● 授業のポイント

① 委譲（delegate）について学習します。
② Strategy パターンについて学習します。
③ 継承と委譲のちがいについて学習します。

● 学習項目

・委譲（delegate）
・Strategy パターン

● J2Kad22D「ECC ソフト株式会社①（委譲）」

１．委譲（delegate）

ECCSoft クラスの work メソッドでは、ここで何かの対応をするわけでなく、下請け A（SubA クラス）もしくは下請け B（SubB クラス）の work メソッドへ処理を丸投げします。main メソッドからすれば ECCSoft へ発注しているにもかかわらず実は下請けクラスへ処理が丸投げされる手法を**委譲（delegate）**と呼びます。クラスの継承と並んで重要なテクニックです。

```
public void work() {
    sub.work();              // ~~丸投げ~~委譲（delegate）
}
```

● J2Kad22C「ECC ソフト株式会社②（下請けの追加）」

１．下請けの追加（クラス外から追加する方法）

下請け C を追加します。ただし J2Kad22D のように ECCSoft クラス内で下請けクラスのインスタンスを生成していると、下請けが増えるごとに ECCSoft クラスを修正する必要があります。ここではクラス外（課題では main メソッド）で下請けクラスのインスタンスを生成し、ECCSoft クラスへ渡す（登録する）ようにします。これにより ECCSoft クラスの work メソッドは登録された下請けクラスへ委譲するだけの処理になります。下請けクラスが増えても ECCSoft クラスの修正は発生しません。

● J2Kad22B「スーパーコンピュータ ECC1000（Strategy パターン）」

１．アルゴリズムの切り替え（Strategy パターン）

委譲を使うことによってアルゴリズムの切り替えを行うことができます。ここでは ECC1000 の計算処理を、委譲を使って切り替えます。このようにクラスから振る舞い（アルゴリズム）部分を独立させて、変更や交換を容易にできるようにした手法を **Strategy パターン**と呼びます。

● J2Kad22A「ジョブチェンジ！」

１．継承（Before）と委譲（After）

RPG キャラクターのジョブチェンジを作ります。課題作成前は RPGCharacter を継承して Fighter や Mage、Monk といった職業クラスを作っていました。この場合、キャラクターが Fighter を選択したら Fighter クラスのインスタンスが生成されます。Mage を選択したら Mage クラスのインスタンス、Monk を選択したら Monk クラスのインスタンスが生成されます。クラスが異なるためそれぞれ別々のインスタンスを生成することになり、キャラクターのデータ（本課題では name）も個々に異なります。データはそのままでジョブのみ変更するためには継承ではなく委譲を使う必要があります。委譲であればインスタンスは同じままで振る舞い（ジョブ）を切り換えることができます。

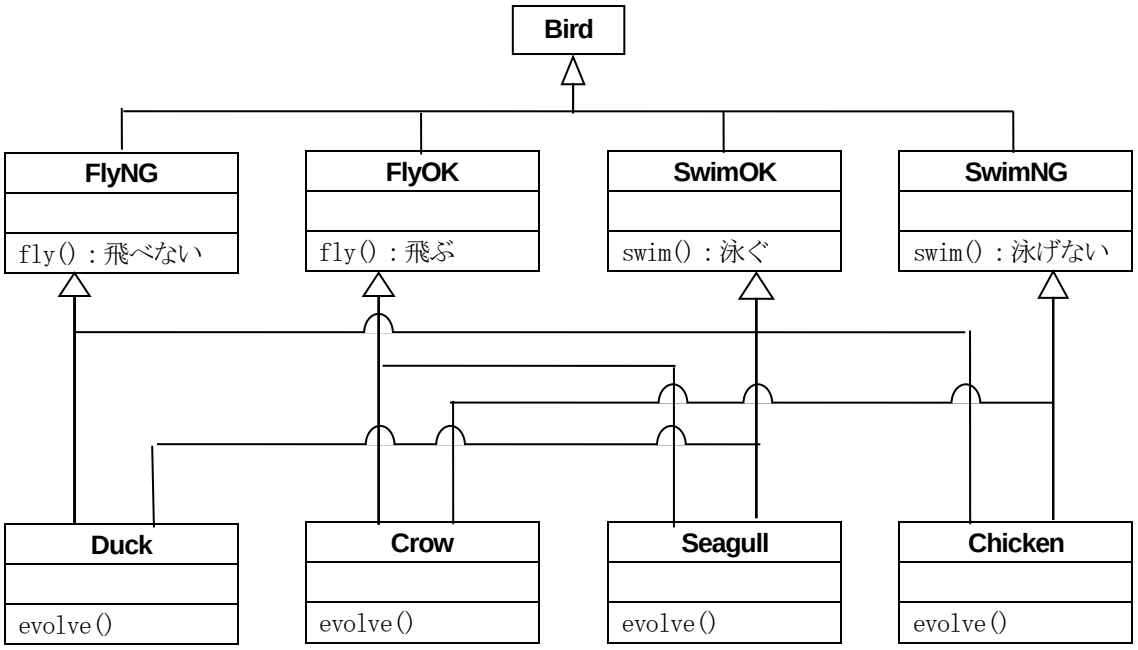
課題自体は J2Kad22B と同じような内容ですが、ここでは振る舞いとして独立させるのは何かを考えてクラスを作る必要があります。

● J2Kad22S「ポケット Duck！」

１．多重継承（Java ではできませんが）とクラス爆発

野鳥は 4 種類（Duck・Crow・Seagull・Chicken）ありますが、fly メソッドは「飛べる」「飛べない」の 2 種類しかありません。swim メソッドは「泳げる」「泳げない」の 2 種類です。もしクラス継承での効率化を考えると Bird クラスを継承して FlyOK（飛べる鳥）クラス・FlyNG（飛べない鳥）クラス、SwimOK（泳げる鳥）クラス・SwimNG（泳げない鳥）クラスを作成し、これらを（Java ではできない）多重継承して Duck・Crow・Seagull・Chicken を作ることになります。

