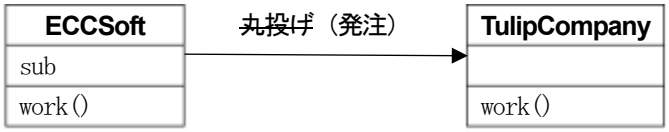
● J2Kad21D「ECC ソフト株式会社①（委譲）」

ECC がソフト開発会社を立ち上げた！その名も「信頼と実績の ECC ソフト株式会社」、どんな課題も真摯に解決してくれる。ただその実態は請け負った仕事をそのまま下請けに丸投げするだけらしい。下請け会社チューリップ商会（TulipCompany クラス）を作成し、ECC ソフト（ECCSoft クラス）がチューリップ商会に丸投げする処理（work メソッド）を追加せよ。



TulipCompany クラスの仕様（新規ファイル「TulipCompany.java」に作成）

メンバ	仕様
public void work()	″チューリップ商会「下請けはつらいなー！」″と表示する。

ECCSoft クラスに追加するメンバ（ファイル「ECCSoft.java」）

メンバ	仕様
public TulipCompany sub	下請け会社（チューリップ商会）への参照。初期値は TulipCompany のインスタンスを設定。
public void work()	仕事をする。ただしその実態は下請け会社（sub）の work メソッドを呼び出すだけ。

main メソッドの処理

① 「0：ECC ソフトに仕事を発注する」を選択したとき、ECCSoft の work メソッドを呼び出す。

課題完成時の画面

```
信頼と実績の ECC ソフト株式会社です！
どんな課題でも私たちが真摯に解決します！！
どうしますか？（0：ECC ソフトに仕事を発注する、-1：もういい）＞0
チューリップ商会「下請けはつらいなー！」

どうしますか？（0：ECC ソフトに仕事を発注する、-1：もういい）＞0
チューリップ商会「下請けはつらいなー！」

どうしますか？（0：ECC ソフトに仕事を発注する、-1：もういい）＞-1
```

ECC ソフトに仕事を発注しても
チューリップ商会に丸投げする。

● J2Kad21C「ECC ソフト株式会社②（下請けの切り換え）」※J2Kad21D をコピーして作成

ECC ソフトの業績が好調だ！そこでさらに下請けを増やすことにした。新しい下請けはペーパー企画(PaperPlanning)とグローバル商事（GlobalTrading)。下請け先を「1：下請けを変更する」で変更できるようにせよ。

ヒント：TulipCompany、PaperPlanning、GlobalTrading のスーパークラスとして Subcontractor を考える。

Subcontractor クラスの仕様（新規ファイル「Subcontractor.java」に作成）

メンバ	仕様
public String getName()	会社名を返す。
public void work()	抽象メソッド

下請けクラスの仕様（各サブクラスのファイルを新規作成）

メンバ	仕様
public String getName()	各サブクラスの会社名を返す。
public void work()	PaperPlanning：″ペーパー企画「ただいま企画検討中です！」″と表示 GlobalTrading：″グローバル商事「我が社のネットワークを使って孫請けを探します！」″と表示

ECCSoft クラスに追加するメンバ（ファイル「ECCSoft.java」）

メンバ	仕様
public **Subcontractor** sub	TulipCompany への参照→Subcontractor への参照に変更。
public void setSubcontractor(Subcontractor sub)	sub フィールドに引数 sub を設定し、 「～に変更します！」（～は会社名）と表示する。

main メソッドの処理

①　「1：下請けを変更する」を追加、下請け会社を選択して設定する。

課題完成時の画面

```
      ⋮
どうしますか？（0：ECC ソフトに仕事を発注する、1：下請けを変更する、-1：もういい）＞1
どこにしますか？（0：チューリップ商会、1：ペーパー企画、2：グローバル商事）＞1
ペーパー企画に変更します！

どうしますか？（0：ECC ソフトに仕事を発注する、1：下請けを変更する、-1：もういい）＞0
ペーパー企画「ただいま企画検討中です！」

どうしますか？（0：ECC ソフトに仕事を発注する、1：下請けを変更する、-1：もういい）＞1
どこにしますか？（0：チューリップ商会、1：ペーパー企画、2：グローバル商事）＞2
グローバル商事に変更します！

どうしますか？（0：ECC ソフトに仕事を発注する、1：下請けを変更する、-1：もういい）＞0
グローバル商事「我が社のネットワークを使って孫請けを探します！」
      ⋮
```

