応用Javaプログラミング
補足1

1. オブジェクトの直列化
2. インスタンスの浅いと深い

参照型を直列化する必要性

下記は1つの図形を表すクラスである.

そのインスタンスのリストをファイルに保存したいとする.

```
package painter ;
import  javax.awt.* ;
public class DrawCommand {
    private Color color ; // 図形の色
    private float thickness ; // 図形の線の太さ
    private Shape shape ; // 図形の形

    //コンストラクタ
    public DrawCommand( ) { }
    public DrawCommand(Color color, float thickness, Shape shape) {
        this.color = color; this.thickness = thickness; this.shape = shape ;
    }
} // DrawCommandクラスの終わり
```

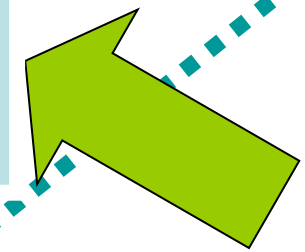
Colorは参照型であるので,
その値はメモリ上のアドレス
(その図形の色に関する情報
が置かれたメモリのアドレス).

Shapeも参照型なので
同じ問題が起こる。

問題:colorフィールドの値(アドレス)をそのまま
ファイルに保存したとすると,プログラムを再起動
したとき,保存した図の色情報を復元できない.

問題点の解決法

解決法：colorフィールドの値（アドレス）を
そのままファイルに保存しないで，そのアドレス
をシリアル番号で置き換え，かつ，そのアドレス
に置かれた**色情報**もファイルに保存する．



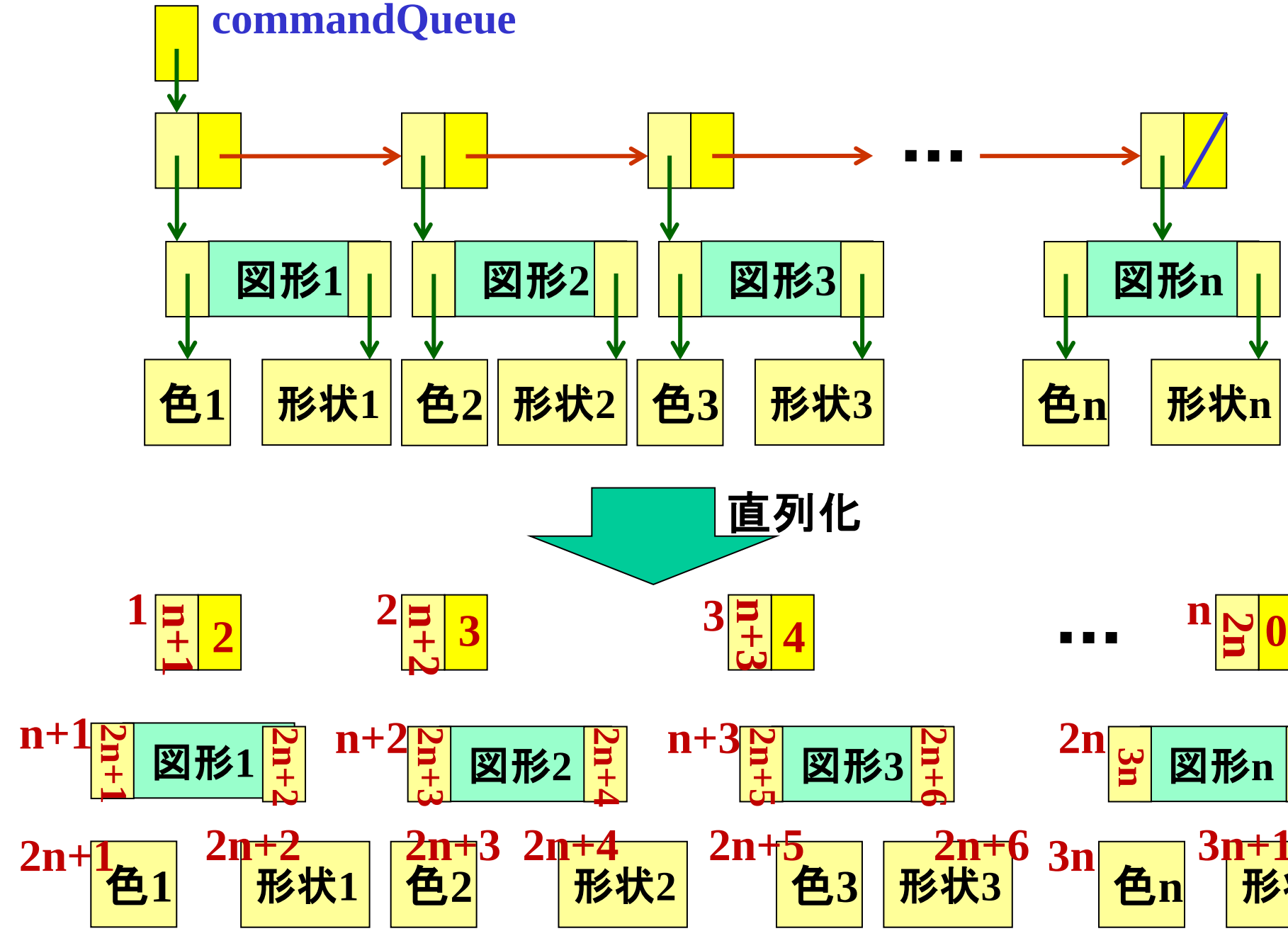
上記アイディアを実現するためには，
DrawCommandクラスを**直列化**可能に
する必要がある．

