2023/2024(1)

EF234302 Object Oriented Programming

Lecture #11

Thread, Race & Deadlock-Livelock

Misbakhul Munir IRFAN SUBAKTI

司馬伊凡

Мисбакхул Мунир Ирфан Субакти

Thread: What is that?

- A thread is basically a piece of code that is concurrently executing with our main program
- We have already implicitly used threads in our assignments ago
- When we write an `ActionListener`, its `actionPerformed` method, when called as a result of the user interacting with the GUI, is executed in a separate thread to our main program (this thread is called the *Event Dispatch Thread* or EDT)
- Threads are massively *useful* and can simplify the design of a program. For instance, when writing our GUI for the `predictive text` assignment, there was no need to periodically check if the user had pressed a button or not. We wrote code that setup our program and code that knew what to do when a button was clicked and let another thread deal with the complexities of mouse I/O.
- As an analogy, we could think of threads as multitasking for our computer. The JVM allocates a small amount of time to every thread that's running in some order, switching between time allocation for each thread in turn. This gives the appearance that the computer is performing several tasks '*at once*', when in fact, just one is running at any time.

Thread: More about

- Java is a multi-threaded *programming language* which means we can develop multi-threaded program using Java.
- Threads allows a program to operate more efficiently by doing multiple things at the same time.
- Threads can be used to perform complicated tasks in the background *without interrupting the main program*.
- Concept of threads arise due to *multiprocessor architecture* in *distributed systems*.
  - Internet, mainframe servers, quantum computers are examples of distributed computing systems where *inter-process communication arise* between *multiple processing units* in which threads are *used*

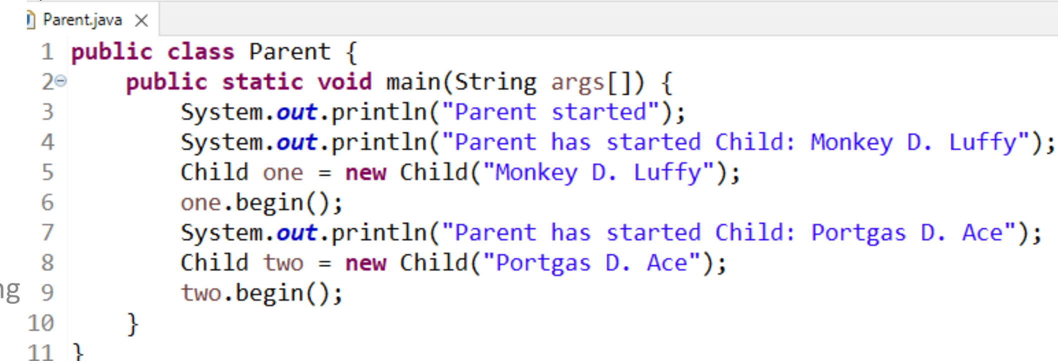
Thread: Why?

- When Swing, servlets, Remote Method Invocation (RMI), or Enterprise JavaBeans (EJB) – Jakarta Enterprise Beans (JEB) technologies are used, we may already be using threads without realising it.
- So, why? Because by using threads in our Java programs:
  - Make the *User Interface* (UI) more *responsive*
  - Take advantage of *multiprocessor* systems
  - Simplify *modelling*
  - Perform *asynchronous* or *background* processing
- Parallelism in Java program → *multiple threads*, take full advantage of *multiple cores* by serving *more clients* and *serving* them *faster*

Parent-Child

- The example we will use takes the following analogy. We have a parent process, which directs a number of child processes to independently counting a number. It tells them to do this indefinitely, until it decides otherwise. We'll begin with a *non-threaded* version of the processes, which will make the problem apparent. The `Parent` class:

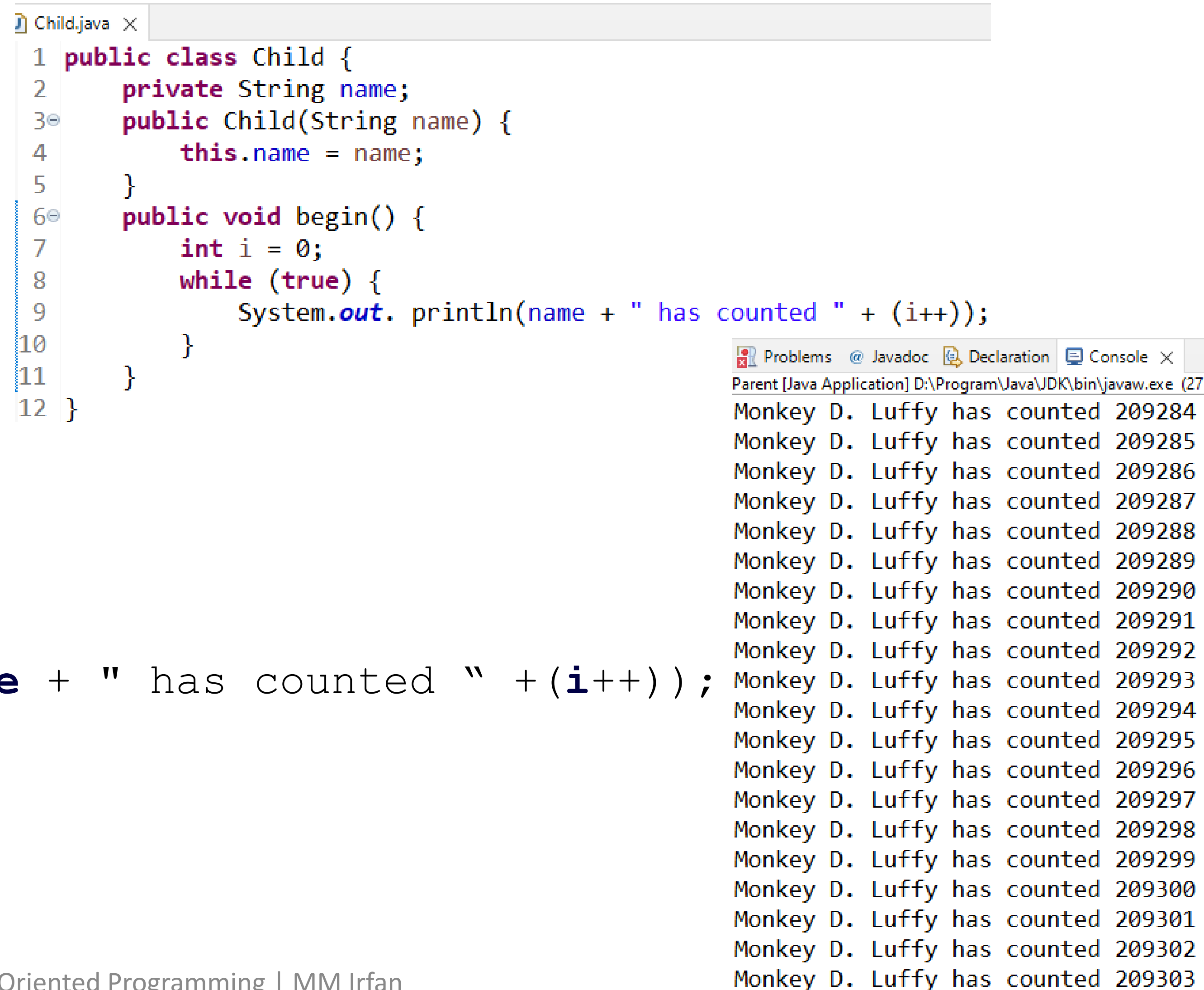
```
public class Parent {
    public static void main(String args[]) {
        System.out.println("Parent started");
        System.out.println("Parent has started Child: Monkey D. Luffy");
        Child one = new Child("Monkey D. Luffy");
        one.begin();
        System.out.println("Parent has started Child: Portgas D. Ace");
        Child two = new Child("Portgas D. Ace");
        two.begin();
    }
}
```



Parent-Child (continued)

- The `Child` class can be seen below.

```
public class Child {
    private String name;
    public Child(String name) {
        this.name = name;
    }
    public void begin() {
        int i = 0;
        while (true) {
            System.out.println(name + " has counted " +(i++));
        }
    }
}
```



Parent-Child (continued)

- When we run the `Parent` program, what will happen?
- Well, we'll start the first child one, and tell it to call the `begin` method/function.
- Unfortunately, full control is now passed over to `Child`'s one, which enters an infinite loop.
- We never get to start `Child`'s two, because the first child, i.e., one, never stops.
- Furthermore, the `Parent` has no way to stop the first child, i.e., one!
- If we want both children to run indefinitely, we need some way to transfer control back to the parent.
- Threads are the solution.
- As we said earlier, more than one instance of a thread can run at once.
- If we make a class that implements `Runnable`, we can harness this power.
- The `Runnable` interface has a method `run()`, which is actually the part which can be run in parallel with other threads.
- However, it is never called explicitly by the programmer. Instead, we pass our runnable object to the `Thread` class and call `start()` — more on this later.
- First we need to override the `run()` method:

