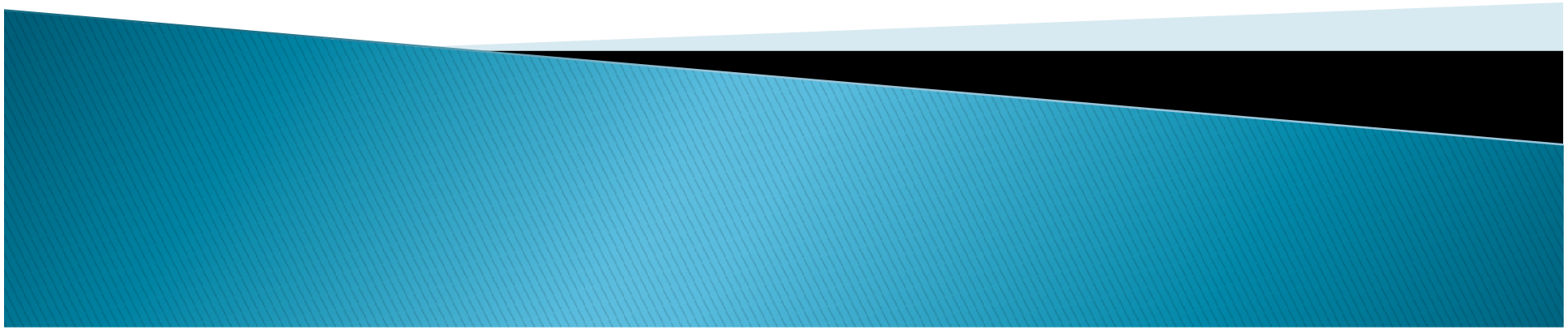
C#でゲームを作ろう2017

第6回　担当:ten



自己紹介

- ID:ten
- 京都大学工学部情報学科2回生
  - 計算機科学コース
- 第40代会長
- ゲーム制作して競プロしてる
- パズルと音ゲー

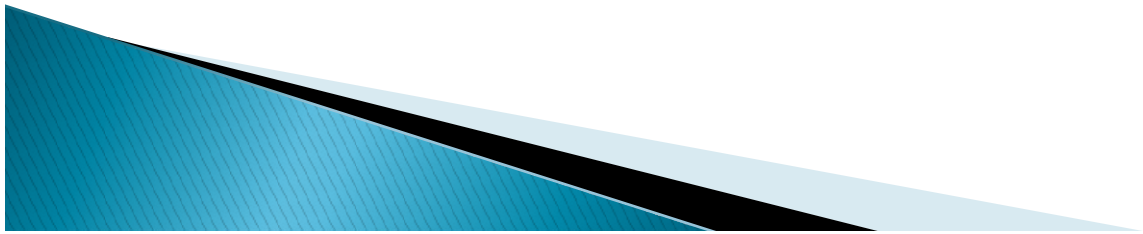
Twitter

Slackとか

自己紹介をしましょう（短縮版）

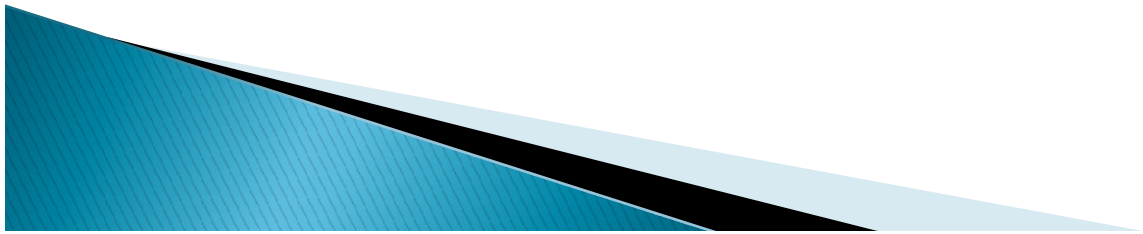
- KMC-ID (or 本名)
- 所属

- ※すべて任意です



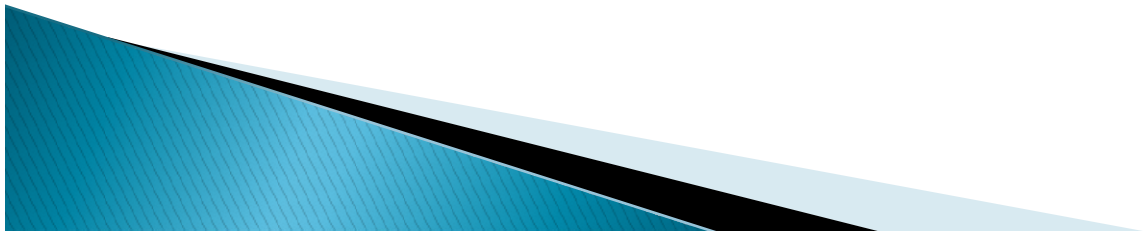
ゲームを考えてきてね

- 夏休み後あたりに発表会をする予定
- 夏休み前に内容を終えるので、そこから自分のゲームを作ってもらう予定
- というわけでゲームの内容を考えてきてください
- いわゆる「３分ゲー」のようなもので大丈夫です



