MDDチュートリアル

目次

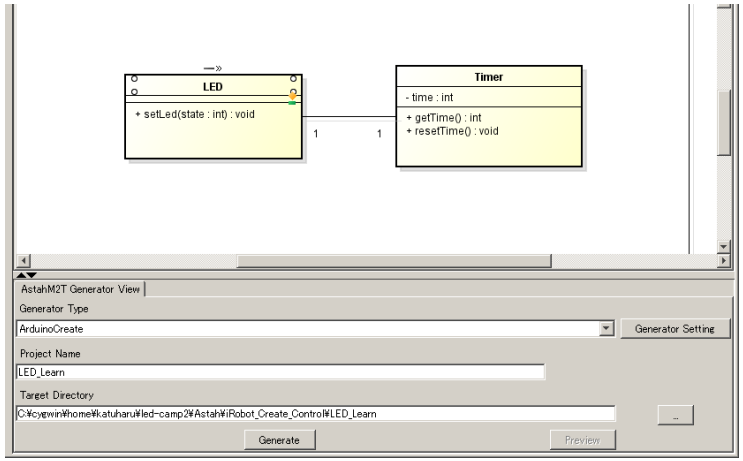
- MDDチュートリアル
- LEDちかちか
- クラス図を作ってみよう
- ステートマシン図を作ってみよう
- コード生成
- コンパイル
- 最後に

