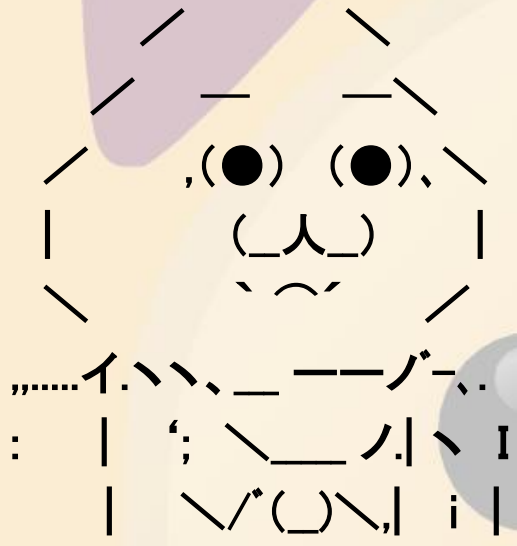
匠の伝承w

マルチな時代の設計と開発

パート4

スピーカー自己紹介



ゆーちです。

ハンドル名です。

本名は、内山康広といいます。

48歳です。

おっさんです。_|￣|○

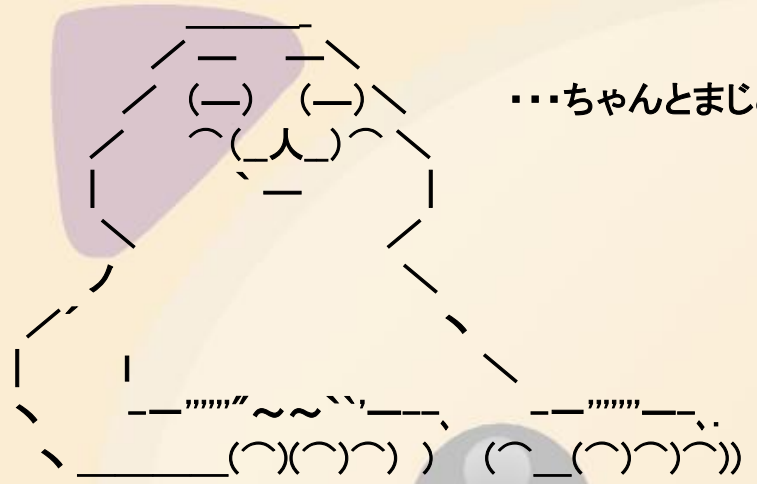
株式会社シーソフト代表取締役です。

現役のエンジニアです。プログラム書いてます。

にこにこカレンダーシートを販売しています。
2ちゃんねらーではありません。

Special thanks for 2ch.

