オブジェクト指向プログラミング

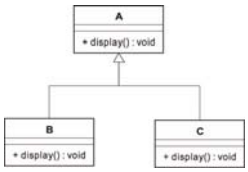
OO編　第4回

ポリモーフィズムを利用した完全なデータ抽象

- 目的
  - ポリモーフィズムの仕組みと利点欠点を理解する
- キーワード
  - ポリモーフィズム，継承，インターフェイス

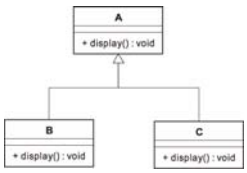
違う型に代入できる

- 今まで
  - A x1 = new A();
  - B x2 = new B();
  - C x3 = new C();
- ポリモーフィズム
  - A x1 = new B();
  - A x2 = new C();



なぜできるか

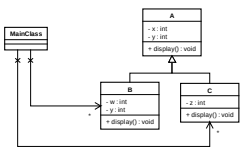
- Super x = Sub();
- サブクラスのインスタンスならメソッド，属性呼び出しが保証できるから
  - ex)
  - Super x = Sub();
  - x.display();
- 練習問題1
  - サブクラスのメソッド優先



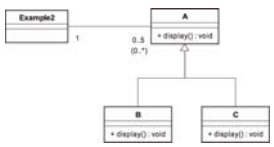
なぜそんなことが必要か

- 同じものとして扱えると便利なときが多いから

ただの継承

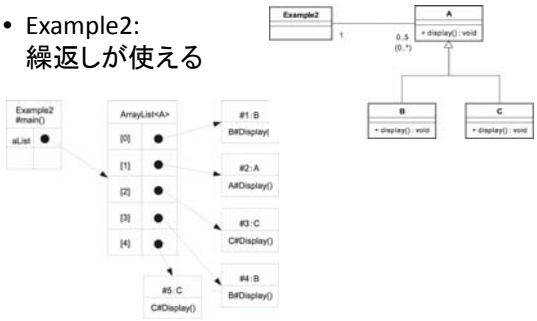


ポリモーフィズム



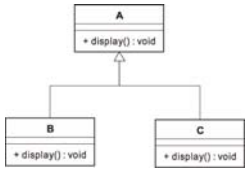
同じ配列に入れられる

- Example2:
  繰返しが使える



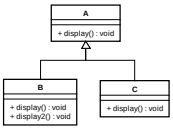
代入のルール

- OK
  - Super x = Super();
  - Sub x = Sub();
  - Super x = Sub();
- NG
  - Sub x1 = Super();



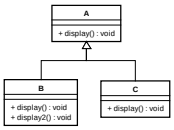
キャストエラー : Example3

- B x = new A();
- x.display2();



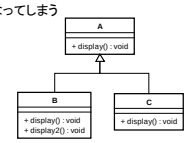
ダウンキャスト:Example4

- 一時的にSubClassとしてみたい場合
  - A x = new B();
  - x.display();
  - B y = (B)x;
  - y.display2();
- インスタンスで考えないと理解不可能なことに注意せよ
- 参照であることに注意せよ



instanceof演算子 : Example5

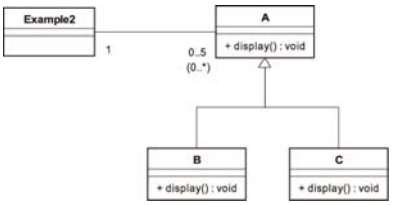
- どのインスタンスなのか知りたい場合
- 問題:
  - Aクラスの配列XにはAかBかCのインスタンスが入っている
  - しかし，どのクラスのインスタンスかは分からなくなってしまう



- aList[0] instanceof A = true or false
- aList[0] instanceof B = true or false
- aList[0] instanceof C = true or false
- aList[1] instanceof A = true or false
- aList[1] instanceof B = true or false
- aList[1] instanceof C = true or false
- aList[2] instanceof A = true or false
- aList[2] instanceof B = true or false
- aList[2] instanceof C = true or false

