Ｊａｖａ™プログラミング能力認定試験

１級実技試験

テーマプログラム

［第6版］

サーティファイ
情報処理能力認定委員会

従業員派遣管理システム

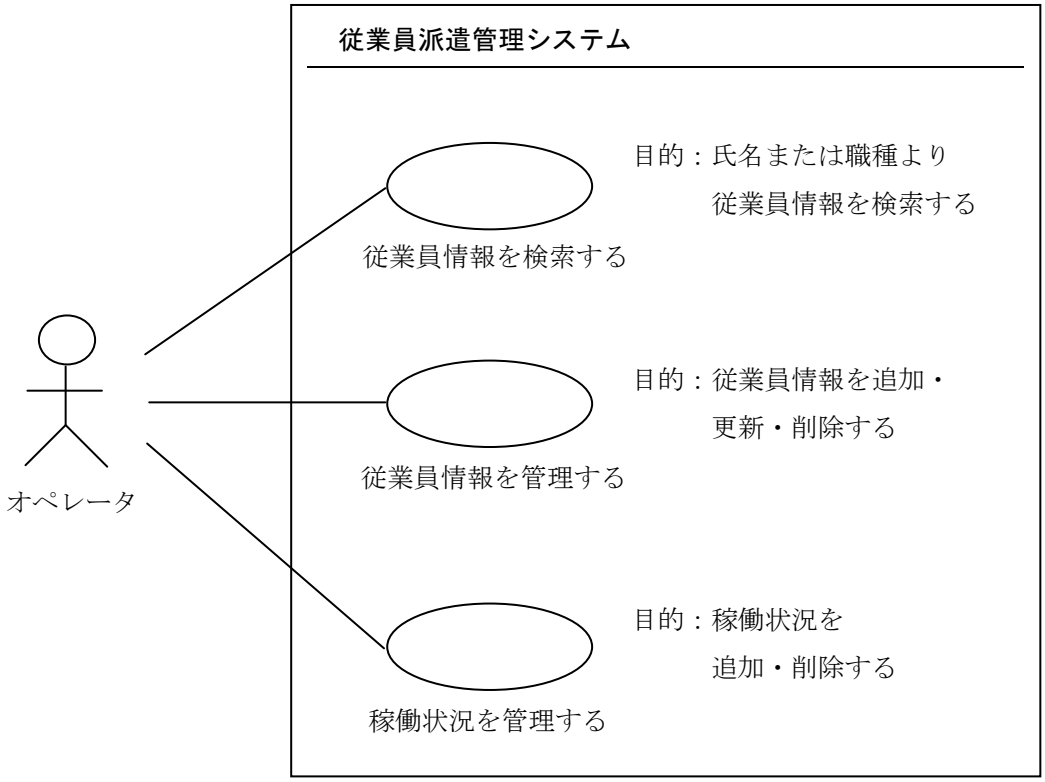
システム仕様書

１．概要

本システムは，派遣会社の従業員を管理することを目的としている。

また，本システムは，Javaのコンソールアプリケーションであり，データはCSV形式のテキストファイルで管理される。

２．ユースケース図



３．イベントフロー

３．１．「従業員情報を検索する」ユースケースについて

（１）メインフロー

このユースケースは，オペレータが「従業員検索」を選択するところから始まる。オペレータが検索方法コードを入力すると，検索方法コードによって(１−１)と(１−２)の処理に分かれる(オペレータが入力した検索方法コードが正しくない場合は，下記の例外フローE-1 の処理を行う)。

（１−１）「氏名から検索」の機能コードが選択された場合

オペレータが氏名に含まれる文字列を入力すると，システムは入力された文字列を含む従業員情報を検索し，表示する。オペレータが検索結果一覧より一覧終了コードまたは従業員 ID を入力すると，入力された内容によって(１−３)と(１−４)の処理に分かれる。

（１−２）「職種から検索」の機能コードが選択された場合

オペレータが職種を入力すると，システムは入力された職種に属する従業員情報を検索し，表示する。オペレータが検索結果一覧を参照し，従業員 ID を入力すると(１−４)の処理へ遷移する。また，一覧終了コードを入力すると(１−３)の処理へ遷移する。

（１−３）「検索結果一覧終了」の一覧終了コードが選択された場合

システムはこのユースケースを終了し，（１）メインフローの検索方法の選択に戻る。

（１−４）「従業員 ID」が入力された場合

システムは入力された従業員 ID に該当する従業員の詳細情報を表示する。

（２）例外フロー

E-1 検索方法が正しくない。検索方法一覧を表示し，検索方法コードを入力し直す。

３．２．「従業員情報を管理する」ユースケースについて

（１）メインフロー

このユースケースは，オペレータが「従業員管理」を選択するところから始まり，選択された機能コードによって(１−１)から(１−３)の処理に分かれる。

（１−１）「追加」の機能コードが選択された場合

オペレータが従業員情報を入力すると，システムは入力された従業員情報に，従業員マスタに登録されている最大の従業員 ID に 1 を加えた従業員 ID を割り当てて，従業員マスタに登録し，割り当てられた従業員 ID を画面に表示する。

（１−２）「更新」の機能コードが選択された場合

オペレータが従業員 ID を指定すると，システムは該当する従業員情報を従業員マスタから取り出して表示する。オペレータが更新する項目番号と更新値を入力すると，システムは入力された値で従業員マスタを更新する。

