**每日作业卷**

**JAVA基础第13天**

**多线程**

# 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。

### 操作步骤描述

请参考讲义1.1：多线程介绍。

正在运行的程序。确切的来说，当一个程序进入内存运行，即是一个或多个进程在运行，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。

### 操作步骤描述

请参考讲义1.1：多线程介绍。

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。

## 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。

### 操作步骤描述

请参考讲义1.1：多线程介绍。

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。

### 操作步骤描述

请参考讲义1.2：多线程程序运行的原理。

同一时刻CPU只能执行一个线程

## 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。

### 操作步骤描述

请参考讲义1.3：主线程。

当一个程序启动时，就有一个进程被操作系统创建，与此同时一个线程也立刻运行，该线程通常叫做程序的主线程。

## 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。

### 操作步骤描述

参考讲义1.4 ：Thread类。

public void start() 使该线程开始执行

public void run() 线程要执行的业务逻辑方法，相当于该线程的"main方法"

## 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。

### 操作步骤描述

参考讲义1.4 ：Thread类。

继承Thread：

1、指定线程执行目标：定义一个Thread类的子类，重写run方法，将相关逻辑实现

public void run() 线程要执行的业务逻辑方法，相当于该线程的"main方法"

2、创建自定义的线程子类对象

3、开启线程动作public void start() 使该线程开始执行

实现Runnable接口：

1、指定线程执行目标：定义Runnable线程执行目标实现类，重写run方法，指定目标逻辑

2、通过指定线程执行目标的构造方法创建线程对象

public Thread(Runnable target)

a)创建线程执行目标对象

b)通过线程执行目标创建线程对象

3、开启线程动作 (start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；

### 操作步骤描述

* 第一种方式：继承Thread类

1.定义一个子线程的类，继承Thread类；

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；

3.定义一个测试类；

4.在main方法中打印主线程的名称；

5.在main方法中创建子线程对象；

6.调用子线程对象的start方法，开启子线程；

**public** **class** MyThread **extends** Thread {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

System.***out***.println(**this**.getName()+"在运行");

}

}

**public** **class** MyThreadDemo {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

MyThread myThread = **new** MyThread();

myThread.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；

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；

3.定义一个测试类；

4.在main方法中打印主线程的名称；

5.在main方法中创建一个子任务对象；

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；

7.调用Thread类对象的start方法开启子线程；

**public** **class** MyRunnable **implements** Runnable {

@Override

**public** **void** run() {

Thread thread = Thread.*currentThread*();

System.***out***.println(thread.getName() + "在运行");

}

}

**public** **class** MyRunnableDemo {

**public** **static** **void** main(String[] args) {

MyRunnable myRunnable = **new** MyRunnable();

Thread thread = **new** Thread(myRunnable);

thread.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。

### 操作步骤描述

请参考讲义1.6.2：实现Runnable的好处。

实现Runnable接口避免了单继承的局限性

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。

## 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。

### 操作步骤描述

请参考讲义2.1：线程安全。

当多个线程操作共享的数据时，共同修改该数据。可能出现，当第1个线程想操作A状态的数据时，发现已经被第二个线程修改为了B状态，于是无法完成本想完成的任务。

## 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。

### 操作步骤描述

1.定义一个子任务类，实现Runnable接口，用来模拟电影票。

1.1 在子任务类中定义一个变量，用来表示总票数量；

1.2 在子任务类中，重写Runnable接口的run方法；

1.3在run方法中，模拟卖票，使用同步代码块或者同步方法保证同一时间只能有一个线程（窗口）在卖票，每卖一张票，就让票数量减一；

2.定义一个测试：

1.1在测试类的main方法中，创建一个票对象，三个子线程对象，创建三个子线程时，把票对象和窗口名作为参数传递给线程的构造方法；

1.2 开启三个子线程；

**public** **class** MyRunnable **implements** Runnable {

**private** **int** ticket = 100;

**private** Object object = **new** Object();

@Override

**public** **void** run() {

**while** (**true**) {

**synchronized** (object) {

Thread thread = Thread.*currentThread*();

**if** (ticket > 0) {

System.***out***.println(thread.getName() + "正在卖第" + (ticket--) + "张票");

} **else** {

System.***out***.println(thread.getName() + "已售完");

**break**;

}

}

}

}

}

**public** **class** MyRunnableDemo {

**public** **static** **void** main(String[] args) {

MyRunnable ticket = **new** MyRunnable();

Thread thread1 = **new** Thread(ticket, "窗口1");

Thread thread2 = **new** Thread(ticket, "窗口2");