New Parent-Child

```java
public class NewChild implements Runnable {
    private String name;
    public NewChild(String name) {
        this.name = name;
    }
    @Override
    public void run() {
        Random r = new Random();
        try {
            int i = 0;
            while (true) {
                int rand = r.nextInt(2000);
                System.out.println(name + " has counted to " + (i++)
                    + " and will now sleep for " + rand + "ms");
                Thread.sleep(rand);
            }
        } catch (InterruptedException e) {
            System.out.println(name + " interrupted; ending");
        }
    }
}
```

NewChild.java

```java
1  import java.util.Random;
2  public class NewChild implements Runnable {
3      private String name;
4      public NewChild(String name) {
5          this.name = name;
6      }
7      @Override
8      public void run() {
9          Random r = new Random();
10         try {
11             int i = 0;
12             while (true) {
13                 int rand = r.nextInt(2000);
14                 System.out.println(name + " has counted to " + (i++)
15                     + " and will now sleep for " + rand + "ms");
16                 Thread.sleep(rand);
17             }
18         } catch (InterruptedException e) {
19             System.out.println(name + " interrupted; ending");
20         }
21     }
22 }
```

- Note that in this version, we increment `i` then wait for a random amount of time between `0` and `2` seconds. Now the `NewParent` can have some control over the `Child` threads:

New Parent-Child (continued)

```java
public class NewParent {
        public static void main (String args[]) {
                System.out.println("Parent started" ) ;
                System.out.println("Parent is starting Child: Monkey D. Luffy");
                NewChild one = new NewChild("Monkey D. Luffy");
                Thread threadOne = new Thread(one);
                threadOne.start();
                System.out.println("Parent is starting Child: Portgas D. Ace");
                NewChild two = new NewChild("Portgas D. Ace");
                Thread threadTwo = new Thread(two);
                threadTwo.start();
                System.out.println("Parent will sleep for 10 seconds");
                try {
                        Thread.sleep(10000);
                        System.out.println("Parent has woken up");
                }
                catch (InterruptedException e) {
                        System.out.println("Somebody has awaken the Parent" + e);
                        // This actually won't happen.
                }
                finally {
                        // Interrupt children
                        threadOne.interrupt();
                        threadTwo.interrupt();
                }
                System.out.println("Parent ended");
        }
}
```


NewParent.java

```java
1  public class NewParent {
2      public static void main (String args[]) {
3          System.out.println("Parent started" ) ;
4          System.out.println("Parent is starting Child: Monkey D. Luffy");
5          NewChild one = new NewChild("Monkey D. Luffy");
6          Thread threadOne = new Thread(one);
7          threadOne.start();
8          System.out.println("Parent is starting Child: Portgas D. Ace");
9          NewChild two = new NewChild("Portgas D. Ace");
10         Thread threadTwo = new Thread(two);
11         threadTwo.start();
12         System.out.println("Parent will sleep for 10 seconds");
13         try {
14             Thread.sleep(10000);
15             System.out.println("Parent has woken up");
16         }
17         catch (InterruptedException e) {
18             System.out.println("Somebody has awaken the Parent" + e);
19             // This actually won't happen.
20         }
21         finally {
22             // Interrupt children
23             threadOne.interrupt();
24             threadTwo.interrupt();
25         }
26         System.out.println("Parent ended");
27     }
28 }
```

New Parent-Child: Output

Problems @ Javadoc Declaration Console

<terminated> NewParent [Java Application] D:\Program\Java\JDK\bin\javaw.exe (27 Nov 2021, 20:35:05 – 20:35:16)

```
Parent started
Parent is starting Child: Monkey D. Luffy
Parent is starting Child: Portgas D. Ace
Parent will sleep for 10 seconds
Portgas D. Ace has counted to 0 and will now sleep for 479ms
Monkey D. Luffy has counted to 0 and will now sleep for 545ms
Portgas D. Ace has counted to 1 and will now sleep for 379ms
Monkey D. Luffy has counted to 1 and will now sleep for 817ms
Portgas D. Ace has counted to 2 and will now sleep for 610ms
Monkey D. Luffy has counted to 2 and will now sleep for 235ms
Portgas D. Ace has counted to 3 and will now sleep for 379ms
Monkey D. Luffy has counted to 3 and will now sleep for 1292ms
Portgas D. Ace has counted to 4 and will now sleep for 446ms
Portgas D. Ace has counted to 5 and will now sleep for 1355ms
Monkey D. Luffy has counted to 4 and will now sleep for 208ms
Monkey D. Luffy has counted to 5 and will now sleep for 174ms
Monkey D. Luffy has counted to 6 and will now sleep for 1010ms
Portgas D. Ace has counted to 6 and will now sleep for 760ms
Monkey D. Luffy has counted to 7 and will now sleep for 1377ms
Portgas D. Ace has counted to 7 and will now sleep for 419ms
```

```
Portgas D. Ace has counted to 8 and will now sleep for 1647ms
Monkey D. Luffy has counted to 8 and will now sleep for 605ms
Monkey D. Luffy has counted to 9 and will now sleep for 612ms
Portgas D. Ace has counted to 9 and will now sleep for 592ms
Monkey D. Luffy has counted to 10 and will now sleep for 1840ms
Portgas D. Ace has counted to 10 and will now sleep for 1867ms
Monkey D. Luffy has counted to 11 and will now sleep for 768ms
Portgas D. Ace has counted to 11 and will now sleep for 968ms
Monkey D. Luffy has counted to 12 and will now sleep for 1483ms
Portgas D. Ace has counted to 12 and will now sleep for 442ms
Parent has woken up
Parent ended
Monkey D. Luffy interrupted; ending
Portgas D. Ace interrupted; ending
```

New Parent-Child: Explanation

- Let's look at what's happening.
- Each time the parent tells a thread object to `start`, it calls the `run` method of its `Runnable`, and, importantly, *control is passed back to* the `Parent`.
- That means that we can start *both* children, which will independently start counting and sleeping for random amounts of time.
- The `Parent` wants to give the children ten seconds to count, and then "stop" them, i.e., `interrupt` them.
- To program this, we use the command `Thread.sleep(long millis)`, which puts any thread "*to sleep*" for `millis` milliseconds.
- Note that although we haven't said that `Parent` is a thread, implicitly it is—it just happens that there are no other `Parent` threads running.

New Parent-Child: Explanation (continued)

- So, the parent waits for ten seconds.
- Now, we want it to *interrupt* the execution of the children.
- To do this, it calls the `interrupt` method on each child, which (if you look back at the `NewChild` class) causes the child to exit the `while` loop and continue with whatever was below the `try` block (as we catch an `InterruptedException`).
- Note that it is completely *the child's choice* what to do when interrupted. We could have written code which ignores this exception and loops again.
- The parent can only request that the child stops.
- In this case, when the children are interrupted, and an `InterruptedException` is thrown, the child immediately stops.
- Note that we could also ask the child to **stop** when finished, i.e., see the *next page*.
- When we run the code, both children count independently, and the parent correctly stops them.
- However, this perfect execution may not always happen when we are dealing with a shared resource.

