# 多线程作业

一、填空题

1. 处于运行状态的线程在某些情况下，如执行了 sleep（睡眠）方法，或等待 I/O 设备等资源，将让出 CPU 并暂时停止自己的运行，进入____阻塞_____状态。
2. 处于新建状态的线程被启动后，将进入线程队列排队等待 CPU，此时它已具备了运行条件，一旦轮到享用 CPU 资源就可以获得执行机会。上述线程是处于__就绪__ 状态。
3. 一个正在执行的线程可能被人为地中断，让出 CPU 的使用权，暂时中止自己的执行，进入_阻塞__状态。
4. 在 Java 中编写实现多线程应用有两种途径：一种是继承 Thread 类创建线程，另一种是实现__Runnable__　　接口创建线程。
5. 在线程控制中，可以调用_____join()_____方法，阻塞当前正在执行的线程，等插队线程执行完后后再执行阻塞线程。
6. 多线程访问某个共享资源可能出现线程安全问题，此时可以使用_____synchronized_____关键字来实现线程同步，从而避免安全问题出现，但会影响性能，甚至出现死锁。在线程通信中，调用 wait()可以是当前线程处于等待状态，而为了唤醒一个等待的线程，需要调用的方法是_____notifyAll()_____。
7. 在线程通信中，可以调用 wait()、notify()、notifyAll()三个方法实现线程通信，这三个方法都是_____Object_____类提供的 public 方法，所以任何类都具有这三个方法。

二、选择题

**1.** 下列关于 **Java** 线程的说法正确的是（A）。（选择一项）

**A** 每一个 Java 线程可以看成由代码、一个真实的 CPU 以及数据三部分组成

**B** 创建线程的两种方法中，从 Thread 类中继承方式可以防止出现多父类的问题

**C** Thread 类属于 java.util 程序包

**D** 使用 new Thread(new X()).run();方法启动一个线程

**2.** 以下选项中可以填写到横线处，让代码正确编译和运行的是（A）。（选择一项）

```
public class Test implements Runnable {
    public static void main(String[] args) {
        ____________________________________
        __ t.start();
        System.out.println("main");
    }
    public void run() {
        System.out.println("thread1!");
```

```
    }
}
```

**A** Thread t = new Thread(new Test());

**B** Test t = new Test();

**C** Thread t = new Test();

**D** Thread t = new Thread();

**3.** 如下代码创建一个新线程并启动线程，问**:**四个选项中可以保证正确代码创建 **target** 对象，并能编译正确的是（C）？（选择一项）

```
public static void main(String[] args) {
    Runnable target=new MyRunnable( );
    Thread myThread=new Thread(target);
}
```

