C++ゲームプログラミング講習
上級 第4回

クラス

クラスの書き方

```
1  class Hoge {↓
2      private int width;↓
3      private int height;↓
4      public int x;↓
5      public int y;↓
6      ↓
7      public Hoge() {↓
8          System.out.println("Hogeが生成されました");↓
9      }↓
10     ↓
11     public Hoge(int x1, int y1) {↓
12         x = x1;↓
13         y = y1;↓
14         System.out.printf(↓
15             "Hogeがx=" + x + " y=" + y + "で生成されました");↓
16     }↓
17     ↓
18     private void Method() {↓
19         System.out.println("Method");↓
20     }↓
21     ↓
22     public void Update() {↓
23         System.out.println("Update");↓
24     }↓
25     ↓
26 }[EOF]
```

クラスの書き方

```
3  class Hoge {
4  private :
5      int width;
6      int height;
7
8      void Method() {
9          printf("Method¥n");
10     }
11
12 public :
13     int x;
14     int y;
15
16     Hoge() {
17         printf("Hogeが生成されました¥n");
18     }
19
20     Hoge(int x1, int y1) {
21         x = x1;
22         y = y1;
23         printf("Hogeがx=%d y = %d で生成されました¥n", x, y);
24     }
25
26     void Update() {
27         printf("Update¥n");
28     }
29 };
```

クラスの書き方

```
class Hoge {
private :
    int width;
    int height;

    void Method() {
        printf("Method¥n");
    }

public :
    int x;
    int y;

    Hoge() {
        printf("Hogeが生成されました¥n");
    }

    Hoge(int x1, int y1) {
        x = x1;
        y = y1;
        printf("Hogeがx=%d y = %d で生成されました¥n", x, y);
    }

    void Update() {
        printf("Update¥n");
    }
};
```

クラスの書き方

```
3  class Hoge {
4  private :
5      int width;
6      int height;
7
8      void Method() {
9          printf("Method¥n");
10     }
11
12 public :
13     int x;
14     int y;
15
16     Hoge() {
17         printf("Hogeが生成されました¥n");
18     }
19
20     Hoge(int x1, int y1) {
21         x = x1;
22         y = y1;
23         printf("Hogeがx=%d y = %d で生成されました¥n", x, y);
24     }
25
26     void Update() {
27         printf("Update¥n");
28     }
29 };
```

クラスの書き方

```
3  class Hoge {
4  private :
5      int width;
6      int height;
7
8      void Method() {
9          printf("Method¥n");
10     }
11
12 public :
13     int x;
14     int y;
15
16     Hoge() {
17         printf("Hogeが生成されました¥n");
18     }
19
20     Hoge(int x1, int y1) {
21         x = x1;
22         y = y1;
23         printf("Hogeがx=%d y = %d で生成されました¥n", x, y);
24     }
25
26     void Update() {
27         printf("Update¥n");
28     }
29 };
```

インスタンスの生成

- 静的生成
- 動的生成

インスタンスの静的生成

- コンパイル時にオブジェクトが生成される

```
42      Hoge hoge1;
43
44      Hoge hoge2(10, 20);
45      hoge1.Update();
```

インスタンスの静的生成

- コンパイル時にオブジェクトが生成される

```
42      Hoge hoge1;
43
44      Hoge hoge2(10, 20);
45      hoge1.Update();
```

```
Hogeが生成されました
Hogeがx=10 y = 20 で生成されました
Update
```

インスタンスの静的生成

- コンパイル時にオブジェクトが生成される

```
42      Hoge hoge1;
43
44      Hoge hoge2(10, 20);
45      hoge1.Update();
```

```
Hogeが生成されました
Hogeがx=10 y = 20 で生成されました
Update
```

- new演算子を使ってないのに既にインスタンスが作られている
- コンストラクタで引数を受け取りたいときもこんな感じ
- 変数や関数へのアクセスは**ドット(.)** を使用

クラスの動的生成

- プログラムの実行時に生成される

```
42      Hoge *hoge1;
43      Hoge *hoge2;
44
45      printf("まだ生成されてない¥n");
46
47      hoge1 = new Hoge();
48      hoge2 = new Hoge(10, 20);
49
50      hoge1->x = 0;
51      hoge1->y = 5;
52      hoge1->Update();
```

