Ｃ＃の基礎について②

コンストラクタとデストラクタ

発表の目的

学ばせてもらったことをどれぐらい理解できているか。
理解した内容はあっているか。
確認と理解を深めることを目的としています。

アジェンダ

1. コンストラクタ

・コンストラクタ
・デフォルトコンストラクタ

2. デストラクタ

3. ラムダ式

1.コンストラクタ

- コンストラクタ
  クラスの生成時に実行される関数。

- コンストラクタの特徴
  ・コンストラクタを定義する場合は、
  クラスと同名の関数を作成する。
  ・定義がない場合、
  デフォルトコンストラクタが実行される。

1.コンストラクタ

- コンストラクタの定義

```
class  Color
{
    public  int  r;
    public  int  g;
    public  int  b;

    Color()
    {  r  =  g  =  b  =  0;  }

    Color(int  n)
    {  r  =  g  =  b  =  n;  }

    Color(int  red,  int  green;  int  blue;)
    {
      r  =  red;
      g  =  green;
      b  =  blue;
    }
}
```

1.コンストラクタ

- コンストラクタの呼び出しと実行

```
class  Test
{
  static  void  Main()
  {
      Color  color    =  new  Color();
      Color  color2   =  new  Color(5);
      Color  color3   =  new  Color(1,2,3);
  }
}
```

Color()で初期化

Color(int)で初期化

Color(int,int,int)で初期化

1.コンストラクタ

コンストラクタについては以上です。
次はデフォルトコンストラクタについて。

1.コンストラクタ

- デフォルトコンストラクタ
  コンストラクタを定義しない場合、
  デフォルトコンストラクタが呼ばれ、実行される。

- デフォルトコンストラクタの特徴
  コンストラクタを定義しない場合に、
  自動的に作成されるが、コンストラクタを一つでも
  定義すると、デフォルトコンストラクタは
  自動的に作成されないため、定義の省略ができない。

1.コンストラクタ

- デフォルトコンストラクタ

```
class  Color
{
}
class  Test
{
  static  void  Main()
  {
    Color  color  =  new  Color();
  }
}
```

1.コンストラクタ

```
class  Color
{
  Color(){ }
}
class  Test
{
  static  void  Main()
  {
    Color  color  =  new  Color();
  }
}
```

1.コンストラクタ

```
class  Color
{
  Color(int  r){ //  処理 }
}
class  Test
{
  static  void  Main()
  {
    Color  color  =  new  Color();
  }
}
```

1.コンストラクタ

```
class Color
{
  Color(int r){ // 処理 }
  Color(){ }
}
class Test
{
  static void Main()
  {
    Color color = new Color();
  }
}
```

1.コンストラクタ

デフォルトコンストラクタについては以上です。
次はデストラクタについて。

アジェンダ

1. コンストラクタ
  - コンストラクタ
  - デフォルトコンストラクタ
2. **デストラクタ**
3. ラムダ式

2.デストラクタ

- デストラクタ

インスタンスが破棄される時に実行される関数。

- デストラクタの特徴

インスタンスの寿命は
.NET Framework自体が管理をしてるため、
いつ破棄が行われるかわかりません。
そのため、Ｃ＃ではデストラクタを書かれることは
あまりありません。
領域の解放はデストラクタで明記しなくても
ガベージコレクション(garbage collection)が行います。

2.デストラクタ

```
class  Color
{
  Color(){ }    //コンストラクタ
  ~Color(){ }  //デストラクタ
}
class  Test
{
  static  void  Main()
  {
    Color  color  =  new  Color();
  }
}
```

2.デストラクタ

デストラクタについては以上です。

次はラムダ式について。

アジェンダ

1. コンストラクタ
   - コンストラクタ
   - デフォルトコンストラクタ
2. デストラクタ
3. **ラムダ式**

3.ラムダ式

ラムダ式については長くなるため、
今回は一部分のみのまとめとしました。

- ラムダ式とは

アルゴリズムと高階関数をより簡単に扱うことを
主目的として導入されたもの。

「ラムダ式によるデザインパターン in C#」より一部抜粋。

3.ラムダ式

- ラムダ式について

ラムダ式はその名の通り、式の表記になります。
ラムダ式自体を調べると、無名関数や匿名関数などの名前を目にしますが、
ラムダ式はラムダ演算子を使用した式をさします。
ラムダ演算子は「**=>**」この記載です。
ラムダ演算子を使用して処理を記載することにより、
無名関数や匿名関数と呼ばれるものになります。

3.ラムダ式

```
x  =>  x  *  x
```

左辺値のパラメータxを２乗する内容です。
単純な内容ですが、
それでも関数にすると最低でも数行必要になります。
それをラムダ演算子を使用すれば短く、
明示的に書くことができます。

3.ラムダ式

```
Function((x,y) => x * y);
```

これでは計算するだけです。
答えの戻り値は以下のように受け取ります。

```
Function(int n)
Function f = (x,y) => { x * y };
```

これでパラメータx yの計算後の値がnumに代入されます。

Ｃ＃の基礎について②

本日の発表は以上です。