要素技術とモデルを開発に使おう: コードとモデル図を対応づけてみよう

ETロボコン実行委員会 – er-info@etrobo.jp – 2.3, 2019-04-12 16:46:58 | 2019年用

sample00（ウォーカー）を動かしてみよう

`sample00` をビルドして動かしてみましょう。

どんな動きをするプログラムでしょうか。

リスト 1. `sample00/app.cpp`

```
#include "app.h"
#include "util.h"

#include "Motor.h"
#include "Clock.h"

using namespace ev3api;

/**
 * メインタスク
 */
// tag::main_task_1[]
void main_task(intptr_t unused) {

  Motor leftWheel(PORT_C);
  Motor rightWheel(PORT_B);
  Clock clock;

  const int8_t pwm = (Motor::PWM_MAX) / 6;
  const uint32_t duration = 2000;

  init_f(__FILE__);
  while(1) {
    msg_f("Forwarding...", 1);
    leftWheel.setPWM(pwm);
    rightWheel.setPWM(pwm);
    clock.sleep(duration);
// end::main_task_1[]
// tag::main_task_2[]
    msg_f("Backwarding...", 1);
    leftWheel.setPWM(-pwm);
    rightWheel.setPWM(-pwm);
    clock.sleep(duration);

    // 左ボタンを長押し、それを捕捉する
    if (ev3_button_is_pressed(LEFT_BUTTON)) {
      break;
```

```
    }
  }

  msg_f("Stopped.", 1);
  leftWheel.stop();
  rightWheel.stop();
  while(ev3_button_is_pressed(LEFT_BUTTON)) {
    ;
  }

  ext_tsk();
}
// end::main_task_2[]
```

どんな動作をするプログラム？

ビルドの方法はわかりますか。

```
$ cd （EV3RTのインストールディレクトリ）/hrp2/sdk/beginners
$ make app=sample00
```

ビルドできたら、床の上などで動かしてみましょう。

ロボットは、次のような動きをします。

- 2000msごとに、前進と後退を繰り返す
- 左ボタンをしばらく押し続けていると、そのうち押されたことが捕捉され、停止する

それはプログラムのどこに書いてありますか

- 前進する、後退するという処理はどれですか
- ボタンが押されるという処理はどれですか

その前に...

そもそも、何をしているのかプログラムから読み取れますか？

図に表して確認してみましょう。

サンプルモデル図のプロジェクトを作る

astah* でモデル図を作成するプロジェクトを用意します。

1. デスクトップに演習用のディレクトリ（フォルダー） `beginners` を作ります

2. **astah* Professional** を起動します（ライセンスは設定できていますか）
3. 「ファイル」→「プロジェクトの新規作成」でプロジェクトを作成します
4. プロジェクトのファイル名は `sample00` にしましょう（ファイル名は `sample00.asta` になります）
5. 「ファイル」→「プロジェクトを保存」で、 `beginners` ディレクトリに保存します

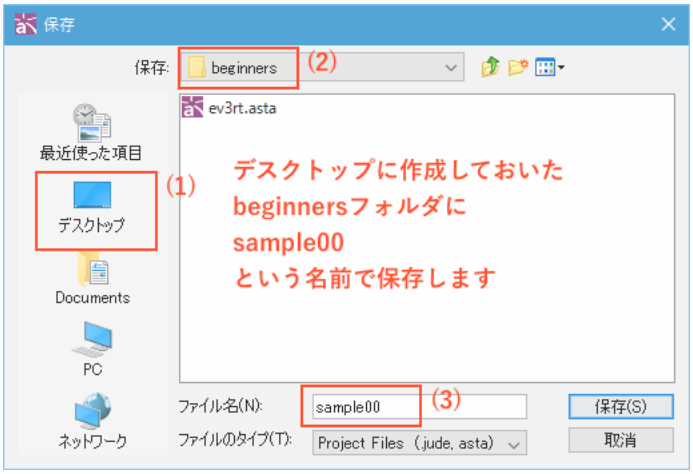

図 1. プロジェクトを保存する

ライブラリの参照プロジェクトを追加する

EV3RTの C++ 用のクラス群が定義されているプロジェクトを用意してあるので、これを自分が作成したプロジェクトから参照できるようにします。

1. 「ファイル」→「参照プロジェクト管理」で参照プロジェクト管理ダイアログを開きます
2. 「追加」ボタンで「ファイルの指定」ダイアログを開きます
3. 「相対パス」をチエックして、ファイル名に `ev3rt.asta` を指定し「了解」ボタンを押します
4. 参照プロジェクト管理ダイアログを閉じます
5. 構造ツリーに `ev3api` というパッケージが追加されていることを確認しましょう