多重継承（Java では NG、C++では OK）を使ったクラス構成（こんなの絶対いやだ）

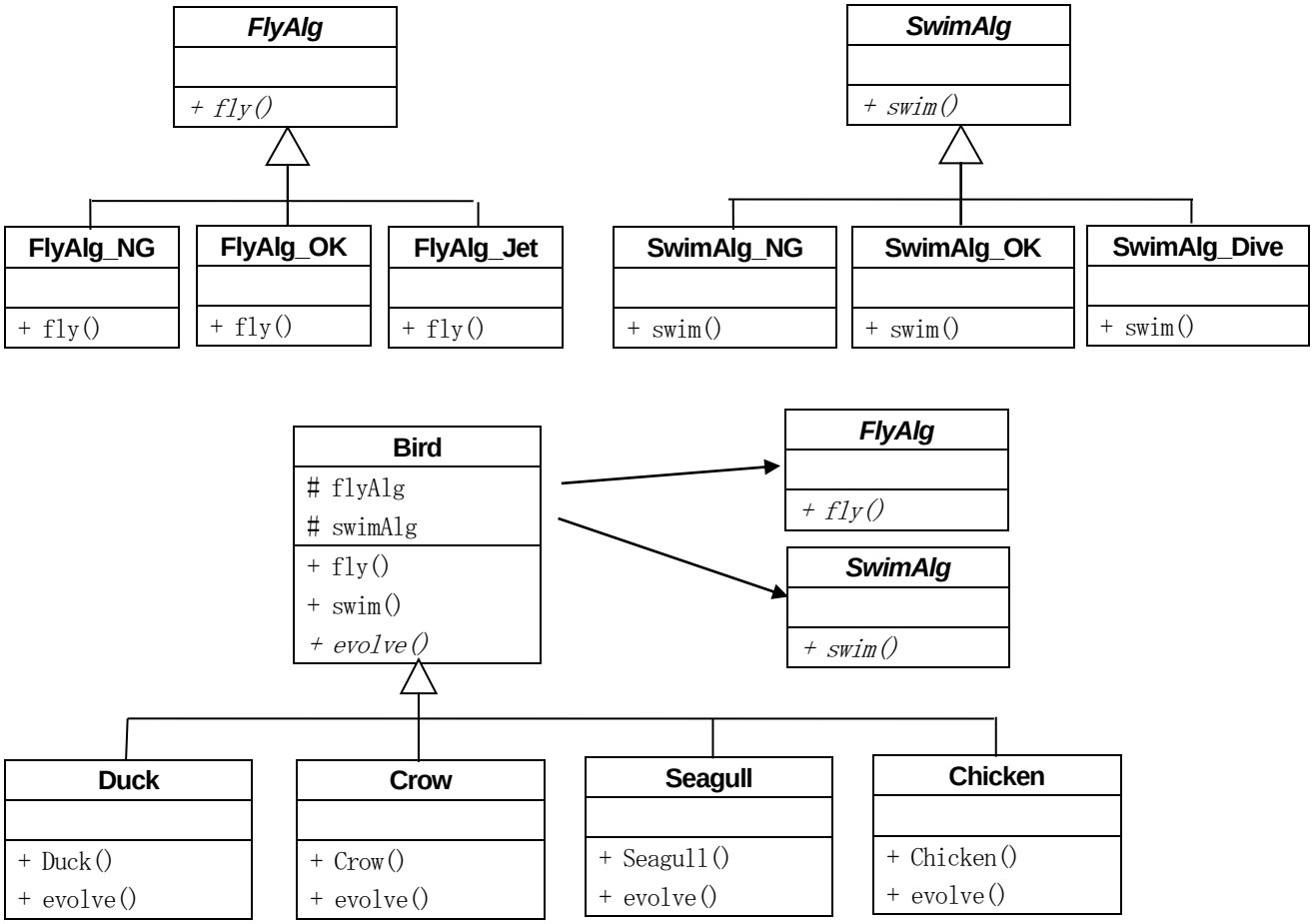


これに進化したときの「ジェット噴射」や「潜る」を加えると fly×swim の組み合わせなのでクラス数がとんでもないことになります。またクラスの継承関係は動的に変更することができないので、Duck が進化して飛べるようになったとしても対応することができません。

２．Strategy パターンによる解決

Strategy パターンを使って、fly 部分と swim 部分を独立したアルゴリズムクラス（それぞれ FlyAlg クラス、SwimAlg クラス）として作成します。各野鳥クラスは自分の特徴にあった FlyAlg クラスと SwimAlg クラスを「装備」します。進化したときにはアルゴリズムクラスの切り替えを行えば対応することができます。

最終的なクラス構成（Strategy パターン適用）



● J2Kad22X「ファミレス ECC」

１．Composite パターン（詳細は年明け授業にて）

Composite パターンとは容器と中身を同一視して再帰的な構造を作るパターンです。例えば「フォルダ」の中に「ファイル」だけでなく「フォルダ」があったり、さらにその「フォルダ」の中にも「ファイル」や「フォルダ」があったりという構造です。課題では MenuList クラスが「容器」、MenuItem クラスが「中身」に該当します。Composite パターンを使えばメニュー表示は簡潔になります（詳細は後日授業にて）。

ちなみに何も考えずに指定されたメニューを片っ端から表示するという力わざもありますが、ここではそれでも OK とします（せっかくがんばって作ったので）。