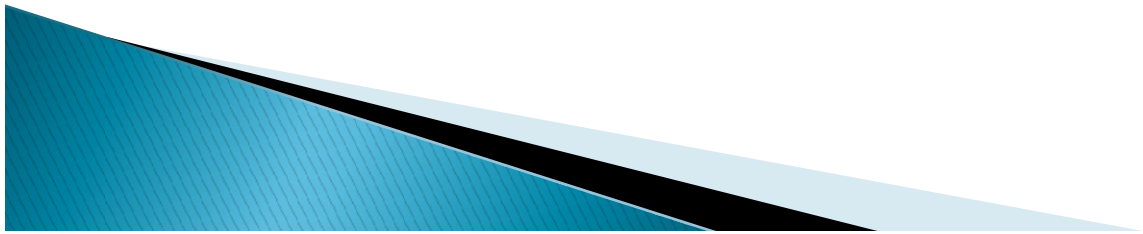
Slack

- #csgame



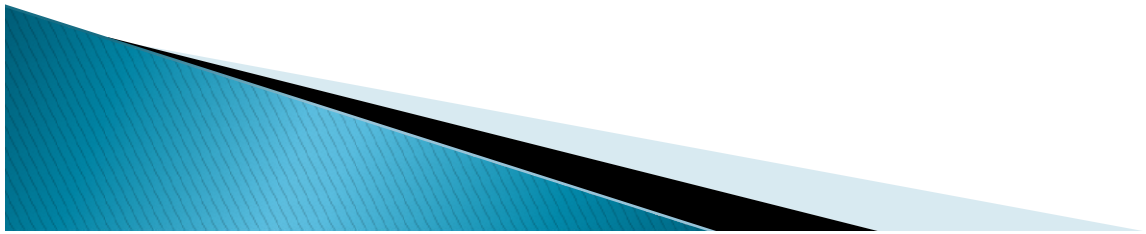
Github

- https://github.com/kmc-jp/csgame2017



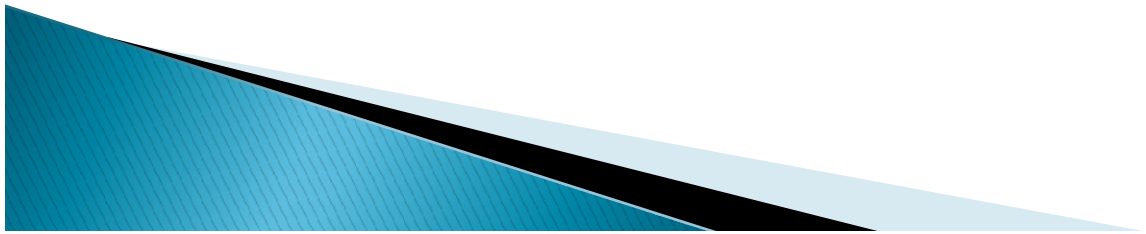
今日やること

- クラス



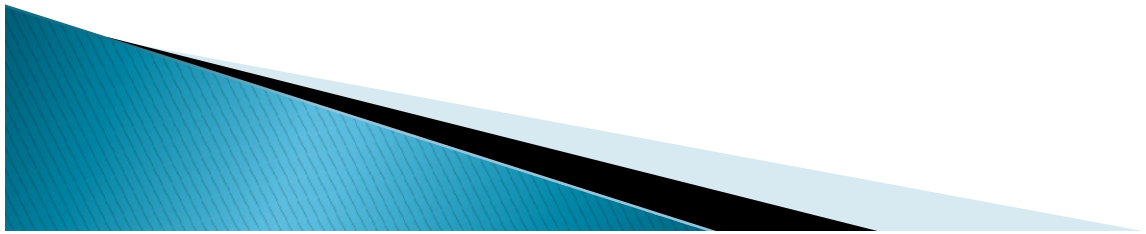
今日やること

- 今日は概念的な話になります
- シューティングは来週まで待って！！！



くらすのふくしゅー

- クラスって何だっけ



くらすのふくしゅー

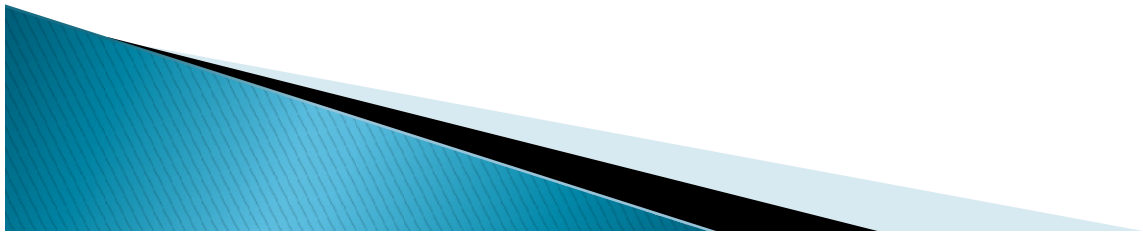
- こんなの。

```
class Enemy
{
    private int hp;

    public void Damage(int _damage)
    {
        hp -= _damage;
        Console.WriteLine("敵に" + _damage + "のダメージ！");
    }

    public Enemy(int _hp)
    {
        hp = _hp;
    }
}
```

くらすのふくしゅー

- Main関数内。

```
class Program
{
    static void Main(string[] args)
    {