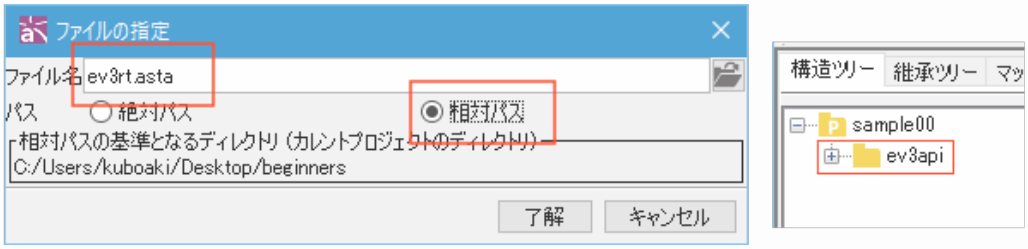

図 2. 参照プロジェクトとして `ev3rt.asta` を相対指定する

コードの登場人物をオブジェクト図に表す

`sample00` のコードを、「モノ」と「働き」に着目して整理します。次の手順に従って図に表してみましょう。

1. 「図」→「クラス図」でクラス図を追加し、図の名前を `sample00`のオブジェクト図 にします
2. コードの中から「モノ」を探してオブジェクトにします
   - たとえば、 `leftWheel` （左車輪）は `sample00.cpp` で使っているインスタンスですから「モノ」とみなせそうです
3. 「モノ」と識別した対象をオブジェクトとして図に描きます
   - クラス図のパレットから「インスタンス仕様」を選択し、アイコンが反転したら、マウスをダイアグラムエディター上でクリックします
   - 「インスタンス仕様0」のように仮の名前が表示されるので、これを `leftWheel` に変更します

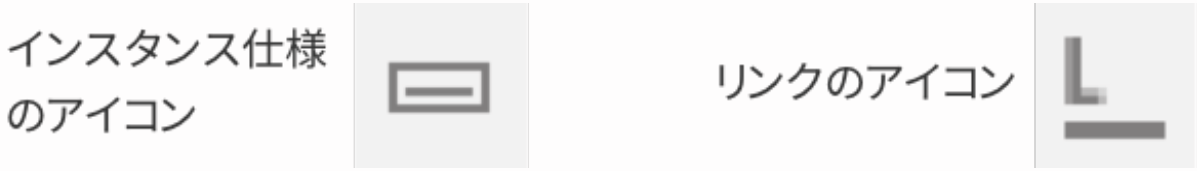

図 3. インスタンス仕様とリンクのアイコン

4. 同様にして、コード上にある `rightWheel` 、 `clock` も「モノ」とみなしてオブジェクトにします
5. `main_task` もオブジェクトとして作成します
   - `main_task` は、RTOSから呼び出されるCの関数ので、通常なら「働き」にあたりますが、この演習では呼び出す側のクラスのインスタンスとみなして扱うことにします
6. 「働き」を呼び出している側と呼びだされている側の間に、リンクを引きます
   - あるオブジェクトが別のオブジェクトの「働き」を呼び出して使っているなら、その間にリンクの線を引きます（関数とその関数を呼び出す側の関係と考えてみてください）
   - クラス図のパレットから「リンク」を選択し、アイコンが反転したら、一方のオブジェクトの上でマウスのボタンを押し、他方のオブジェクトの上まで移動したらボタンを離します
   - たとえば、`leftWheel` の `setPWM` は、モーターにPWM値を設定しているので、`leftWheel` の働きといえそうですが、これを呼び出して使っている「モノ」とは何でしょうか

