**每日作业卷答案**

**java基础第13天**

# 关卡1

## 训练案例1

### 训练考核知识点

进程的概念；

### 训练描述

请描述什么是进程。

### 操作步骤答案

进程指正在运行的程序。确切的来说，当一个程序进入内存运行，即变成一个进程，进程是处于运行过程中的程序，并且具有一定独立功能。

## 训练案例2

### 训练考核知识点

讲义1.1：多线程介绍。

### 训练描述

请描述什么是线程。

### 操作步骤答案

线程是进程中的一个执行单元，负责当前进程中程序的执行，一个进程中至少有一个线程。一个进程中是可以有多个线程的，这个应用程序也可以称之为多线程程序。

## 训练案例3

### 训练考核知识点

讲义1.1：多线程介绍。

### 训练描述

请描述进程与线程之间的关系，并举例说明。

### 操作步骤答案

一个程序运行后至少有一个进程，一个进程中可以包含多个线程,但一个进程中至少包含一个线程。比如使用迅雷软件下载网络文件时，同时下载多个文件，就使用到了多线程下载。

## 训练案例4

### 训练考核知识点

讲义1.2：多线程程序运行的原理。

### 训练描述

请描述多线程程序运行的原理。

### 操作步骤答案

大部分操作系统都支持多进程并发运行，现在的操作系统几乎都支持同时运行多个

程序。比如：现在我们上课一边使用编辑器，一边使用录屏软件，同时还开着画图板，

dos窗口等软件。此时，这些程序是在同时运行，”感觉这些软件好像在同一时刻运行

着“。实际上，CPU(中央处理器)使用抢占式调度模式在多个线程间进行着高速的切换。 对于CPU的一个核而言，某个时刻，只能执行一个线程，而 CPU的在多个线程间切换 速度相对我们的感觉要快，看上去就是在同一时刻运行。

## 训练案例5

### 训练考核知识点

讲义1.3：主线程。

### 训练描述

请描述什么主线程。

### 操作步骤答案

jvm启动后，必然有一个执行路径(线程)从main方法开始的，一直执行到main方

法结束，这个线程在java中称之为主线程（main线程）。

## 训练案例6

### 训练考核知识点

讲义1.4 ：Thread类。

### 训练描述

请描述Thread类中的start（）方法与run方法的区别。

### 操作步骤答案

线程对象调用run方法不开启线程，仅是对象调用方法。线程对象调用start开启

线程，并让jvm调用run方法在开启的线程中执行。

## 训练案例7

### 训练考核知识点

讲义1.4 ：Thread类。

### 训练描述

请描述创建线程的两种方式。

### 操作步骤答案

* 第一种方式是将类声明为 Thread 的子类。该子类应重写 Thread 类的 run 方法。创建对象，开启线程。run方法相当于其他线程的main方法。
* 第二种方方式是声明一个实现 Runnable 接口的类。该类然后实现 run 方法。然后创建Runnable的子类对象，传入到某个线程的构造方法中，开启线程。。

## 训练案例8

### 训练考核知识点

讲义1.5：创建线程方式一继承Thread类。

### 训练描述

请编写程序，分别打印主线程的名称和子线程的名称。

要求使用两种方式实现：

第一种方式：继承Thread类。

第二种方法：实现Runnable接口。

### 操作步骤答案

* 第一种方式：继承Thread类

|  |
| --- |
| /\*  \* 1.定义一个子线程的类，继承Thread类；  \*/  **public** **class** SubThread **extends** Thread{  /\*  \*2.在子线程类中重写run方法，在run方法中打印子线程的名称；  \*/  **public** **void** run() {  // 打印子线程的名称  System.***out***.println("subThread:"+Thread.*currentThread*().getName());  }  } |

|  |
| --- |
| /\*  \* 3.定义一个测试类，  \*/  **public** **class** ThreadDemo {  **public** **static** **void** main(String[] args) {  // 4.在main方法中打印主线程的名称；  System.***out***.println("main:"+Thread.*currentThread*().getName());  // 5.在main方法中创建子线程对象；  SubThread st = **new** SubThread();  // 6.调用子线程对象的start方法，开启子线程。  st.start();  }  } |

