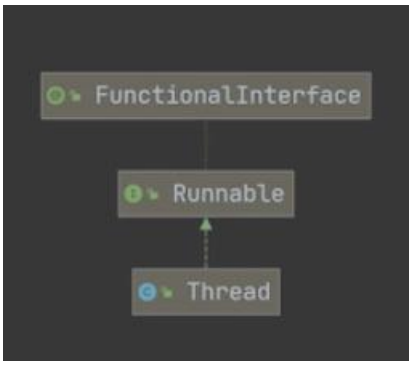
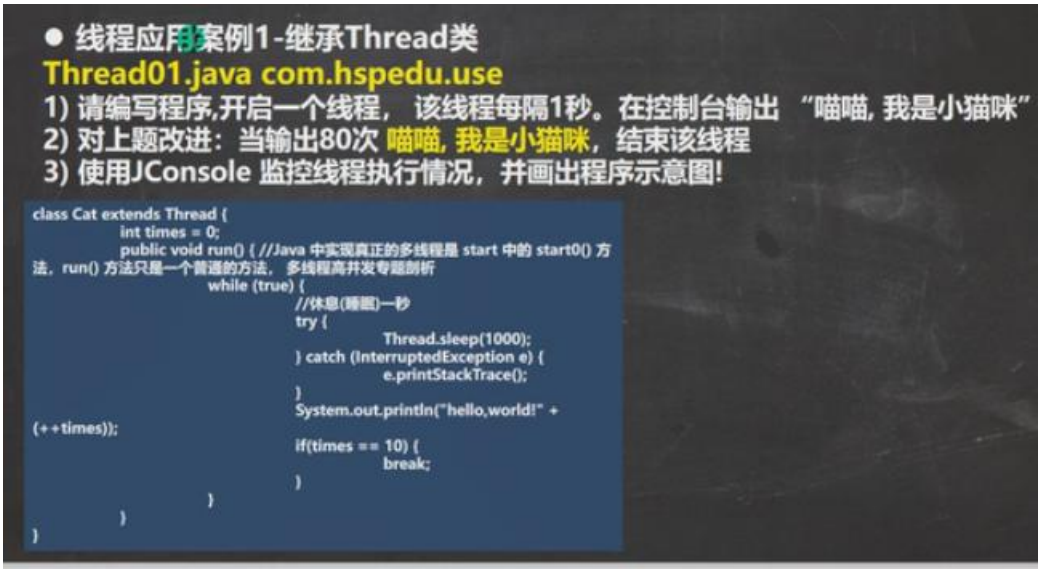
多线程基础

线程：
1. 线程由进程创建，是进程的一个实体；
2. 一个进程可以拥有多个线程
3. 单线程：同一个时刻，只允许执行一个线程
4. 多线程：同一个时刻，可以执行多个线程，比如：一个 QQ 进程，可以同时打开多个聊天窗口；一个迅雷进程，可以同时下载多个文件
5. 并发：同一时刻，多个任务交替执行，造成一种“貌似同时”的错觉，简而言之就是单核 CPU 实现的多任务就是并发
6. 并行：同一个时刻，多个任务同时执行，多核 CPU 可以实现并行

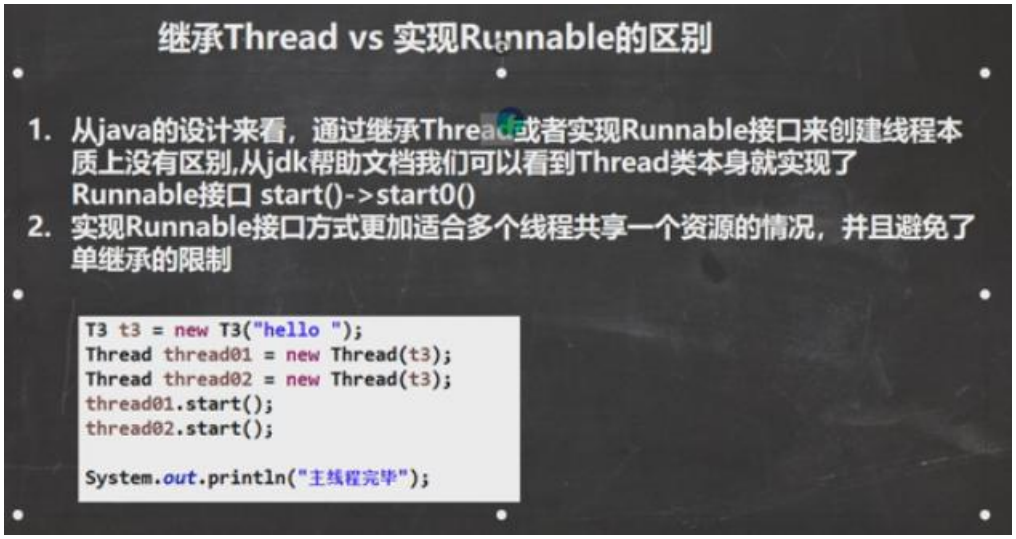
创建线程的两种方式：java 中线程使用的 2 种方式
1. 继承 Thread 类，重写 run 方法
2. 实现 Runnable 接口，重写 run 方法