● J2Kad21B「スーパーコンピュータ ECC1000（Strategy パターン）」

ECC ホームエレクトロニクスがスーパーコンピュータ ECC1000（サウザンド）を開発した！1 から 10 までの整数を足し合わせる高度な演算を、アルゴリズムを切り換えて処理できるというスグレモノだ！ECC1000 クラスを作成し、アルゴリズム（SumAlg）を切り換えて動作するかどうか確認せよ。

SumAlg インターフェイス（作成済み）

メソッド	仕様
void sum(int n)	1 から n までの整数を足し合わせた結果を表示する。

演算アルゴリズム（SumAlg インターフェイスを実装、作成済み）

クラス	特徴
NobitaAlg	「のび太」をモデルにしたアルゴリズム。計算が苦手。
SuneoAlg	「スネ夫」をモデルにしたアルゴリズム。普通に計算する。
DekisugiAlg	「出木杉」をモデルにしたアルゴリズム。頭が良すぎるため、物事を複雑に考える。
SizukaAlg	「しずか」をモデルにしたアルゴリズム。物事を別の角度からシンプルに考える。

ECC1000 クラスの仕様（ファイル「ECC1000.java」）

メンバ	説明
private SumAlg alg	使用するアルゴリズム
public void setAlg(SumAlg alg)	アルゴリズムのセッター（フィールド alg に引数 alg を設定する）
public void sum(int n)	設定されたアルゴリズムに 1～n までの合計を計算させる。 設定されていないときは「アルゴリズムがセットされていません」と表示する。

main メソッドの処理

① 使用するアルゴリズムを選択（マイナスの値で終了）。
② 指定されたアルゴリズムを ECC1000 に設定する（1～4 以外のときは null を設定）。
③ 1 から 10 までの加算結果を表示して、①へ戻る。