前回までのおさらい



・・・ちゃんとまじめな話をしたんだお。

PART 1

開発者はプロセス指向にとらえがち。

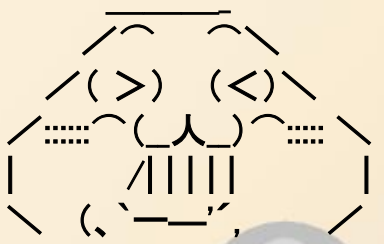
オブジェクト指向は『モノ』をとらえる。

『モノ』に対する時間軸のイベントを列挙。

時間軸へのイベントが『状態』を作る。

開発は『状態』別に分けて考える。

PART 2



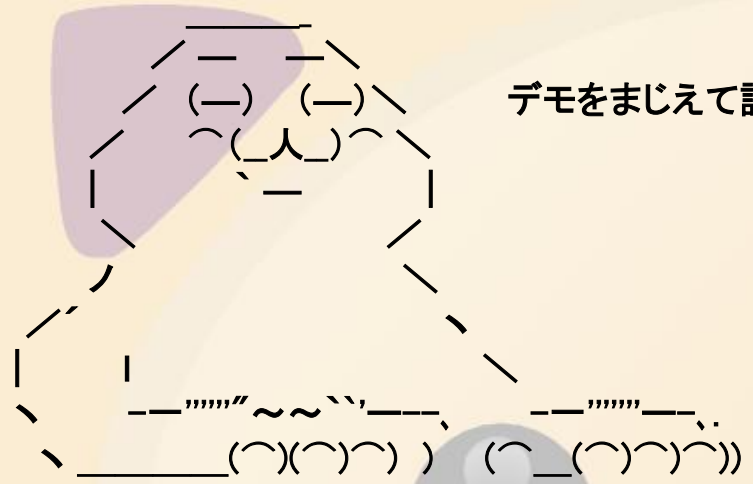
ぐだぐだだったお。

処理の依存性を切り離す

非同期の事象はループを分断して考える。

イベントトレース図→状態遷移表。

ステートパターンの実装。



デモをまじえて説明したんだお。

PART 3

クラスに「時間軸」を考える。

オブザーバパターンによるイベント通知。

クラスの依存性を少なくし独立性を高める。

本日のテーマ

依存性をなくすことのメリット。

DIってなんだ？

依存性をなくすって

コピー処理の例(PART2)

```
Copy(string DestFileName, string SrcFileName);
        ↓
Copy(string DestFileName, string SrcFileName, DWORD *CopyTime);
        ↓
Copy(string DestFileName, string SrcFileName, DWORD *CopyTime,
    CALLBACK *OnProgress);
        ↓
Copy(string DestFileName, string SrcFileName, DWORD *CopyTime,
    CALLBACK *OnProgress, bool &Canceled);
```

書き込み時間を計測

プログレスバーを表示

中止ボタンに応答

「コピー」だけを独立できないの？

コピーロジッククラスの例(PART3)

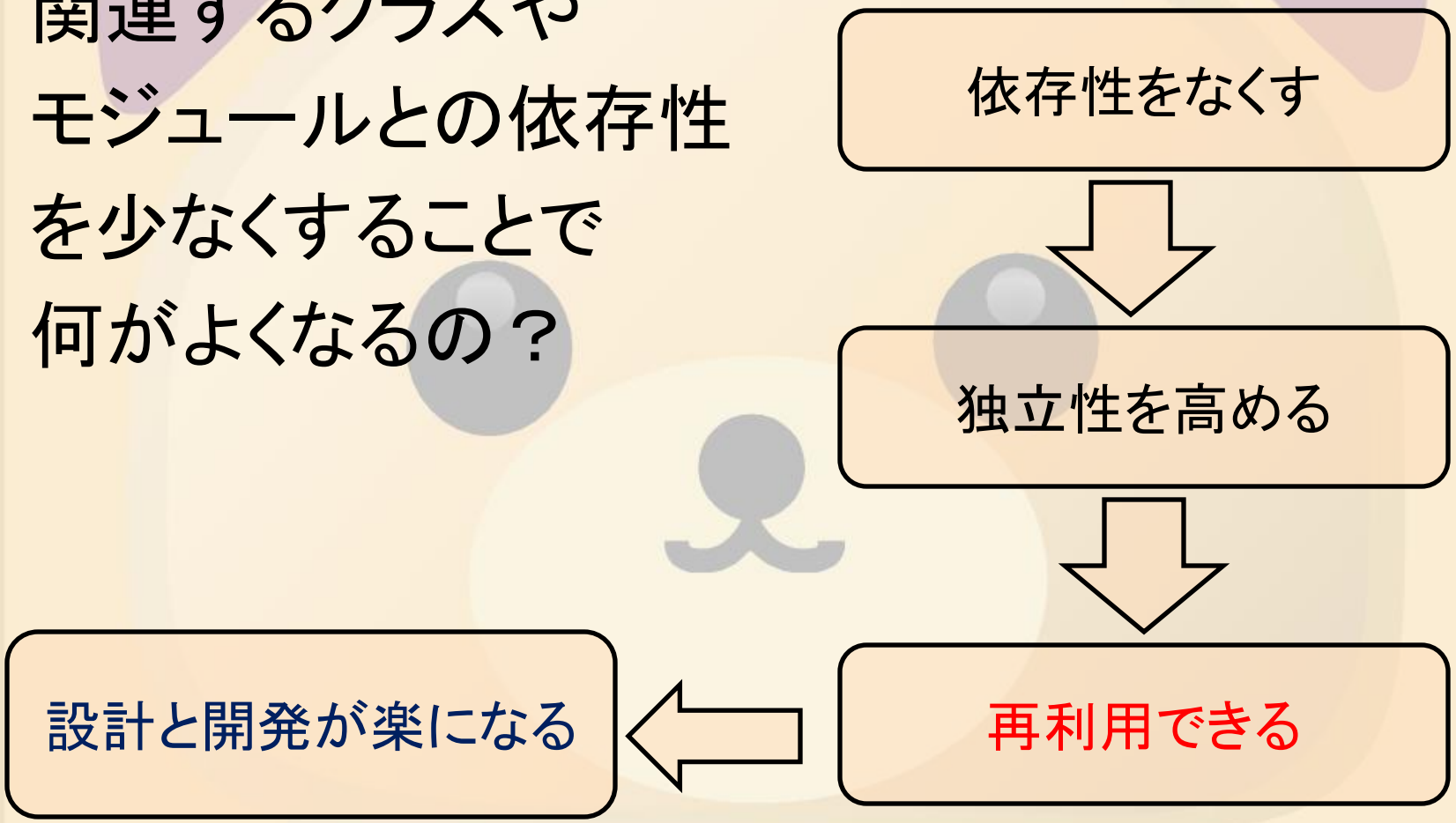
```
class CopyFileLogic
{
private:
	HANDLE			Source;
	HANDLE			Dest;
	LARGE_INTEGER		FileSize;
	char			*Buffer;
	static const int	BufferSize = 32768;
	// コピー状態の通知者
	ObserverSubject *Subject;
public:
	CopyFileLogic();
	virtual ~CopyFileLogic();
	bool Open( CString& source, CString& dest );
	void Close();
	bool Execute();
	// コピー状態の観察者の追加と削除
	void AddObserver( Observer *observer );
	void RemoveObserver( Observer *observer );
};
```

