

【ねらい】

コンストラクタとインスタンス生成について理解します。また、`null` リテラルについても理解します。

1 コンストラクタとは

コンストラクタは、`new` 演算子によってインスタンスが作られるときに実行される特殊なメソッドです。コンストラクタはクラスと同じ名前で作られ、通常新しいインスタンスのメンバを初期化するのに用いられます。次の例ではフィールドの定義に続いて `Train` クラスのコンストラクタを定義しています。

<pre>class Train { public bool Express; public Directions Direction; public Stations CurrentStation; public Train() { Express = false; Direction = Directions.下り; CurrentStation = Stations.東京; } }</pre>	<p><code>Train</code> クラスの定義。</p> <p>列車が急行かどうかを表すフィールド。 現在の進行方向を表すフィールド。 現在の停車駅を表すフィールド。</p> <p><code>Train</code> クラスのコンストラクタ。</p> <p><code>Express</code> フィールドの初期化。 <code>Direction</code> フィールドの初期化。 <code>CurrentStation</code> フィールドの初期化。</p>
--	---

`new` 演算子によって `Train` クラスの新しいインスタンスが作られる時に、このコンストラクタが呼び出されます。上の例では `Train` クラスのインスタンス生成時に、フィールドが決められた値で初期化されるようにしています。

コンストラクタは戻り値を返しません、キーワードの `void` は指定しません。上の例ではコンストラクタは引数を持っていませんが、引数を持つコンストラクタを定義することもできます。コンストラクタをオーバーロードして、引数の数や型が異なる複数のコンストラクタを同時に定義することもできます。たとえば上記コンストラクタに加えて、引数の数が異なる次のコンストラクタを定義することもできます。

<pre>public Train(bool _express) {</pre>	<p>急行かどうかを指定してインスタンスを生成するコンストラクタ。(進行方向と現在の駅は固定値。)</p>
--	---

<pre> Express = _express; Direction = Directions.下り; CurrentStation = Stations.東京; } public Train(bool _express, Directions _direction, Stations _currentStation) { Express = _express; Direction = _direction; CurrentStation = _currentStation; } </pre>	<p>Express フィールドの初期化。</p> <p>Direction フィールドの初期化。</p> <p>CurrentStation フィールドの初期化。</p> <p>急行かどうか、進行方向、現在の停車駅のすべてを指定してインスタンスを生成するコンストラクタ。</p> <p>Express フィールドの初期化。</p> <p>Direction フィールドの初期化。</p> <p>CurrentStation フィールドの初期化。</p>
--	---

※サンプル プログラム

サンプル プログラムの「Application1」プロジェクトにこのサンプルがあります。

これらコンストラクタを利用する場合には、`new` 演算子によるインスタンス生成の際に、引数を指定するようにします。引数の型や数に応じたコンストラクタが呼び出されます。

<pre> Train train1 = new Train(true); Train train2 = new Train(false, Directions.上り, Stations.京都); </pre>	<p>Train クラスのインスタンス生成 (引数 1 つ)。</p> <p>Train クラスのインスタンス生成 (引数 3 つ)。</p>
--	---

クラスにコンストラクタが定義されていない場合には、引数を持たない既定のコンストラクタが自動的に追加されます。既定のコンストラクタでは、フィールドは既定値 (`int` 型であれば `0` など) で初期化されます。

あまり使用されませんが、コンストラクタに `static` キーワードを指定した、静的コンストラクタも定義できます。静的コンストラクタは、最初のインスタンスを作成する前か、または静的メンバが参照される前に自動的に呼び出されます。静的コンストラクタは静的フィールドを初期化する際などに使用します。

2 null リテラル

コンストラクタから離れますが、`null` リテラルについてここで紹介しておきます。参照型の変数に対して、この変数が参照するオブジェクトがない状態を指定したい場合には `null` リテラルを指定します。

<pre> Train train3 = null; </pre>	<p>変数 <code>train3</code> に <code>null</code> リテラルを代入。</p>
-----------------------------------	--

`null` は参照型の既定値となっているため、たとえば初期値を指定せずにクラスの配列を宣言した場合などに、`null` が既に代入されている場合があります。下の例では `trainArray[0]` などの要素はすべて `null` になります。

```
Train[] trainArray = new Train[5];
```

Train 型の配列 trainArray の生成。

上の例で trainArray[0] のメンバである trainArray[0].Direction などをおのまま参照した場合には、NullReferenceException という実行時例外が発生し、エラーになります。参照型の変数の値が null である可能性がある場合には、次の例のように事前に確認してからメンバを参照するようにします。

```
if (train != null)
{
    Directions direction = train.Direction;
}
```

変数 train の値が null でない場合。

train の Direction を変数 direction に代入。

◆ 演習課題

- 次のようなパズルを実施するクラスをコーディングします。

テーブルの上に正方形型にコインが並べられています。初期状態ではすべてのコインの裏面が上を向いています。この中からあるコインを選択して、そのコインと同じ行か列にあるすべてのコインをひっくり返します。

例) コインの表面を○で、裏面を×で表しています。

```
× × × ×
× × × ×
× × × ×
× × × ×
```

↓ (1, 1) を選択

```
× ○ × ×
○ ○ ○ ○
× ○ × ×
× ○ × ×
```

↓ (0, 3) を選択

```
○ × ○ ○
○ ○ ○ ×
× ○ × ○
× ○ × ○
```

パズルの目的はこの操作を繰り返して、できるだけ早くすべてのコインを表向きにすることです。この課題で作成するプログラムは、このパズルを自分で解くプログラムではなく、利用者がパズルを解きやすいように、利用者の操作に応じてコインをひっくり返して表示するプログラムです。そのために現在のコインの状態を表すクラス Board をコーディングします。

Board クラスには、現在のコインの状態をさきほどの例のような○×の文字列にして返すメソッド GetString と、コインの位置を指定された場合にそのコインと同じ行か列にあるコイン

をひっくり返す操作を行うメソッド **Play** を実装してください。現在のコインの状態を記憶するために、2次元配列の内部フィールドも **Board** 内に持つ必要があります。

さらに **Board** クラスのコンストラクタもコーディングしてください。コンストラクタでは、縦横に並べるコインの数を指定できるようにします。つまり、**new Board(5)** で 5 行 5 列に裏向きにコインが並んだ状態のインスタンスを生成できるようにしてください。

※解答例

サンプル プログラムに演習の解答例があります。