New Parent-Child v02: Stop the children

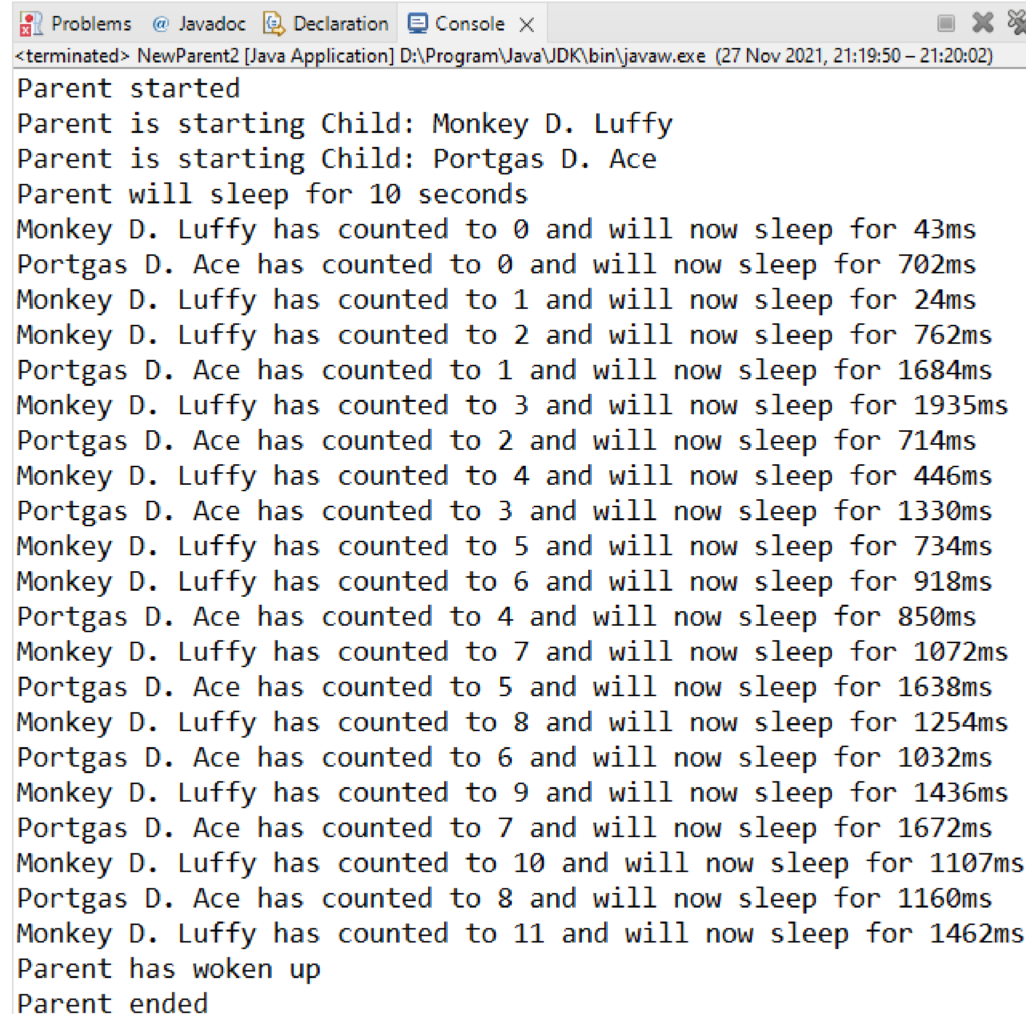
NewChild2.java

```java
import java.util.Random;
public class NewChild2 implements Runnable {
    private String name;
    private boolean stopStatus= false;
    public NewChild2(String name) {
        this.name = name;
    }
    @Override
    public void run() {
        Random r = new Random();
        try {
            int i = 0;
            while (!stopStatus) {
                int rand = r.nextInt(2000);
                System.out.println(name + " has counted to " + (i++)
                    + " and will now sleep for " + rand + "ms");
                Thread.sleep(rand);
            }
        } catch (InterruptedException e) {
            System.out.println(name + " interrupted; ending");
        }
    }
    public void stop() {
        stopStatus = true;
    }
}
```

NewParent2.java

```java
public class NewParent2 {
    public static void main (String args[]) {
        System.out.println("Parent started" ) ;
        System.out.println("Parent is starting Child: Monkey D. Luffy");
        NewChild2 one = new NewChild2("Monkey D. Luffy");
        Thread threadOne = new Thread(one);
        threadOne.start();
        System.out.println("Parent is starting Child: Portgas D. Ace");
        NewChild2 two = new NewChild2("Portgas D. Ace");
        Thread threadTwo = new Thread(two);
        threadTwo.start();
        System.out.println("Parent will sleep for 10 seconds");
        try {
            Thread.sleep(10000);
            System.out.println("Parent has woken up");
        }
        catch (InterruptedException e) {
            System.out.println("Somebody has awaken the Parent" + e);
            // This actually won't happen.
        }
        finally {
            // Ask the children to stop themselves
            one.stop();
            two.stop();
        }
        System.out.println("Parent ended");
    }
}
```

New Parent-Child v02: Output



Race

- Consider the following. We have a number of **thread children** which all increment and decrement a common *counter*:

```
public class Counter {
    private int c = 0;
    public void increment() {
        c++;
    }
    public void decrement() {
        c--;
    }
    public int get() {
        return c;
    }
}
```

Race (continued)

- Seems simple!
- Now, if we change each child to increment and *immediately* decrement the counter by one, a *million times*.
- Because we know that the children will terminate eventually, we don't need to interrupt them, the parent can instead simply wait for the thread to finish, by calling its `join()` method
- What if we didn't know if the thread would terminate?
- The parent will then just get the value of the counter.
- When run, there's a problem!
- We should at the end get something like:

```
counter should be 0 and is 0
```

- But, in fact we get something like:

```
counter should be 0 and actually is 645
```

- What's going on? Well, this is known as a race condition.

Race (continued)

- Let’s say we have two threads. The ‘*correct*’ flow of execution should be:

1. Child 1 increments counter
2. Child 1 decrements counter
3. Child 2 increments counter
4. . . .

- What happens is a little *confusing*. In order to increment the counter, the `c++` operation first gets the value of `c`, and then increases that value and *stores it back* in `c`. If we slow down what might **go wrong**, it should become clear:

1. Child 1 gets value of counter as 0
2. Child 2 gets value of counter as 0
3. Child 2 sets value of counter to be 1 (it should be 1)
4. Child 1 also sets value of counter to be 1 (it should be 2)
5. Child 2 gets value of counter as 1
6. Child 2 sets value of counter to be 0 (it should be 1)
7. Child 1 gets value of counter as 0
8. Child 1 sets value of counter as -1 (it should be 0)

Parent-Child: Race

Counter.java

```java
package counter;
public class Counter {
    private int c = 0;
    public void increment() {
        c++;
    }
    public void decrement() {
        c--;
    }
    public int get() {
        return c;
    }
}
```

Child.java

```java
package counter;
public class Child implements Runnable {
    private String name;
    private Counter counter;
    public Child(String name, Counter counter) {
        this.name = name;
        this.counter = counter;
    }
    @Override
    public void run() {
        for (int i = 0; i < 400; i++) {
            System.out.println(name + " increments counter");
            counter.increment();
            System.out.println(name + " decrements counter");
            counter.decrement();
            System.out.println(name + " gets value of counter as "
                    + counter.get());
        }
    }
}
```

Parent.java

```java
package counter;
public class Parent {
    public static void main (String args[])
            throws InterruptedException {
        System.out.println("Parent started" ) ;
        System.out.println("Parent is starting Child: 01");
        Counter counter = new Counter();
        Child one = new Child("01", counter);
        Thread threadOne = new Thread(one);
        threadOne.start();
        System.out.println("Parent is starting Child: 02");
        Child two = new Child("02", counter);
        Thread threadTwo = new Thread(two);
        threadTwo.start();
        System.out.println("Parent is starting Child: 03");
        Child three = new Child("03", counter);
        Thread threadThree = new Thread(three);
        threadThree.start();
        // Wait for threads to finish
        threadOne.join();
        threadTwo.join();
        threadThree.join();
        System.out.println("Parent ended");
    }
}
```

Parent-Child: Race (Output)

Problems @ Javadoc Declaration Consc
<terminated> Parent (1) [Java Application] D:\Program\

```
Parent started
Parent is starting Child: 01
Parent is starting Child: 02
Parent is starting Child: 03
01 increments counter
02 increments counter
02 decrements counter
01 decrements counter
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
02 gets value of counter as 1
02 increments counter
03 gets value of counter as 0
01 gets value of counter as 0
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
01 decrements counter
```

```
03 increments counter
03 decrements counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
03 gets value of counter as 1
03 increments counter
01 gets value of counter as 1
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
03 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 1
02 increments counter
02 decrements counter
02 gets value of counter as 1
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
```

```
01 decrements counter
01 gets value of counter as 1
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
01 increments counter
01 decrements counter
```

```
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
```

Parent-Child: Race (Output)

```
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
03 increments counter
03 decrements counter
03 gets value of counter as 0
01 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
```

```
<terminated> Parent (1) [Java Application] D:\Program\.
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
01 decrements counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
03 increments counter
01 gets value of counter as 1
```

```
01 increments counter
01 decrements counter
03 decrements counter
02 decrements counter
02 gets value of counter as 0
03 gets value of counter as 1
03 increments counter
03 decrements counter
01 gets value of counter as 2
01 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
01 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
02 increments counter
02 decrements counter
```

```
03 increments counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 1
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
03 decrements counter
01 decrements counter
01 gets value of counter as 0
03 gets value of counter as 1
03 increments counter
03 decrements counter
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
```