オブザーバパターン

処理中の経過や変化を見てる人に通知

依存性をなくすメリット

関連するクラスや
モジュールとの依存性
を少なくすることで
何がよくなるの？

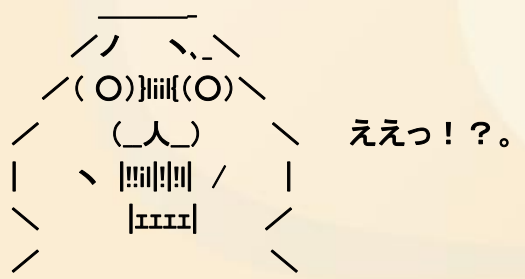


DI(Dependency Injection)

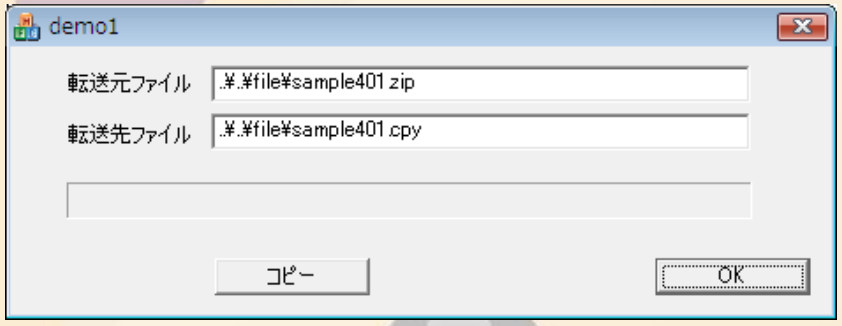
IoC(Inversion of Control)とも呼ばれます。

Strategy パターン、Factory パターンを使って、依存性を切り離そうという考え方です。

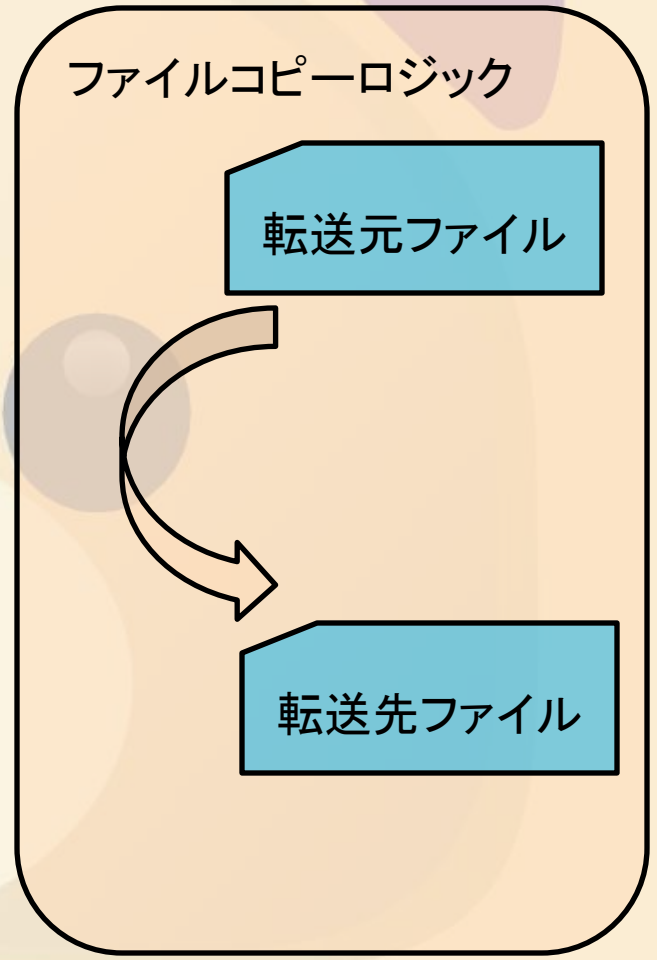
世界中のあちこちで解説してありますので、詳細は略。



再び「ファイルのコピー」を考える



フォームの構成要素
・ファイル名の入力コントロール
・終了のためのボタン
・コピー開始のためのボタン
・プログレスバー



プログラムの構造

```
Formのクラス{
    Logic ＝ new CopyFileLogic();
    OnButtonCopy(){
        Logic->SetSourceFile( Edit1->Text );
        Logic->SetDestnationFile( Edit2->Text );
        Logic->Execute();
    }
    OnOk(){
        ExitProgram();
    }
    OnSetProgressSize( int Size){
        //プログレスバーのサイズ設定
    }
    OnProgressChanged( int Position ){
        //プログレスバーを移動
    }
    とかなんとか。
```