（メソッドなしの）
インターフェース
Serializableを
実装すればよい．

Colorは参照型であるので，
その値はメモリ上のアドレス
（**その図形の色に関する情報**
が置かれたメモリのアドレス）．

問題：colorフィールドの値（アドレス）をそのまま
ファイルに保存したとすると，プログラムを再起動
したとき，保存した図の**色情報**を復元できない．

図形データのリスト


commandQueue
図形1
図形2
図形3
図形n
色1
形状1
色2
形状2
色3
形状3
色n
形状n
直列化
1
n+1
2
2
n+2
3
3
n+3
4
n
2n
0
n+1
2n+1
図形1
2n+2
n+2
2n+3
図形2
2n+4
n+3
2n+5
図形3
2n+6
2n
3n
図形n
3n+1
2n+1
色1
2n+2
形状1
2n+3
色2
2n+4
形状2
2n+5
色3
2n+6
形状3
3n
色n
3n+1
形状n

DrawCommandクラス

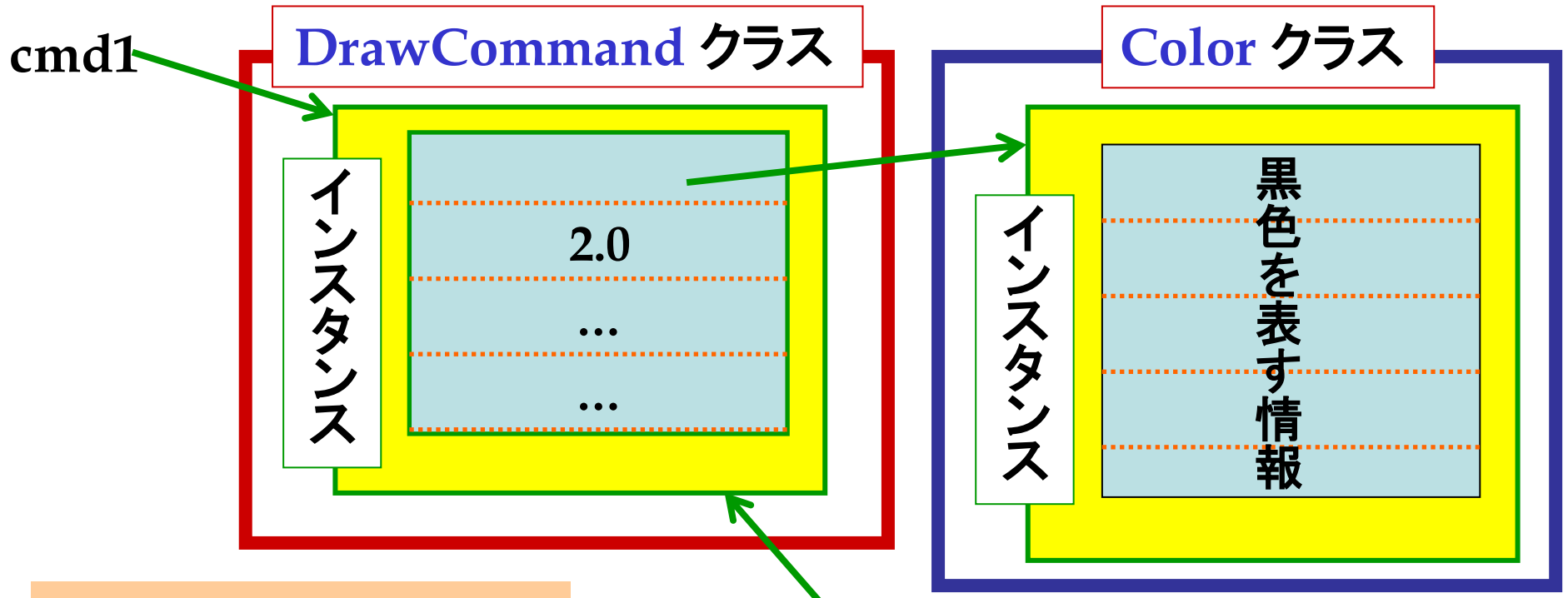
```
package painter ;
import  javax.awt.* ;

public class DrawCommand implements Serializable {
    private Color color ; // 図形の色
    private float thickness ; // 図形の線の太さ
    private Shape shape ; // 図形の形

    //コンストラクタ
    public DrawCommand( ) { }
    public DrawCommand(Color color, float thickness, Shape shape) {
        this.color = color; this.thickness = thickness; this.shape = shape ;
    }
} // DrawCommandクラスの終わり
```

命令の実行に必要なデータ.

補足1：インスタンスのコピー

DrawCommandクラスのインスタンスを例にして話を進める.

```
DrawCommand  cmd1 = new DrawCommand(Color.BLACK, 2.0F, … ) ;
```




```
cmd1.thickness = 0.5 ;
```

