目标：

Threads, 共 37 个slide(373-410);

知识点：

一. 什么是线程：

进程是指运行中的应用程序，每一个进程都有自己独立的内存空间。一个应用程序可以同时启动多个进程。例如每打开一个IE浏览器窗口，就启动了一个新的进程。同样，每次执行JDK的java.exe程序，就启动了一个独立的Java虚拟机进程，该进程的任务是解析并执行Java程序代码。

线程是指进程中的一个执行流程。一个进程可以由多个线程组件。即在一个进程中可以同时运行多个不同的线程，它们分别执行不同的任务，当进程内的多个线程同时运行时，这种运行方式称为**并发运行**。

线程与进程的主要区别在于：每个进程都需要操作系统为其分配独立的内存地址空间，而同一进程中的所有线程在同一块地址空间中工作，这些线程可以共享同一块内存和系统资源。比如共享一个对象或者共享已经打开的一个文件。

二. 主线程

在java虚拟机进程中，执行程序代码的任务是由线程来完成的。每当用java命令启动一个Java虚拟机进程时，Java虚拟机都会创建一个主线程。该线程从程序入口main()方法开始执行。

计算机中机器指令的真正执行者是CPU，线程必须获得CPU的使用权，才能执行一条指令。

三. 线程的创建和启动

前面我们提到Java虚拟机的主线程，它从启动类的main()方法开始运行。此外，用户还可以创建自己的线程，它将和主线程并发运行。创建线程有两种方式，如下：

getName():返回当前调用的线程的名称
名称默认Thread-0 表示第一个线程
0这个数字以此类推,表示多个线程
. 继承java.lang.Thread类;
. 实现Runnable接口;

1. 扩展java.lang.Thread类

Thread类代表线程类，它的最主要的两个方法是：

. **run**()——包含线程运行时所执行的代码；
. **start**()——用于启动线程；

public void run(); //没有抛异常，所以子类重写亦不能抛异常

1) 主线程与用户自定义的线程并发运行

a. Thread类的run()方法是专门被自身的线程执行的，主线程调用Thread类的run()方法，违背了Thread类提供run()方法的初衷；
b. Thread thread = Thread.currentThread(); 返回当前正在执行这行代码的线程引用；
String name = thread.getName(); 获得线程名字；

每个线程都有默认名字，主线程默认的名字为main, 用户创建的第一个线程的默认名字为"Thread-0"， 第二个线程的默认名字为"Thread-1"， 依引类推。Thread类的**setName**()方法可以显示地设置线程的名字；

2) 多个线程共享同一个对象的静态变量
多个线程各自操作自己的实例变量
继承Thread只有操作静态变量时才是共享数据,实例变量不是共享数据
实现Runnable操作实例变量就是共享数据(只创建一个runnable对象)
多线程多用实现Runnable,少用继承Thread
3) 不要随便覆盖Thread类的start()方法

创建了一个线程对象，线程并不自动开始运行，必须调用它的start()方法。对于以下代码：

```
Machine machine = new Machine();
machine.start();
```

4) 一个线程只能被启动一次

```
Machine machine = new Machine();
machine.start();
machine.start();    //抛出IllegalThreadStateException异常 运行时异常
```

5)主线程也有可能在子线程结束之前结束。并且子线程不受影响，不会因为主线程的结束而结束。

2. 实现Runnable接口
相同条件下,一般使用Runnable来实现多线程的构建 java支持单继承多实现.
Java不允许一个类继承多个类，因此一旦一个类继承了Thread类，就不能再继承其他的类。为了解决这一问题，Java提供了java.lang.Runnable接口，它有一个run()方法，定义如下：

public void run();

启动：Thread(Runnable runnable) //当线程启动时，将执行参数runnable所引用对象的run()方法；

1. 王政泽
2020年8月15日 上午11:49:56
就是睡眠的线程

2. 王政泽
2020年8月15日 上午11:50:09
解除睡眠后等待获取锁定

四. 线程状态

线程在它的生命周期中会处于各种不同的状态；

1. 新建状态(New)

用new语句创建的线程对象处于新建状态， 此时它和其他Java对象一样；仅在堆区中被分配了内存；

2. 就绪状态(Runnable)

当一个线程对象创建后，其他线程调用它的start()方法， 该线程就进入就绪状态，Java虚拟机会为它创建方法调用栈。处于这个状态的线程位于可运行池中， 等待获得CPU的使用权。

3. 运行状态(Running)