Thread thread3 = **new** Thread(ticket, "窗口3");

thread1.start();

thread2.start();

thread3.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。

### 训练步骤描述

请参考讲义：2.4线程状态。

新建状态，可运行状态，运行状态，阻塞状态，等待状态，死亡状态

新建状态通过start()方法变为可运行状态

可运行状态线程获得CPU后进入运行状态

运行状态线程失去CPU后进入可运行状态，等待同步锁，调用阻塞IO方法后进入阻塞状态，调用wait(),join(),sleep()方法后进入等待状态，线程run()完成，或者出现异常时，进入死亡状态

阻塞状态线程获得同步锁，阻塞IO方法返回时，进入可运行状态

等待转台的线程调用notify()方法唤醒，join()线程终止，sleep()时间到后进入可运行状态

# 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；

### 操作步骤描述

1.定义一个子任务类，实现Runnable接口：

1.1 定义一个ArrayList集合对象，用于存放子线程的名称；

1.2 定义一个变量，用来记录正在运行的线程个数；

1.3定义一个构造方法，把进程总数作为参数进来；

1.4 在构造方法中，给正在运行的线程个数赋值为线程总数；

1.5 重写Runnable接口的run方法 ；

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；

1.7 正在运行的线程个数减1；

1.8 所有的线程运行结束后遍历集合，打印所有线程的名称；

2.定义一个测试类：

2.1 在main方法中使用Random得到一个1到10的随机数；

2.2 调用子任务类的构造方法创建子任务对象，把随机数传递给构造方法；

2.3 循环创建n的子线程对象，并开启子线程；

**public** **class** MyRunnable1 **implements** Runnable {

**public** ArrayList<String> name = **new** ArrayList<String>();

**private** **int** count;

**private** **static** Object *object* = **new** Object();

@Override

**public** **void** run() {

**synchronized** (*object*) {

Thread thread = Thread.*currentThread*();

name.add(thread.getName());

count--;

}

**if** (count <= 0) {

**for** (String string : name) {

System.***out***.println(string);

}

}

}

**public** MyRunnable1(**int** total) {

**this**.count = total;

}

}

**public** **class** MyRunnableDemo {

**public** **static** **void** main(String[] args) {

Random random = **new** Random();

**int** count = random.nextInt(10) + 1;

System.***out***.println(count);

MyRunnable1 myRunnable1 = **new** MyRunnable1(count);

**for** (**int** i = 0; i < count; i++) {

Thread thread = **new** Thread(myRunnable1, "线程" + (i + 1));

thread.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；

### 操作步骤描述

1.定义一个隧道类，实现Runnable接口：

1.1 定义一个变量，用来记录通过隧道的人数；

1.2 重写Runnable的run方法；

1.3 定义一个同步方法，模拟每个人通过隧道需要5秒钟：

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；

1.3.2 改变通过的人次；

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；

1.4 调用通过隧道的方法；

2.定义一个测试类：

2.1 在main方法中创建一个隧道类对象；

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；

**public** **class** Tunnel **implements** Runnable {

**private** **int** count = 0;

@Override

**public** **void** run() {

**try** {

throughTunnel();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**public** **synchronized** **void** throughTunnel() **throws** InterruptedException {

Thread.*sleep*(5000);

Thread thread = Thread.*currentThread*();

System.***out***.println(thread.getName() + "通过隧道,已过" + (++count) + "人");

}

}

**public** **class** TunnelDemo {

**public** **static** **void** main(String[] args) {

String[] strings = { "张三", "李四", "王五", "张三1", "张三2", "张三3", "张三4", "张三5", "张三6", "张三7" };

Tunnel tunnel = **new** Tunnel();

**for** (**int** i = 0; i < 10; i++) {

Thread thread = **new** Thread(tunnel, strings[i]);

thread.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；

### 操作步骤描述

1.定义一个传感器类,实现Runnable接口：

1.1 重写Runnable的run方法；

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；

1.3 使用Random模拟得到随机的温度,湿度和风速；

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；

1.5 打印温度,湿度和风速, 模拟计算机读取传感器的数据；

2.定义一个传感器的测试类：

2.1 在main方法中创建传感器类的对象；

2.2 在main方法中循环创建100个子线程对象,并把传感器对象传递给构造方法；

2.3 每创建一个子线程，就开启，读取传感器的获取的大气环境数据；

**public** **class** Sensors **implements** Runnable {

**private** **static** Object *object* = **new** Object();

@Override

**public** **void** run() {

**synchronized** (*object*) {

Random random = **new** Random();

System.***out***.println("获取温度中...");

**try** {

Thread.*sleep*(1666);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**int** temperature = random.nextInt(30);

System.***out***.println("获取湿度中...");

**try** {

Thread.*sleep*(1666);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**int** humidity = random.nextInt(70);

System.***out***.println("获取风速中...");

**try** {

Thread.*sleep*(1668);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**int** windSpeed = random.nextInt(221);

System.***out***.println("读取数据中...");