**A** **public class** MyRunnable **extends** Runnable { **public void** run( ) { } }

**B** **public class** MyRunnable **extends** Runnable { **void** run( ) { } }

**C** **public class** MyRunnable **implements** Runnable{ **public void** run( ) { } }

**D** **public class** MyRunnable **implements** Runnable{ **void** run( ) { }

**4.** 当线程调用 **start( )**后，其所处状态为（C）。（选择一项）

**A** 阻塞状态 **B.** 运行状态 **C.** 就绪状态

**D.** 新建状态

**5.** 下列关于 **Thread** 类提供的线程控制方法的说法中，错误的是（C）。（选择一项）

**A** 线程 A 中执行线程 B 的 join()方法，则线程 A 等待直到 B 执行完成

**B** 线程 A 通过调用 interrupt()方法来中断其阻塞状态

**C** 若线程 A 调用方法 isAlive()返回值为 false，则说明 A 正在执行中，也可能是可运行状态

**D** currentThread()方法返回当前线程的引用

**6.** 下列关于线程的优先级说法中，正确的是（BC ）。（选择两项）

**A** 线程的优先级是不能改变的

**B** 线程的优先级是在创建线程时设置的

**C** 在创建线程后的任何时候都可以重新设置

**D** 线程的优先级的范围在 1–100 之间

**7.** 以下选项中关于 **Java** 中线程控制方法的说法正确的是（A D）。（选择二项）

**A** join ( ) 的作用是阻塞指定线程等到另一个线程完成以后再继续执行

**B** sleep ( ) 的作用是让当前正在执行线程暂停，线程将转入就绪状态

**C** yield ( ) 的作用是使线程停止运行一段时间，将处于阻塞状态

**D** setDaemon( )的作用是将指定的线程设置成后台线程

**8.** 在多个线程访问同一个资源时，可以使用（A）关键字来实现线程同步，保证对资源安全访问。（选择一项）

**A** synchronized

**B** transient

**C** static

**D** yield

**9.** **Java** 中线程安全问题是通过关键字（C）解决的**?**。（选择一项）

**A** finally

**B** wait( )

**C** synchronized

**D** notify( )

**10.** 以下说法中关于线程通信的说法错误的是（D ）**?**。（选择一项）

**A** 可以调用 wait()、notify()、notifyAll()三个方法实现线程通信

**B** wait()、notify()、notifyAll()必须在 synchronized 方法或者代码块中使用

**C** wait()有多个重载的方法，可以指定等待的时间

**D** wait()、notify()、notifyAll()是 Object 类提供的方法，子类可以重写

三、判断题

1. 进程是线程 Thread 内部的一个执行单元，它是程序中一个单一顺序控制流程。( ×)
2. Thread 类实现了 Runnable 接口。(√ )
3. 一个进程可以包括多个线程。两者的一个主要区别是：线程是资源分配的单位，而进程 CPU 调度和执行的单位。 （ ×）
4. 用 new 关键字建立一个线程对象后，该线程对象就处于新生状态。处于新生状态的线程有自己的内存空间，通过调用 start 进入就绪状态。（√）
5. A 线程的优先级是 10，B 线程的优先级是 1，那么当进行调度时一定会先调用 A （ ×）
6. 线程可以用 yield 使低优先级的线程运行。 （ ×）
7. Thread.sleep( )方法调用后，当等待时间未到，该线程所处状态为阻塞状态。当等待时间已到，该线程所处状态为运行状态。 （ ×）
8. 当一个线程进入一个对象的一个 synchronized 方法后，其它线程不可以再进入该对象同步的其它方法执行。（√）
9. wait 方法被调用时，所在线程是会释放所持有的锁资源。 sleep 方法不会释放。（√）
10. wait、notify、notifyAll 是在 Object 类中定义的方法。（√）
11. notify 是唤醒所在对象 wait pool 中的第一个线程。 （ ×）

四、简答题

1. 简述程序、进程和线程的联系和区别。

程序是静态的，进程是动态进行的，线程是在进程的基础上进行的。

2. 创建线程的两种方式分别是什么？各有什么优缺点。

采用继承 Thread 类方式：

(1) 优点：编写简单，如果需要访问当前线程，无需使用 Thread.currentThread()方法，直接使用 this，即可获得当前线程。

(2) 缺点：因为线程类已经继承了 Thread 类，所以不能再继承其他的父类。

采用实现 Runnable 接口方式：

(1) 优点：线程类只是实现了 Runnable 接口，还可以继承其他的类和实现多个接口。

(2) 缺点：编程稍微复杂，如果需要访问当前线程，必须使用 Thread.currentThread()方法。



3. sleep、yield、join 方法的区别?

1.sleep()方法

在指定时间内让当前正在执行的线程暂停执行，但不会释放“锁标志”。不推荐使用。

sleep()使当前线程进入阻塞状态，在指定时间内不会执行。

2.yield 方法

暂停当前正在执行的线程对象。

yield()只是使当前线程重新回到可执行状态，所以执行 yield()的线程有可能在进入到可执行状态后马上又被执行。

yield()只能使同优先级或更高优先级的线程有执行的机会。

3.join 方法

等待该线程终止。

等待调用 join 方法的线程结束，再继续执行。

4. synchronize 修饰的语句块，如下面的代码。是表示该代码块运行时必须获得 account 对象的锁。如果没有获得，会有什么情况发生?

```
synchronized (account)
    { if(account.money-
    drawingNum<0){ return;
    }
}
```

使用多线程时，出现多次提现问题，提现总金额大于账户总余额。

5. 请你简述 sleep( )和 wait( )有什么区别?

sleep 方法属于 Thread 类中方法，表示让一个线程进入睡眠状态，等待一定的时间之后，自动醒来进入到可运行状态，不会马上进入运行状态，因为线程调度机制恢复线程的运行也需要时间，一个线程对象调用了 sleep 方法之后，并不会释放他所持有的所有对象锁，所以也就不会影响其他进程对象的运行。

wait 属于 Object 的成员方法，一旦一个对象调用了 wait 方法，必须要采用 notify()和 notifyAll()方法唤醒该线程;如果线程拥有某个或某些对象的同步锁，那么在调用了 wait()后，这个线程就会释放它持有的所有同步资源，而不限于这个被调用了 wait()方法的对象。

6. 死锁是怎么造成的？用文字表达。再写一个代码示例。

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

```
public static void main(String[] args) {
    StringBuffer s1 = new StringBuffer();
    StringBuffer s2 = new StringBuffer();
    new Thread(() -> {
      synchronized (s1) {
        s1.append("a");
        s2.append("1");
```

```
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
        synchronized (s2) {
            s1.append("b");
            s2.append("2");
            System.out.println(s1);
            System.out.println(s2);
        }
    }).start();
    new Thread(() -> {
        synchronized (s2) {
            s1.append("c");
            s2.append("3");
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
        synchronized (s1) {
            s1.append("d");
            s2.append("4");
            System.out.println(s1);
            System.out.println(s2);
        }
    }).start();
}
```

7. Java 中实现线程通信的三个方法及其作用。

wait():导致当前线程等待，直到其他线程调用该同步监视器的 notify()方法或 notifyAll()方法来唤醒该线程。

notify():唤醒在此同步监视器上等待的单个线程。

notifyAll():唤醒在此同步监视器上等待的所有线程。

8. 为什么不推荐使用 stop 和 destroy 方法来结束线程的运行?