        Enemy enemy = new Enemy(5000);
    }
}
```



publicとprivateって何

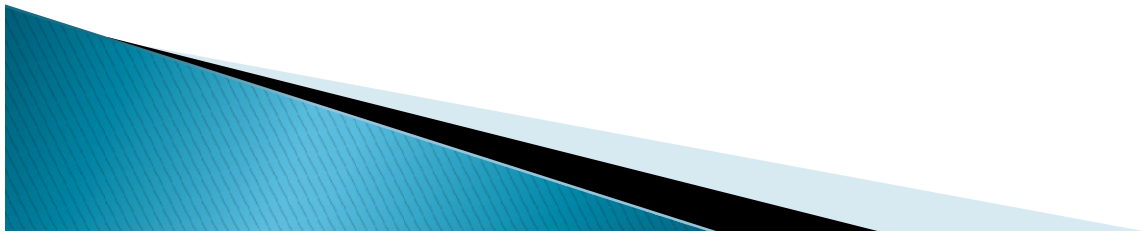
- これって何だろう

```
class Enemy
{
    private int hp;

    public void Damage(int _damage)
    {
        hp -= _damage;
        Console.WriteLine("敵に" + _damage + "のダメージ！");
    }

    public Enemy(int _hp)
    {
        hp = _hp;
    }
}
```

publicとprivateって何

- アクセス修飾子
- クラス外から変数や関数を操作できるか？

- public
  - 操作できる
- private
  - 操作できない



publicとprivateって何

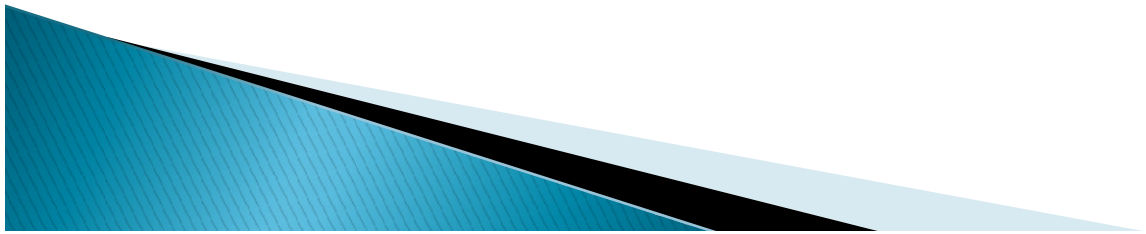
- hp
  - privateな変数→アクセスできない
- Damage
  - publicな関数→アクセスできる

```
static void Main(string[] args)
{
    Enemy enemy = new Enemy(5000);

    enemy.Damage(100);
    enemy.hp -= 10;
}
```