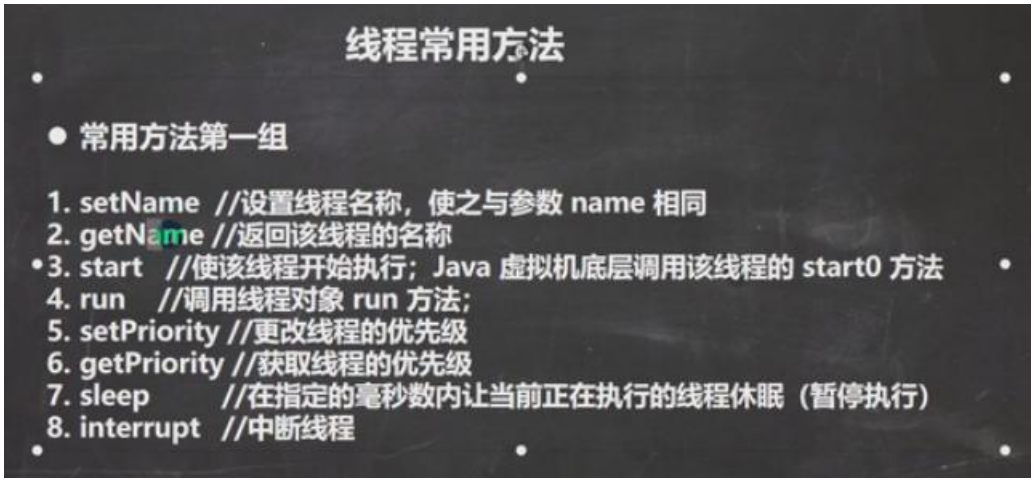




线程的基本使用 2---实现 Runnable 接口

1. Java 是单继承的，在某些情况下一个类可能已经继承了某个父类，这时再用继承 Thread 类的方法来创建线程是不太可能了
2. Java 设计者提供了通过实现 Runnable 接口来创建线程的方法





线程常用方法

- 注意事项和细节

1. start 底层会创建新的线程，调用run，run 就是一个简单的方法调用，不会启动新线程
2. 线程优先级的范围
3. interrupt，中断线程，但并没有真正的结束线程。所以一般用于中断正在休眠线程
4. sleep:线程的静态方法，使当前线程休眠

java.lang.Thread

线程常用方法

- 常用方法第二组

1. yield:线程的礼让。让出cpu，让其他线程执行，但礼让的时间不确定，所以也不一定礼让成功

2. join：线程的插队。插队的线程一旦插队成功，则肯定先执行完插入的线程所有的任务
案例：创建一个子线程，每隔1s 输出 hello, 输出 20 次, 主线程每隔1秒，输出 hi，输出 20次.要求: 两个线程同时执行，当主线程输出 5次后，就让子线程运行完毕，主线程再继续，

教育

Synchronized

- 同步具体方法-Synchronized

1. 同步代码块
```
synchronized (对象) {  // 得到对象的锁，才能操作同步代码
    // 需要被同步代码;
}
```
2. synchronized还可以放在方法声明中，表示整个方法-为同步方法

```
public synchronized void m (String name){
    //需要被同步的代码
}
```

