vehicle locations. This means that if thread A calls `getLocations` and thread B later modifies the location of some of the points, those changes are reflected in the Map returned to thread A.

4.3.2. Independent State Variables

We can also delegate thread safety to more than one underlying state variable as long as those underlying state variables are independent, meaning that the composite class does not impose any invariants involving the multiple state variables.

`VisualComponent` in Listing 4.9 is a graphical component that allows clients to register listeners for mouse and keystroke events. It maintains a list of registered listeners of each type, so that when an event occurs the appropriate listeners can be invoked. But there is no relationship between the set of mouse listeners and key listeners; the two are independent, and therefore `VisualComponent` can delegate its thread safety obligations to two underlying thread-safe lists.

Listing 4.9. Delegating Thread Safety to Multiple Underlying State Variables.

```
public class VisualComponent {
    private final List<KeyListener>
keyListeners
                                        =
new CopyOnWriteArrayList<KeyListener>();
```

```
    private final List<MouseListener>
mouseListeners
                                        =
new CopyOnWriteArrayList<MouseListener>();

  public void addKeyListener(KeyListener
listener) {
        keyListeners.add(listener);
    }

  public void
addMouseListener(MouseListener listener) {
        mouseListeners.add(listener);
    }

  public void
removeKeyListener(KeyListener listener) {
        keyListeners.remove(listener);
    }

  public void
removeMouseListener(MouseListener
listener) {
        mouseListeners.remove(listener);
    }

}
```

`VisualComponent` uses a `CopyOnWriteArrayList` to store each listener list; this is a thread-safe List implementation particularly suited for managing listener lists (see Section 5.2.3). Each List is thread-safe, and because there are no constraints coupling the state of one to the state of the other, `VisualComponent` can delegate its thread safety responsibilities to the underlying `mouseListeners` and `keyListeners` objects.

4.3.3. When Delegation Fails

Most composite classes are not as simple as `VisualComponent`: they have invariants that relate their component state variables. `NumberRange` in Listing 4.10 uses two `AtomicIntegers` to manage its state, but imposes an additional constraint—that the first number be less than or equal to the second.

Listing 4.10. Number Range Class that does Not Sufficiently Protect Its Invariants. Don't do this.

```
public class NumberRange {

  // INVARIANT: lower <= upper
    private final AtomicInteger lower =
new AtomicInteger(0);
    private final AtomicInteger upper =
new AtomicInteger(0);

  public void setLower(int i) {
        //Warning - unsafe check-then-act
        if(i > upper.get()) {
            throw new
IllegalArgumentException(
                    "Can't set lower to "
+ i + " > upper ");
        }
        lower.set(i);
    }

  public void setUpper(int i) {
        //Warning - unsafe check-then-act
        if(i < lower.get()) {
            throw new
IllegalArgumentException(
                    "Can't set upper to "
+ i + " < lower ");
        }
```

```
            upper.set(i);
        }

      public boolean isInRange(int i){
            return (i >= lower.get() && i <=
upper.get());
        }

}
```

`NumberRange` is **not thread-safe**; it does not preserve the invariant that constrains lower and upper. The `setLower` and `setUpper` methods attempt to respect this invariant, but do so poorly. Both `setLower` and `setUpper` are check-then-act sequences, but they do not use sufficient locking to make them atomic. If the number range holds (0, 10), and one thread calls `setLower(5)` while another thread calls `setUpper(4)`, with some unlucky timing both will pass the checks in the setters and both modifications will be applied. The result is that the range now holds (5, 4)—**an invalid state**. So **while the underlying AtomicIntegers are thread-safe, the composite class is not**. Because the underlying state variables `lower` and `upper` are not independent, `NumberRange` cannot simply delegate thread safety to its thread-safe state variables.

`NumberRange` could be made thread-safe by using locking to maintain its invariants, such as guarding lower and upper with a common lock. It

> must also avoid publishing lower and upper to prevent clients from subverting its invariants.
>
> If a class has compound actions, as `NumberRange` does, delegation alone is again not a suitable approach for thread safety. In these cases, the class must provide its own locking to ensure that compound actions are atomic, unless the entire compound action can also be delegated to the underlying state variables.
>
> **If a class is composed of multiple independent thread-safe state variables and has no operations that have any invalid state transitions, then it can delegate thread safety to the underlying state variables.**

edited Oct 11, 2017 at 17:50

community wiki
3 revs
overexchange