。

* 第二种方式：实现Runnable接口

|  |
| --- |
| /\*  \* 1.定义一个子任务类，实现Runnable接口。  \*/  **public** **class** SubRunnable **implements** Runnable {  @Override  **public** **void** run() {  // 2.在子任务类中重写run方法，在run方法中打印子线程的名称。  System.***out***.println("SubRunnable:"+Thread.*currentThread*().getName());  }  } |

|  |
| --- |
| /\*  \* 3.定义一个测试类。  \*/  **public** **class** RunnableDemo {  **public** **static** **void** main(String[] args) {  // 4.在main方法中打印主线程的名称。  System.***out***.println("RunnableDemo:"+Thread.*currentThread*().getName());  // 5.在main方法中创建一个子任务对象。  SubRunnable r = **new** SubRunnable();  // 6.在main方法中创建一个Thread类的对象，并把子任务对象传递给Thread类的构造方法。  Thread t = **new** Thread(r);  // 7.调用Thread类对象的start方法开启子线程。  t.start();  }  } |

## 训练案例9

### 训练考核知识点

讲义1.6.2：实现Runnable的好处。

### 训练描述

请描述使用实现Runnable接口的方式创建线程的好处。

### 操作步骤答案

第二种方式实现Runnable接口避免了单继承的局限性，所以较为常用。实现

Runnable接口的方式，更加的符合面向对象，线程分为两部分，一部分线程对象，

一部分线程任务。

第一种方式继承Thread类，线程对象和线程任务耦合在一起。一旦创建Thread 类的子类对象，既是线程对象，有又有线程任务。

实现runnable接口，将线程任务单独分离出来封装成对象，类型就是Runnable

接口类型。Runnable接口对线程对象和线程任务进行解耦。

## 训练案例10

### 训练考核知识点

讲义2.1：线程安全。

### 训练描述

请描述在什么样的情况下会出现线程安全的问题。

### 操作步骤答案

线程安全问题都是由全局变量及静态变量引起的。若每个线程中对全局变量、静态

变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程

同时执行写操作，一般都需要考虑线程同步，否则的话就可能影响线程安全。

## 训练案例11

### 训练考核知识点

讲义2.2：线程同步。

### 训练描述

使用3个线程模拟电影院3个窗口同时卖票的过程，总票数量为100张。在卖票的

过程要考虑到线程同步的问题，不能出现多卖、少卖、重复卖票的情况。

### 操作步骤答案

|  |
| --- |
| /\*  \* 1.定义一个子任务类，实现Runnable接口，用来模拟电影票。  \*/  **public** **class** Ticket **implements** Runnable{  // 1.1 定义一个变量，用来表示总票数量  **int** ticket = 100;  // 1.2 定义一个锁对象，用来监视同步代码块  Object lock = **new** Object();  /\*  \* 1.3 在子任务类中，重写Runnable接口的run方法  \*/  @Override  **public** **void** run() {  // 1.4 在run方法中，模拟卖票  **while**(**true**){  // 1.5 使用同步代码块使用锁对象保证同一时间只能有一个线程（窗口）在卖票  **synchronized**(lock){  // 1.6 如果剩余票数大于0就继续卖票，否则就不卖票  **if**(ticket>0){  **try** {  // 1.7 前后两次卖票之间睡眠100毫秒  Thread.*sleep*(10);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  // 1.8 模拟正在卖票  System.***out***.println(Thread.*currentThread*().getName()+"正在卖第"+(ticket--)+"张票！");  }  }  }  }  } |

|  |
| --- |
| /\*  \* 2.定义一个测试  \*/  **public** **class** TicketDemo {  **public** **static** **void** main(String[] args) {  // 2.1 在main方法中创建票对象  Ticket ticket = **new** Ticket();  // 2.2 创建3个子线程对象，模拟3个售票窗口  Thread t1 = **new** Thread(ticket, "窗口1");  Thread t2 = **new** Thread(ticket, "窗口2");  Thread t3 = **new** Thread(ticket, "窗口3");  // 2.3 开启三个子线程，模拟开始卖票  t1.start();  t2.start();  t3.start();  }  } |

## 训练案例12

### 训练考核知识点

讲义2.4：线程状态。

### 训练描述

请描述线程的几个状态。

### 训练步骤答案

线程包含以下几个状态：

1. 新建状态：

调用构造方法创建线程对象后，线程就会处于新建状态。

2.就绪状态：

一个新创建的线程并不自动开始运行，要执行线程，必须调用线程的start()