处于这个状态的线程占用CPU，执行程序代码。在并发运行环境中， 如果计算机只有一个CPU, 那么任何时刻只会有一个线程处于这个状态。如果计算机有多个CPU, 那么同一时刻可以让几个线程占用不同的CPU，使它们都处于运行状态。只有处于就绪状态的线程才有机会转到运行状态。

4. 阻塞状态(Blocked)

指线程因为某些原因放弃CPU， 暂时停止运行。当线程处于阻塞状态时，Java虚拟机不会给线程分配CPU，直到线程重新进入就绪状态，它才有机会转到运行状态。

阻塞状态可分为三种：

1. 位于对象等待池中的阻塞状态(Blocked in objects' wait pool): 运行状态时，执行某个对象的**wait**()方法；
2. 位于对象锁池中的阻塞状态(Blocked in object's lock pool): 当线程处于运行状态，试图获得某个对象的同步锁时，如该对象的同步锁已经被其他线程占用，Java虚拟机就会把这个线程放到这个对象的锁池中；
. 其他阻塞状态(Otherwise Blocked): 当前线程执行了**sleep**()方法，或者调用了其他线程的join()方法，或者发出了I/O请求时，就会进入这个状态。

当一个线程执行System.out.println()或者System.in.read()方法时，就会发出一个I/O请求，该线程放弃cpu, 进入阻塞状态，直到I/O处理完毕，该线程才会恢复运行。

例如：ch13.MachineOfBlockStatus.java

5. 死亡状态(Dead)

当线程退出run()方法时，就进入死亡状态，该线程结束生命周期。线程有可能是正常执行完run()方法退出，也有可能是遇到异常而退出。不管线程正常结束还是异常结束，都不会对其他线程造成影响。

例如：ch13.MachineOfDeadStatus.java

五. 线程调度

计算机通常只有一个CPU, 在任意时刻只能执行一条机器指令，每个线程只有获得CPU的使用权才能执行指令。所谓多线程的并发运行，其实是指从宏观上看，各个线程轮流获得CPU的使用权，分别执行各自的任务。在可运行池中，会有多个处于就绪状态的线程在等待CPU，Java虚拟机的一项任务就是负责线程的调度。线程的调度是指按照特定的机制为多个线程分配CPU的使用权。有两种调度模型：

. 分时调度模型：让所有线程轮流获得CPU的使用权，并且平均分配每个线程占用CPU的时间片。

. 抢占式调度模型：优先让可运行池中优先级高的线程占用CPU，如果可运行池中线程的优先级相同，那么就随机选择一个线程，使其占用CPU。处于可运行状态的线程会一直运行，直至它不得不放弃CPU。Java虚拟机采用。

一个线程会因为以下原因而放弃CPU:

. Java虚拟机让当前线程暂时放弃CPU，转到就绪状态；
. 当前线程因为某些原因而进入阻塞状态；
. 线程运行结束；

线程的调度不是跨平台的，它不仅取决于Java虚拟机，还依赖于操作系统。在某些操作系统中，只要运行中的线程没有阻塞，就不会放弃CPU；在某些操作系统中，即使运行中的线程没有遇到阻塞，也会在运行一段时间后放弃CPU，给其他线程运行机会。

讲解：ch13.MachineOfAttemper.java

1. stop

Thread类的stop()方法可以强制终止一个线程，但从JDK1.2开始废弃了stop()方法。在实际编程中，**一般是在受控制的线程中定义一个标志变量，其他线程通过改变标志变量的值，来控制线程的自然终止、暂停及恢复运行。**

2. isAlive:

final boolean **isAlive**（）：判定某个线程是否是活着的(该线程如果处于可运行状态、运行状态和阻塞状态、对象等待队列和对象的锁池中返回true)

3. Thread.sleep(5000);

放弃CPU， 转到阻塞状态。当结束睡眠后，首先转到就绪状态，如有其它线程在运行，不一定运行，而是在可运行池中等待获得CPU。 线程在睡眠时如果被中断，就会收到一个InterrupedException异常，线程跳到异常处理代码块。

4. boolean otherThread.isInterrupted()：

测试某个线程是否被中断，与static boolean interrupt()不同，对它的调用不会改变该线程的“中断”状态。

5. static boolean Thread.interrupt()：

执行中断操作,它会将当前线程的“中断”状态改为false。

6. public void join();

挂起 谁调用,没被调用的被挂起(建议)
e.g a和b线程
a.join() b线程被挂起

例：ch13.JoinTest.java

六. 线程的同步

线程的职责就是执行一些操作，而多数操作都涉及到处理数据。这里有一个程序处理实例变量count:

```
if(count>0){
    try{
        Thread.sleep(1000);
    }catch(InterruptedException e){
        e.printStackTrace();
    }
    System.out.println(count--);
}
```