Parent-Child: Race (Output)

```
02 increments counter
02 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
03 decrements counter
02 gets value of counter as 0
02 increments counter
03 gets value of counter as 1
03 increments counter
01 decrements counter
01 gets value of counter as 2
01 increments counter
03 decrements counter
03 gets value of counter as 2
03 increments counter
03 decrements counter
```

```
02 decrements counter
02 gets value of counter as 1
02 increments counter
02 decrements counter
02 gets value of counter as 1
02 increments counter
02 decrements counter
02 gets value of counter as 1
02 increments counter
02 decrements counter
02 gets value of counter as 1
03 gets value of counter as 2
03 increments counter
01 decrements counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
```

```
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
```

```
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
01 gets value of counter as 1
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
03 increments counter
03 decrements counter
```

Parent-Child: Race (Output)

```
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
03 decrements counter
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
01 decrements counter
01 gets value of counter as 1
01 increments counter
01 decrements counter
01 gets value of counter as 1
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
```

```
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
03 decrements counter
03 gets value of counter as 1
03 increments counter
03 decrements counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
03 gets value of counter as 1
03 increments counter
03 decrements counter
01 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
01 decrements counter
03 gets value of counter as 0
03 increments counter
02 increments counter
02 decrements counter
```

```
02 gets value of counter as 1
02 increments counter
02 decrements counter
03 decrements counter
01 gets value of counter as 0
01 increments counter
01 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
02 gets value of counter as 1
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
```

...

```
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
03 decrements counter
03 gets value of counter as 0
Parent ended
```

Race: Solution

- This is just for three children. Think about the problem extended to *several children running a million times*.
- The problem is that no one child has **exclusive access** to the counter at *any one time*, and the *operation* to *increment* or *decrement* is **not atomic** (that is to say, it *doesn't happen in one go*).
- Luckily, there are a couple of *solutions* to this.
- The simplest is to change the counter, so that its increment and decrement methods are `synchronized`. This means that each thread accessing the counter must *synchronise* with other thread doing so, in a '*one at a time*' way:

```
public synchronized void increment() {
    i++;
}
```

- We could also ask the thread itself to `synchronise` what it does. The terminology is slightly different:

```
synchronized(counter) {
    counter.increment();
}
```

Race: Solution (continued)

- The later will need to rely on all threads correctly synchronizing on the counter before accessing it.
- The `counter` next to the `synchronized` statement means that threads must *synchronise on the* `counter` *object*.
- Adding either of these statements makes the necessary operations **atomic** and accessible by only one thread at a time, and solves the problem.
- In short, any object which will be accessed by *two or more threads* and *whose fields will be changed by one or methods* **should only be accessed in synchronized blocks or methods**.

Parent-Child: Synchronized

Counter.java

```java
package counter;
public class Counter {
    private int c = 0;
    public void increment() {
        c++;
    }
    public void decrement() {
        c--;
    }
    public int get() {
        return c;
    }
}
```

Child.java

```java
package counter;
public class Child implements Runnable {
    private String name;
    private Counter counter;
    public Child(String name, Counter counter) {
        this.name = name;
        this.counter = counter;
    }
    @Override
    public void run() {
        for (int i = 0; i < 400; i++) {
            System.out.println(name + " increments counter");
            synchronized (counter) {
                counter.increment();
                System.out.println(name + " decrements counter");
                counter.decrement();
                System.out.println(name + " gets value of counter as "
                        + counter.get());
            }
        }
    }
}
```

Parent.java

```java
package counter;
public class Parent {
    public static void main (String args[])
            throws InterruptedException {
        System.out.println("Parent started" ) ;
        System.out.println("Parent is starting Child: 01");
        Counter counter = new Counter();
        Child one = new Child("01", counter);
        Thread threadOne = new Thread(one);
        threadOne.start();
        System.out.println("Parent is starting Child: 02");
        Child two = new Child("02", counter);
        Thread threadTwo = new Thread(two);
        threadTwo.start();
        System.out.println("Parent is starting Child: 03");
        Child three = new Child("03", counter);
        Thread threadThree = new Thread(three);
        threadThree.start();
        // Wait for threads to finish
        threadOne.join();
        threadTwo.join();
        threadThree.join();
        System.out.println("Parent ended");
    }
}
```

Parent-Child: Synchronized (Output)

```
Problems  @ Javadoc  Declaration  Conso
<terminated> Parent (1) [Java Application] D:\Program\J
Parent started
Parent is starting Child: 01
Parent is starting Child: 02
Parent is starting Child: 03
01 increments counter
02 increments counter
01 decrements counter
03 increments counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
```

```
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
```

```
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
```

```
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
```

Parent-Child: Synchronized (Output)

```
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
```

```
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
01 decrements counter
03 increments counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
```

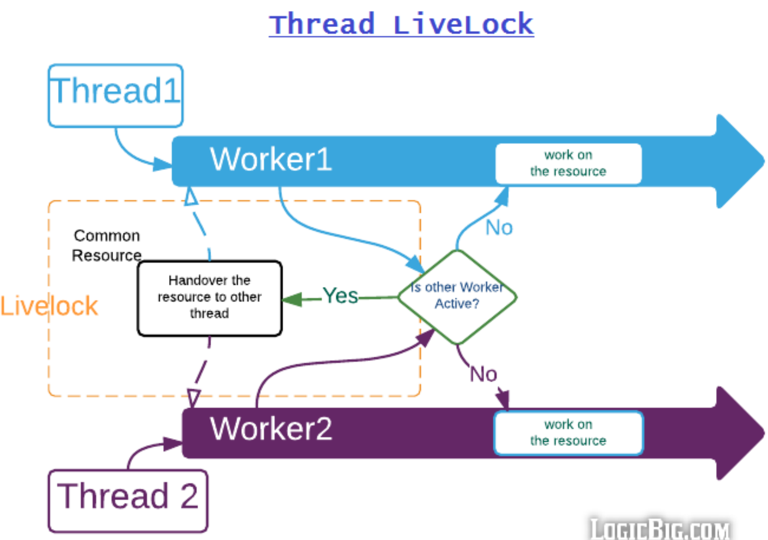
```
02 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
03 decrements counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
01 decrements counter
01 gets value of counter as 0
03 decrements counter
01 increments counter
03 gets value of counter as 0
03 increments counter
01 decrements counter
01 gets value of counter as 0
01 increments counter
02 decrements counter
```

...

```
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
02 increments counter
02 decrements counter
02 gets value of counter as 0
Parent ended
```

Deadlock & livelock

- We will talk about a different problem to do with synchronisation between multiple threads accessing a number of resources:
  - Deadlock. E.g., the dining philosopher problem
  - Livelock. E.g., the situation of side-stepping to avoid walking into someone, but they also side-step. Other example: sharing resource problem (see later).



Deadlock: Dining philosopher problem

- Five silent philosophers sit at a round table with bowls of *spaghetti*. Forks are *placed between* each pair of adjacent philosophers.
- Each philosopher must alternately think and eat. However, a philosopher *can only* eat spaghetti when they have *both* left and right forks. *Each fork* can be held by only *one philosopher at a time* and so a philosopher can *use the fork* only *if it is* not being used by another philosopher. After an individual philosopher *finishes eating*, they need to put down both forks so that the forks become available to *others*. A philosopher can *only* take the fork on their *right* or the one on their *left* as they become *available* and they *cannot start eating before getting both forks*.
- Eating is *not limited* by the remaining *amounts of spaghetti* or *stomach space*; an *infinite supply* and an *infinite demand* are assumed.
- The problem is how to design a discipline of behaviour (a *concurrent* algorithm) such that no philosopher will **starve**; i.e., each can *forever continue* to alternate between *eating and thinking*, assuming that *no philosopher* can *know* when *others* may *want to eat or think*.