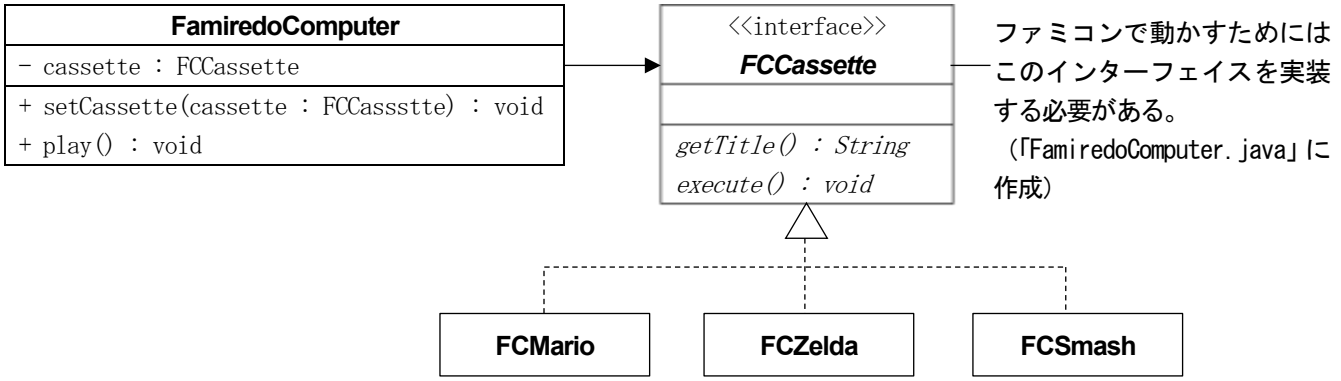
課題完成時の画面

```
スーパーコンピュータ ECC1000 を開発した！
アルゴリズムをセットすればどんな高度な計算でもできます！！
計算アルゴリズムをセットしてください（0：Nobita、1：Suneo、2：Dekisugi、3：Sizuka、-1：終了）＞0
のび太：え～っ、わかんないよ～

計算アルゴリズムをセットしてください（0：Nobita、1：Suneo、2：Dekisugi、3：Sizuka、-1：終了）＞1
スネ夫：こんなの簡単さ。順番に足していけばいいんだ。
スネ夫：1 を足して
スネ夫：2 を足して
スネ夫：3 を足して
スネ夫：4 を足して
スネ夫：5 を足して
スネ夫：6 を足して
スネ夫：7 を足して
スネ夫：8 を足して
スネ夫：9 を足して
スネ夫：10 を足して
スネ夫：答えは 55 だよ！

計算アルゴリズムをセットしてください（0：Nobita、1：Suneo、2：Dekisugi、3：Sizuka、-1：終了）＞2
出木杉：こんなの簡単さ。再帰処理を使えばいいんだ。
出木杉：ここは再帰処理 10 だよ。
出木杉：ここは再帰処理 9 だよ。
      ⋮
出木杉：ここは再帰処理 2 だよ。
出木杉：ここは再帰処理 1 だよ。
出木杉：呼び出し元へ戻るね。
出木杉：呼び出し元へ戻るね。
      ⋮
出木杉：呼び出し元へ戻るね。
出木杉：呼び出し元へ戻るね。
出木杉：答えは 55 だよ！

計算アルゴリズムをセットしてください（0：Nobita、1：Suneo、2：Dekisugi、3：Sizuka、-1：終了）＞3
しずか：逆順に並べてみればいいんだわ。
  1  2  3  4  5  6  7  8  9 10
 10  9  8  7  6  5  4  3  2  1
しずか：上と下を足すと・・・
 11 11 11 11 11 11 11 11 11 11
しずか：そうか！11 ＊ 10 を計算して半分にすればいいんだわ！
しずか：答えは 55 よ！

計算アルゴリズムをセットしてください（0：Nobita、1：Suneo、2：Dekisugi、3：Sizuka、-1：終了）＞4
アルゴリズムがセットされていません！

計算アルゴリズムをセットしてください（0：Nobita、1：Suneo、2：Dekisugi、3：Sizuka、-1：終了）＞-1
```

● J2Kad21A「ファミコン開発秘話！」

ECC ホームエレクトロニクスがゲーム機の開発に着手した！カセットを交換するだけでいろんなゲームが遊べる画期的なゲーム機だ。その名も「ファミレドコンピュータ」、略して「ファミコン」。どこかで聞いたことのある名称だが、そんなこと気にしていてはいい大人になれない。ファミコン設計図（クラス図）をもとにファミコンを開発しゲームソフトで遊べるようにせよ。

ファミコン設計図



FamiredoComputer クラスの仕様（新規ファイル「FamiredoComputer.java」）

メソッド	仕様
setCassette メソッド	cassette フィールドに引数 cassette を設定する。 ・引数が null のとき「カセットを取り外しました！」と表示する。 ・そうでないとき「～をセットしました！」（～はカセットのタイトル）と表示する。
play メソッド	ゲームで遊ぶ。 ・cassette フィールドが null のとき「カセットがセットされていません！」 ・そうでないとき、cassette の execute メソッドを呼び出す。

ゲームソフトの仕様（FCCassette インターフェイスを実装、ゲームごとにファイルを作成すること）

メソッド	仕様
getTitle メソッド	タイトルを返す。FCMario：「マリオ」、FCZelda：「ゼルダ」、FCSmash：「スマブラ」
execute メソッド	ゲームを実行する。各クラスで適当にメッセージを表示すること。

main メソッドの処理

① FamiredoComputer のインスタンスを生成する。
② 「0：遊ぶ」「1：カセットをセットする」「-1：1 時間遊んだのでやめる」を選択する。
③ -1（マイナスの値）のとき終了する。
④ 0 のとき FamiredoComputer で遊ぶ（play メソッド）。
⑤ 1 のとき遊ぶゲームを選択（0：マリオ、1：ゼルダ、2：スマブラ）、FamiredoComputer に選択したゲームのカセットをセットする（setCassette メソッド）。
⑥ ②へ戻る。