クラスの動的生成

- プログラムの実行時に生成される

```
42      Hoge *hoge1;
43      Hoge *hoge2;
44
45      printf("まだ生成されてない¥n");
46
47      hoge1 = new Hoge();
48      hoge2 = new Hoge(10, 20);
49
50      hoge1->x = 0;
51      hoge1->y = 5;
52      hoge1->Update();
```

```
まだ生成されてない
Hogeが生成されました
Hogeがx=10 y = 20 で生成されました
Update
```

クラスの動的生成

- プログラムの実行時に生成される

```
42      Hoge *hoge1;
43      Hoge *hoge2;
44
45      printf("まだ生成されてない¥n");
46
47      hoge1 = new Hoge();
48      hoge2 = new Hoge(10, 20);
49
50      hoge1->x = 0;
51      hoge1->y = 5;
52      hoge1->Update();
```

・クラスの**ポインタ**を用意する
・**new演算子**でインスタンスを生成する
・変数や関数へのアクセスは **->** を使用する

new演算子

- Javaでも使うnew演算子
- メモリを確保しインスタンスを展開する
- コンストラクタを実行
- インスタンスのポインタを返す演算子

デストラクタ

- **コンストラクタ**→インスタンスが生成されたとき自動で呼ばれる関数
- **デストラクタ**→インスタンスが削除されたとき自動で呼ばれる関数

デストラクタ

- **コンストラクタ**→インスタンスが生成されたとき自動で呼ばれる関数
- **デストラクタ**→インスタンスが削除されたとき自動で呼ばれる関数

delete演算子

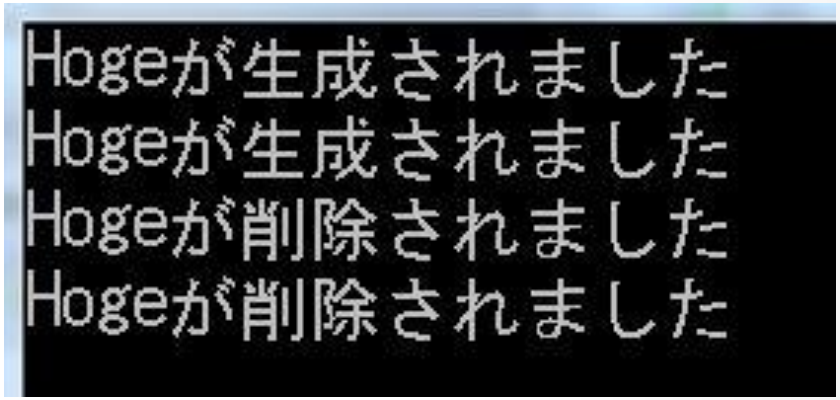
- C++には**GC(ガーベジコレクション)**がない
- メモリの管理は自分でしないといけない
- new演算子で確保したメモリは**delete演算子**で開放する

```
27      Hoge *hoge1;
28      Hoge *hoge2;
29
30      hoge1 = new Hoge();
31      hoge2 = new Hoge();
32
33      delete hoge1;
34      delete hoge2;
```

delete演算子

- C++には**GC(ガーベジコレクション)**がない
- メモリの管理は自分でしないといけない
- new演算子で確保したメモリは**delete演算子**で開放する

```
27      Hoge *hoge1;
28      Hoge *hoge2;
29
30      hoge1 = new Hoge();
31      hoge2 = new Hoge();
32
33      delete hoge1;
34      delete hoge2;
```



まとめ

- private : の下に書くとprivate
- public : の下に書くとpublic

- Hoge hoge; で静的生成。ドットでアクセス
- Hoge *hoge = new Hoge(); で動的生成。-> でアクセス

- new演算子はインスタンスのポインタを返す
- メモリを確保したらdelete演算子で削除

課題

- 前々回作った、3パターンの敵をクラス化する

- ヒント :
- 敵1をEnemyA, 敵2をEnemyB, 敵3をEnemyCとしよう
- 考えられる変数は？(x座標、y座標、画像のhandle....
- 描画の処理をvoid Draw()
- 更新(移動)の処理をvoid Update() にまとめよう

次回

- ポインタ、クラスと難しい話が続いたので
- 息抜きに、画像描画(エフェクトとか)についてやろうと思います