エラーが吐かれる

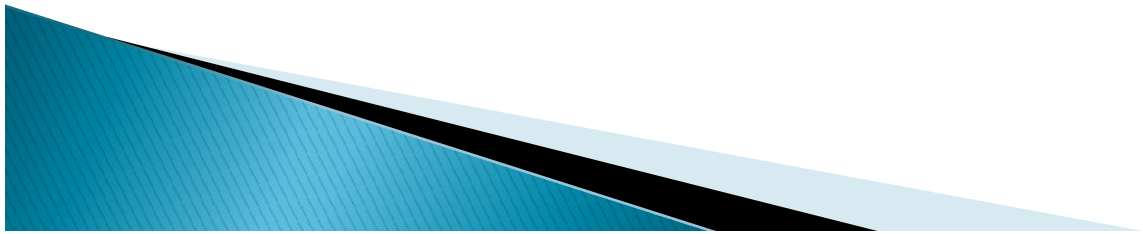
publicとprivateって何

- アクセス修飾子
- あとで増えます



継承

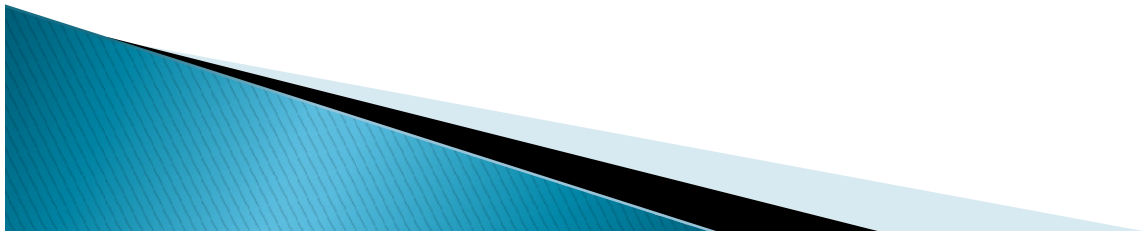
- 今日のメイン



継承

- さっきEnemyのクラスを作った
- 他にもスライムとかゾンビとか不死鳥とかを表すクラスを作りたい

- これらのクラスは
  Enemyの性質を持っていて欲しい
  - HPがあったり
  - ダメージを受けたり



継承

- そこで継承
- Enemyの持つ変数、関数を引き継ぎ、新たなクラスを作る

```
class Phoenix : Enemy
{
    private int revivalNum;

    public Phoenix(int _hp, int _revivalNum)
        :base(_hp){
            revivalNum = _revivalNum;
        }
}
```

継承

- 継承のしかた
- class クラス名 : 継承するクラス名

```
class Phoenix : Enemy
{
    private int revivalNum;

    public Phoenix(int _hp, int _revivalNum)
        :base(_hp){
            revivalNum = _revivalNum;
        }
}
```

継承

- コンストラクタの書き方
- public クラス名(引数)
  :base(継承元のクラスに渡す引数)

```
class Phoenix : Enemy
{
    private int revivalNum;

    public Phoenix(int _hp, int _revivalNum)
        :base(_hp){
            revivalNum = _revivalNum;
        }
}
```

継承

- 継承元のコンストラクタが呼ばれてから
  このクラスのコンストラクタが呼ばれる

```
class Phoenix : Enemy
{
    private int revivalNum;

    public Phoenix(int _hp, int _revivalNum)
        :base(_hp){
            revivalNum = _revivalNum;
        }
}
```

継承

- Main関数
- PhoenixはEnemyを継承している
  - Damage関数がそのまま使える

```
class Program
{
    static void Main(string[] args)
    {
        Phoenix phoenix = new Phoenix(500, 10);
        phoenix.Damage(600);
    }
}
```

