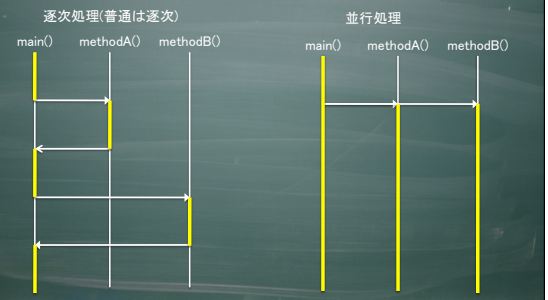
高度情報処理演習B

マルチスレッドと新入出力(java.nio)

スレッドとは

- 並行処理を書くための手段
- プロセスとの違い
  - プロセスよりも軽い
  - 複数スレッドはメモリ領域を共有
    - プロセスは別メモリとなる
- Javaは容易にスレッドを作成/利用できる
  - Threadクラス
  - Runnableインタフェース

逐次処理と並行処理の違い



スレッドの作成と実行1

- Threadクラスの継承
  1. Threadクラスを継承してクラスを作る
  2. run()メソッドをオーバライドする
  3. クラスをインスタンス化する
  4. インスタンス化したオブジェクトのstart()メソッドを実行する

スレッドの作成と実行2

- Runnableインタフェースの実装
  1. Runnableインタフェースを実装してクラスを作る
  2. run()メソッドをオーバライドする
  3. Threadクラスのオブジェクトを生成する
     - その際，コンストラクタの引数にRunnableインタフェースを実装したクラスのオブジェクトを指定する
  4. インスタンス化したオブジェクト(Thread)のstart()メソッドを呼び出す

スレッドはなぜ(いつ)必要か？

- 主にサーバプログラムで必要
- EchoServerは実はクライアント1台しかつながらない．
  - process()メソッドを呼び出すとstart()に制御が返ってこない(クライアントが切断するまで)．
  - 2台目が接続を試みても，accept()の呼び出しが行われない

そこでThreadの登場！

演習問題(quiz4th-1)をやってみましょう

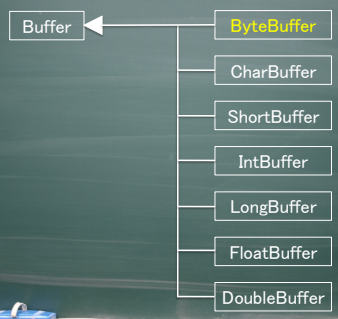
NewIO

- JDK1.4で導入されたIOパッケージ
- 特徴
  - Bufferクラス
  - Channelクラス
  - ノンブロッキングIO

Buffer

- プリミティブ型に特化したコンテナ
  - 配列以上コレクション以下
  - サイズ不変
  - 型の混合は不可
  - シーケンシャル/ランダムアクセス可能
  - position, limit, capacity // 重要
  - ヒープ外へのメモリへの直接アクセス
  - ByteBuffer以外入出力できない

Bufferのクラス階層



Bufferの使い方(ByteBufferを例に)

- 生成

```
ByteBuffer buffer = ByteBuffer.allocate(1024);
// ヒープ外に領域を確保
// ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
```

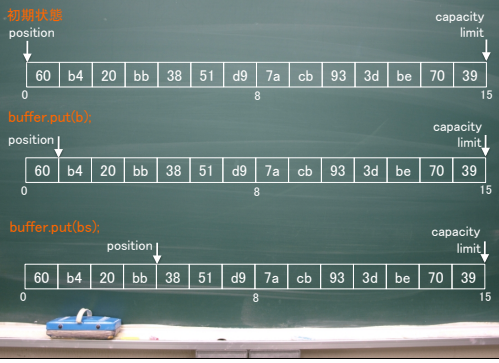
- 読み込み

```
byte b = buffer.get() // 1バイト読み込み
byte[] bs = new byte[3];
buffer.get(bs);       // 3バイト読み込み
```

- 書き込み

```
byte b = (byte)10;
buffer.put(b);              // 1バイト書き込み
byte[] bs = {10, 11, 12};
buffer.put(bs);             // 3バイト書き込み
```

position, limit, capacity

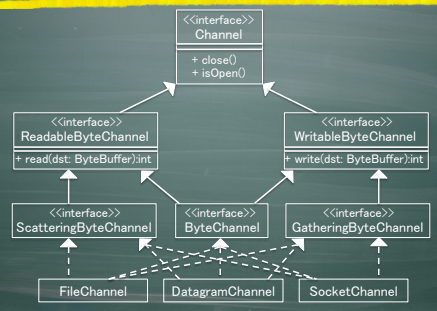


position, limit, capacityを操作するメソッド

- 詳しくはJavaDocを参照

メソッド名	機能
position(int newPosition)	positionをnewPositionに移動する
limit(int newLimit)	limitをnewLimitに移動する
clear()	position = 0, limit = capacityに移動する
flip()	limitをpositionの位置に移動させ，positionを0にする
rewind()	limitはそのままで，position=0に移動する
remaining()	現在のpositionからlimitまでの要素数を返す

Channel – ストリームに代わるクラス群



Channelの使い方 – FileChannelの例

1. ストリームの作成
2. チャネルの取り出し
3. ByteBufferを利用した読み書き

```
FileInputStream inStream = new FileInputStream();
FileChannel inChannel = inStream.getChannel();
.........
ByteBuffer buffer = ByteBuffer.allocate(1024);
inChannel.read(buffer);
......
```

ノンブロッキングIO

- ブロックするとはどういうことか

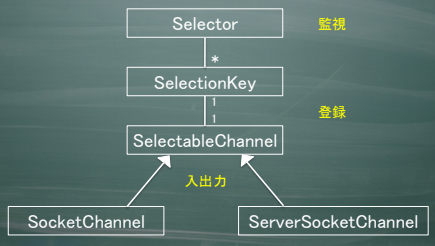
```
InputStream stream = socket.getInputStream();
int c = stream.read();
OutputStream out = new FileOutputStream("test");
out.write(c);
```

これ以上処理が進まない

- こうならないためにThreadを利用
  - Threadはコストがかかる
- そこで，ノンブロッキングIOが登場
  - 主にサーバプログラムで利用

Selector – ノンブロッキングIOの主役

- Selector – IOの入力を監視する



ノンブロッキングIOの処理手順

1. Channelを生成し，ノンブロッキングモードに設定
2. Selectorに登録
3. selectメソッドを実行
4. I/Oを処理が発生し，selectメソッドから戻る
5. I/Oの種類を調べ，その種類に応じた処理を行う
6. 再びselectメソッドを実行


I/Oの種類
● java.nio.channels.SelectionKeyに定義
種類
意味
SelectionKey.OP_ACCEPT
新規接続の受付
SelectionKey.OP_CONNECT
ソケットの接続(サーバに接続)
SelectionKey.OP_READ
データの読み込み
SelectionKey.OP_WRITE
データの書き込み