# 请大家阅读文档时，在视图里勾选导航窗格，在左边显示章节目录方便浏览。

# 一、填空题

1. 创建线程的方式有\_**继承Thread类**和**实现Runnable接口**。

2. 程序中可能出现一种情况：多个线种互相等待对方持有的锁，而在得到对方的锁之前都不会释放自己的锁，这就是\_\_\_\_**死锁**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

3. 若在线程的执行代码中调用yield方法后，则该线程将**申请释放CPU资源，挂起进入ready，给其它进程调度机会**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

4.线程程序可以调用\_\_\_\_\_\_\_\_\_**sleep()\_**\_\_\_\_\_\_\_\_\_\_\_方法，使线程进入睡眠状态，可以通过调用\_\_\_\_\_\_\_\_\_**setPriority()**\_\_\_\_\_\_\_\_\_\_方法设置线程的优先级。

5. 获得当前线程id的语句是\_\_**Thread.currentThread().getId()**\_\_\_。

# 二、单项选择题

1. 能够是线程进入死亡状态的是\_\_\_\_**C\_**\_\_\_\_\_\_。

A. 调用Thread类的yield方法

B. 调用Thread类的sleep方法

C. 线程任务的run方法结束

D. 线程死锁

1. 给定下列程序：
2. public class Holder {
3. private int data = 0;
4. public int getData () {return data;}
5. public synchronized void inc (int amount) {
6. int newValue = data + amount;
7. try {Thread.sleep(5);
8. } catch (InterruptedException e) {}
9. data = newValue;
10. }
11. public void dec (int amount) {
12. int newValue = data - amount;
13. try {Thread.sleep(1);
14. } catch (InterruptedException e) {}
15. data = newValue;
16. }
17. }
18. public static void main (String [] args) {
19. ExecutorService es = Executors.newCachedThreadPool();
20. Holder holder = new Holder ();
21. int incAmount = 10, decAmount = 5, loops = 100;
22. Runnable incTask = () -> holder.inc(incAmount);
23. Runnable decTask = () -> holder.dec(decAmount);
24. for (int i = 0; i < loops; i++) {
25. es. execute(incTask);
26. es. execute(decTask);
27. }
28. es. shutdown ();
29. while (! es. isTerminated ()) {}
30. }

下列说法正确的是\_\_\_\_\_**B\_**\_\_\_\_\_。

A. 当一个线程进入holder对象的inc方法后，holder对象被锁住，因此其他线程不能进入inc方法和dec方法

B. 当一个线程进入holder对象的inc方法后，holder对象被锁住，因此其他线程不能进入inc方法，但可以进入dec方法

C. 当一个线程进入holder对象的dec方法后，holder对象被锁住，因此其他线程不能进入dec方法和inc方法

D. 当一个线程进入holder对象的dec方法后，holder对象被锁住，因此其他线程不能进入dec方法，但可以进入inc方法

1. 给定下列程序：
2. public class Test2\_3 {
3. private static Object lockObject = new Object ();
4. */\*\**
5. \* 计数器
6. \*/
7. public static class Counter {
8. private int count = 0;
9. public int getCount () {return count;}
10. public void inc () {
11. synchronized (lockObject) {
12. int temp = count + 1;
13. try {Thread.sleep(5);} catch (InterruptedException e) {}
14. count = temp;
15. }
16. }
17. public void dec () {
18. synchronized (lockObject) {
19. int temp = count - 1;
20. try {Thread.sleep(5);} catch (InterruptedException e) {}
21. count = temp;
22. }
23. }
24. }
25. public static void main (String [] args) {
26. ExecutorService es = Executors.newCachedThreadPool();
27. Counter counter1 = new Counter ();
28. Counter counter2 = new Counter ();
29. int loops1 = 10, loops2 = 5;
30. Runnable incTask = () -> counter1.inc ();
31. Runnable decTask = () -> counter2.dec ();
32. for (int i = 0; i < loops1; i++) {es. execute(incTask);}
33. for (int i = 0; i < loops2; i++) {es. execute(decTask);}
34. es. shutdown ();
35. while (! es. isTerminated ()) {}
36. }
37. }

下面说法正确的是\_\_\_\_\_**C**\_\_\_\_\_\_。

A. incTask的执行线程进入counter1对象的inc方法后，counter1对象被上锁，会阻塞decTask的执行线程进入counter2对象的dec方法

B. incTask的执行线程进入counter1对象的inc方法后，counter1对象被上锁，不会阻塞decTask的执行线程进入counter2对象的dec方法

C. incTask的执行线程进入对象counter1的inc方法后，lockObject对象被上锁，会阻塞decTask执行线程进入counter2对象的方法dec

D. incTask的执行线程进入对象counter1的inc方法后，lockObject对象被上锁，不会阻塞decTask执行线程进入counter2对象的方法dec

1. 给定下列程序：
2. public class Test2\_4 {
3. public static class Resource {
4. private int value = 0;
5. public int sum (int amount) {
6. int newValue = value + amount;
7. try {Thread.sleep(5);} catch (InterruptedException e) {}
8. return newValue;
9. }
10. public int sub (int amount) {
11. int newValue = value - amount;
12. try {Thread.sleep(5);} catch (InterruptedException e) {}
13. return newValue;
14. }
15. }
16. public static void main (String [] args) {
17. ExecutorService es = Executors.newCachedThreadPool();
18. Resource r = new Resource ();
19. int loops1 = 10, loops2 = 5, amount = 5;
20. Runnable sumTask = () -> r.sum(amount);
21. Runnable subTask = () -> r.sub(amount);
22. for (int i = 0; i < loops1; i++) {es. execute(sumTask);}
23. for (int i = 0; i < loops2; i++) {es. execute(subTask);}
24. es. shutdown ();
25. while (! es. isTerminated ()) {}
26. }
27. }