MDDチュートリアル

今回の実習では、astah* professionalに「m2tプラグイン」を導入してUMLのクラス図、ステートマシン図からArduinoのコードを生成します。

そこで、事前実習として、このプラグインの簡単な使い方を覚えていただきます。

m2tプラグインは「クラス図」「ステートマシン図」の2つのモデル図からソースコードを生成できます。



LEDちかちか

MDDチュートリアルとしてLEDを周期的に点滅させるアプリケーションのモデルからArduinoのソースコードを自動生成してみます。

以下のような仕様を満たすものとします

- Arduino基板上のチップLEDを1秒毎に点滅させる
- Ledの点灯、滅灯を制御するLedクラスを作成する
- 時間を計測するTimerクラスを作成する

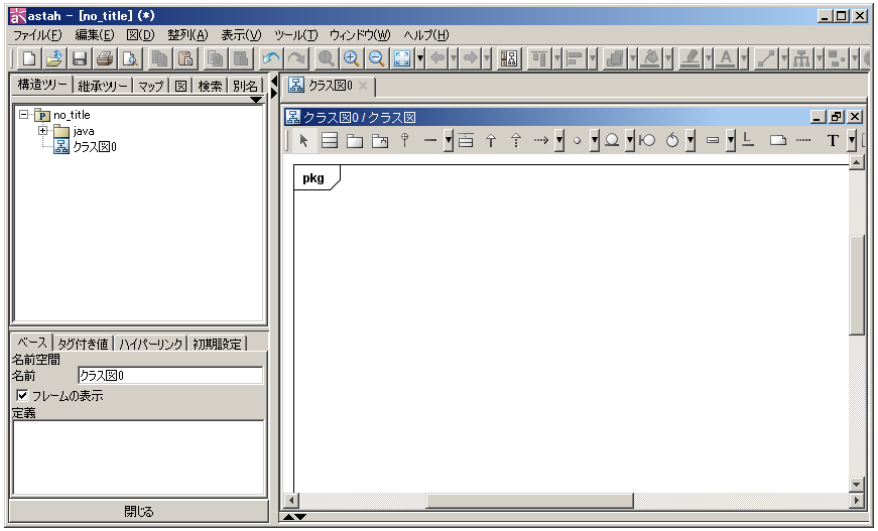
クラスを作る-1

まず、クラス図を作成します。

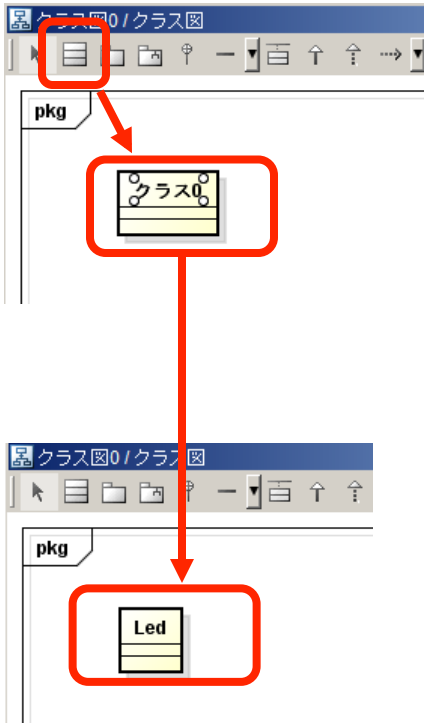
astah*を起動し、「ファイル -> プロジェクトの新規作成」を選択してください。これで、新規プロジェクトが作成されます。

プロジェクトの保存は「ファイル -> プロジェクトを保存」を選択します。適当なディレクトリに保存してください。

「図 -> クラス図」を選択すると、新規クラス図が作成されます。



クラス図を作る-2

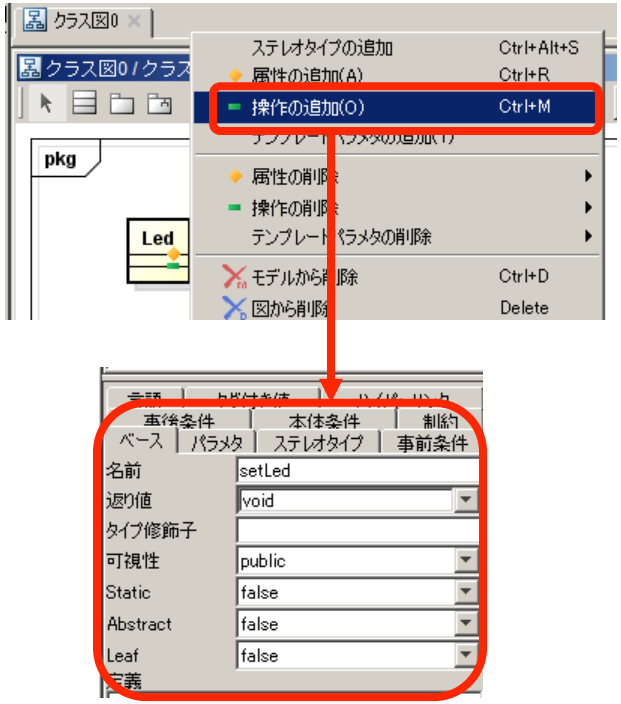


クラスを選択します。

その後、キャンパスをクリックするとクラスが配置されます。

「クラス0」と書かれている部分をクリックすると、クラス名を自由に変更できるので「Led」と変更します。

クラス図を作る-3



クラスを右クリックし「操作の追加」を選択します。操作とは、「クラスが持つ関数」のことです。

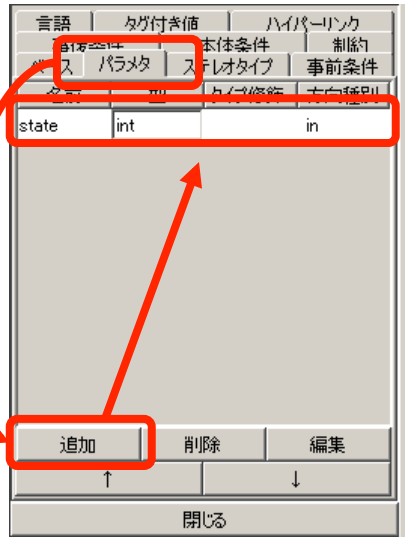
左側のビューに、情報を入力する欄がありますので、図のように入力します。

名前は直接クラスから入力することもできます。

※今回の実習では、名前、返り値以外の情報はソースコードに反映されません。それら以外は以下のように生成されます。

タイプ修飾子：空白
可視性　：public
Static　：false
Abstract　：false
Leaf　：false

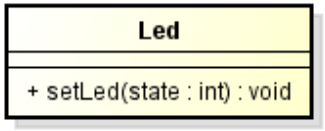
クラス図を作る-4



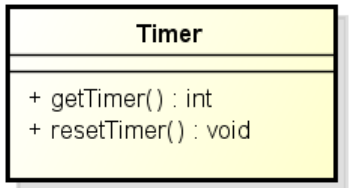
Ledクラス上の「setLed」が選択された状態で、左側のビューのパラメタタブを選択します。ここで「setLed」にどのような引数を与えるかを指定できます。
「追加」ボタンを押し、左図のように設定してください。