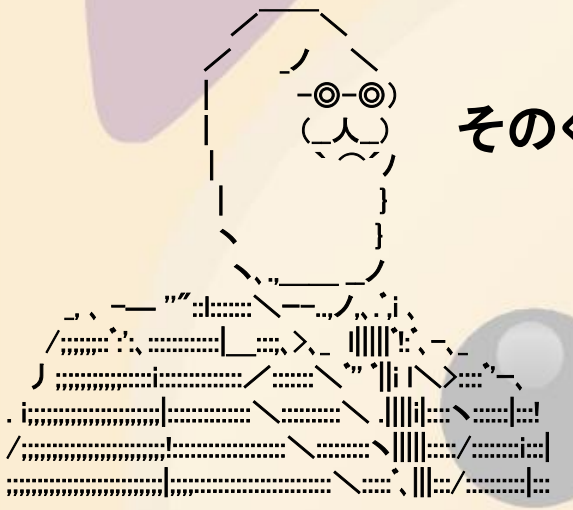
この辺がクラスの依存関係

DIを使ってみる。

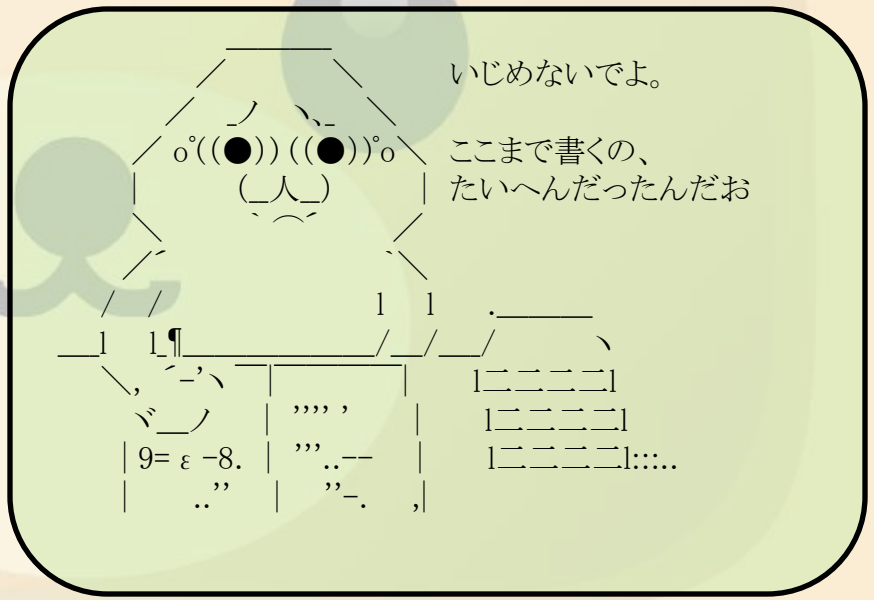
DEMO

Spring.NETを使ってみる

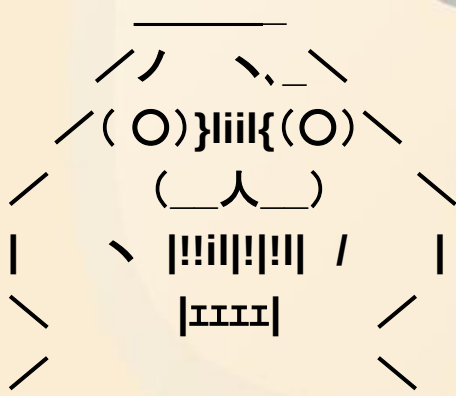
どのへんが「匠の伝承」なのか、について（笑）



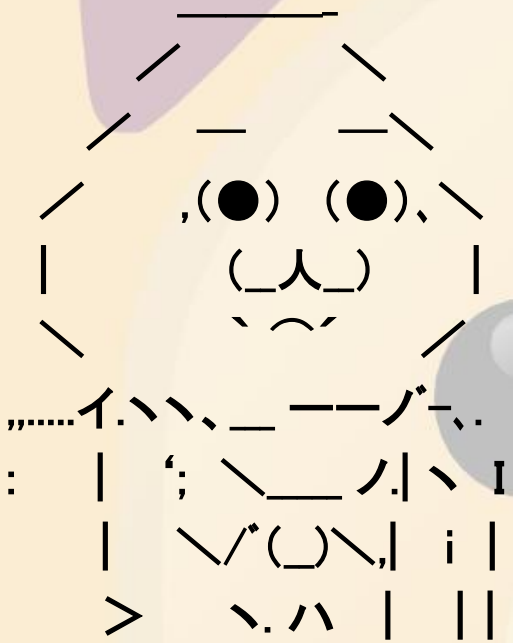