作成したsample00のオブジェクト図 1

この図では各インスタンスの「スロットの表示」を「オフ」にしています。

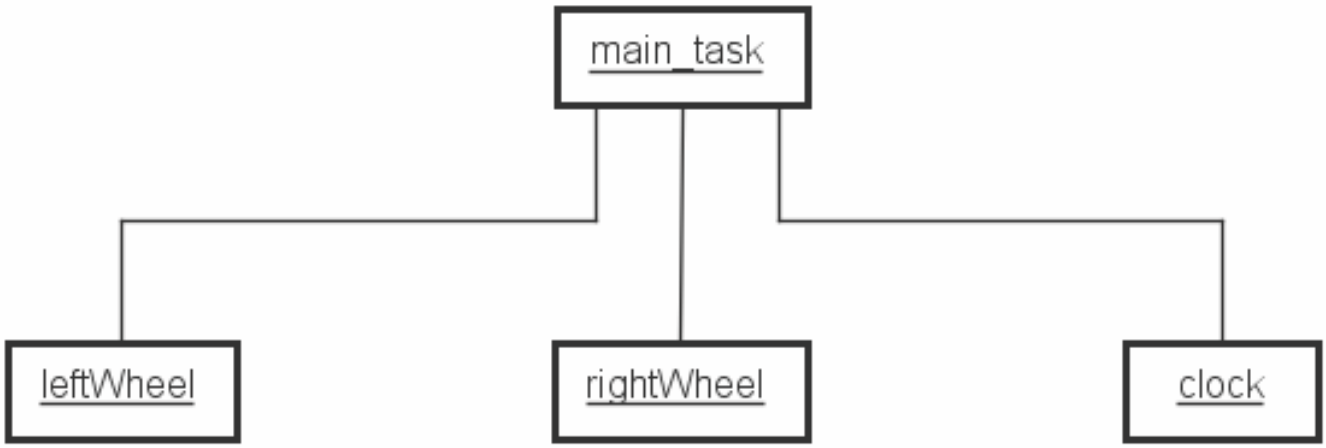

図 4. sample00 のオブジェクト図

インスタンスをクラスと関係づける

それぞれのインスタンスと関連するクラスがわかるようクラスを指定してみましょう。

1. leftWheel を選択します（4隅にハンドルが現れます）
2. 画面左下のプロパティエディターから「ベース」を選択し、「名前」が先ほど選択した leftWheel なのを確認します
3. ベースクラスのリストから Motor - ev3api を選びます
4. leftWheel の図が Motor クラスのインスタンスとして表示されます
5. 同様にして rightWheel 、 clock もクラスを関連づけます