3. 如何理解:
就好像 某小伙伴上厕所前先把门关上(上锁),完事后再出来(解锁),那么其它小伙伴就可在使用厕所了，如图:

4. 使用synchronized 解决售票问题

- 基本介绍

1. Java语言中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。
2. 每个对象都对应于一个可称为“互斥锁”的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。
3. 关键字synchronized 来与对象的互斥锁联系。当某个对象用synchronized修饰时，表明该对象在任一时刻只能由一个线程访问
4. 同步的局限性：导致程序的执行效率要降低
5. 同步方法（非静态的）的锁可以是this，也可以是其他对象(要求是同一个对象)
6. 同步方法（静态的）的锁为当前类本身。

互斥锁

- 注意事项和细节

1. 同步方法如果没有使用static修饰：默认锁对象为this
2. 如果方法使用static修饰，默认锁对象:当前类.class

3. 实现的落地步骤：
- 需要先分析上锁的代码
- 选择同步代码块或同步方法
- 要求多个线程的锁对象为同一个即可！

线程的死锁

韩顺平教育

- 基本介绍

多个线程都占用了对方的锁资源，但不肯相让，导致了死锁，在编程是一定要避免死锁的发生.

- 应用案例

妈妈：你先完成作业，才让你玩手机
小明：你先让我玩手机，我才完成作业.

- 下面操作会释放锁

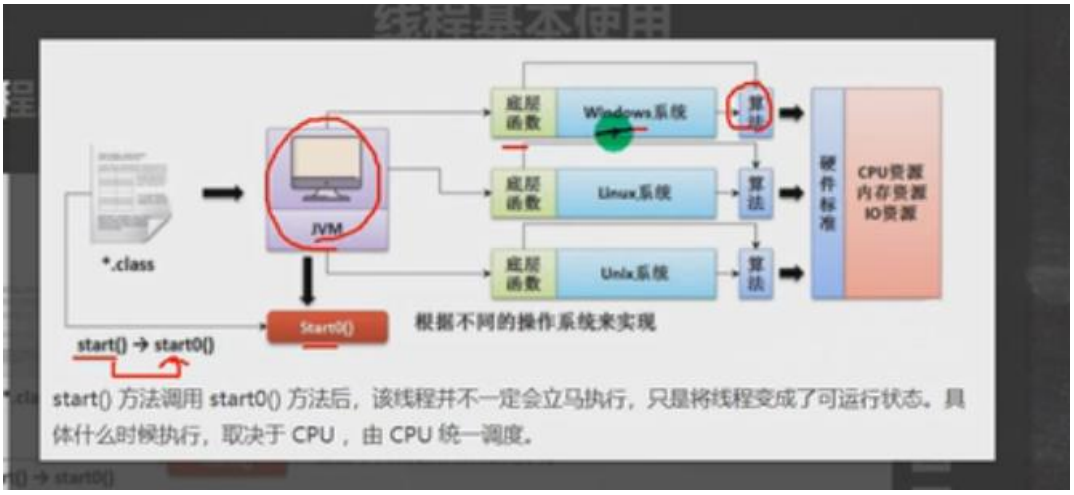
1. 当前线程的同步方法、同步代码块执行结束
   案例：上厕所，完事出来
2. 当前线程在同步代码块、同步方法中遇到break、return。
   案例：没有正常的完事，经理叫他修改bug，不得已出来

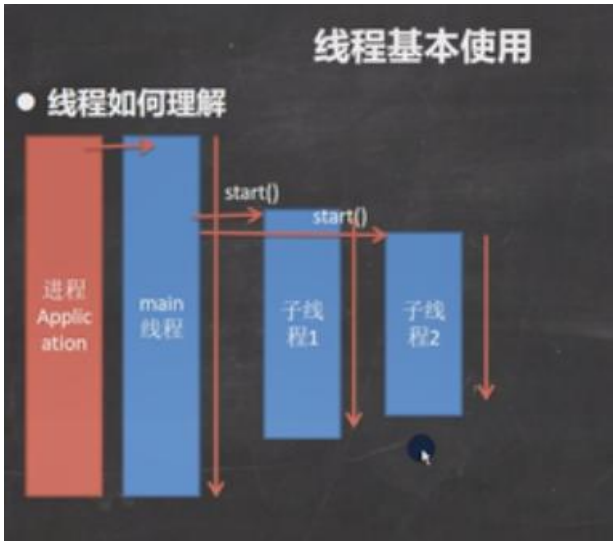
3. 当前线程在同步代码块、同步方法中出现了未处理的Error或Exception，导致异常结束
   案例：没有正常的完事，发现忘带纸，不得已出来
4. 当前线程在同步代码块、同步方法中执行了线程对象的wait()方法，当前线程暂停，并释放锁。
   案例：没有正常完事，觉得需要酝酿下，所以出来等会再进去