**try** {

Thread.*sleep*(10);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("温度：" + temperature + "℃ ,湿度：" + humidity + "% ,风速：" + windSpeed + "m/s");

}

}

}

**public** **class** SensorsDemo {

**public** **static** **void** main(String[] args) {

Sensors sensors = **new** Sensors();

**for** (**int** i = 0; i < 100; i++) {

Thread thread = **new** Thread(sensors);

thread.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。

### 操作步骤描述

1. 定义一个购票的类：

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；

1.2 定义一个变量用来记录购票人手中一张RMB的面值；

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；

1.4 重写Thread类的run方法；

1.5 定义售票找零的方法：

1.5.1 如果购票人给的RMB面值是5元的，卖票成功，打印卖票的信息；

1.5.2 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，没有零钱，就调用wait方法等待 ，让其它人买票；

1.5.3 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，收到一张20元的RMB,20元的RMB张

数应该加1，卖票成功，打印卖票的信息；

1.5.4 如果购票人给的一张10元的，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；

1.5.5 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，售票员收到了一张10元的，10元的RMB张数应该加1，打印售票

成功的信息；

1.5.6最后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；

1.6 在run方法中，调用买票找零的规则方法，售票；

2.定义一个测试类：

2.1 在main方法中创建三个购票类的对象；

2.2 分别调用start方法，开启子线程，开始购票；

**public** **class** BuyTicket **extends** Thread {

**private** **static** **int** *fiveCount* = 3;

**private** **static** **int** *tenCount* = 0;

**private** **static** **int** *twentyCount* = 0;

**private** String name;

**private** **int** count = 0;

@Override

**public** **void** run() {

**try** {

getchange(count);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**public** BuyTicket(String name, **int** count) {

**this**.name = name;

**this**.count = count;

}

**public** **synchronized** **void** getchange(**int** count) **throws** InterruptedException {

**if** (count == 5) {

*fiveCount* += 1;

System.***out***.println(name + "购票成功");

System.***out***.println("实付：5元 ，找零：0元");

} **else** **if** (count == 20) {

**if** (*fiveCount* >= 3) {

*fiveCount* -= 3;

*twentyCount* += 1;

System.***out***.println(name + "购票成功");

System.***out***.println("实付：20元 ，找零：15元");

} **else** **if** (*fiveCount* < 3 && *fiveCount* > 0 && *tenCount* > 0) {

*fiveCount* -= 1;

*tenCount* -= 1;

*twentyCount* += 1;

System.***out***.println(name + "购票成功");

System.***out***.println("实付：20元 ，找零：15元");

} **else** {

wait();

}

} **else** **if** (count == 10) {

**if** (*fiveCount* > 0) {

*tenCount* += 1;

*fiveCount* -= 1;

System.***out***.println(name + "购票成功");

System.***out***.println("实付：10元 ，找零：5元");

} **else** {

wait();

}

}

System.***out***.println("零钱：5元" + *fiveCount* + "张，10元" + *tenCount* + "张，20元" + *twentyCount* + "张");

notify();

}

}

**public** **class** BuyTicketDemo {

**public** **static** **void** main(String[] args) {

BuyTicket consumer1 = **new** BuyTicket("张三", 20);

BuyTicket consumer2 = **new** BuyTicket("李四", 10);

BuyTicket consumer3 = **new** BuyTicket("赵武", 5);

consumer1.start();

consumer2.start();

consumer3.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；

### 训练步骤描述

1. 定义一个子线程类：

1.1 创建一个对象，用作锁对象；

1.2 重写Thread类的run方法；

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；

1.4 调用getPriority方法获得当前线程的优先级；

1.5 如果线程的优先级为10,就打印5次“线程1正在运行”；

1.6 如果线程的优先级为6,就打印10次“线程2正在运行”；

2. 定义测试类：

2.1 在main方法中创建线程任务类的对象；

2.2 创建两个线程对象；

2.3 调用setPriority方法给线程对象设置优先级；

2.4 开启线程；

**public** **class** MyThread **extends** Thread {

**private** Object object = **new** Object();

**private** String name;

**public** MyThread(String name) {

**this**.name = name;

}

@Override

**public** **void** run() {

**synchronized** (object) {

**int** priority = **this**.getPriority();

**if** (priority == 10) {

**for** (**int** i = 0; i < 5; i++) {

System.***out***.println(name + "正在运行");

}

} **else** **if** (priority == 6) {

**for** (**int** i = 0; i < 10; i++) {

System.***out***.println(name + "正在运行");

}

}

}

}

}

**public** **class** MyThreadDemo {

**public** **static** **void** main(String[] args) {

MyThread thread1 = **new** MyThread("线程1");

thread1.setPriority(10);

MyThread thread2 = **new** MyThread("线程2");

thread2.setPriority(6);

thread1.start();

thread2.start();

}

}