※今回の実習では、名前、型以外の情報はソースコードに反映されません。
それら以外は以下のように生成されます。

タイプ修飾子：空白
方向種別　　：inout



クラス図を作る-5

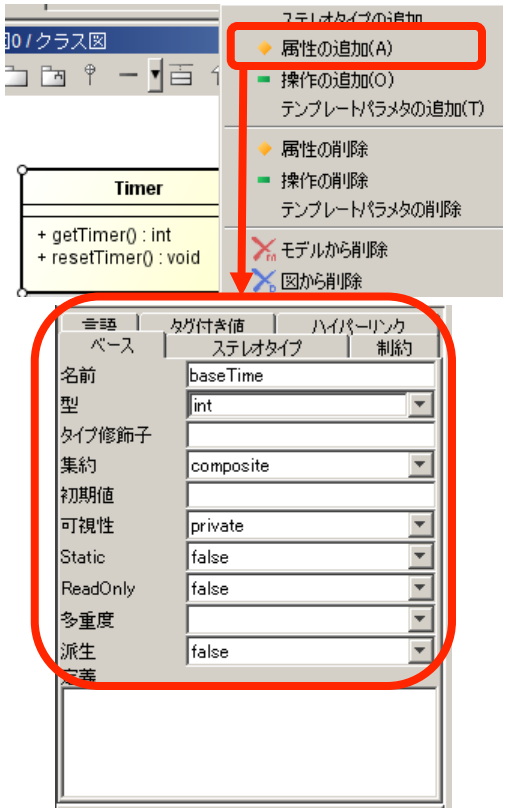


次に、Timerクラスを作成します。
これまでの内容を元に、左図のようなTimerクラスを作成してください。

操作の動作内容はそれぞれ以下のようになります。

getTimer　：resetTimer()からの経過時間[ms]を返す
resetTimer：getTiemr()が返す時間の基準時間を設定する

クラス図を作る-6



Timerクラスでは、基準時間からの相対時間をgetTimerで取得し、resetTimerで基準時間を設定します。なので、Timerクラスには「基準時間」が必要になります。

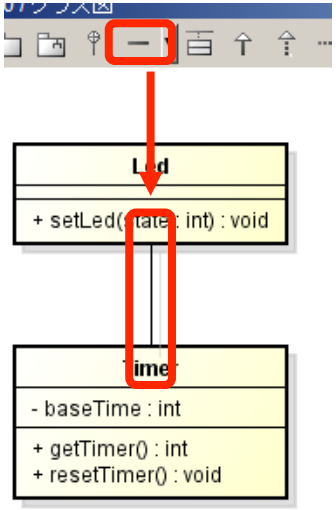
Timerクラスを右クリックし、「属性の追加」を選択してください。属性は、「クラスが持つ変数」のことです。

左側のビューに、情報を入力する欄がありますので左図のように設定してください。

※今回の実習では、名前、型以外の情報はソースコードに反映されません。それら以外は以下のように生成されます。

タイプ修飾子　:空白
集約　:public
初期値　:なし
可視性　:public
static　:false
ReadOnly　:false
多重度　:空白
派生　:false