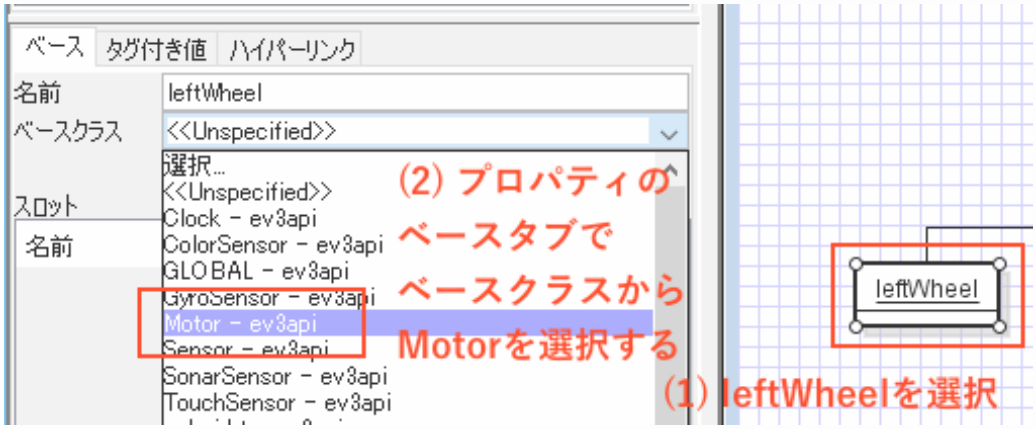

図 5. leftWheel に Motor クラスを割り当てる

作成したsample00のオブジェクト図２

クラスと関連づけたので、「オブジェクト名：クラス名」という表示に変わっています。

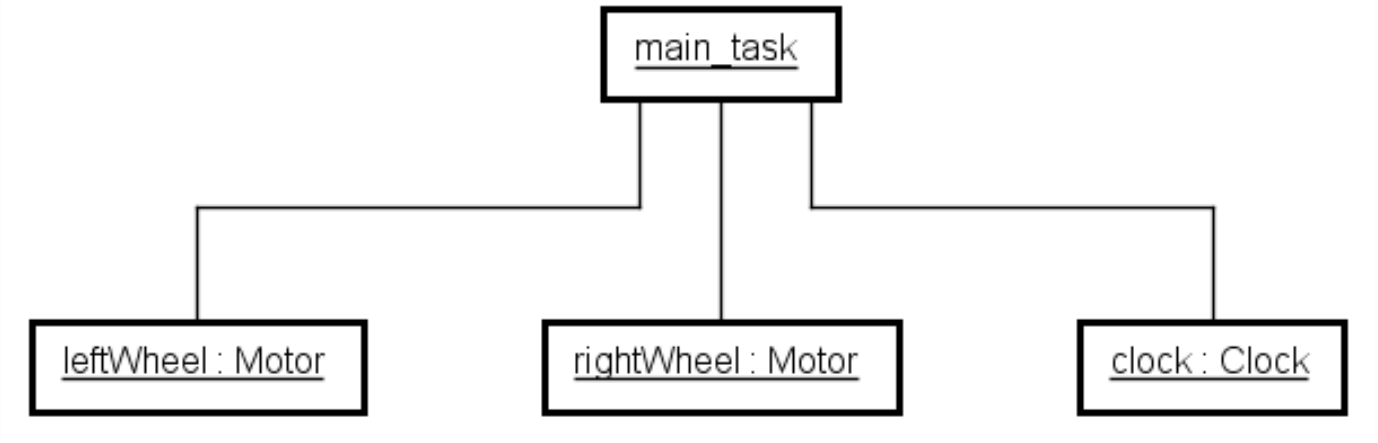

図 6. sample00 のオブジェクト図（クラスと関連づけ後）

クラス図に表してみる

クラス図を作成して、インスタンス同士の関係をクラス同士の関係で表してみましょう。

1. 「図」→「クラス図」でクラス図を追加し、図の名前を sample00のクラス図 にします
2. クラス図のパレットから「クラス」を選択し、アイコンが反転したら、マウスをダイアグラムエディター上でクリックします
3. 「クラス0」を、 sample00.cpp で便宜上クラスとして扱うことにした main_task に変更します
4. 「構造ツリー」の ev3api パッケージを展開して Motor クラスをダイアグラムエディターにドラッグ＆ドロップします
5. 同様にして Clock クラスを配置します
6. 使っている側のクラスから、使われている側のクラスへ、参照に使っているインスタンスごとに単方向関連を引きます
7. 関連の端にマウスを移動すると「→」がポップアップするので、 leftWheel などの関連端名をつけます
8. 「その他の表示/非表示」→「名前空間の表示」→「親の表示」を設定するとクラス名の前にパッケージの名前が表示されます

作成した sample00 のクラス図

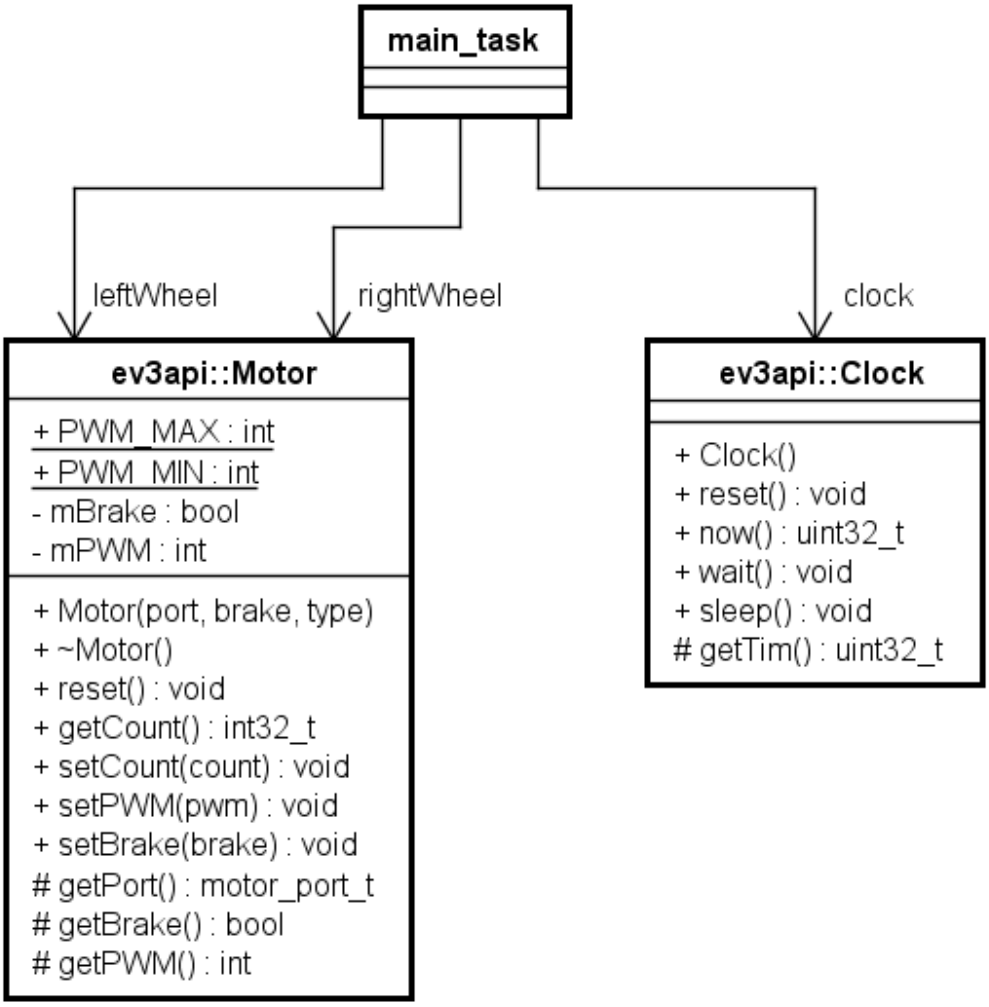

図 7. `sample00` のクラス図

作成した図から何が言えるでしょうか

- どんなクラスを使っているかわかりますか
- どんなクラスのインスタンス（オブジェクト）を使っているかわかりますか
- どんなことがやりたいシステムかわかりますか
- どんな処理をするシステムかわかりますか