継承

- 継承元のクラスの変数に入れられる
- えらい
- List<Enemy>に入れたりとか…

```
class Program
{
    static void Main(string[] args)
    {
        Enemy phoenix = new Phoenix(500, 10);
        phoenix.Damage(600);
    }
}
```

関数のオーバーライド

```
class Phoenix : Enemy
{
    private int revivalNum;

    public override void Damage(int _damage)
    {
        base.Damage(_damage);
        if (hp <= 0 && revivalNum > 0)
        {
            revivalNum--;
            hp += 100;
        }
    }

    public Phoenix(int _hp,int _revivalNum)
        :base(_hp){
            revivalNum = _revivalNum;
        }
}
```

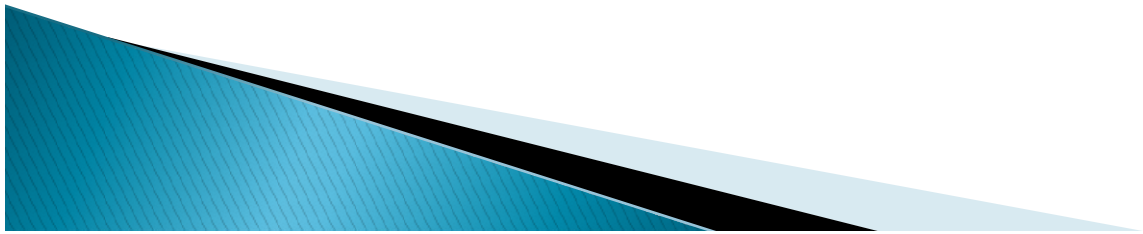
関数のオーバーライド

- Damage関数に追加で別の挙動をさせたい
  or　まったく別の挙動をさせたい
- そんなときにオーバーライド

```
public override void Damage(int _damage)
{
    base.Damage(_damage);
    if (hp <= 0 && revivalNum > 0)
    {
        revivalNum--;
        hp += 100;
    }
}
```

関数のオーバーライド

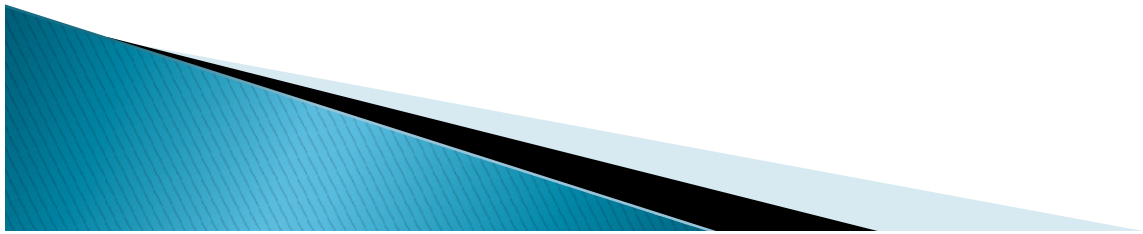
- Damage関数に追加で別の挙動をさせたい
  or　まったく別の挙動をさせたい
- そんなときにオーバーライド



関数のオーバーライド

- オーバーライドするには、継承元のクラスの関数を「virtual」にする必要がある

```
virtual public void Damage(int _damage)
{
    hp -= _damage;
    Console.WriteLine("敵に" + _damage + "のダメージ！");
}
```



関数のオーバーライド

- また、継承したクラスの関数には「override」をつける必要がある

```
public override void Damage(int _damage)
{
    base.Damage(_damage);
    if (hp <= 0 && revivalNum > 0)
    {
        revivalNum--;
        hp += 100;
    }
}
```

関数のオーバーライド

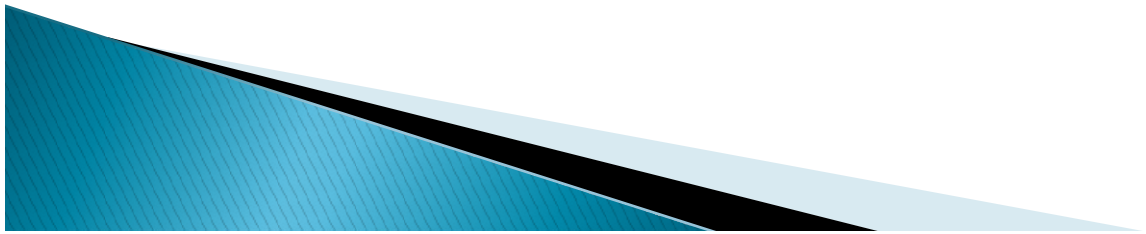
- base.Damage();
- 継承元のDamage関数を呼び出している

```
public override void Damage(int _damage)
{
    base.Damage(_damage);
    if (hp <= 0 && revivalNum > 0)
    {
        revivalNum--;
        hp += 100;
    }
}
```