- 下面操作不会释放锁

1. 线程执行同步代码块或同步方法时，程序调用Thread.sleep()、Thread.yield()方法暂停当前线程的执行，不会释放锁
   案例：上厕所，太困了，在坑位上眯了一会

2. 线程执行同步代码块时，其他线程调用了该线程的suspend()方法将该线程挂起，该线程不会释放锁。
   提示：应尽量避免使用suspend()和resume()来控制线程， 方法不再推荐使用





继承Thread vs 实现Runnable的区别

1. 从java的设计来看，通过继承Thread或者实现Runnable接口来创建线程本质上没有区别,从jdk帮助文档我们可以看到Thread类本身就实现了Runnable接口
2. 实现Runnable接口方式更加适合多个线程共享一个资源的情况，并且避免了单继承的限制

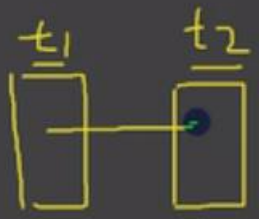
```
T3 t3 = new T3("hello ");
Thread thread01 = new Thread(t3);
Thread thread02 = new Thread(t3);
thread01.start();
thread02.start();

System.out.println("主线程完毕");
```

```
        //如果希望main线程去控制t1 线程的终止，必须可以修改 loop
        //让t1 退出run方法，从而终止 t1线程 -> 通知方式

        //让主线程休眠 10 秒，再通知 t1线程退出
        System.out.println("main线程休眠10s...");
        Thread.sleep(10 * 1000);
        t1.setLoop(false);
    }
}

class T extends Thread {
    private int count = 0;
    //设置一个控制变量
    private boolean loop = true;
    @Override
    public void run() {
```



Synchronized

- 同步具体方法-Synchronized

1. 同步代码块
   synchronized (对象) { // 得到对象的锁，才能操作同步代码
   // 需要被同步代码;
   }
2. synchronized还可以放在方法声明中，表示整个方法-为同步方法

```
public synchronized void m (String name){
        //需要被同步的代码
}
```

3. 如何理解:
   就好像 某小伙伴上厕所前先把门关上(上锁),完事后再出来(解锁),那么其它小伙伴就可在使用厕所了，如图:

4. 使用synchronized 解决售票问题

Synchronized

- 线程同步机制

1. 在多线程编程，一些敏感数据不允许被多个线程同时访问，此时就使用同步访问技术，保证数据在任何时刻，最多有一个线程访问，以保证数据的完整性。

2. 也可以这里理解：线程同步，即当有一个线程在对内存进行操作时，其他线程都不可以对这个内存地址进行操作，直到该线程完成操作， 其他线程才能对该内存地址进行操作.

互斥锁

- 基本介绍

1. Java在Java语言中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。
2. 每个对象都对应于一个可称为“互斥锁”的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。
3. 关键字synchronized 来与对象的互斥锁联系。当某个对象用synchronized修饰时，表明该对象在任一时刻只能由一个线程访问
4. 同步的局限性：导致程序的执行效率要降低
5. 同步方法（非静态的）的锁可以是this，也可以是其他对象(要求是同一个对象)
6. 同步方法（静态的）的锁为当前类本身。