などなど考えてみたところ...

sample00 は:

`Motor` クラスと `Clock` クラスを使っているアプリケーションであるというほかは、どのようなことをやりたいのか書いていないコードになっている

ということがわかりました。

システムの処理を担当するクラスを追加する

決まった走行（前進・後退を繰り返す）をロボットにさせたいのに、 sample00 にはそのことがわかるクラスがありません。

> このロボットがやる仕事を担当するクラスを作って、仕事の担当者としての名前をつけてみましょう。

クラス図にWalkerクラスを追加する

sample00 のクラス図を元に、 Walker クラスを追加し、 run メソッドを追加した sample01 のクラス図を作成しましょう。

1. 「ファイル」→「プロジェクトの別名保存」で sample01 として保存します（ファイル名は sample01.asta になります）
2. クラス図の名前を sample01のクラス図 に変更します
3. クラス図のパレットからクラスをドラッグ＆ドロップして、新しいクラス Walker を追加します
4. Walker クラスにコンストラクタと run メソッドを追加します
   - デストラクタは作成せず、コンパイラに任せることにします
5. pwm 、 duration は Walker クラスの属性にします
6. main_task は自分の仕事を Walker に移譲するので、関連を引き直します
   - Motor クラス、 Clock クラスは Walker クラスと一緒に使うので、コンポジションにしておきます