課題作成前の画面

ファミコンを買ってもらった！
ゲームソフトも買ってもらった！
どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**0**
カセットがセットされていません！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**1**
どれをセットしますか？（0：マリオ、1：ゼルダ、2：スマブラ）＞**0**
マリオをセットした！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**0**
マリオで遊んでいます！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**1**
どれをセットしますか？（0：マリオ、1：ゼルダ、2：スマブラ）＞**1**
ゼルダをセットした！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**0**
ゼルダで遊んでいます！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**1**
どれをセットしますか？（0：マリオ、1：ゼルダ、2：スマブラ）＞**2**
スマブラをセットした！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**0**
スマブラで遊んでいます！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**1**
どれをセットしますか？（0：マリオ、1：ゼルダ、2：スマブラ）＞**3**
カセットを取り外した！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**0**
カセットがセットされていません！

どうしますか？（0：遊ぶ、1：カセットをセットする、-1：1時間遊んだのでやめる）＞**-1**

● J2Kad21S「ジョブチェンジ！」

ECC ソフトに依頼して RPG のジョブチェンジの処理を作成した。しかしジョブは選択できるがキャラクターの名前がどんどん変わっていく！まるでキャラクター（インスタンス）が入れ替わっているようだ！！キャラクターの名前が変わらないように（インスタンスが入れ替わらないように）処理を修正せよ。

課題作成前の画面

```
あなたは RPG のキャラクターです！
今から世界を救う冒険に出かけます！
でもその前に職業を選んでください！！

フックさん、こんにちは！
ジョブチェンジしますか？（0：戦士、1：魔法使い、2：モンク、-1：これでいい）>0
武器で攻撃します！
盾で防御します！

ファラデーさん、こんにちは！
ジョブチェンジしますか？（0：戦士、1：魔法使い、2：モンク、-1：これでいい）>1
攻撃の魔法を唱えます！
防御の魔法を唱えます！

オイラーさん、こんにちは！
ジョブチェンジしますか？（0：戦士、1：魔法使い、2：モンク、-1：これでいい）>3
攻撃方法を知らない！
防御方法を知らない！
```

ジョブチェンジと同時にキャラクターの名前も変わる。

課題完成時の画面

```
あなたは RPG のキャラクターです！
今から世界を救う冒険に出かけます！
でもその前に職業を選んでください！！

フーリエさん、こんにちは！
ジョブチェンジしますか？（0：戦士、1：魔法使い、2：モンク、-1：これでいい）>0
武器で攻撃します！
盾で防御します！

フーリエさん、こんにちは！
ジョブチェンジしますか？（0：戦士、1：魔法使い、2：モンク、-1：これでいい）>1
攻撃の魔法を唱えます！
防御の魔法を唱えます！

フーリエさん、こんにちは！
ジョブチェンジしますか？（0：戦士、1：魔法使い、2：モンク、-1：これでいい）>3
攻撃方法を知らない！
防御方法を知らない！
```