クラス図を作る-7

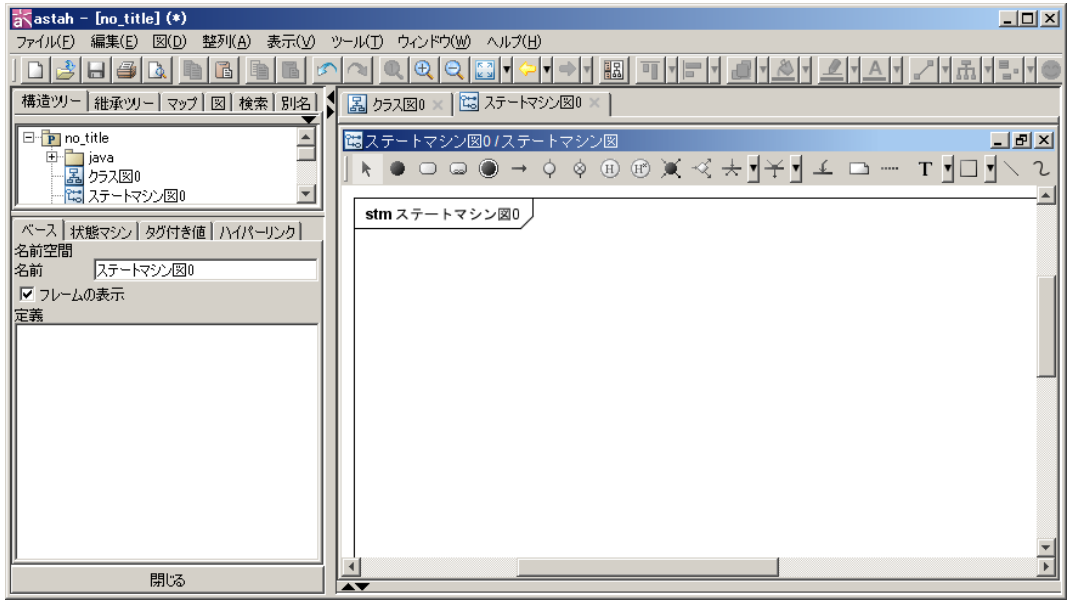


最後に、クラス間の関係を示します。
ツールバーの「関連」を選択し、各クラス間をつなぎます。

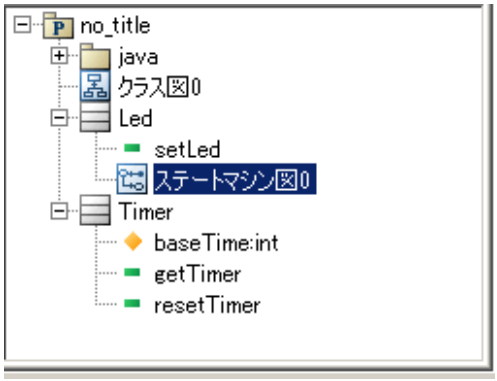
クラス間の関係には「関連」、「集約」、「コンポジション」、「依存」、「汎化」、「実現」がありますが、今回は「関連」のみが使用できます。