下面说法正确的是\_\_\_\_\_**C**\_\_\_\_\_\_。

A. 由于方法sum和sub都没有采取任何同步措施，所以sumTask和subTask的执行线程都可以同时进入共享资源对象r的sum方法或sub方法，造成对象r的实例成员value的值不一致；

B. 由于方法sum和sub都没有采取任何同步措施，所以sumTask和subTask的执行线程都可以同时进入共享资源对象r的sum方法或sub方法，造成方法内局部变量newValue和形参amount的值不一致；

C. 虽然方法sum和sub都没有采取任何同步措施，但Resource类的sum和sub里的局部变量newValue和形参amount位于每个线程各自的堆栈里互不干扰，同时多个线程进入共享资源对象r的sum方法或sub方法后，对实例数据成员value都只有读操作，因此Resource类是线程安全的

D. 以上说法都不正确

5. 给定下列程序：

1. public class Test2\_5 {
2. public static class Resource {
3. private static int value = 0;
4. public static int getValue () {return value;}
5. public static void inc (int amount) {
6. synchronized (Resource. Class) {
7. int newValue = value + amount;
8. try {Thread.sleep(5);} catch (InterruptedException e) {}
9. value = newValue;
10. }
11. }
12. public synchronized static void dec (int amount) {
13. int newValue = value - amount;
14. try {Thread.sleep(2);} catch (InterruptedException e) {}
15. value = newValue;
16. }
17. }
18. public static void main (String [] args) {
19. ExecutorService es = Executors.newCachedThreadPool();
20. int incAmount = 10, decAmount = 5, loops = 100;
21. Resource r1 = new Resource ();
22. Resource r2 = new Resource ();
23. Runnable incTask = () -> r1.inc(incAmount);
24. Runnable decTask = () -> r2.dec(decAmount);
25. for (int i = 0; i < loops; i++) {es. execute(incTask); es. execute(decTask);}
26. es. shutdown ();
27. while (! es. isTerminated ()) {}
28. }
29. }

下面说法**错误的**的是\_\_\_\_\_\_**B**\_\_。

A. 同步的静态方法public synchronized static void dec (int amount) {} 等价于public static void dec (int amount) {synchronized (Resource. class) {}}

B. incTask的执行线程访问的对象r1，decTask访问的是对象r2，由于访问的是不同对象，因此incTask的执行线程和decTask的执行线程之间不会同步

C. 虽然incTask的执行线程和decTask的执行线程访问的是Resource类不同对象r1和r2，但由于调用的是Resource类的同步静态方法，因此incTask的执行线程和decTask的执行线程之间是被同步的

D. 一个线程进入Resource类的同步静态方法后，这个类的所有静态同步方法都被上锁，而且上的是对象锁，被锁的对象是Resource.class。但是这个锁的作用范围是Resource类的所有实例，即不管线程通过Resource类的哪个实例调用静态同步方法，都将被阻塞

6. 假设一个临界区通过Lock锁进行同步控制，当一个线程拿到一个临界区的Lock锁，进入该临界区后，该临界区被上锁。这时下面的说法正确的是\_\_\_\_\_**D**\_\_\_\_\_\_。

A. 如果在临界区里线程执行Thread.sleep方法，将导致线程进入阻塞状态，同时临界区的锁会被释放；如果在临界区里线程执行Lock锁的条件对象的await方法，将导致线程进入阻塞状态，同时临界区的锁会被释放

B.如果在临界区里线程执行Thread.sleep方法，将导致线程进入阻塞状态，同时临界区的锁不会被释放；如果在临界区里线程执行Lock锁的条件对象的await方法，将导致线程进入阻塞状态，同时临界区的锁不会被释放

C. 如果在临界区里线程执行Thread.sleep方法，将导致线程进入阻塞状态，同时临界区的锁会被释放；如果在临界区里线程执行Lock锁的条件对象的await方法，将导致线程进入阻塞状态，同时临界区的锁不会被释放

D. 如果在临界区里线程执行Thread.sleep方法，将导致线程进入阻塞状态，同时临界区的锁不会被释放；如果在临界区里线程执行Lock锁的条件对象的await方法，将导致线程进入阻塞状态，同时临界区的锁会被释放

# 三、问答题

1：有三个线程T1，T2，T3，怎么确保它们按指定顺序执行：首先执行T1，T1结束后执行T2，T2结束后执行T3，T3结束后主线程才结束。请给出示意代码。

 public static void main(String[] args) throws InterruptedException {

*// 创建 T1 线程*

        Thread T1 = new Thread(() -> {

            System.out.println("T1 执行");

            try {

                sleep(1000); *// 模拟 T1 执行时间*

            } catch (InterruptedException e) {

                throw new RuntimeException(e);

            }

            System.out.println("T1 执行完毕");

        });

*// 创建 T2 线程*

        Thread T2 = new Thread(() -> {

            System.out.println("T2 执行");

            try {

                sleep(1000); *// 模拟 T2 执行时间*

            } catch (InterruptedException e) {

                throw new RuntimeException(e);

            }

            System.out.println("T2 执行完毕");

        });

*// 创建 T3 线程*

        Thread T3 = new Thread(() -> {

            System.out.println("T3 执行");

            try {

                sleep(1000); *// 模拟 T3 执行时间*

            } catch (InterruptedException e) {

                throw new RuntimeException(e);

            }

            System.out.println("T3 执行完毕");

        });

*// 启动线程*

        T1.start();

        T1.join();  *// 等待 T1 执行完*

        T2.start();

        T2.join();  *// 等待 T2 执行完*

        T3.start();

        T3.join();  *// 等待 T3 执行完*

        System.out.println("主线程结束");

    }