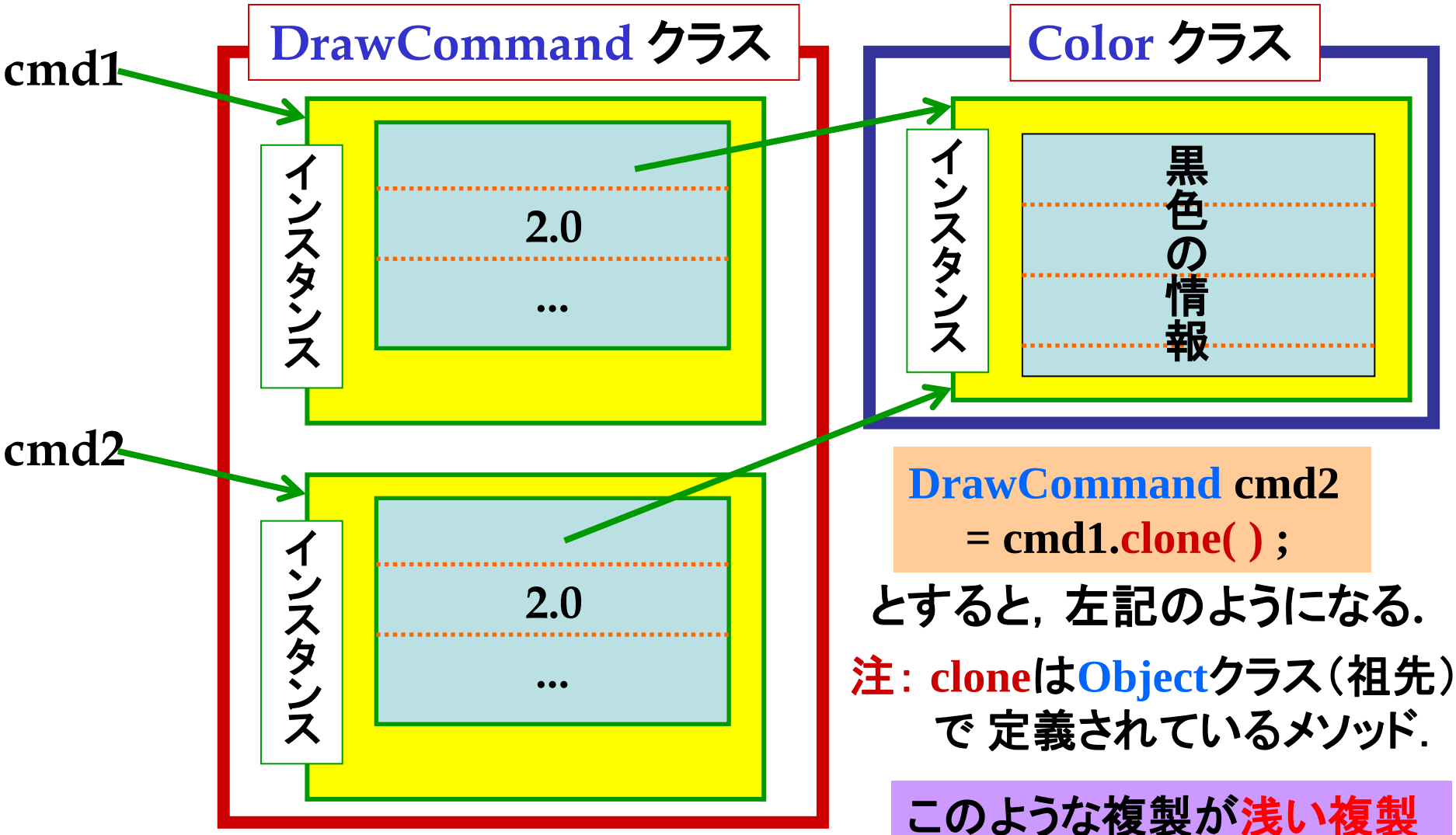
のようにすると,
cmd2.thickness も 0.5 になる.

```
DrawCommand cmd2 = cmd1;
```

とすると，cmd2 は cmd1 と同じインスタンスを指す.

補足2:インスタンスの浅い複製(clone)

```
DrawCommand cmd1 = new DrawCommand(Color.BLACK, 2.0F, … ) ;
```




```
DrawCommand cmd2
= cmd1.clone( ) ;
```

とすると，左記のようになる.

注: cloneはObjectクラス(祖先)で 定義されているメソッド.

このような複製が浅い複製(shallow copy)と呼ばれる.