方法。在调用start()方法后，运行run()方法之前，线程就会处于就绪状态。

3.运行状态：

当线程获得CPU的资源后，才会进入运行状态，调用run方法。

4.阻塞状态：

当正在运行的线程还没有运行结束，暂时让出CPU资源，让其它处于就绪

状态的线程获得CPU资源。

5.死亡状态：

当线程的run方法运行结束后，线程就结束了，此时线程就会死亡状态。

# 关卡2

## 训练案例1

### 训练考核知识点

讲义2.2：线程同步。

### 训练描述

请按如下要求编写多线程程序：

1.得到一个随机的整数n，创建n个子线程对象。

2.要求在子线程中把当前线程的名称作为元素添加一个集合中。

3.当n个线程的名称都添加到集合中，遍历集合打印每个线程的名称。

### 操作步骤答案

|  |
| --- |
| /\*  \* 1.定义一个子任务类，实现Runnable接口。  \*/  **public** **class** AddThreadRunnable **implements** Runnable {  // 1.1 定义一个ArrayList集合对象，用于存放线程的名称。  List<String> list = **new** ArrayList<String>();  // 1.2 定义一个变量，用来记录正在运行的线程个数  **int** threadCount;  // 1.3 定义一个构造方法，参数threadCount是用来传递线程总个数  **public** AddThreadRunnable(**int** threadCount) {  // 1.4 给正在运行的线程个数赋值为线程总数  **this**.threadCount = threadCount;  }  /\*  \* 1.5 重写Runnable接口的run方法  \*/  @Override  **public** **void** run() {  // 1.6 使用同步代码块，保证同一时间只能向集合中添加一个线程名称  **synchronized** (list) {  **try** {  // 1.7 让子线程睡眠1毫秒  Thread.*sleep*(1);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  // 1.8 把当前线程名称添加到集合中  list.add(Thread.*currentThread*().getName()+":"+threadCount);  // 1.9 正在运行的线程个数减1  threadCount--;  // 1.10 所有的线程运行结束后遍历集合，打印所有线程的名称  **if** (threadCount == 0) {  System.***out***.println("所有线程已经添加到集合中");  **for** (**int** x = 0; x < list.size(); x++) {  System.***out***.println(list.get(x));  }  }  }  }  } |

|  |
| --- |
| /\*  \* 2.定义一个测试类  \*/  **public** **class** AddThreadDemo {  **public** **static** **void** main(String[] args) {  // 2.1 在main方法中使用Random得到一个随机数  Random r = **new** Random();  **int** n = r.nextInt(10)+1;    // 2.2 创建子任务对象，并把线程总数传递过去  AddThreadRunnable fr = **new** AddThreadRunnable(n);  // 2.3 循环创建n个子线程对象，并开启  **for**(**int** x=0;x<n;x++){  Thread t = **new** Thread(fr);  t.start();  }  }  } |

## 训练案例2

### 训练考核知识点

讲义2.2：线程同步。

### 训练描述

请按要求编写多线程应用程序，模拟多个人通过一个山洞：

1.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；

2.随机生成10个人，同时准备过此山洞，显示每次通过山洞人的姓名；

### 操作步骤答案

|  |
| --- |
| /\*  \* 1.定义一个隧道类，实现Runnable接口  \*/  **public** **class** Tunnel **implements** Runnable {  // 1.1 定义一个变量，用来记录通过隧道的人数  **private** **int** crossNum = 0;    /\*  \* 1.2 重写Runnable的run方法  \*/  @Override  **public** **void** run() {  // 1.4 调用通过隧道的方法  cross();  }    /\*  \* 1.3 定义一个同步方法，模拟每个人通过隧道需要5秒钟  \*/  **public** **synchronized** **void** cross(){  // 1.3.1 子线程睡眠5秒钟，模拟每个人通过隧道需要5秒钟  **try** {  Thread.*sleep*(5000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  // 1.3.2 改变通过的人次  crossNum++;  // 1.3.3 打印线程名称及其通过隧道的顺序，模拟人通过隧道及其顺序  System.***out***.println(Thread.*currentThread*().getName()+"已经通过隧道，TA是第"+crossNum+"通过的！");  }  } |

|  |
| --- |
| /\*  \* 2.定义一个测试类  \*/  **public** **class** TunnelDemo {  **public** **static** **void** main(String[] args) {  // 2.1 在main方法中创建一个隧道类对象  Tunnel tul = **new** Tunnel();    // 2.2 在main方法中，循环创建10个子线程对象，通过构造方法把隧道对象和线程名（作为人的姓名）传递进去，并开启子线程  **for** (**int** i = 1; i <= 10; i++) {  Thread t = **new** Thread(tul,"p"+i);  t.start();  }  }  } |