Dining philosopher: Code (HerongYang.com)

Philosopher.java

```java
1  import java.util.*;
2  /*
3   * Based on HerongYang.com's work
4   */
5  public class Philosopher extends Thread {
6      public static final int numberOfThreads = 5;
7      public static Object[] listOfLocks = new Object[numberOfThreads];
8      public static String[] dinerTable = new String[4 * numberOfThreads];
9      public static String[] lockedDiner = new String[4 * numberOfThreads];
10     public static Random randomGenerator = new Random();
11     public static int unitOfTime = 500;
12     private int threadIndex;
13     private static String array2String(Object arr[], String delimiter) {
14         StringBuilder sb = new StringBuilder();
15         for (Object obj : arr)
16             sb.append(obj.toString()).append(delimiter);
17         return sb.substring(0, sb.length() - 1);
18     }
19     private static String getInfo(String s[]) {
20         String result = "";
21         for (int i = 0; i < numberOfThreads; i++) {
22             if (s[4 * i].equals("^")) {
23                 result += "Fork:Idle  ";
24             } else {
25                 result += "Fork:Taken ";
26             }
27             if (s[4 * i + 2].equals("_O_")) {
28                 result += "[P" + (i + 1) + ":No fork] ";
29             } else if (s[4 * i + 2].equals("<O>")) {
30                 result += "[P" + (i + 1) + ":2 forks] ";
31             } else {
32                 result += "[P" + (i + 1) + ": 1 fork] ";
33             }
34         }
35         return result;
36     }
37     public Philosopher(int i) {
38         threadIndex = i;
39     }
40     public void run() {
41         while (!isInterrupted()) {
42             try {
43                 sleep(unitOfTime * randomGenerator.nextInt(6));
44             } catch (InterruptedException e) {
45                 break;
46             }
47             // Try to get the fork on the left
48             Object leftLock = listOfLocks[threadIndex];
49             synchronized (leftLock) {
50                 int i = 4 * threadIndex;
51                 dinerTable[i] = " ";
52                 dinerTable[i + 1] = "^";
53                 dinerTable[i + 2] = "<O ";
54                 try {
55                     sleep(unitOfTime * 1);
56                 } catch (InterruptedException e) {
57                     break;
58                 }
59                 // Try to get the fork on the right
60                 Object rightLock = listOfLocks[(threadIndex + 1) % numberOfThreads];
61                 synchronized (rightLock) {
62                     dinerTable[i + 2] = "<O>";
63                     dinerTable[i + 3] = "^";
64                     dinerTable[(i + 4) % (4 * numberOfThreads)] = " ";
65                     try {
66                         sleep(unitOfTime * 1);
67                     } catch (InterruptedException e) {
68                         break;
69                     }
70                     dinerTable[i] = "^";
71                     dinerTable[i + 1] = " ";
72                     dinerTable[i + 2] = "_O_";
```

Dining philosopher: Code (HerongYang.com)

```
73                 dinerTable[i + 3] = " ";
74                 dinerTable[(i + 4) % (4 * numberOfThreads)] = "^";
75             }
76         }
77     }
78 }
79 public static void main(String[] a) {
80     for (int i = 0; i < numberOfThreads; i++)
81         listOfLocks[i] = new Object();
82     for (int i = 0; i < numberOfThreads; i++) {
83         dinerTable[4 * i] = "^";
84         dinerTable[4 * i + 1] = " ";
85         dinerTable[4 * i + 2] = "_O_";
86         dinerTable[4 * i + 3] = " ";
87         lockedDiner[4 * i] = " ";
88         lockedDiner[4 * i + 1] = "^";
89         lockedDiner[4 * i + 2] = "<O ";
90         lockedDiner[4 * i + 3] = " ";
91     }
92     for (int i = 0; i < numberOfThreads; i++) {
93         Thread t = new Philosopher(i);
94         t.setDaemon(true);
95         t.start();
96     }
97     String lockedString = array2String(lockedDiner, "");
98     System.out.println("The diner table:");
99     long step = 0;
100        while (true) {
101            step++;
102            String sDinerTable = array2String(dinerTable, "");
103            System.out.println(sDinerTable + "    " + step + "\t" + getInfo(dinerTable));
104            if (lockedString.equals(sDinerTable)) {
105                break;
106            }
107            try {
108                Thread.sleep(unitOfTime);
109            } catch (InterruptedException e) {
110                System.out.println("Interrupted.");
111            }
112        }
113        System.out.println("The diner is locked.");
114    }
115 }
```

Dining philosopher: Output

Problems   @ Javadoc   Declaration   Console

<terminated> Philosopher [Java Application] D:\Program\Java\JDK\bin\javaw.exe (28 Nov 2021, 22:58:00 – 22:58:12)

```
The diner table:
^ _O_  ^<O  ^ _O_ ^ _O_  ^<O     1       Fork:Idle   [P1:No fork] Fork:Taken [P2: 1 fork] Fork:Idle   [P3:No fork] Fork:Idle   [P4:No fork] Fork:Taken [P5: 1 fork]
  _O_  ^<O>^  _O_ ^ _O_  ^<O>    2       Fork:Taken [P1:No fork] Fork:Taken [P2:2 forks] Fork:Taken [P3:No fork] Fork:Idle   [P4:No fork] Fork:Taken [P5:2 forks]
^ _O_ ^ _O_ ^ _O_ ^ _O_ ^ _O_    3       Fork:Idle   [P1:No fork] Fork:Idle   [P2:No fork] Fork:Idle   [P3:No fork] Fork:Idle   [P4:No fork] Fork:Idle   [P5:No fork]
^ _O_ ^ _O_  ^<O  ^ _O_  ^<O     4       Fork:Idle   [P1:No fork] Fork:Idle   [P2:No fork] Fork:Taken [P3: 1 fork] Fork:Idle   [P4:No fork] Fork:Taken [P5: 1 fork]
 ^<O  ^ _O_  ^<O   ^<O   ^<O     5       Fork:Taken [P1: 1 fork] Fork:Idle   [P2:No fork] Fork:Taken [P3: 1 fork] Fork:Taken [P4: 1 fork] Fork:Taken [P5: 1 fork]
 ^<O>^  _O_  ^<O   ^<O   ^<O     6       Fork:Taken [P1:2 forks] Fork:Taken [P2:No fork] Fork:Taken [P3: 1 fork] Fork:Taken [P4: 1 fork] Fork:Taken [P5: 1 fork]
  _O_ ^ _O_  ^<O   ^<O   ^<O>    7       Fork:Taken [P1:No fork] Fork:Idle   [P2:No fork] Fork:Taken [P3: 1 fork] Fork:Taken [P4: 1 fork] Fork:Taken [P5:2 forks]
^ _O_  ^<O   ^<O   ^<O>^  _O_    8       Fork:Idle   [P1:No fork] Fork:Taken [P2: 1 fork] Fork:Taken [P3: 1 fork] Fork:Taken [P4:2 forks] Fork:Taken [P5:No fork]
^ _O_  ^<O   ^<O>^  _O_ ^ _O_    9       Fork:Idle   [P1:No fork] Fork:Taken [P2: 1 fork] Fork:Taken [P3:2 forks] Fork:Taken [P4:No fork] Fork:Idle   [P5:No fork]
 ^<O   ^<O>^  _O_ ^ _O_ ^ _O_    10      Fork:Taken [P1: 1 fork] Fork:Taken [P2:2 forks] Fork:Taken [P3:No fork] Fork:Idle   [P4:No fork] Fork:Idle   [P5:No fork]
 ^<O>^  _O_ ^ _O_ ^ _O_ ^ _O_    11      Fork:Taken [P1:2 forks] Fork:Taken [P2:No fork] Fork:Idle   [P3:No fork] Fork:Idle   [P4:No fork] Fork:Idle   [P5:No fork]
^ _O_  ^<O  ^ _O_ ^ _O_ ^ _O_    12      Fork:Idle   [P1:No fork] Fork:Taken [P2: 1 fork] Fork:Idle   [P3:No fork] Fork:Idle   [P4:No fork] Fork:Idle   [P5:No fork]
^ _O_  ^<O>^  _O_  ^<O   ^<O     13      Fork:Idle   [P1:No fork] Fork:Taken [P2:2 forks] Fork:Taken [P3:No fork] Fork:Taken [P4: 1 fork] Fork:Taken [P5: 1 fork]
  _O_ ^ _O_  ^<O   ^<O   ^<O>    14      Fork:Taken [P1:No fork] Fork:Idle   [P2:No fork] Fork:Taken [P3: 1 fork] Fork:Taken [P4: 1 fork] Fork:Taken [P5:2 forks]
 ^<O  ^ _O_  ^<O   ^<O>^  _O_    15      Fork:Taken [P1: 1 fork] Fork:Idle   [P2:No fork] Fork:Taken [P3: 1 fork] Fork:Taken [P4:2 forks] Fork:Taken [P5:No fork]
 ^<O>^  _O_  ^<O>^  _O_ ^ _O_    16      Fork:Taken [P1:2 forks] Fork:Taken [P2:No fork] Fork:Taken [P3:2 forks] Fork:Taken [P4:No fork] Fork:Idle   [P5:No fork]
^ _O_  ^<O  ^ _O_ ^ _O_ ^ _O_    17      Fork:Idle   [P1:No fork] Fork:Taken [P2: 1 fork] Fork:Idle   [P3:No fork] Fork:Idle   [P4:No fork] Fork:Idle   [P5:No fork]
^ _O_  ^<O   ^<O  ^ _O_ ^ _O_    18      Fork:Idle   [P1:No fork] Fork:Taken [P2: 1 fork] Fork:Taken [P3: 1 fork] Fork:Idle   [P4:No fork] Fork:Idle   [P5:No fork]
^ _O_  ^<O   ^<O>^  _O_ ^ _O_    19      Fork:Idle   [P1:No fork] Fork:Taken [P2: 1 fork] Fork:Taken [P3:2 forks] Fork:Taken [P4:No fork] Fork:Idle   [P5:No fork]
 ^<O   ^<O>^  _O_ ^ _O_  ^<O     20      Fork:Taken [P1: 1 fork] Fork:Taken [P2:2 forks] Fork:Taken [P3:No fork] Fork:Idle   [P4:No fork] Fork:Taken [P5: 1 fork]
 ^<O   ^<O  ^ _O_  ^<O   ^<O     21      Fork:Taken [P1: 1 fork] Fork:Taken [P2: 1 fork] Fork:Idle   [P3:No fork] Fork:Taken [P4: 1 fork] Fork:Taken [P5: 1 fork]
 ^<O   ^<O   ^<O   ^<O   ^<O     22      Fork:Taken [P1: 1 fork] Fork:Taken [P2: 1 fork] Fork:Taken [P3: 1 fork] Fork:Taken [P4: 1 fork] Fork:Taken [P5: 1 fork]
The diner is locked.
```