補足2（続き）：Objectクラスの cloneメソッドはそのまま使えない

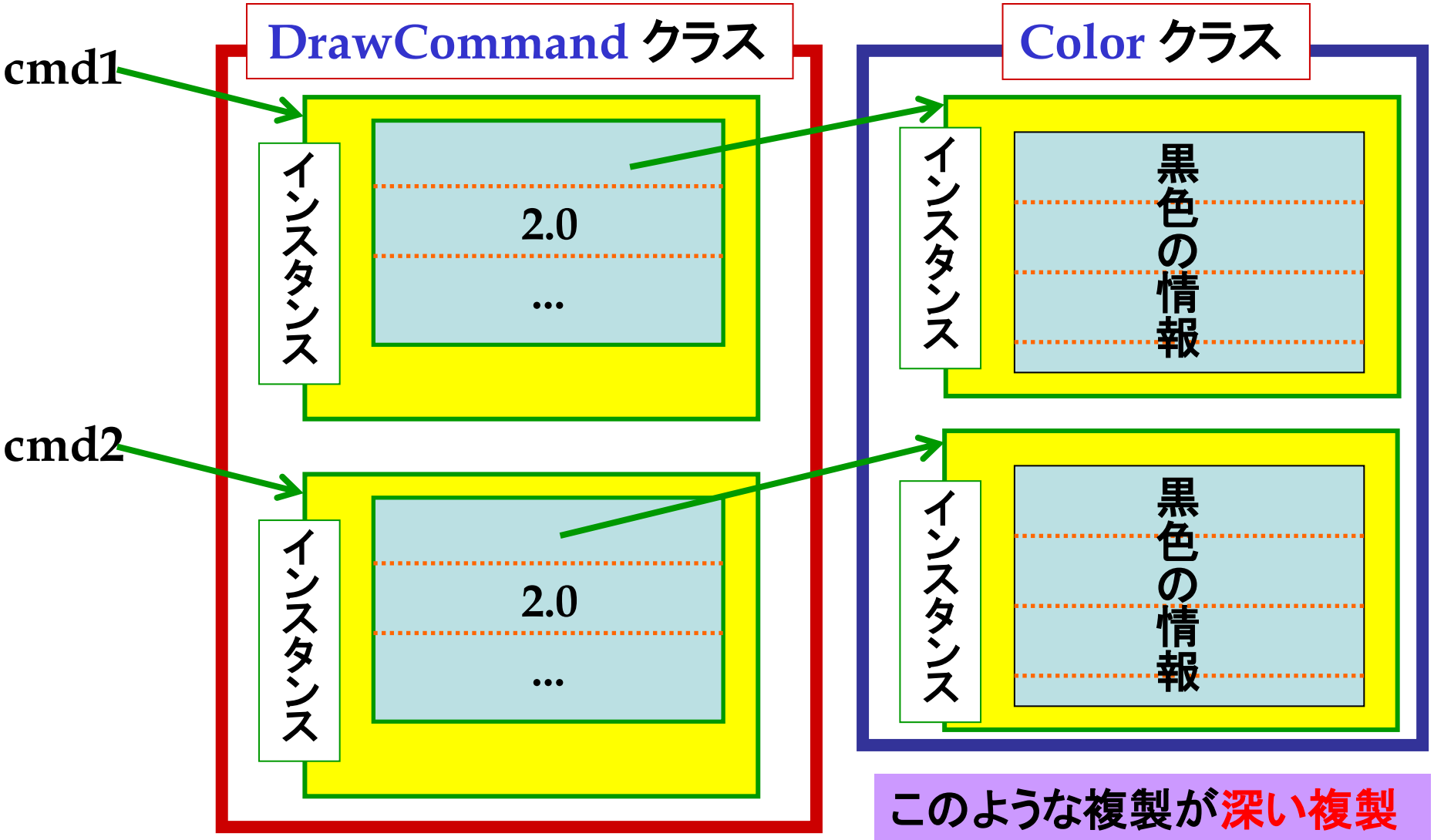
`DrawCommand cmd2 = cmd1.clone( );` のようにしてDrawCommandクラスのインスタンスの浅い複製を行うには，DrawCommandクラスで Cloneable インターフェースを実装し，Objectクラスの cloneメソッドを下記のようにオーバーライドすべき．

```
public class DrawCommand implements Serializable, Cloneable {
  ...（省略）...
    @Override
    public DrawCommand clone( ) {
        DrawCommand  c = null ;  // 浅い複製はまだできていない
        try {
            c = (DrawCommand)super.clone( ) ;   // 浅い複製を作る
        } catch( CloneNotSupportedException ex ) {
            return null ;  //浅い複製を作れなかったら nullを返す
        }
        return  c ;  // 作った浅い複製を返す
    }
}
```