作成したsample01のクラス図

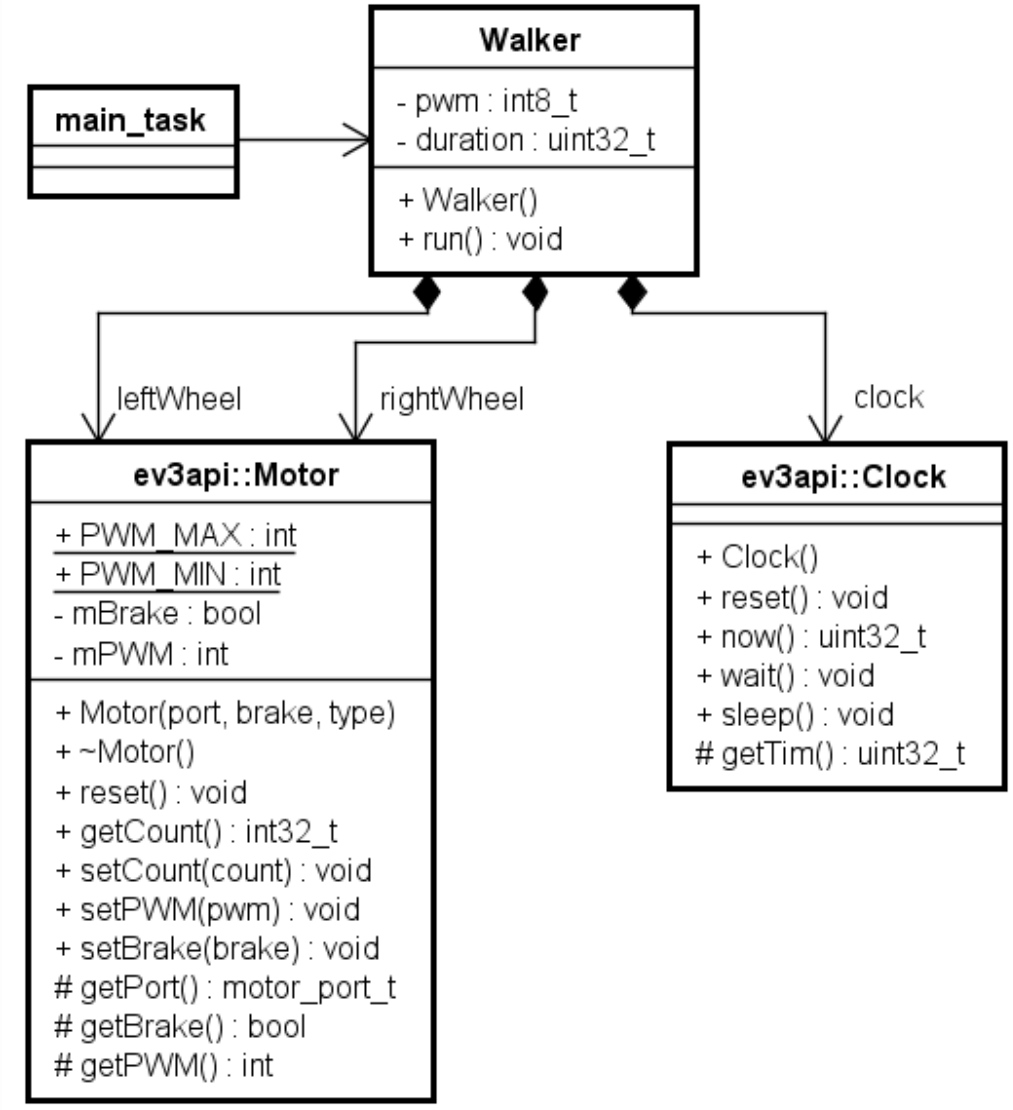

図 8. `sample01` のクラス図

sample01のコードを作成する

モデル図に合わせて、コードを変更しましょう。

1. サンプルコードの `sample00` ディレクトリをそっくりコピーして `sample01` ディレクトリを作りましょう
2. ファイルは分割しないで、 `app.cpp` の中に `Walker` クラスを作成しましょう
3. クラス図に従って `Walker` クラスを作成します
4. `Walker` クラスの `run` メソッドに `main_task` の処理を移動します
5. `main_task` は `Walker` クラスのインスタンスの作成と `run` メソッドの呼び出しを担当します

作成したコードは次ページ以降に掲載してあります。

コードが作成できたら、ビルドして、動作を確認しましょう。

```
$ cd （EV3RTのインストールディレクトリ）/hrp2/sdk/beginners
$ make app=sample01
```

リスト 2. sample01/app.cpp （その１）

```
#include "app.h"
#include "util.h"

#include "Motor.h"
#include "Clock.h"

using namespace ev3api;

class Walker {
public:
  Walker();
  void run();

private:
  Motor leftWheel;              (1)
  Motor rightWheel;             (1)
  Clock clock;                  (1)

  const int8_t pwm = (Motor::PWM_MAX) / 6;
  const uint32_t duration = 2000;
};
```

(1) Motor クラスと Clock クラスのインスタンスは、 Walker のインスタンスと共に作成・破棄する

リスト 3. sample01/app.cpp （その２）

```
Walker::Walker():
  leftWheel(PORT_C), rightWheel(PORT_B) {
}

void Walker::run() {
  init_f(__FILE__);
  while(1) {
    msg_f("Forwarding...", 1);
    leftWheel.setPWM(pwm);
    rightWheel.setPWM(pwm);
    clock.sleep(duration);

    msg_f("Backwarding...", 1);
    leftWheel.setPWM(-pwm);
    rightWheel.setPWM(-pwm);
    clock.sleep(duration);

    // 左ボタンを長押し、それを捕捉する
```

```
    if (ev3_button_is_pressed(LEFT_BUTTON)) {
      break;
    }
  }

  msg_f("Stopped.", 1);
  leftWheel.stop();
  rightWheel.stop();
  while(ev3_button_is_pressed(LEFT_BUTTON)) {
    ;
  }
}
```

リスト 4. sample01/app.cpp （その３）

```
void main_task(intptr_t unused) {

  Walker walker;                ①

  walker.run();                 ②

  ext_tsk();
}
```

① Walker クラスのインスタンスを作成

② run メソッドを実行

ここまでのまとめ

sample01 のコードとモデル図を作成しました。

この演習から何が言えるでしょうか。

- ロボットがやりたいことを担当するクラスを追加しました
  - 図やコードを読んだ時に、何がしたいのかわかるようになりました
  - やりたいことに「名前」がつきました
  - Walker クラスの run メソッドに処理を走行の詳細を閉じ込めることができました
- main_task は Walker クラスを使うだけになりました
- クラス図を変更し、それに合わせてコードを作成しました
  - クラス図とコードが対応していて、どちらで検討しても他方でも辻褄が合うようになりました

Walker クラスを別ファイルに分割してみよう:

sample01 では、演習を簡便に済ませるために、 app.c ファイルの中に Walker クラスを作りました。 さらに進めて、 Walker クラスを別ファイルに分割した場合のサンプル sample01_01 を用意してあります。参考にしてみてください。