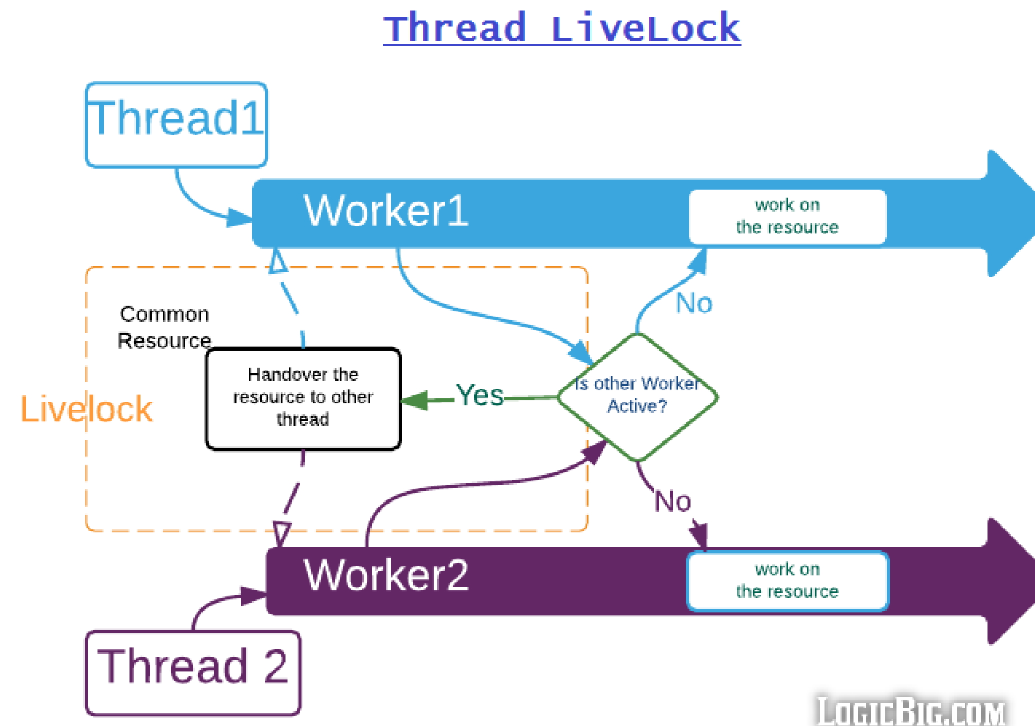
Dining philosopher: Solution (Wikipedia)

- Resource hierarchy solution
  - Dijkstra: assign a partial order to the resources, i.e., the forks
- Arbitrator solution
  - A philosopher can only pick up both forks or none by introducing an arbitrator
- Limiting the number of diners in the table
  - William Stallings: allow a maximum of *n*-1 philosopher to sit down at any time
- Chandy/Misra solution
  - Allow arbitrary agents to contend for an arbitrary number of resources
  - Completely distributed, requires no central authority after initialisation
  - Violates the requirement that philosophers don’t speak to each other

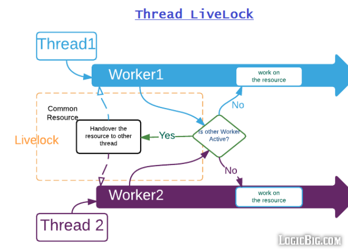
Livelock: Sharing resource problem (logicbig.com)

- Two threads want to access a shared common resource via a `Worker` object but when they see that other `Worker` (invoked on another thread) is also '*active*', they attempt to *hand over the resource to other worker* and *wait for it to finish*.
- If initially we make *both workers active* they will suffer from livelock.

Thread LiveLock



Sharing resource problem: Code (logicbig.com)



Worker.java

```java
1  package sharingres;
2  public class Worker {
3      private String name;
4      private boolean active;
5      public Worker(String name, boolean active) {
6          this.name = name;
7          this.active = active;
8      }
9      public String getName() {
10         return name;
11     }
12     public boolean isActive() {
13         return active;
14     }
```

```java
15     public synchronized void work(CommonResource commonResource, Worker otherWorker) {
16         while (active) {
17             // Wait for the resource to become available.
18             if (commonResource.getOwner() != this) {
19                 try {
20                     wait(10);
21                 } catch (InterruptedException e) {
22                     // ignore
23                 }
24                 continue;
25             }
26             // If other worker is also active let it do it's work first
27             if (otherWorker.isActive()) {
28                 System.out.println(getName() +
29                         " : handover the resource to the worker " + otherWorker.getName());
30                 commonResource.setOwner(otherWorker);
31                 continue;
32             }
33             // Now use the commonResource
34             System.out.println(getName() + ": working on the common resource");
35             active = false;
36             commonResource.setOwner(otherWorker);
37         }
38     }
39 }
```

Sharing resource problem: Code (logicbig.com)

Thread LiveLock
Thread1
Worker1
Common Resource
Livelock
Yes
No
Worker2
Thread 2

CommonResource.java

```java
package sharingres;
public class CommonResource {
    private Worker owner;
    public CommonResource(Worker d) {
        owner = d;
    }
    public Worker getOwner() {
        return owner;
    }
    public synchronized void setOwner(Worker d) {
        owner = d;
    }
}
```

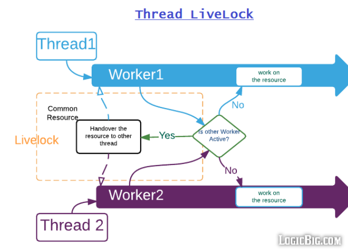
Livelock.java

```java
package sharingres;
public class Livelock {
    public static void main(String[] args) {
        final Worker worker1 = new Worker("Worker 1", true);
        final Worker worker2 = new Worker("Worker 2", true);
        final CommonResource s = new CommonResource(worker1);
        new Thread(() -> {
            worker1.work(s, worker2);
        }).start();
        new Thread(() -> {
            worker2.work(s, worker1);
        }).start();
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Livelock [Java Application] D:\Program\Java\JDK\bin\javaw.exe (29 Nov 2021, 08:57:36 – 08:57:50)
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
```

Sharing resource problem: Solution (logicbig.com)