Synchronized

- 同步具体方法-Synchronized

1. 同步代码块

```
synchronized (对象) {  // 得到对象的锁，才能操作同步代码
    // 需要被同步代码;
}
```

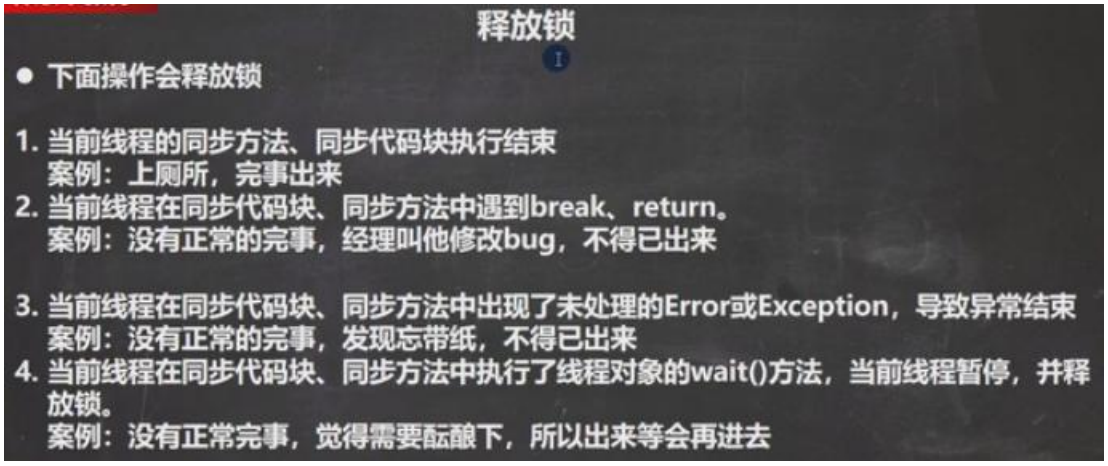
2. synchronized还可以放在方法声明中，表示整个方法-为同步方法

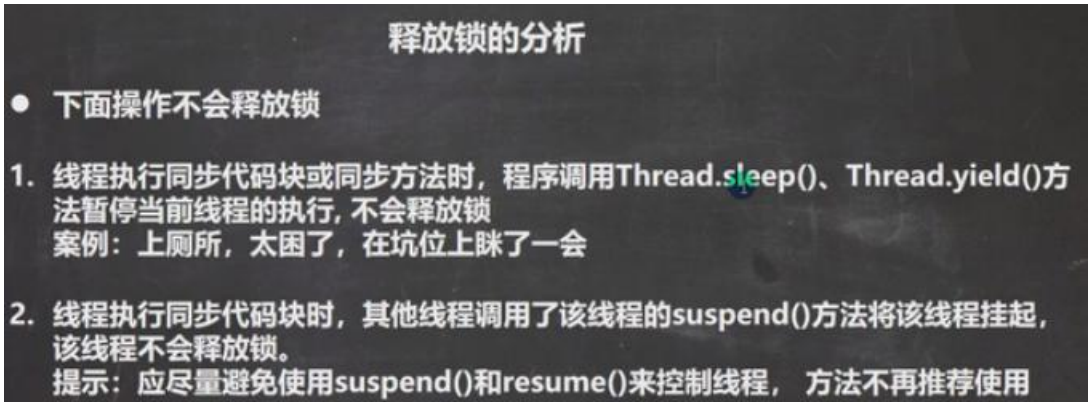
```
public synchronized void m (String name){
    //需要被同步的代码
}
```

3. 如何理解:
就好像 某小伙伴上厕所前先把门关上(上锁),完事后再出来(解锁),那么其它小伙伴就可在使用厕所了，如图:



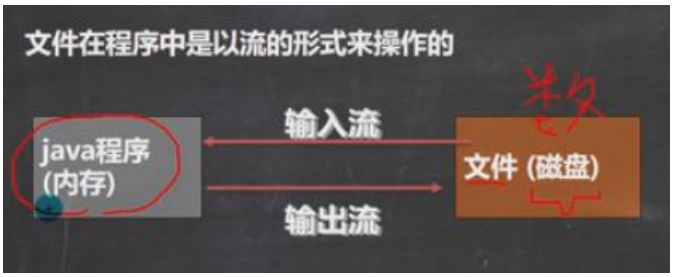
4. 使用synchronized 解决售票问题





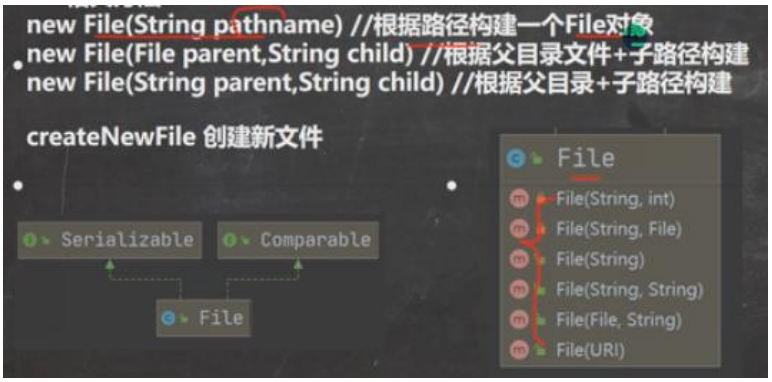
IO 流

文件：是保存数据的地方

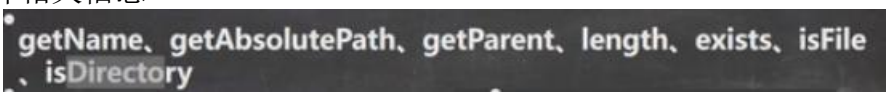


流：数据在数据源（文件）和程序（内存）之间经历的路径
输入流：数据从数据源（文件）到程序（内存）的路径
输出流：数据从程序（内存）到数据源（文件）的路径

常用的文件操作：
创建文件对象相关构造器和方法：

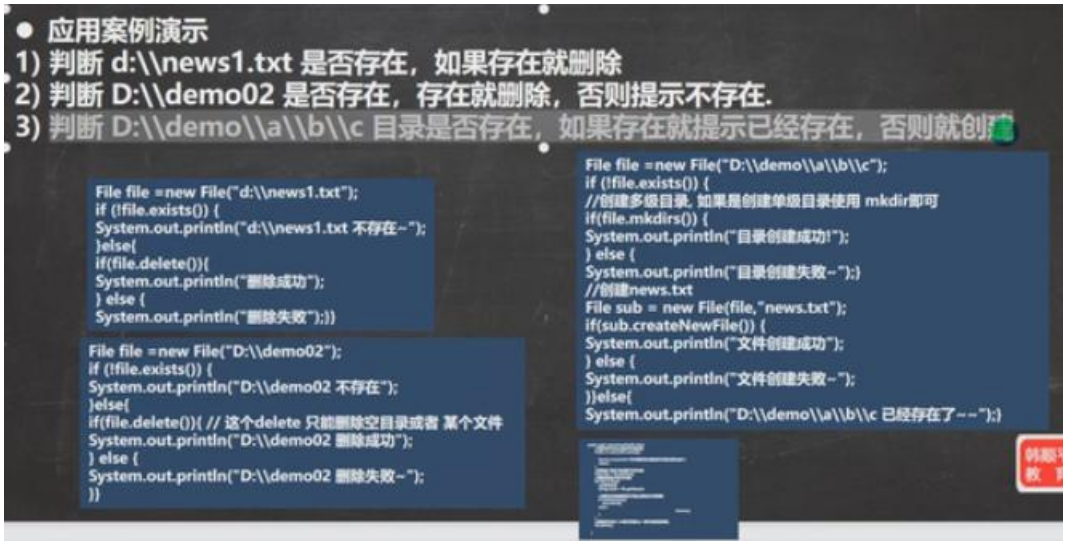