abstract

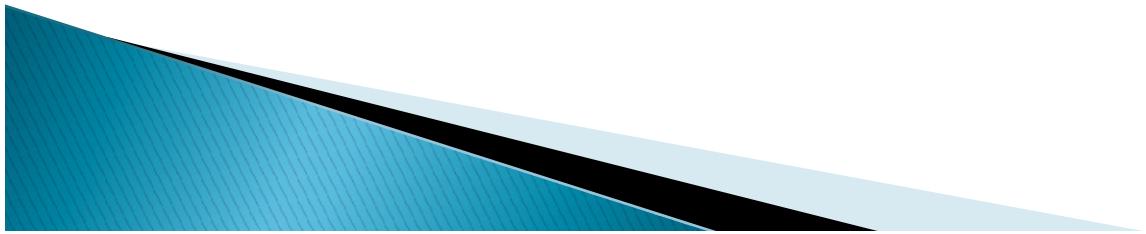
- 「virtual」の代わりに「abstract」にもできる
  - オーバーライドされる専用の関数になる]

- abstract class
  - 抽象クラス
  - すべてのメソッドが抽象でなければならない
  - classの前にabstractをつける



ところで

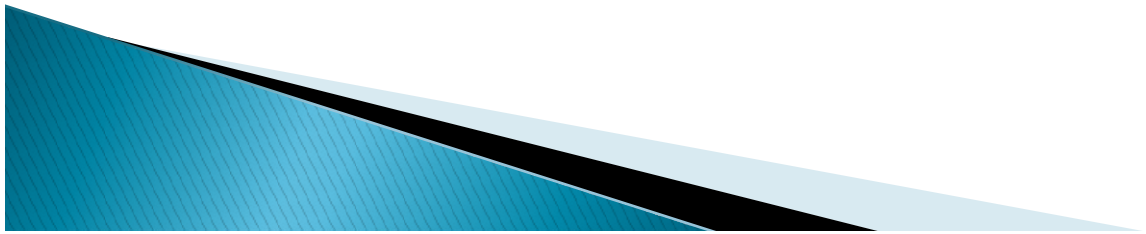
- Enemy内にあるhpはprivate
- 同じクラス内でしかアクセスできない
- →Phoenix内でhpがアクセスできない！？やばい



protected

- hpを「private」の代わりに「protected」にする
- 同じクラス内 or 継承されたクラスでのみアクセス可能になる

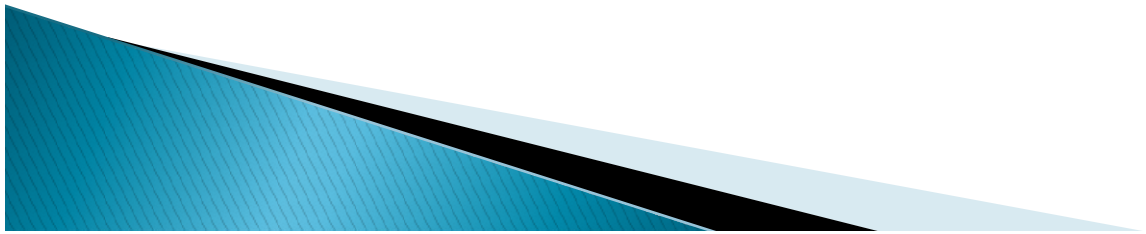
```
class Enemy
{
    protected int hp;
```



いろんな修飾子

- privateとか以外にもいろんな修飾子がある

- https://docs.microsoft.com/ja-jp/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers



お疲れ様でした

- 次回は6/10(土)
- シューティングの続きが作れる！！！！！！！！！！！！！！！！！！！！！！！！！
- 再来週は40thなのでおやすみ