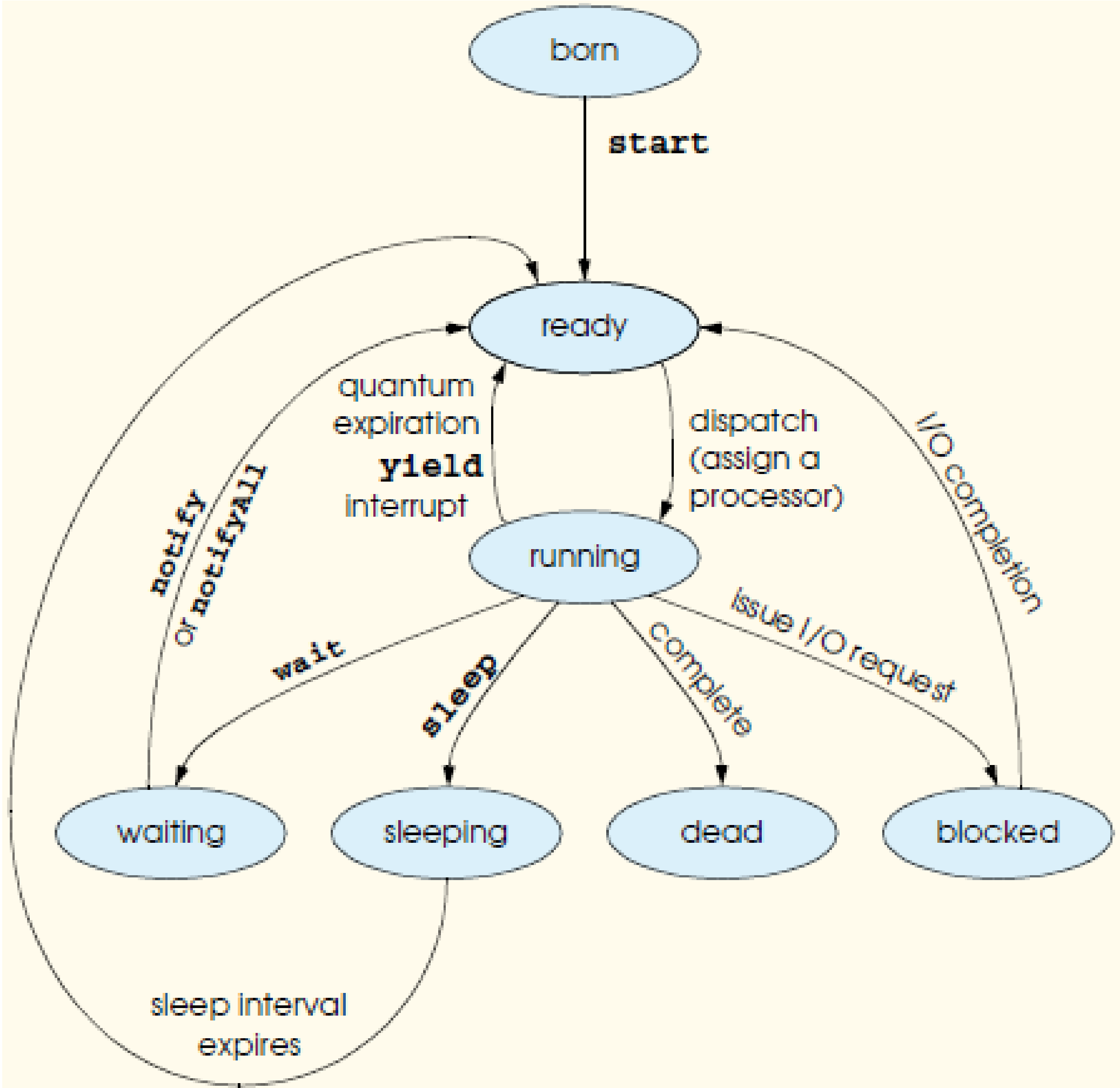


- We can fix the issue by processing the common resource sequentially rather than in different threads simultaneously.
- Just *like* deadlock, there's *no general guideline to avoid* livelock, but we have to be *careful* in scenarios where we *change the state of common objects also being used by other threads*, for example in above scenario, the `Worker` object.

Thread: Another example

- Reference
  - Deitel, H.M. and Deitel, P.J. (2002) Java: How to Program. Prentice Hall, 4th Edition, Upper Saddle River, NJ, USA.

Thread: Life cycle



Producer-consumer (`ProCon`): Output

Demonstrating Thread Synchronisation

```
Produced 1 into cell 0      write 1    read 0    buffer:  1 -1 -1 -1 -1
Consumed 1 from cell 0      write 1    read 1    buffer:  1 -1 -1 -1 -1
BUFFER EMPTY
Produced 2 into cell 1      write 2    read 1    buffer:  1  2 -1 -1 -1
Produced 3 into cell 2      write 3    read 1    buffer:  1  2  3 -1 -1
Produced 4 into cell 3      write 4    read 1    buffer:  1  2  3  4 -1
Produced 5 into cell 4      write 0    read 1    buffer:  1  2  3  4  5
Produced 6 into cell 0      write 1    read 1    buffer:  6  2  3  4  5
BUFFER FULL WAITING TO PRODUCE 7
Consumed 2 from cell 1      write 1    read 2    buffer:  6  2  3  4  5
Produced 7 into cell 1      write 2    read 2    buffer:  6  7  3  4  5
BUFFER FULL WAITING TO PRODUCE 8
Consumed 3 from cell 2      write 2    read 3    buffer:  6  7  3  4  5
Produced 8 into cell 2      write 3    read 3    buffer:  6  7  8  4  5
BUFFER FULL WAITING TO PRODUCE 9
Consumed 4 from cell 3      write 3    read 4    buffer:  6  7  8  4  5
Produced 9 into cell 3      write 4    read 4    buffer:  6  7  8  9  5
BUFFER FULL WAITING TO PRODUCE 10
Consumed 5 from cell 4      write 4    read 0    buffer:  6  7  8  9  5
Produced 10 into cell 4     write 0    read 0    buffer:  6  7  8  9  10
BUFFER FULL
ProduceInteger finished producing values
Terminating ProduceInteger

Consumed 6 from cell 0      write 0    read 1    buffer:  6  7  8  9  10
Consumed 7 from cell 1      write 0    read 2    buffer:  6  7  8  9  10
Consumed 8 from cell 2      write 0    read 3    buffer:  6  7  8  9  10
Consumed 9 from cell 3      write 0    read 4    buffer:  6  7  8  9  10
Consumed 10 from cell 4     write 0    read 0    buffer:  6  7  8  9  10
BUFFER EMPTY
ConsumeInteger retrieved values totaling: 55
Terminating ConsumeInteger
```

Demonstrating Thread Synchronization

```
Produced 1 into cell 0      write 1    read 0    buffer:  1 -1 -1 -1 -1
Produced 2 into cell 1      write 2    read 0    buffer:  1  2 -1 -1 -1
Produced 3 into cell 2      write 3    read 0    buffer:  1  2  3 -1 -1
Produced 4 into cell 3      write 4    read 0    buffer:  1  2  3  4 -1
Consumed 1 from cell 0      write 4    read 1    buffer:  1  2  3  4 -1
Produced 5 into cell 4      write 0    read 1    buffer:  1  2  3  4  5
Produced 6 into cell 0      write 1    read 1    buffer:  6  2  3  4  5
BUFFER FULL WAITING TO PRODUCE 7
Consumed 2 from cell 1      write 1    read 2    buffer:  6  2  3  4  5
Produced 7 into cell 1      write 2    read 2    buffer:  6  7  3  4  5
BUFFER FULL
Consumed 3 from cell 2      write 2    read 3    buffer:  6  7  3  4  5
Produced 8 into cell 2      write 3    read 3    buffer:  6  7  8  4  5
BUFFER FULL WAITING TO PRODUCE 9
Consumed 4 from cell 3      write 3    read 4    buffer:  6  7  8  4  5
Produced 9 into cell 3      write 4    read 4    buffer:  6  7  8  9  5
BUFFER FULL WAITING TO PRODUCE 10
Consumed 5 from cell 4      write 4    read 0    buffer:  6  7  8  9  5
Produced 10 into cell 4     write 0    read 0    buffer:  6  7  8  9  10
BUFFER FULL
ProduceInteger finished producing values
Terminating ProduceInteger

Consumed 6 from cell 0      write 0    read 1    buffer:  6  7  8  9  10
Consumed 7 from cell 1      write 0    read 2    buffer:  6  7  8  9  10
Consumed 8 from cell 2      write 0    read 3    buffer:  6  7  8  9  10
Consumed 9 from cell 3      write 0    read 4    buffer:  6  7  8  9  10
Consumed 10 from cell 4     write 0    read 0    buffer:  6  7  8  9  10
BUFFER EMPTY
ConsumeInteger retrieved values totaling: 55
Terminating ConsumeInteger
```