Walkerクラスの課題を考えよう

この節は、技術教育の中では紹介だけにします。演習時間に余裕があるときに実施してください。
この章（ text01 ）の終わりへ

sample01 では、決まった走行（前進・後退を繰り返す）をロボットにさせるために、その動作を担当する Walker クラスを作成し、決まった走行をするという動作を担当する run メソッドを用意しました。

それでは、 sample01 のクラス図とコードをもう一度よく見てみましょう。決まった走行としてやりたかったこと（前進・後退を繰り返す）がわかるでしょうか。

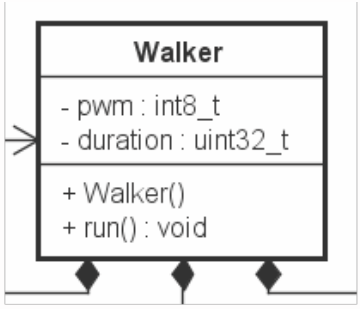

図 9. sample01 のクラス図（ Walker クラス部分の抜粋）

このクラス図を見ても、「前進する」「後退する」といった動作があるとは分からないですね。

リスト 5. sample01/app.cpp （ run メソッドの冒頭部分の抜粋）

```
1.  void Walker::run() {
2.    init_f(__FILE__);
3.    while(1) {
4.      msg_f("Forwarding...", 1);
5.      leftWheel.setPWM(pwm);
6.      rightWheel.setPWM(pwm);
7.      clock.sleep(duration);
8.
9.      msg_f("Backwarding...", 1);
10.     leftWheel.setPWM(-pwm);
11.     rightWheel.setPWM(-pwm);
12.     clock.sleep(duration);
13.
14.     // 左ボタンを長押し、それを捕捉する
```

このコードを見ても、メッセージはあるものの、「前進する」「後退する」といった動作がどの部分なのかわからないですね。

Walkerクラスに操作を追加する

`sample01` の `Walker` クラスでは、「前進する」「後退する」といった動作がわかりませんでした。 動作がわかるようにするには、該当する処理を「前進する」など **動作の名前** で呼ぶことができればよい、つまり **メソッド** にすればよいですね。

`sample01` のクラス図を元に、 `Walker` クラスに `forward` メソッドなどを追加した `sample02` のクラス図を作成しましょう。

1. 「ファイル」→「プロジェクトの別名保存」で `sample02` として保存します（ファイル名は `sample02.asta` になります）
2. クラス図の名前を `sample02のクラス図` に変更します
3. `Walker` クラスに「前進する」 `forward` メソッドを追加します
   - 「後退する」 `back` メソッド、「停止する」 `stop` メソッドも追加しておきましょう
4. 新しく追加した3つのメソッドは、可視性を `protected` にしておきます
   - 可視性を `protected` にすると、メッソド名の前に `#` がつきます
   - `protected` なメソッドは `Walker` クラスの外部からは呼び出せませんが、`Walker` クラスを継承したクラスは利用できるメソッドになります