そのくらいのことは、もう一般常識だよ、君。



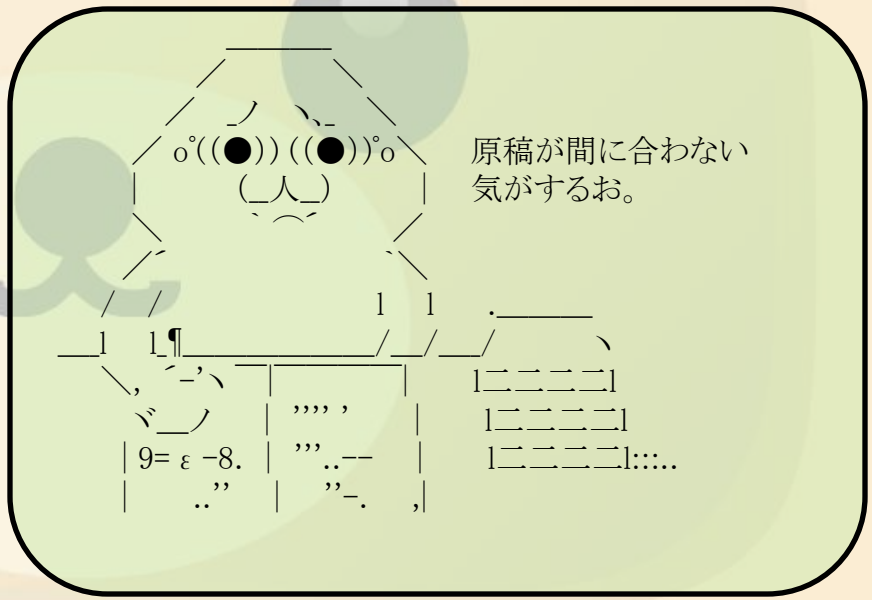
ここまでは、実は前フリです。



ええっ！？。

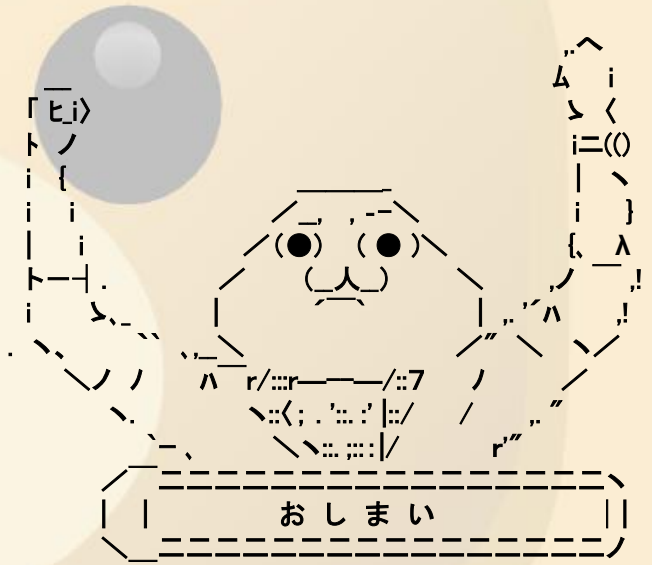


ざんねんながら、時間が迫ってきたようです。
次回、本シリーズの本領に踏み込んでいきます。
世界中の技術者を幸せに（謎）



ご静聴ありがとうございました。

m(_._)m



Special thanks for Yaruo charactors