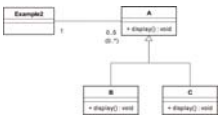
ポリモーフィズムの基本形

- この形を暗記する

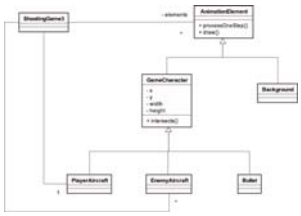


シューティングゲームへ応用

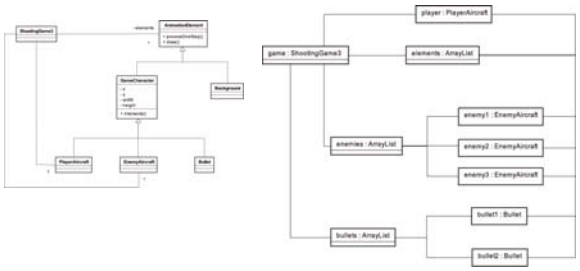
基本形



応用例

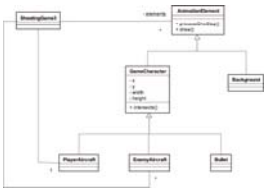


オブジェクト図



描画のプログラムが簡単

- private void processOneStep(BCanvas canvas){
  for(int i=0; i < elementSize; i++){
  elements[i].processOneStep(canvas);
  }
  }



あたり判定のプログラムが簡単

- 今までの方法
  - intersectsPlayer(),intersectsEnemy(), intersectsBullet()を別々に作っていた
  - クラスが増えるたびに
- ポリモーフィズムを使えば
  - intersects(ShootingCharacter x)
  - で，（サブクラスなら）何が来ても，あたり判定が出来る

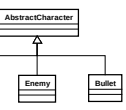
簡略化イメージ図

Playerクラスにあったメソッド

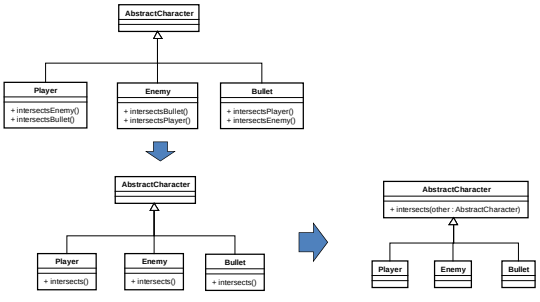
```
public boolean intersects(Enemy enemy) {
 int player_leftX = this.getX();
 int player_rightX = this.getX() + this.getWidth();
 int enemy_leftX = enemy.getX();
 int enemy_rightX = enemy.getX() + enemy.getWidth();
 int player_topY = this.getY();
 int player_bottomY = this.getY() + this.getHeight();
 int enemy_topY = enemy.getY();
 int enemy_bottomY = enemy.getY() + enemy.getHeight();
return (enemy_leftX < player_rightX && enemy_rightX > player_leftX
  && enemy_topY < player_bottomY && enemy_bottomY > player_topY);
}
```

```
public boolean intersects(Bullet bullet) {
 int player_leftX = this.getX();
 int player_rightX = this.getX() + this.getWidth();
 int bullet_leftX = bullet.getX();
 int bullet_rightX = bullet.getX() + bullet.getWidth();
 int player_topY = this.getY();
 int player_bottomY = this.getY() + this.getHeight();
 int bullet_topY = bullet.getY();
 int bullet_bottomY = bullet.getY() + bullet.getHeight();
return (bullet_leftX < player_rightX && bullet_rightX > player_leftX
  && bullet_topY < player_bottomY && bullet_bottomY > player_topY);
}
```

```
public boolean intersects(AbstractCharacter other) {
 int player_leftX = this.getX();
 int player_rightX = this.getX() + this.getWidth();
 int other_leftX = other.getX();
 int other_rightX = other.getX() + other.getWidth();
 int player_topY = this.getY();
 int player_bottomY = this.getY() + this.getHeight();
 int other_topY = other.getY();
 int other_bottomY = other.getY() + other.getHeight();
return (other_leftX < player_rightX && other_rightX > player_leftX
  && other_topY < player_bottomY && other_bottomY > player_topY);
}
```

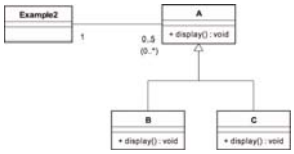


継承もできる



ポリモーフィズムの基本形

- この形を暗記する



・Aはインターフェイスとしての役割を持つ
・サブクラスによって振る舞いが変わる＝多態性
・目的と手段の関係（HCPチャートに関係）
・意味的にはif文と等価→if文を排除できる

練習問題:時計の設計を考える

考察

- 機能継承は多用するな！
- ポリモ/合成による設計をつかえ！