## 训练案例3

### 训练考核知识点

讲义2.2：线程同步。

### 训练描述

编写线程同步模拟应用程序:

1. 大气环境数据为:温度,湿度,风速；

2. 一个大气环境传感器测量环境数据需要5秒时间；

3. 一个计算机读取传感器获得数据需要0.01秒时间；

4. 模拟100个计算机读取大气环境传感器获取的随机温度,湿度,风速；

### 操作步骤答案

|  |
| --- |
| /\*  \* 1.定义一个传感器类,实现Runnable接口  \*/  **public** **class** Sensor **implements** Runnable {  Object obj = **new** Object();  /\*  \* 1.1 重写Runnable的run方法  \*/  @Override  **public** **void** run() {    **synchronized** (obj) {  // 1.2 子线程睡眠5秒钟,模拟传感器获得温度,湿度,风速需要的5秒钟时间  **try** {  Thread.*sleep*(5000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }    // 1.3 使用Random模拟得到随机的温度,湿度和风速  Random r = **new** Random();  // 温度  **int** temperature = r.nextInt(50)+1;  // 湿度  **int** humidity = r.nextInt(100)+1;  // 风速  **int** windScale = r.nextInt(40)+1;    // 1.4 子线程睡眠0.01秒,模拟计算机读取传感器的数据需要的0.01秒时间  **try** {  Thread.*sleep*(10);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  // 1.5 打印温度,湿度和风速, 模拟计算机读取传感器的数据  System.***out***.println(Thread.*currentThread*().getName()+"-当前温度："+temperature+"度");  System.***out***.println(Thread.*currentThread*().getName()+"-当前湿度："+humidity+"%");  System.***out***.println(Thread.*currentThread*().getName()+"-当前风速："+windScale+"米/秒");  System.***out***.println("--------------------------");  }  }  } |

|  |
| --- |
| /\*  \* 2.定义传感器的测试类  \*/  **public** **class** SensorDemo {  **public** **static** **void** main(String[] args) {  // 2.1 在main方法中创建传感器类的对象  Sensor s = **new** Sensor();  // 2.2 在main方法中创建100个子线程对象,并把传感器对象传递给构造方法,模拟100个计算机  **for**(**int** x=1;x<=100;x++){  Thread t = **new** Thread(s);  // 2.3 开启子线程,开启读取传感器的获取的大气环境数据  t.start();  }  }  } |

# 关卡3

## 训练案例1

### 训练考核知识点

讲义2.2：线程同步。

### 训练描述

编写多线程同步程序，模拟3个人排除买票，张某、李某和赵某买电影票，售票员只有3张五元的钱，电影票5元一张。张某拿20元一张的RMB排在李某的前面，李某排在赵某的前面拿一张10元的RMB买票，赵某拿一张5元的RMB买票。

**提示：**

1. Thread类的wait()方法：是指让当前线程处于等待状态，让释放线程锁对象，让其它线程能够获得锁对象，直到其它线程调用notify()或者notifyAll()方法唤醒后才能继续执行。
2. Thread类的notifyAll()方法：是指唤醒所有正在等待锁对象的线程。