多个线程在操纵共享资源——实例变量时，有可能引起共享资源的竞争。为了保证每个线程能正常执行操作，保证共享资源能正常访问和修改。Java引入了同步进制，具体做法是在有可能引起共享资源竞争的代码前加上synchronized标记。这样的代码被称为同步代码块。

【同步代码块】：

语法格式：

```
synchronized（同步对象）{
        //需要同步的代码
}
```

但是一般都把当前对象this作为同步对象或者是当前对象的镜像(类.class)

. 如果这个锁已经被其他线程占用，Java虚拟机就会把这个消费者线程放到this指定对象的锁池中，线程进入阻塞状态。在对象的锁池中可能会有许多等待锁的线程。等到其他线程释放了锁，Java虚拟机会从锁池中随机取出一个线程，使这个线程拥有锁，并且转到就绪状态。

. 假如这个锁没有被其他线程占用，线程就会获得这把锁，开始执行同步代码块。在一般情况下，线程只有执行完同步代码块，才会释放锁，使得其他线程能够获得锁。

如果一个方法中的所有代码都属于同步代码，则可以直接在方法前用synchronized修饰。

【同步方法】

也可以采用同步方法。

语法格式为synchronized 方法返回类型 方法名（参数列表）{

// 其他代码

}

```
public synchronized String pop(){...}
        等价于
public String pop(){
    synchronized(this){...}
}
```

线程同步的特征：

1. 如果一个同步代码块和非同步代码块同时操纵共享资源，仍然会造成对共享资源的竞争。
因为当一个线程执行一个对象的同步代码块时，其他线程仍然可以执行对象的非同步代码块。
2. 每个对象都有唯一的同步锁。
3. 在静态方法前面也可以使用synchronized修饰符。此时该同步锁的对象为类对象。
4. 当一个线程开始执行同步代码块时，并不意味着必须以不中断的方式运行。进入同步代码块的线程也可以执行Thread.sleep()或者执行Thread.yield()方法，此时它并没有释放锁，只是把运行机会(即CPU)让给了其他的线程。
5. synchnozied声明不会被继承。

同步是解决共享资源竞争的有效手段。当一个线程已经在操纵共享资源时，其他共享线程只能等待。为了提升并发性能，应该使同步步代码块中包含尽可能少的操作，使得一个线程能尽快释放锁，减少其他线程等待锁的时间。

课堂练习：ch13.Account.java

七. 线程的通信

. Object.wait(): 执行该方法的线程释放对象的锁，Java虚拟机把该线程放到该对象的等待池中。该线程等待其它线程将它唤醒；
. Object.notify(): 执行该方法的线程唤醒在对象的等待池中等待的一个线程。Java虚拟机从对象的等待池中随机选择一个线程，把它转到对象的锁池中。如果对象的等待池中没有任何线程，那么notify()方法什么也不做。
. Object.notifyAll()：会把对象的等待池中的所有线程都转到对象的锁池中。

假如t1线程和t2线程共同操纵一个s对象，这两个线程可以通过s对象的wait()和notify()方法来进行通信。通信流程如下：

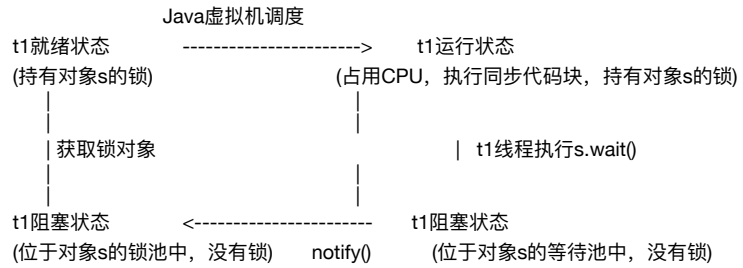
1. 当t1线程执行对象s的一个同步代码块时，t1线程持有对象s的锁，t2线程在对象s的锁池中等待；