作成したsample02のクラス図

下図は、 `Motor` クラスと `Clock` クラスの詳細を非表示にしてあります。

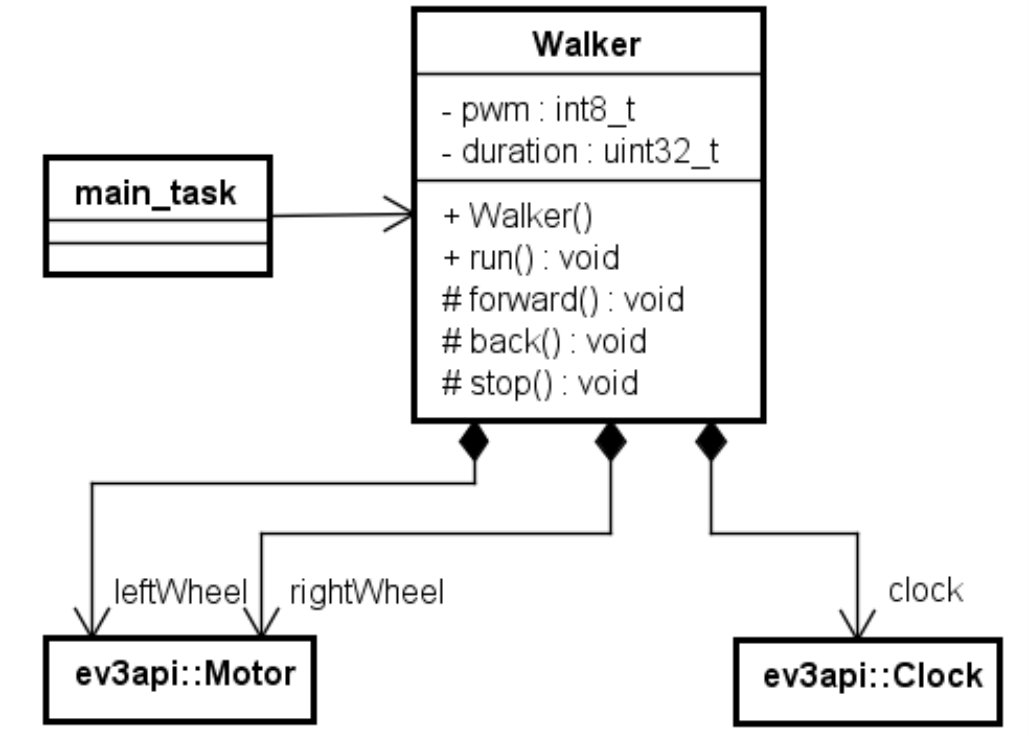

図 10. sample02 のクラス図

sample02のコードを作成する

モデル図に合わせて、コードを変更しましょう。

1. サンプルコードの sample01 ディレクトリをそっくりコピーして sample02 ディレクトリを作りましょう
2. ファイルは分割しないで、 app.cpp の中に Walker クラスを作成しましょう
3. クラス図に従って Walker クラスを修正します
4. run メソッドの前進している処理を抜き出して forward メソッドを作ります
5. 同様にして、 back メソッド、 stop メソッドを作ります
6. 追加した操作を使って run メソッドを修正します

作成したコードは次ページ以降に掲載してあります。

コードが作成できたら、ビルドして、動作を確認しましょう。

```
$ cd （EV3RTのインストールディレクトリ）/hrp2/sdk/beginners
$ make app=sample02
```

冒頭は、 sample01 と同じです。

リスト 6. sample02/app.cpp （その 1 ）

```
class Walker {
public:
  Walker();
  void run();

private:
  Motor leftWheel;
  Motor rightWheel;
  Clock clock;

  const int8_t pwm = (Motor::PWM_MAX) / 6;
  const uint32_t duration = 2000;

protected:              (1)
  void forward(void);   (2)
  void back(void);      (2)
  void stop(void);      (2)
};
```

(1) protected な属性や操作は、 protected: アクセス指定子から始まる領域に宣言します

(2) forward 、 back 、 stop メソッドを宣言しています

リスト 7. sample02/app.cpp （その２）

```
Walker::Walker():
  leftWheel(PORT_C), rightWheel(PORT_B) {
}

void Walker::forward(void) {  (1)
  msg_f("Forwarding...", 1);
  leftWheel.setPWM(pwm);
  rightWheel.setPWM(pwm);
}

void Walker::back(void) {  (1)
  msg_f("Backwarding...", 1);
  leftWheel.setPWM(-pwm);
  rightWheel.setPWM(-pwm);
}

void Walker::stop(void) {  (1)
  msg_f("Stopped.", 1);
  leftWheel.stop();
  rightWheel.stop();
}
```

(1) forward 、 back 、 stop メソッドを実装しています

リスト 8. sample02/app.cpp （その３）

```
1.  void Walker::run() {
2.    init_f(__FILE__);
3.    while(1) {
4.      forward();                ①
5.      clock.sleep(duration);
6.      back();                   ①
7.      clock.sleep(duration);
8.
9.      // 左ボタンを長押し、それを捕捉する
10.     if (ev3_button_is_pressed(LEFT_BUTTON)) {
11.       break;
12.     }
13.   }
14.
15.   stop();                     ①
16.   while(ev3_button_is_pressed(LEFT_BUTTON)) {
17.     ;
18.   }
19. }
```

① forward 、 back 、 stop メソッドを使って run メソッドを実装しています

main_task は sample01 と同じです。

ここまでのまとめ

sample02 のコードとモデル図を作成しました。

この演習から何が言えるでしょうか。

- ロボットの動作の詳細をクラスのメソッドとして追加しました
  - 詳細な動作に「名前」がついて、目に見えるようになり、その名前で呼べるようにしました
- run メソッドの処理が、追加したメソッドによって、よりわかりやすくなりました
  - 図やコードを読んだ時に、どのような動作があるかが、 sample01 よりも明確になりました
- クラス図を変更し、それに合わせてコードを作成しました
  - sample01 から sample02 へ修正しても、クラス図とコードが対応していて、どちらで検討しても他方でも辻褄が合うことがわかります

本資料について

資料名： 要素技術とモデルを開発に使おう: コードとモデル図を対応づけてみよう （技術教育資料）

この文書は、技術教育「要素技術とモデルを開発に使おう」に使用するETロボコン公式トレーニングのテキストです。

2.3, 2019-04-12 16:46:58, 2019年用

2.3
Last updated 2018-05-15 15:41:53 +0900