调用 stop 和 destroy 方法会出现以下情况

1. 即刻抛出 ThreadDeath 异常，在线程的 run()方法内，任何一点都有可能抛出 ThreadDeath Error，包括在 catch 或 finally 语句中。
2. 释放该线程所持有的所有的锁

Stop 的缺点：stop 这种方法本质上是不安全的

使用 Thread.stop 停止线程会导致它解锁所有已锁定的监视器，即直接释放当前线程已经获取到的所有锁，使得当前线程直接进入阻塞状态

destroy 的缺点：

五、编码题

1. 设计一个多线程的程序如下：设计一个火车售票模拟程序。假如火车站要有 100 张火车票要卖出，现在有 5 个售票点同时售票，用 5 个线程模拟这 5 个售票点的售票情况。

```
package Homeworks.HomeworkDay10;
2 个用法  新 *
class Window implements Runnable{
    2 个用法
    private int ticket = 100;
    新 *
    @Override
    public void run() {
        while (true){
            synchronized (this){
            if(ticket>0){
                try {
                    Thread.sleep( millis: 100);
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
                System.out.println(Thread.currentThread().getName() + ":票号为"+ticket--);
            }else{
                System.out.println(Thread.currentThread().getName() + "车票售空！！！");
                break;
            }
            }
        }

    }
}
```

2. 编写两个线程,一个线程打印 1–52 的整

```
0 个用法  新 *
public class Demo01  {
    新 *
    public static void main(String[] args) {
        Window window=new Window();
        Thread t1 = new Thread(window);
        Thread t2 = new Thread(window);
        Thread t3 = new Thread(window);
        Thread t4 = new Thread(window);
        Thread t5 = new Thread(window);
        t1.setName("窗口1");
        t2.setName("窗口2");
        t3.setName("窗口3");
        t4.setName("窗口4");
        t5.setName("窗口5");
        t1.start();
        t2.start();
        t3.start();
        t4.start();
        t5.start();
    }
}
```

数，另一个线程打印字母 A–Z。打印顺序为

12A34B56C....5152Z。即按照整数和字母的顺序从小到大打印，并且每打印两个整数后，打印一个字母，交替循环打印，直到打印到整数 52 和字母 Z 结束。

要求：

1) 编写打印类 Printer，声明私有属性 index，初始值为 1，用来表示是第几次打印。
2) 在打印类 Printer 中编写打印数字的方法 print(int i)，3 的倍数就使用 wait()方法等待，否则就输出 i，使用 notifyAll()进行唤醒其它线程。
3) 在打印类 Printer 中编写打印字母的方法 print(char c)，不是 3 的倍数就等待，否则就打印输出字母 c，使用 notifyAll()进行唤醒其它线程。
4) 编写打印数字的线程 NumberPrinter 继承 Thread 类，声明私有属性 private Printer p;

在构造方法中进行赋值，实现父类的 run 方法，调用 Printer 类中的输出数字的方法。

5) 编写打印字母的线程 LetterPrinter 继承 Thread 类，声明私有属性 private Printer p;在构造方法中进行赋值，实现父类的 run 方法，调用 Printer 类中的输出字母的方法。
6) 编写测试类 Test，创建打印类对象，创建两个线程类对象，启动线程。

```java
package Homeworks.HomeworkDay10;


public class NumberPrinter extends Thread{

    private Printer p;

    public NumberPrinter(Printer p) { this.p=p; }

    public void run(){
        for (int i = 1; i <= 52 ; i++) {
            p.print(i);
        }
    }
}
```

```java
package Homeworks.HomeworkDay10;

6 个用法  新 *
public class Printer {
    4 个用法
    private int index = 1;
    新 *
    public synchronized void print(int i){
        while (index % 3 == 0){
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
            System.out.print(i);
            index++;
            notifyAll();

    }
    新 *
    public synchronized void print(char c){
        while (index % 3 != 0){
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
            System.out.print(c);
            index++;
            notifyAll();
    }
}
```

```java
package Homeworks.HomeworkDay10;

0 个用法  新 *
public class Test1 {
    新 *
    public static void main(String[] args) {
        Printer p=new Printer();
        Thread t1=new NumberPrinter(p);
        Thread t2=new LetterPrinter(p);
        t1.start();
        t2.start();
    }
}
```

```java
package Homeworks.HomeworkDay10;

/**
 * @author junhaocai
 * @email junhaocai01@gmail.com
 * @date 2023/1/12
 */
1个用法  新 *
public class LetterPrinter extends Thread{
    2个用法
    private Printer p;
    1个用法  新 *
    public LetterPrinter(Printer p){
        this.p = p;
    }

    新 *
    public void run(){
        for (char i = 'A'; i <= 'Z' ; i++) {
            p.print(i);
        }
    }
}
```

六、可选题

1. 设计 4 个线程，其中两个线程每次对 j 增加 1，另外两个线程对 j 每次减少 1。

   要求：使用内部类实现线程，对 j 增减的时候不考虑顺序问题。

2. 编写多线程程序，模拟多个人通过一个山洞的模拟。这个山洞每次只能通过一个人，每个人通过山洞的时间为 5 秒，有 10 个人同时准备过此山洞，显示每次通过山洞人的姓名和顺序。