### 操作步骤答案

|  |
| --- |
| /\*  \* 1.定义一个购票人的类  \*/  **public** **class** TicketBuyer **extends** Thread {  // 1.1 定义三个变量，分别用来记录是售票员手中面值5,10元，20元RMB的张数  **int** money5 = 3, money10 = 0, money20 = 0;  // 1.2 定义一个变量用来记录购票人手中一张RMB的面值  **int** money;  // 1.3 定义一个构造方法，把购票人手中的人民币作为参数传递进去  **public** TicketBuyer(**int** money) {  **this**.money = money;  }  /\*  \* 1.4 重写Thread类的run方法  \*/  @Override  **public** **void** run() {  // 1.6 调用买票找零的规则方法，购票  rule(money);  }  /\*  \* 1.5 定义售票找零的方法  \*/  **public** **synchronized** **void** rule(**int** money) {  // 1.5.1 如果购票人给的RMB面值是5元的  **if** (money == 5) {  // 1.5.2 售票员手中的面值为5元的RMB应该增加一张  money5++;  // 1.5.3 购票成功  System.***out***.println("给你入场券，你的钱正好。");  } **else** **if** (money == 20) {  // 1.5.4 如果购票人给的RMB面值是20元的,但是售票员手中5元的RMB不够3张，不能找给购票人，  **while** (money5 < 3) {  **try** {  // 1.5.5 没有零钱，就等待 ，让其它人买票  wait();  } **catch** (InterruptedException e) {  }  }  // 1.5.6 如果售票员手中5元的RMB大于或者等于3张，就应该找回3张5元的RMB,5元的RMB张数应该减3  money5 = money5 - 3;  // 1.5.7 收到一张20元的RMB,20元的RMB张数应该加1  money20++;  // 打印售票成功的信息  System.***out***.println("给你入场券，你给我20元，找你15元。");  } **else** **if** (money == 10) {  // 1.5.8 如果购票人给的一张10元的  **while** (money5 < 1) {  // 1.5.9 但是售票员手中没有5元的RMB可以找给购票人，就等待，让其它人买票  **try** {  // 没有零钱，等待  wait();  } **catch** (InterruptedException e) {  }  }  // 1.5.10 如果售票员手中有5元的RMB，就找给购票人，5元的RMB张数应该减1  money5 = money5 - 1;  // 1.5.11 售票员收到了一张10元的，10元的RMB张数应该加1  money10++;  // 打印售票成功的信息  System.***out***.println("给你入场券，你给我10元，找你5元。");  }  // 1.5.12 唤醒所有正在等待的线程，通知所有刚才没有零钱可以找回的购票可以买票了  notifyAll();  }  } |

|  |
| --- |
| /\*  \* 2.定义一个购票的测试类  \*/  **public** **class** TicketDemo {  **public** **static** **void** main(String[] args) {  // 2.1 在main方法中创建三个购票者子线程对象  TicketBuyer t1 = **new** TicketBuyer(20);  TicketBuyer t2 = **new** TicketBuyer(10);  TicketBuyer t3 = **new** TicketBuyer(5);    // 2.2 开启三个子线程，开始购票  t1.start();  t2.start();  t3.start();  }  } |

## 训练案例2

### 训练考核知识点

讲义2.2：线程同步。

### 训练描述

请按如下要求编写多线程程序：

1. 创建两个类，一个是测试类，一个是Thread的子类；

2. 在测试类中，创建两个Thread类的子类对象，将其中的一个线程对象的优先级

设置10，另一个线程对象的优先级设置为6。

3. 让优先级为10的线程打印5次“线程1正在运行”，让优先级为6的线程打印

10次“线程2正在运行”；

**提示：**

1. 设置线程优先级的方法为setPriority方法；
2. 得到线程优先级的方法为getPriority方法；

### 训练步骤答案

|  |
| --- |
| /\*  \* 1.定义一个子线程类  \*/  **public** **class** PrintRunnable **implements** Runnable {  // 1.1 创建一个对象，用作锁对象  Object obj = **new** Object();  /\*  \* 1.2 重写Thread类的run方法  \*/  **public** **void** run() {  // 1.3 使用同步代码块，保证每个线程能够按照要求连续打印语句  **synchronized** (obj){  // 1.4 获得当前线程的优先级  **int** priority = Thread.*currentThread*().getPriority();  // 1.5 如果线程的优先级为10,就打印5次“线程1正在运行”  **if**(priority == 10){  **for**(**int** i=1;i<=5;i++){  System.***out***.println(Thread.*currentThread*().getName()+"正在运行");  }  }**else** **if**(priority == 6){  // 1.6 如果线程的优先级为6,就打印10次“线程2正在运行”  **for**(**int** i=1;i<=10;i++){  System.***out***.println(Thread.*currentThread*().getName()+"正在运行");  }  }  }  }  } |

|  |
| --- |
| /\*  \* 2.创建测试类  \*/  **public** **class** ThreadDemo {  **public** **static** **void** main(String[] args) {  // 2.1 在main方法中创建线程任务类的对象  PrintRunnable pr = **new** PrintRunnable();  // 2.2 创建两个线程对象  Thread t1 = **new** Thread(pr,"线程1");  Thread t2 = **new** Thread(pr,"线程2");  // 2.3 调用setPriority方法给线程对象设置优先级  t1.setPriority(10);  t2.setPriority(6);  // 2.4 开启线程  t1.start();  t2.start();  }  } |