追加分

補足3:インスタンスの深い複製

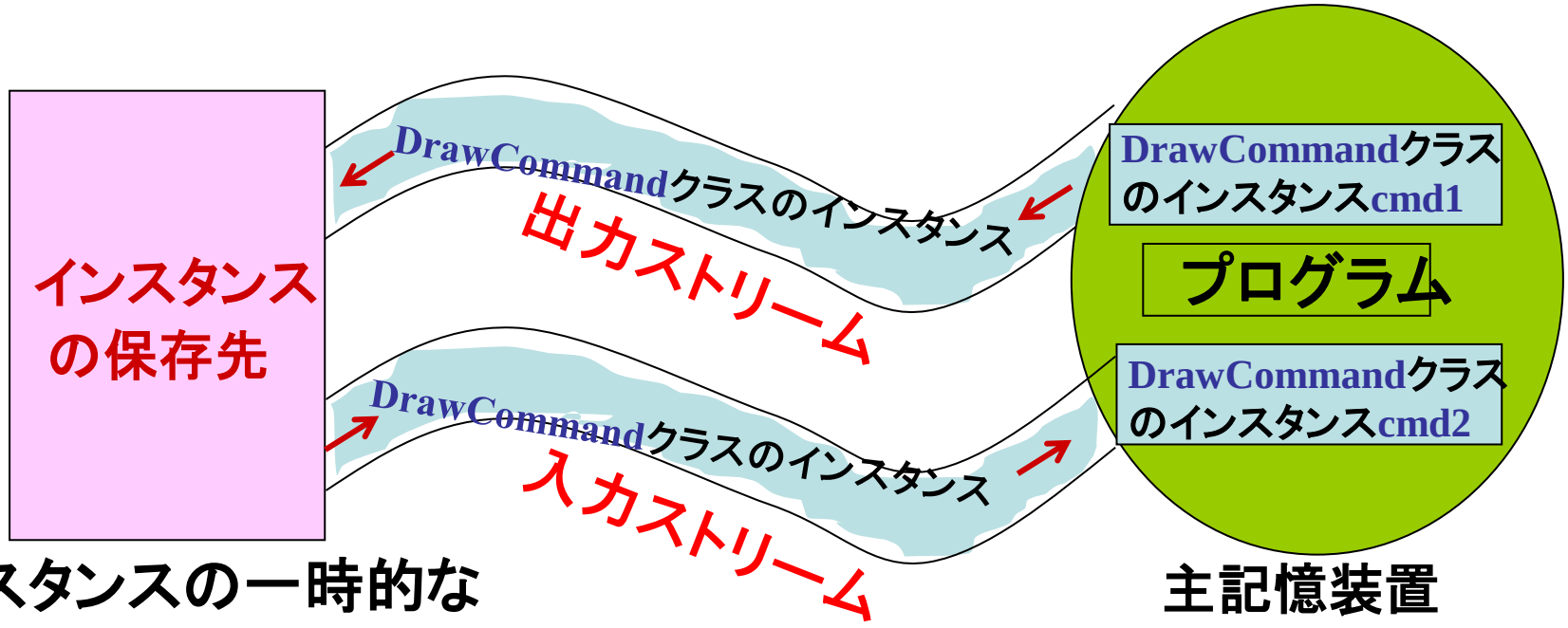
```
DrawCommand cmd1 = new DrawCommand(Color.BLACK, 2.0F, … ) ;
```



このような複製が深い複製（deep copy）と呼ばれる.

補足3(続き):直列化の応用
深い複製を行うメソッドを作ろう!

●アイディア: まず, 複製したいインスタンスを出力ストリームに書き込んで保存しておく. その後, 入力ストリームを介して保存先からインスタンスを読み出して別のインスタンスに記憶する.