ジョブチェンジしてもキャラクターの名前は変わらない。
（キャラクターではなくジョブを変更する）

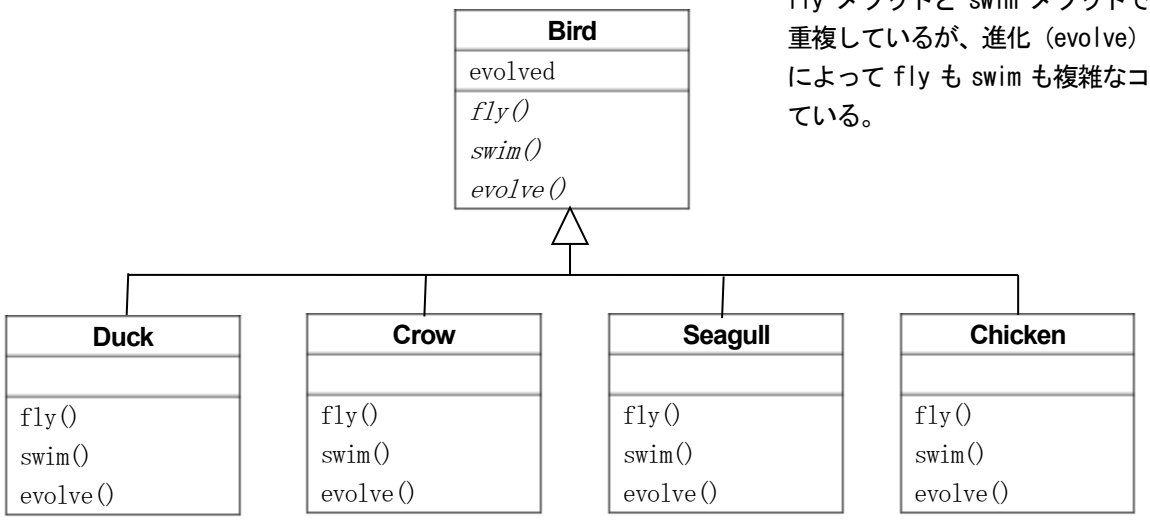
● J2Kad21X「ポケット Duck！」※J2Kad21X クラスは修正なし

世界にはばたく ECC ゲームスでは野鳥をゲットして収集するオリジナルゲーム「ポケット Duck！」を制作している。登場する野鳥は 4 種類（アヒル・カラス・カモメ・ニワトリ）、それぞれ飛ぶ（fly）・泳ぐ（swim）・進化する（evolve）のコマンドに対応している。プログラム開発を ECC ソフトへ発注したところ、「飛ぶ」と「泳ぐ」まで制作できたが、「進化する」を追加した段階で難航しているという報告が来た。今後、さらにさまざまな仕様（例えば「ワープする」とか「深海まで潜る」など）を追加していくことを考えると、このままの設計では「のっぴきならない」（←意味がわからない人は検索）ことになる！今後の開発を順調に進めるためにも、簡潔かつ修正のしやすい設計になるようにクラス設計を見直し、ECC ソフトのコードを修正せよ。

野鳥の仕様（Bird クラスを継承）

クラス	飛ぶ（fly）	泳ぐ（swim）	進化する（evolve）
Duck（アヒル）	飛べない	水面を泳ぐ	空を飛べるようになる
Crow（カラス）	空を飛ぶ	泳げない	ジェット噴射で飛べるようになる
Seagull（カモメ）	空を飛ぶ	水面を泳ぐ	潜れるようになる
Chicken（ニワトリ）	飛べない	泳げない	飛べるようになる&泳げるようになる

課題作成前のクラス構成



fly メソッドと swim メソッドで同じ処理が重複しているが、進化（evolve）が入ることによって fly も swim も複雑なコードになっている。

課題完成時の画面

```
どの野鳥をゲットしますか？（0：アヒル、1：カラス、2：カモメ、3：ニワトリ、-1：終了）>0
ぼくアヒル！
飛べないけど泳げるよ！

何をさせますか？（0：飛ぶ、1：泳ぐ、2：進化、-1：終了）>0
う～ん、飛べない！！

何をさせますか？（0：飛ぶ、1：泳ぐ、2：進化、-1：終了）>1
スイスイ！水面を泳いでいます！！

何をさせますか？（0：飛ぶ、1：泳ぐ、2：進化、-1：終了）>2
空を飛べるようになった！！

何をさせますか？（0：飛ぶ、1：泳ぐ、2：進化、-1：終了）>0
バタバタ！がんばって飛んでいます！！
```