2. t1线程在同步代码块中执行s.wait()方法, t1释放对象s的锁，进入对象s的等待池；

3. 在对象s的锁池中等待锁的t2线程获得了对象s的锁，执行对象s的另一个同步代码块；

4. t2线程在同步代码块中执行s.notify()方法，Java虚拟机把t1线程从对象s的等待池移到对象s的锁池中，在那里等待获得锁。

5. t2线程执行完同步代码块，释放锁。t1线程获得锁，继续执行同步代码块。



课堂练习：男孩赚钱、女孩花钱

八. 线程的死锁

当多个线程共享一个资源的时候需要进行同步，但是过多的同步可能导致死锁

A线程等待B线程持有的锁，而B线程正在等待A持有的锁；

```
/**
* 一个简单的死锁类
* 当类的对象flag=1时（T1 第一个用户），先锁定O1(筷子),睡眠500毫秒，然后锁定O2(碗)；
* 而T1在睡眠的时候另一个flag=0的对象（T2 第二个用户）线程启动，先锁定O2,睡眠500毫秒，等待T1释放O1；
* T1睡眠结束后需要锁定O2才能继续执行，而此时O2已被T2锁定；
* T2睡眠结束后需要锁定O1才能继续执行，而此时O1已被T1锁定；
* T1、T2相互等待，都需要对方锁定的资源才能继续执行，从而死锁。
*/
```

因此避免死锁的一个通用的经验法则是:当几个线程都要访问共享资源A、B时，保证使每个线程都按照同样的顺序去访问它们，比如都先访问A，在访问B

九. 线程让步

Thread.yield()静态方法，如果此时具有相同优先级的其他线程处于就绪状态，暂停自己，运行他人。如果没有相同优先级的可运行线程，则yield()方法什么也不做。

sleep()和yield()方法都是Thread类的静态方法，都会使当前处于运行状态的线程放弃CPU，把运行机会让给别的线程。区别：

. sleep()不考虑其他线程优先级；
  yield()只会给相同优先级或者更高优先级的线程一个运行的机会。
. sleep()转到阻塞状态；
  yield()转到就绪状态；
. sleep()会抛出InterruptedException异常，
  yield()不抛任何异常
. sleep()比yield方法具有更好的可移植性。对于大多数程序员来说，yield()方法的唯一用途是在测试期间人为地提高程序的并发性能，以帮助发现一些隐藏的错误。

十. 调整线程优先级

所有处于就绪状态的线程根据优先级存放在可运行池中，优先级低的线程获得较少的运行机会，优先级高的线程获得较多的运行机会。Thread类的setPriority(int)和getPriority()方法分别用来设置优先级和读取优先级。优先级用整数来表示，取值范围是1-10，Thread类有以下3个静态常量。

. MAX_PRIORITY: 10, 最高;
. MIN_PRIORITY: 1, 最低;
. NORM_PRIORITY: 5, 默认优先级;

其它(过时)：stop(): 中止线程运行；
  resume(): 使暂停线程恢复运行
  suspend(): 暂停线程，不释放锁；
  destroy():销毁

释放对象的锁：

. 执行完同步代码块；
. 执行同步代码块过程中，遇到异常而导致线程终止，释放锁；
. 执行同步代码块过程中，执行了锁所属对象的wait()方法，释放锁进入对象的等待池；

线程不释放锁：

. Thread.sleep()方法，放弃CPU,进入阻塞状态；
. Thread.yield()方法，放弃CPU,进入就绪状态；
. suspend()方法，暂停当前线程，已废弃；

Review questions :

1.(Level 1)What are the differences between two ways to create threads?
  答：继承Thread符合面向对象思想；
  实现Runnable解决多重继承问题；
2.(Level 1)What are the five states for threads?

答：新建、就绪、运行、死亡、阻塞

3.(Level 1)How to solve the concurrent access problem for threads?

答：使用synchronized对有可能引了共享资源竞争的代码进行修饰，保证该段代码在同一时刻只有一个线程访问；

4.(Level 2)What does deadlock means?

答：A线程等待B线程释放锁，而B也正在等待A线程释放锁；

5.(Level 2)A thread wants to make a second thread ineligible for execution. To do this, the first thread can call the yield() method on the second thread?

A. true B.false

答：B

6.(Level 2)A java monitor must either extend Thread or implement Runnable.

A. true B.false

答：B? ?

7.(Level 3)A thread's run() method includes the following lines:

```
1.try{
2.sleep(100);
3.}catch(InterruptedException e){}
```

Assuming the thread is not interrupted, which statement is true?

A.?The code will not complied, because exception can not be caught in a thread's run() method

B. At line 2, the thread will stop running. Exception will resume in, at most, 100 milliseconds

C.?At line 2, the thread will stop running, Exception will resume in exactly 100 milliseconds.

D. At line 2, the thread will stop running, Execution will some time after 100 milliseconds have elapsed.

答：D