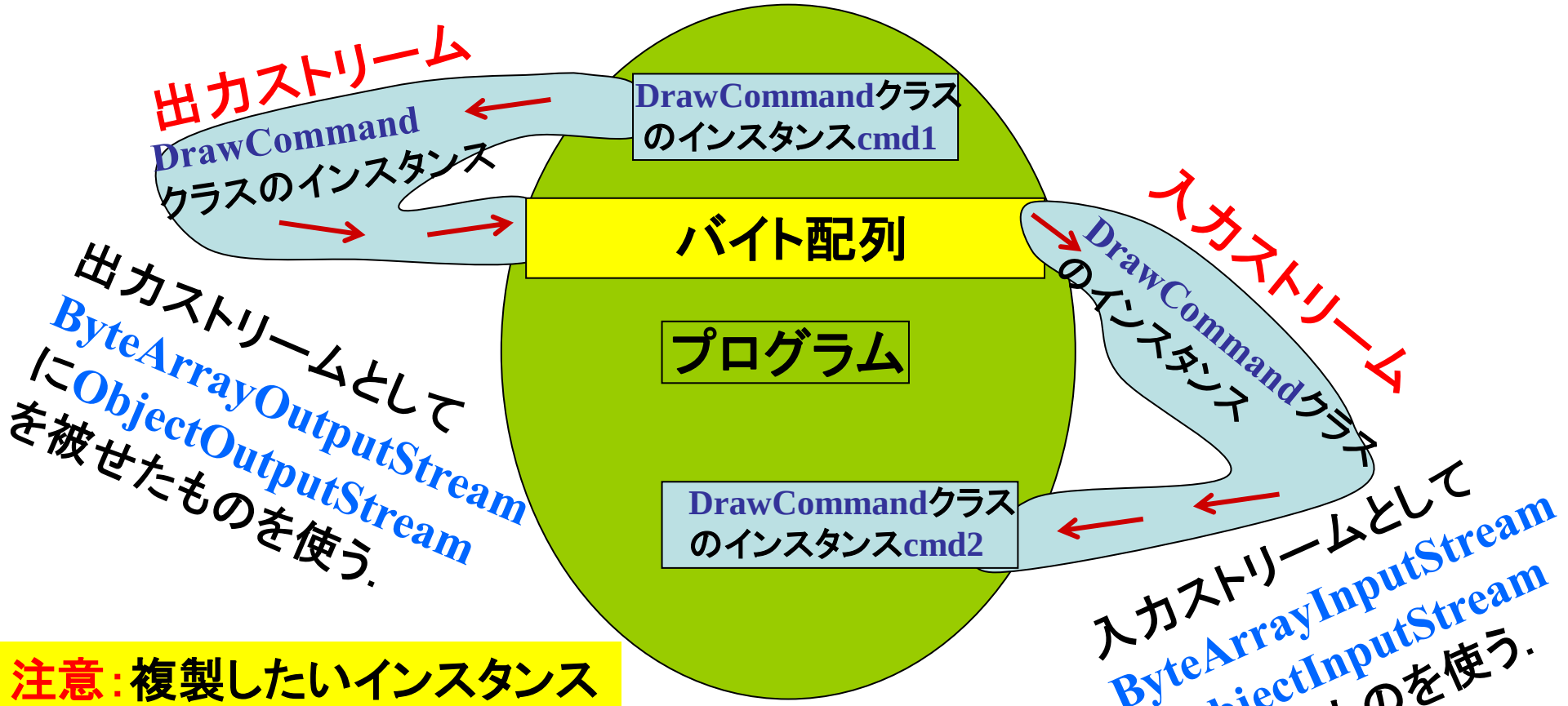


インスタンスの一時的な保存先としてファイルを使うのは自明なやり方.

次頁にて, ファイルを使うより速い方法を紹介する.

補足3(続き):直列化の応用
深い複製を行うメソッドを作ろう!(続き)

●保存先: 保存先として主記憶装置内のバイト配列を使う.



注意:複製したいインスタンスが直列化可能になっているときのみこの方法が使える.

補足3(続き):直列化の応用
深い複製を行うメソッドを作ろう!(続き)

直列化可能なオブジェクトを受け取り,
その深い複製を作って返すメソッド

```
Object deepCopy(Serializable obj) throws IOException,
                                         ClassNoFoundException {
    try ( ByteArrayOutputStream bout = new ByteArrayOutputStream( ) ;
          ObjectOutputStream out = new ObjectOutputStream(bout) ) {
        out.writeObject( obj ) ; // このオブジェクトを出力ストリームに書き込む

        try ( ByteArrayInputStream bin =
                  new ByteArrayInputStream( bout.toByteArray( ) ) ;
              ObjectInputStream in = new ObjectInputStream(bin) ) {
            Object obj2 = in.readObject( ) ; // 入力ストリームから読み出す
            return obj2 ;  // 作った深い複製を返す
        }
    }
}
```

保存

読出

CopyToolsクラス

```
package objectTools ;

import java.io.* ;

public  class CopyTools {
      // privateのデフォルトコンストラクタ。
      // これにより、他のクラスでこのクラスのインスタンスは作れなくなる。
      private CopyTools( ) { }

      // 直列化可能なオブジェクトを受け取り,
      // その深い複製を作って返すメソッド
      public static Object deepCopy(Serializable obj) throws
                                    IOException, ClassNoFoundException {
            … // 前頁の内容
      }
}
```