获取文件相关信息



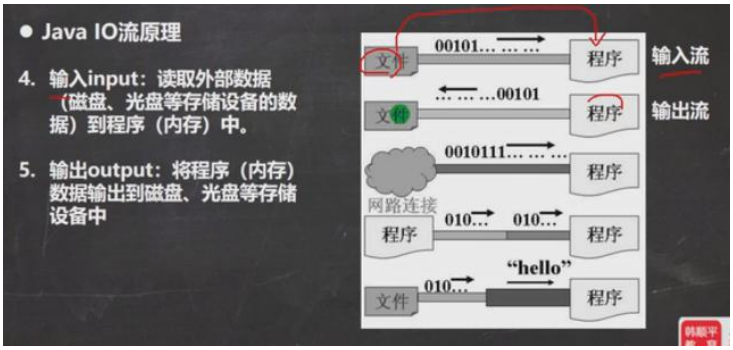
目录的操作和文件删除

mkdir()创建一级目录，mkdies()创建多级目录，delete()删除空目录或文件

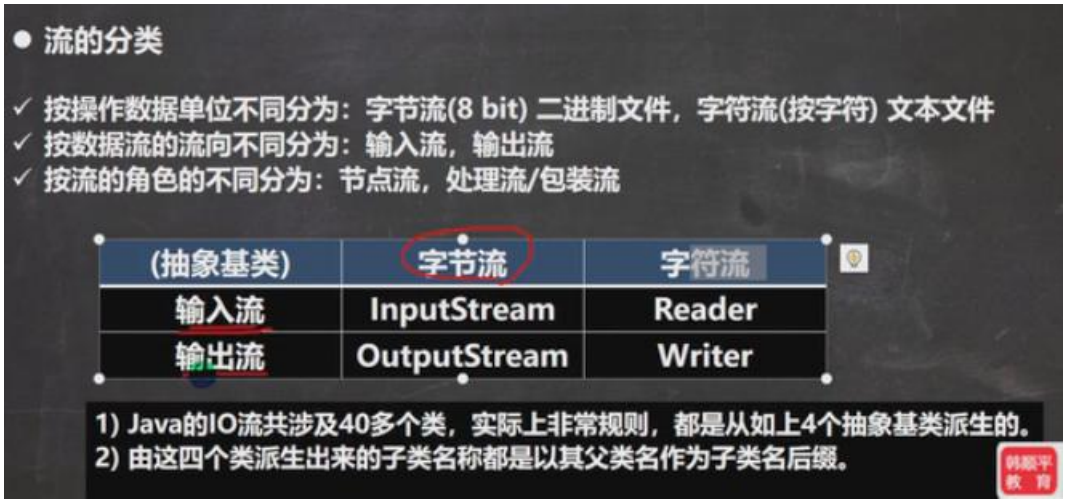


Java IO 流的原理：

1. I/O 是 Input/Output 的缩写，I/O 技术的非常实用的技术，用于处理数据传输，比如读写文件，网络通讯等；
2. Java 程序中，对于数据的输出/输出操作以“流（stream）”的方式进行
3. Java.io 包下提供了各种“流”类和接口，用以获取不同种类的数据，并通过方法输入或输出数据



