問5　共通ライブラリのオブジェクト指向設計（ソフトウェア設計）(H26秋-FE 午後問5)

【解答】
［設問1］　a－カ，b－オ，c－ウ，d－イ
［設問2］　e－エ，f－ア

【解説】
共通ライブラリのオブジェクト指向設計に関する問題である。オブジェクトとは，処理対象となるデータや事象のことで，オブジェクト指向とは，これらを分析し，オブジェクトにオブジェクト固有の属性（データ）と操作（メソッド）をもたせる設計手法である。そして，属性と操作を一体化させたものをオブジェクトとして扱い，このように一体化させることをカプセル化という。

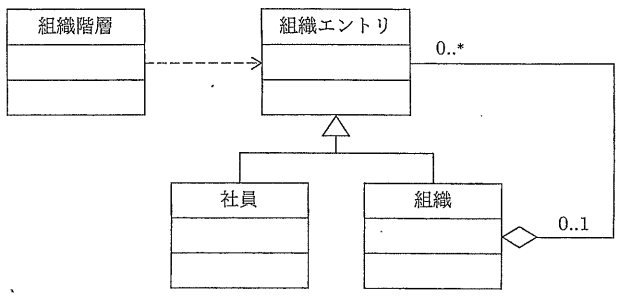
このとき，オブジェクトを作成する型の定義がクラスであり，実体化したものがインスタンスである。

共通ライブラリとは，複数のアプリケーションから利用される共通機能をまとめたものである。これを利用することでソフトウェアの品質安定化と開発の生産性向上を図ることができる。

本問では，組織階層を扱う機能をユーザインタフェースから切り離して，様々な開発プロジェクトで再利用できるように，共通ライブラリとしてオブジェクト指向で設計している。問題文にクラス図を表現する手順が記述されているので，それに従いクラス図を考えればよいが，用語について注意が必要である。まず，クライアントだが，問題文に「共通ライブラリを利用する側のソフトウェアを一律にクライアントと呼ぶ」と記述がある。クライアントサーバモデルのサービス利用側と提供側に模していると考えればよいであろう。次に組織エントリだが，本問独自の用語であり，〔分析のためのクラス図の説明〕(2)に「組織と社員を共通に扱えるようにするために，どちらも組織エントリとして管理する」と定義されている。これは，組織階層を構成するオブジェクト（要素）である組織と社員に共通する属性や操作を一体化し，そのオブジェクトを組織エントリと呼ぶ，という意味である。よって，組織エントリは，組織と社員のスーパクラスもしくはインタフェースであり，組織と社員はこれを継承，又は実装する。この2点に注意し，凡例と問題文を読み取り，穴埋めを考える。

［設問1］
図2の分析のためのクラス図の穴埋めを考える。クラス図はオブジェクトを作成するためのひな形なので，ここでは，クラス図の作成に先駆け，組織階層におけるオブジェクトとその多重度を分析する。まず，オブジェクトを洗い出す。〔分析のためのクラス図の説明〕によれば，オブジェクトに該当するものには，組織階層，親組織，子組織，社員と(2)に記述された組織エントリの五つがある。しかし，親組織と子組織は，親であるか子であるかの違いはあるものの組織としては同じである。よって，オブジェクトは組織階層，組織，社員，組織エントリの四つであり，図2に記述されているクラスの数と一致する。空欄aのクラスには汎化関係を表す△マークが付いているので，空欄bのクラスともう一つのクラスのスーパクラスである。よって，組織と社員を共通に扱えるように定義された（カ）組織エントリと分かる。そして，組織エントリの下位クラスのうち，空欄bには集約を表す◇マークが付いているので，組織エントリは空欄bのクラスの部分である。よって，空欄bのクラスは複数の組織エントリをもつことができる。これに該当するオブジェクトは（オ）組織である。また，「社員は必ず一つの組織だけに属する」と記述があるので，社員が組織の部分になることはあっても，自分の部分となる組織をもつことはない。よって，組織エントリのもう一つの下位クラスが社員である。これにより，残る一つは組織階層であり，組織エントリと依存関係にある。これは，組織階層の表現が組織と社員のスーパクラスである組織エントリの定義に左右されることを表している。したがって，空欄 a は（カ），空欄 b は（オ）が正解である。