ProCon.java

```
ProCon.java
1  package procon;
2
3  // Show multiple threads modifying shared object.
4  import javax.swing.*;
5
6
7  public class ProCon extends JFrame {
8
9      /**
10      * Generate the default serial version UID
11      */
12     private static final long serialVersionUID = 1L;
13
14     // Set up GUI
15     public ProCon() {
16         super("Demonstrating Thread Synchronisation");
17
18         JTextArea outputArea = new JTextArea(20, 30);
19         getContentPane().add(new JScrollPane(outputArea));
20
21         setSize(500, 500);
22         setVisible(true);
23
24         // Set up threads
25         HoldIntegerSynchronized sharedObject =
26                 new HoldIntegerSynchronized(outputArea);
27
28         ProduceInteger producer =
29                 new ProduceInteger(sharedObject, outputArea);
30
31         ConsumeInteger consumer =
32                 new ConsumeInteger(sharedObject, outputArea);
33
34         // Start threads
35         producer.start();
36         consumer.start();
37     }
38
39     // Execute application
40     public static void main(String args[]) {
41         ProCon application = new ProCon();
42
43         application.setDefaultCloseOperation(
44                 JFrame.EXIT_ON_CLOSE);
45     }
46 }
```

UpdateThread.java

UpdateThread.java

```java
1  package procon;
2
3  import javax.swing.*;
4
5  public class UpdateThread extends Thread {
6
7      private JTextArea outputArea;
8      private String messageToOutput;
9
10     // Initialise outputArea and message
11     public UpdateThread(JTextArea output, String message) {
12         outputArea = output;
13         messageToOutput = message;
14     }
15
16     // Method called to update outputArea
17     @Override
18     public void run() {
19         outputArea.append(messageToOutput);
20     }
21 }
```

ProduceInteger.java

ProduceInteger.java

```java
1  package procon;
2
3  import javax.swing.*;
4
5  public class ProduceInteger extends Thread {
6
7      private HoldIntegerSynchronized sharedObject;
8      private JTextArea outputArea;
9
10     // Initialise ProduceInteger
11     public ProduceInteger(HoldIntegerSynchronized shared,
12             JTextArea output) {
13         super("ProduceInteger");
14
15         sharedObject = shared;
16         outputArea = output;
17     }
18
```

```java
19     // ProduceInteger thread loops 10 times and calls
20     // sharedObject's setSharedInt method each time
21     @Override
22     public void run() {
23         for (int count = 1; count <= 10; count++) {
24
25             // Sleep for a random interval
26             // Note: Interval shortened purposely to fill buffer
27             try {
28                 Thread.sleep((int) (Math.random() * 500));
29             } // Process InterruptedException during sleep
30             catch (InterruptedException exception) {
31                 System.err.println(exception.toString());
32             }
33
34             sharedObject.setSharedInt(count);
35         }
36
37         // Update Swing GUI component
38         SwingUtilities.invokeLater(new UpdateThread(outputArea,
39                 "\n" + getName() + " finished producing values"
40                 + "\nTerminating " + getName() + "\n"));
41     }
42 }
```

ConsumeInteger.java

ConsumeInteger.java

```java
package procon;

import javax.swing.*;

public class ConsumeInteger extends Thread {

    private HoldIntegerSynchronized sharedObject;
    private JTextArea outputArea;

    // Initialise ConsumeInteger
    public ConsumeInteger(HoldIntegerSynchronized shared,
            JTextArea output) {
        super("ConsumeInteger");
        sharedObject = shared;
        outputArea = output;
    }

    // ConsumeInteger thread loops until it receives 10
    // from sharedObject's getSharedInt method
    @Override
    public void run() {
        int value, sum = 0;

        do {
            // Sleep for a random interval
            try {
                Thread.sleep((int) (Math.random() * 3000));
            } // Process InterruptedException during sleep
            catch (InterruptedException exception) {
                System.err.println(exception.toString());
            }

            value = sharedObject.getSharedInt();
            sum += value;

        } while (value != 10);

        // Update Swing GUI component
        SwingUtilities.invokeLater(new UpdateThread(outputArea,
                "\n" + getName() + " retrieved values totaling: "
                + sum + "\nTerminating " + getName() + "\n"));
    }
}
```

HoldIntegerSynchronized.java

HoldIntegerSynchronized.java

```
1  package procon;
2
3  // Definition of class HoldIntegerSynchronized that
4  // uses thread synchronisation to ensure that both
5  // threads access sharedInt at the proper times.
6  import java.text.DecimalFormat;
7  import javax.swing.*;
8
9  public class HoldIntegerSynchronized {
10
11     // Array of shared locations
12     private int sharedInt[] = {-1, -1, -1, -1, -1};
13     // Variables to maintain buffer information
14     private boolean writeable = true;
15     private boolean readable = false;
16     private int readLocation = 0, writeLocation = 0;
17     // GUI component to display output
18     private JTextArea outputArea;
19
20     // Initialise HoldIntegerSynchronized
21     public HoldIntegerSynchronized(JTextArea output) {
22         outputArea = output;
23     }
24
```

```
25     // Synchronised method allows only one thread at a time to
26     // invoke this method to set a value in a particular
27     // HoldIntegerSynchronized object
28     public synchronized void setSharedInt(int value) {
29         while (!writeable) {
30
31             // Thread that called this method must wait
32             try {
33
34                 // Update Swing GUI component
35                 SwingUtilities.invokeLater(new UpdateThread(
36                         outputArea, " WAITING TO PRODUCE " + value));
37
38                 wait();
39             } // Process InterrupteException while thread waiting
40             catch (InterruptedException exception) {
41                 System.err.println(exception.toString());
42             }
43         }
44         // Place value in writeLocation
45         sharedInt[writeLocation] = value;
46
47         // Indicate that consumer can read a value
48         readable = true;
49
```

HoldIntegerSynchronized.java (cont'd)

```java
50         // Update Swing GUI component
51         SwingUtilities.invokeLater(new UpdateThread(outputArea,
52                 "\nProduced " + value + " into cell "
53                 + writeLocation));
54 
55         // Update writeLocation for future write operation
56         writeLocation = (writeLocation + 1) % 5;
57 
58         // Update Swing GUI component
59         SwingUtilities.invokeLater(new UpdateThread(outputArea,
60                 "\twrite " + writeLocation + "\tread "
61                 + readLocation));
62 
63         displayBuffer(outputArea, sharedInt);
64 
65         // Test if buffer is full
66         if (writeLocation == readLocation) {
67             writeable = false;
68 
69             // Update Swing GUI component
70             SwingUtilities.invokeLater(new UpdateThread(outputArea,
71                     "\nBUFFER FULL"));
72         }
73 
74         // Tell a waiting thread to become ready
75         notify();
76 
77     } // End method setSharedInt
```

```java
78 
79     // Synchronised method allows only one thread at a time to
80     // invoke this method to get a value from a particular
81     // HoldIntegerSynchronized object
82     public synchronized int getSharedInt() {
83         int value;
84 
85         while (!readable) {
86 
87             // Thread that called this method must wait
88             try {
89 
90                 // Update Swing GUI component
91                 SwingUtilities.invokeLater(new UpdateThread(
92                         outputArea, " WAITING TO CONSUME"));
93                 wait();
94             } // Process InterrupteException while thread waiting
95             catch (InterruptedException exception) {
96                 System.err.println(exception.toString());
97             }
98         }
99 
100        // Indicate that producer can write a value
101        writeable = true;
102
103        // Obtain value at current readLocation
104        value = sharedInt[readLocation];
105
```

HoldIntegerSynchronized.java (cont'd)

```
106          // Update Swing GUI component
107          SwingUtilities.invokeLater(new UpdateThread(outputArea,
108                  "\nConsumed " + value + " from cell "
109                  + readLocation));
110
111          // Update read location for future read operation
112          readLocation = (readLocation + 1) % 5;
113
114          // Update Swing GUI component
115          SwingUtilities.invokeLater(new UpdateThread(outputArea,
116                  "\twrite " + writeLocation + "\tread "
117                  + readLocation));
118
119          displayBuffer(outputArea, sharedInt);
120
121          // Test if buffer is empty
122          if (readLocation == writeLocation) {
123              readable = false;
124
125              // Update Swing GUI component
126              SwingUtilities.invokeLater(new UpdateThread(
127                      outputArea, "\nBUFFER EMPTY"));
128          }
129
130          // Tell a waiting thread to become ready
131          notify();
132
133          return value;
134
135      } // End method getSharedInt
```

```
136
137      // Display contents of shared buffer
138      public void displayBuffer(JTextArea outputArea,
139              int buffer[]) {
140          DecimalFormat formatNumber = new DecimalFormat(" #;-#");
141          StringBuffer outputBuffer = new StringBuffer();
142
143          // Place buffer elements in outputBuffer
144          for (int count = 0; count < buffer.length; count++) {
145              outputBuffer.append(
146                      " " + formatNumber.format(buffer[ count]));
147          }
148
149          // Update Swing GUI component
150          SwingUtilities.invokeLater(new UpdateThread(outputArea,
151                  "\tbuffer: " + outputBuffer));
152      }
153  }
```