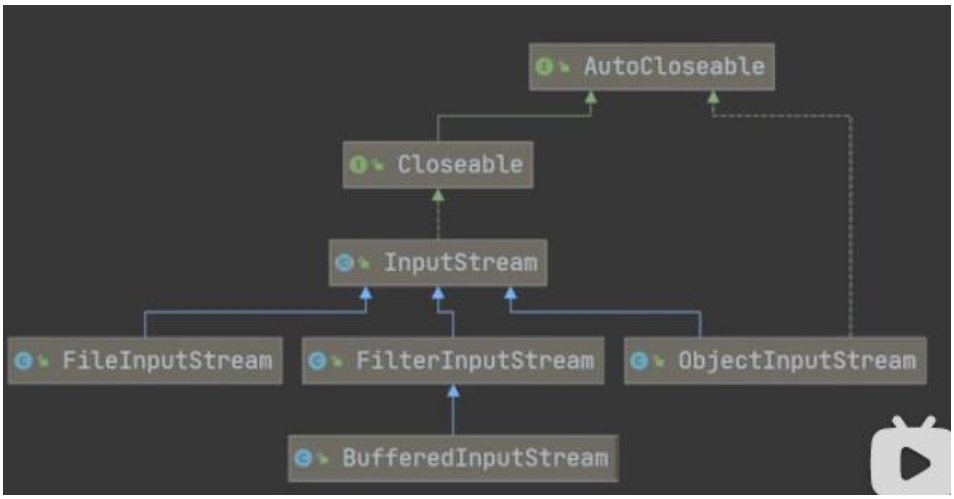
流的分类：



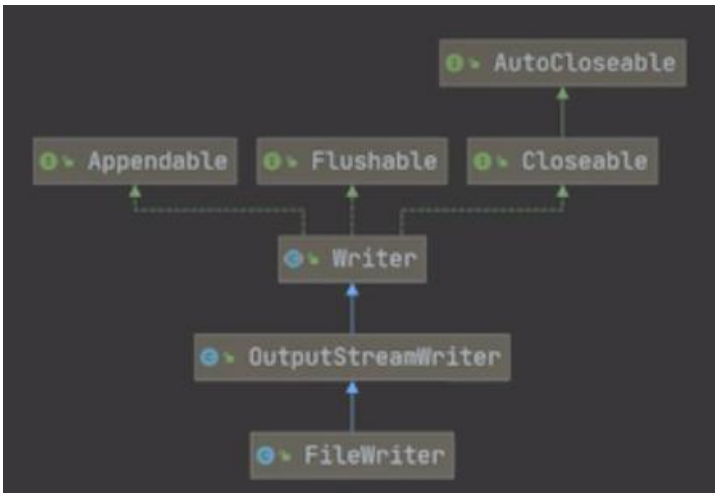
IO 流常用的类

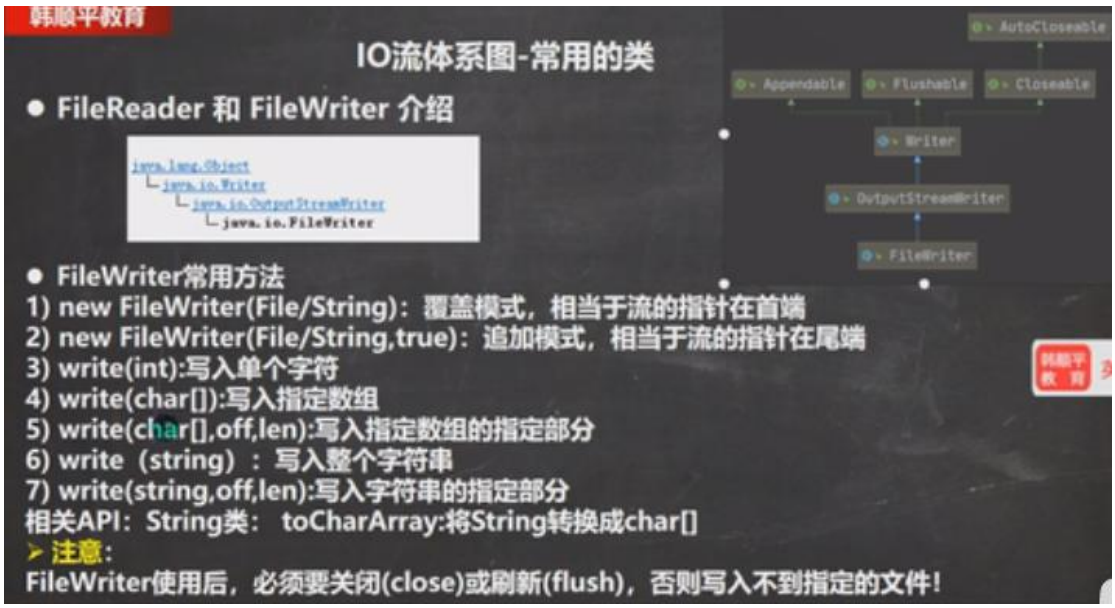
InputStream：字节输入流
InputStream 抽象类是所有类字节输入流的超类

InputStream 的常用子类：
1. FileInputStream：文件输入流
2. BufferedInputStream：缓冲字节输入流
3. ObjectInputStream：对象字节输入流



FileOutputStream 字节输出流


AutoCloseable
Appendable
Flushable
Closeable
Writer
OutputStreamWriter
FileWriter


韩顺平教育
IO流体系图-常用的类
FileReader 和 FileWriter 介绍
java.lang.Object
java.io.Writer
java.io.OutputStreamWriter
java.io.FileWriter
FileWriter常用方法
1) new FileWriter(File/String)：覆盖模式，相当于流的指针在首端
2) new FileWriter(File/String,true)：追加模式，相当于流的指针在尾端
3) write(int):写入单个字符
4) write(char[]):写入指定数组
5) write(char[],off,len):写入指定数组的指定部分
6) write（string）：写入整个字符串
7) write(string,off,len):写入字符串的指定部分
相关API：String类： toCharArray:将String转换成char[]
注意:
FileWriter使用后，必须要关闭(close)或刷新(flush)，否则写入不到指定的文件!