ステートマシン図を作る-1

次にステートマシン図を作成します。
「図 -> ステートマシン図」を選択して、新規ステートマシン図を作成します。



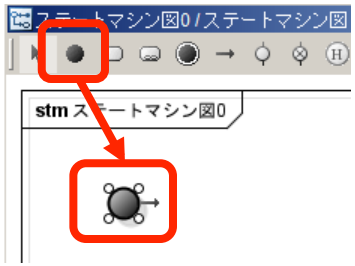
ステートマシン図を作る-2



作成するステートマシンが、どのクラスの状態を表すものなのかを決定します。

左側上部のビューに先ほど作成したステートマシン図が追加されています。これをLedクラスの中にドラッグ&ドロップして移動させてください。

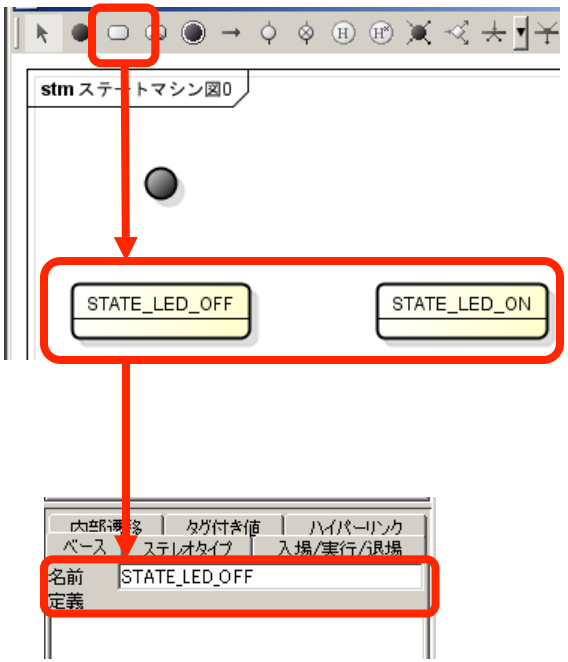
ステートマシン図を作る-3



まずは、状態遷移の起点を作成します。「開始擬似状態」を選択します。その後、キャンパスをクリックすると「開始擬似状態」が配置されます。

開始擬似状態から遷移する状態が初期状態となります。

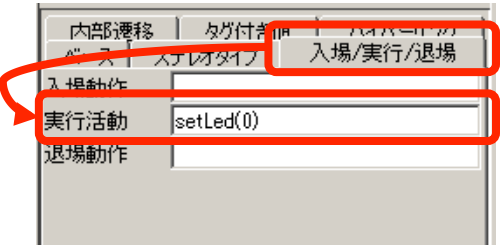
ステートマシン図を作る-4



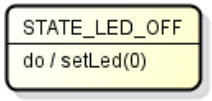
次に「点灯」と、「滅灯」の状態を作成します。
「状態」を選択します。その後、キャンパスをクリックすると「状態」が配置されます。
左のビューに情報を入力する欄があるので、そこに名前を入力します。

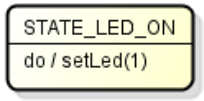
今回は、滅灯の「STATE_LED_OFF」と、点灯の「STATE_LED_ON」の2状態を作成します。

ステートマシン図を作る-5



「STATE_LED_OFF」を選択した状態で、左側のビューの「入場/実行/退場」タブを選択します。
ここでは、入場動作、実行活動、退場動作を指定できます。
入場動作は他の状態から現在の状態に入ったとき、実行活動は現在の状態時に、退場動作は他の状態へ遷移するときに実行されます。



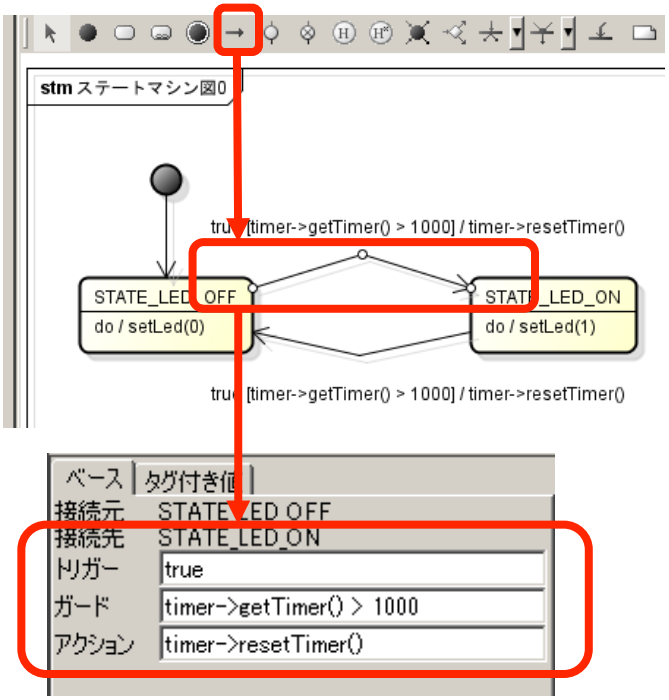


「STATE_LED_OFF」状態ではLEDを滅灯したいため、実行活動で、Ledクラスの setLed(0)を呼び出しLEDを滅灯します。

同様に、「STATE_LED_ON」状態にも実行活動を指定してみてください。

※LEDの点灯はsetLed(1)となります。

ステートマシン図を作る-6



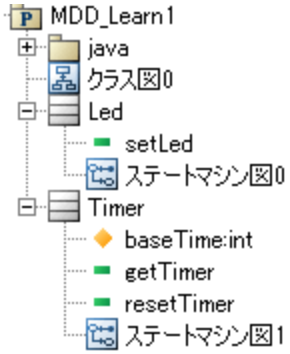
最後に遷移を設定します。遷移を選択して、各状態を左図のように接続してください。
※線をドラッグすると折り曲げることができます。