次に，多重度を分析する。多重度は，一つのインスタンスに対して関連するインスタンスが幾つ存在するかを表す。また，インスタンスは，クラスから作成される具体的なオブジェクトである。空欄cは，ある一つの組織に対する組織エントリの数を表すが，組織エントリは組織の部分なので，問題文の次の記述が該当する「子の組織をもつ場合，子組織の数に制限はない」，「子組織をもたない組織もある」。よって，多重度は，下限が0，上限が*であり，（ウ）0..*と記述できる。同様に空欄dは，ある一つの組織エントリに対する組織の数を表し，問題文の次の記述が該当する「親の組織をもつ場合，親組織は必ず一つであり」，「親組織をもたない組織もある」。よって，多重度は，下限が0，上限が1であり，（イ）0..1と記述できる。したがって，空欄cは（ウ），空欄dは（イ）が正解である。



［設問2］
設問1で作成した図2を基に作成した図3「組織エントリリストを設計するためのクラス図」を，図5「見直し後の設計のためのクラス図」に修正するための設計方針を考える。共通ライブラリの社内レビューでは，クライアント側の実装依存が大きくなることが指摘されている。具体的には，〔指摘の内容〕の「全ての組織エントリを取り出すときや，特定条件を満たす組織エントリだけを取り出したいときなど，クライアント側の実装依存が大きくなる」という記述である。ここで，走査とは，データを探索することである。指摘の具体例として図4「クライアントの走査の例」を見ると，組織だけを読み出したい場合，クライアント側で網掛け部分，つまり社員のデータを読み飛ばさなければならないことが分かる。同様に，社員だけを読み出したい場合は，組織のデータを読み飛ばさなければならない。カプセル化のメリットの一つは，クライアント側はオブジェクト内部のデータ構造を意識せずに処理を実行できる点にあるが，図3の設計ではこれが実現されていない。そこで，〔設計方針〕(2)にあるように，リストを走査する処理を行うためのイテレータという別クラスを生成する。

図3と図5を比較すると，図3の組織エントリリストに該当するクラスが図5のBaseListクラスである。属性と操作は同じものをもっているが，リストを走査するイテレータを生成してクライアントに返すメソッドcreateIteratorが追加されている。このメソッドを実行すると，インタフェースIteratorを実装したBaseIteratorクラスのインスタンスが生成される（図5 <<create>>）。このインスタンスがイテレータである。イテレータは，属性として走査対象のリスト list と走査中の位置currentPositionを保持し，クライアントからの要求に応じて適切な要素を返す（表2「図5のクラス図の説明」）。よって，イテレータに登録するものはリスト自身である。したがって，空欄eは（エ）が正解である。

空欄fは，クライアントが共通ライブラリで提供するメソッドと異なる走査をしたい場合の対応である。共通ライブラリでは，〔設計方針〕(3)にあるように「先頭から順に全ての要素を走査するメソッド」は提供されるので，例えば，社員を読み飛ばす走査や組織を読み飛ばす走査がこれに該当する。その場合，クライアントは〔設計方針〕(4)に従い，アクセス方法を統一するために定義された「インタフェースを用いてリストを走査」しなければならない。インタフェースはListとIteratorが定義されており，それぞれを実装したBaseListクラスとBaseIteratorクラスがあるので，具体的には，これらのクラスを継承して処理をオーバライドすることになる。〔設計方針〕(4)では「アクセス方法の統一を目的として，必要なメソッドをインタフェースとして定義」している。メソッドを追加したり，ListとIteratorを実装した新たなクラスを作成してしまったりしてはアクセス方法の統一がとれなくなってしまう。したがって，空欄fは（ア）が正解である。