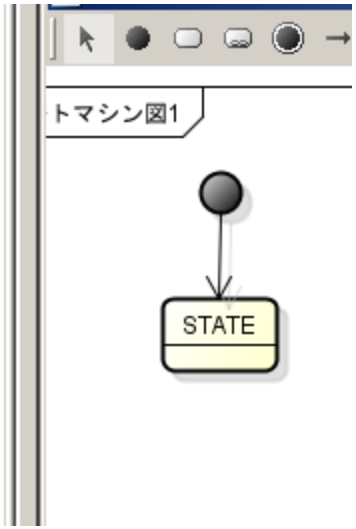
遷移は「トリガー」「ガード」「アクション」を指定でいます。
トリガーは遷移するためのイベントを指定します。
ガードは遷移する際に満たしていなければならない条件を指定します。
アクションは遷移時に行う動作を指定します。

今回は、遷移するイベントは無条件としてtrueとし、ガード条件としてtimerクラスのgetTimer()で取得した値が1000より大きいと指定します。遷移時のアクションとして、timerクラスのresetTimer()を呼び出します。
※他のクラスの操作を実行する場合、クラス名の先頭を小文字にし「->（アロー演算子）」をつけて関数名を記述します

左図のように遷移を記述してみてください。

ステートマシン図を作る-7





プラグインはステートマシン図を持った、クラスに対してコード生成を行います。

今回、Timerクラスは1状態でよいですが、左図のようにステートマシン図を作成して、開始擬似状態から空の状態へ遷移するようなモデルを作成してください。

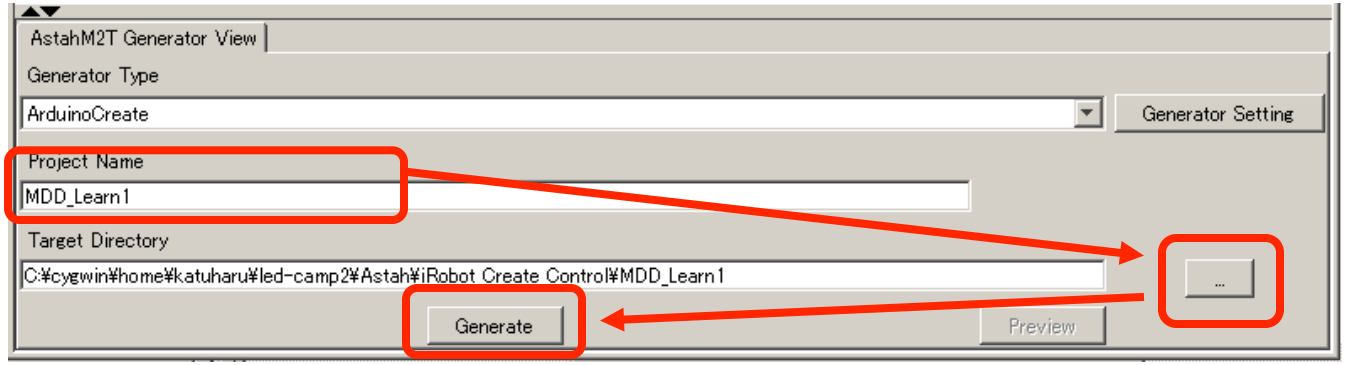
コード生成-1

モデルが完成したら、いよいよコード生成です。
モデルを元にコードを作成します。

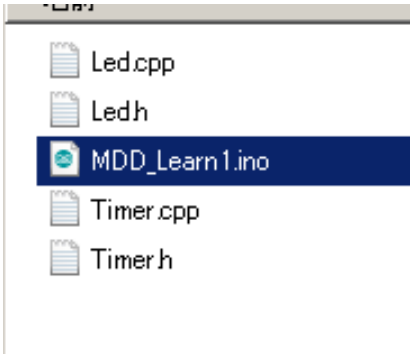
プラグインをインストールしてある場合、astah下部に下図のようなビューがでます。

プロジェクト名と、保存するフォルダを指定して「Generate」をクリックします

※Arduino以外の言語もカスタマイズしてコード生成することもできるため、Generator Typeの設定などもありますが、今回の実習では使用しません。



コード生成-2



Target Directoryで指定したフォルダにコードが生成されていることを確認してください。

生成された.inoファイルをArduino IDEで開くことで、MDDにより自動生成されたコードの編集・コンパイルができます。

コード追加-1

Led.cpp内の「Led関数」、「setLed関数」を右のように追加します。
Led関数はコンストラクタと呼ばれるもので、起動時に呼びだされます。

Arduino基板上についているチップLEDは13pinに接続されているため、pinMode関数で13pinを出力に設定しています。

setLed関数では引数のstateに応じて、digitalWrite関数で13pinの出力を変えています。

```
Led::Led(Create* create){
 this->create=create;
//  led_state=;
 led_event=E_Led_None;
 pinMode(13, OUTPUT);
}


void Led::setLed(int state){
 if(state == 0){
  digitalWrite(13, LOW);
 }else if(state == 1){
  digitalWrite(13, HIGH);
 }else{
  /* stateが0または1以外は何もしない*/
 }
}
```

コード追加-2

Timer.cpp内の「getTimer関数」「resetTimer関数」を右のように変更します。

resetTimer関数では、basetimeにmillis()関数を用いてArduinoが起動してからの時間を格納しています。

※millisはArduinoが起動してからの時間[ms]を、unsigned intで返します・・・ということは？

getTimer関数では、現在の時間とbasetimeの差分を返り値としています。

```
int Timer::getTimer(){
  int nowTime = (int)millis();
  return (nowTime - baseTime);
}

void Timer::resetTimer(){
  baseTime = (int)millis();
}
```

コード追加-3

Led.h
Led.cpp
MDD_Learn1.ino
をそれぞれ右のように
修正します。

MDD_Learn1.inoでは、
Timer timer(&create);
Led led(&create, &timer);
の順序を変更する必要が
あることに注意してください。

```
/* Led.h */
…
#include <Create.h>
#include "Timer.h"
…
/* functions */
 Led(Create*, Timer*);
…
private:
 Create *create;
 Timer *timer;
…
```

```
/* MDD_Learn1.ino */
Timer timer(&create);
Led led(&create, &timer);
```

```
/* Led.cpp */
Led::Led(Create* create, Timer* timer){
 this->create=create;
 //  led_state=;
 led_event=E_Led_None;
 this->timer=timer;
 pinMode(13, OUTPUT);
}
```

コード追加-4

最後に、
MDD_Learn1.inoの
setup関数内の左の2
行をコメントアウトしま
す。

※実習ではSDを使うためこのコードが生成されていますが、今回のLEDちかちかでは使用せず、干渉してしまうためコメントアウトしています。

```
void setup()
{
        //SD.begin(4);
        //CreateDataSDWriter.init();

        create.start();
        create.fullMode();
        delay(2000);
}
```

コンパイル

最後にコンパイルをします。

Arduino IDE上の検証を選択するとコンパイルされます。

エラーがなくコンパイル終了となれば成功です。

最後に

モデルからコード生成までの一連の流れは以上となります。

例題が小規模ということもあり、はじめは少し作業が多く感じるかもしれませんが、初めにモデルをきちんと作成していれば、今後動作に変更があった場合もほぼステートマシン図を修正してコード生成するだけで、動作の変更が可能になります。

今回作成したモデルとソースコードをLED-Camp2の合宿当日にも持ってきてください。
そこで、Arduinoに実際に書き込み、Lちかの動作の確認を行います。

以上です。お疲れさまでした。