（１−３）「削除」の機能コードが選択された場合

オペレータが従業員 ID を指定すると，システムは該当する従業員情報を従業員マスタから取り出して表示する。さらに，システムは削除の意思を確認するメッセージを表示し，オペレータが「削除する」を選択した場合に，指定された従業員情報の削除フラグの項目に，真を格納する(オペレータが「削除しない」を選択した場合は，下記の例外フローE-1 の処理を行う)。

（２）例外フロー

E-1 「削除しない」が選択された。削除の処理を行わず機能コードの選択に戻る。

３．３．「稼働状況を管理する」ユースケースについて

（１）メインフロー

このユースケースは，オペレータが「稼働状況管理」を選択するところから始まり，選択された機能コードによって(１−１)と(１−２)の処理に分かれる。

（１−１）「追加」の機能コードが選択された場合

オペレータが稼働状況を指定すると，システムは入力された稼働状況に，稼働マスタに登録されている最大の稼働 ID に 1 を加えた稼働 ID を割り当て，稼働マスタに登録し，割り当てられた稼働 ID を画面に表示する。稼働状況の入力中，オペレータが顧客 ID を入力する時は顧客情報を顧客マスタから取り出して表示する。

（１−２）「削除」の機能コードが選択された場合

オペレータが従業員 ID を指定すると，システムは該当する従業員の稼働状況を表示する。オペレータが削除する稼働状況の番号を指定すると，システムは削除の意思を確認するメッセージを表示する。オペレータが「削除する」を選択した場合に，指定された稼働状況の削除フラグの項目に，真を格納する(オペレータが「削除しない」を選択した場合は，下記の例外フロー**E-1** の処理を行う)。

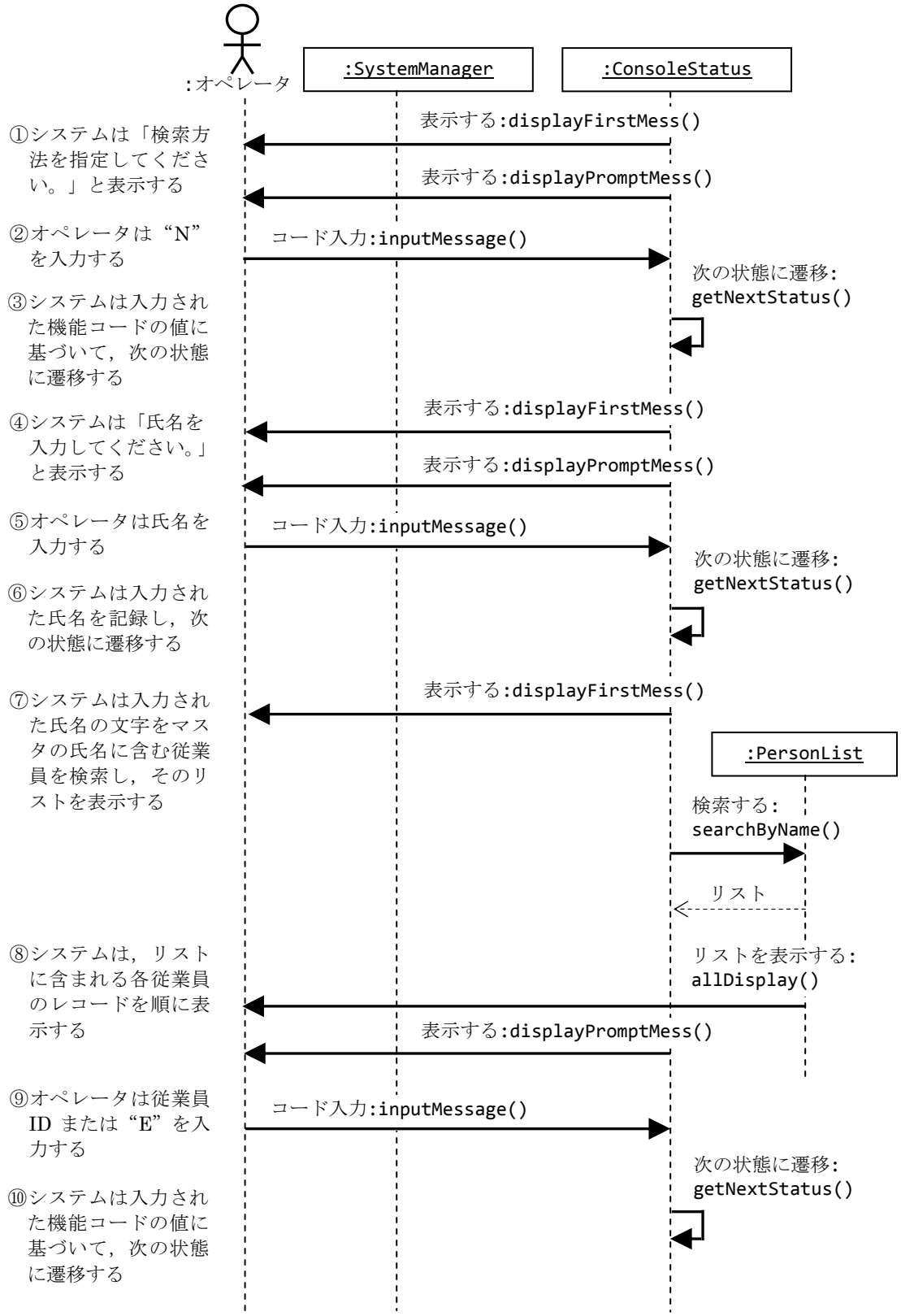
（２）例外フロー

E-1 「削除しない」が選択された。削除の処理を行わず機能コードの選択に戻る。

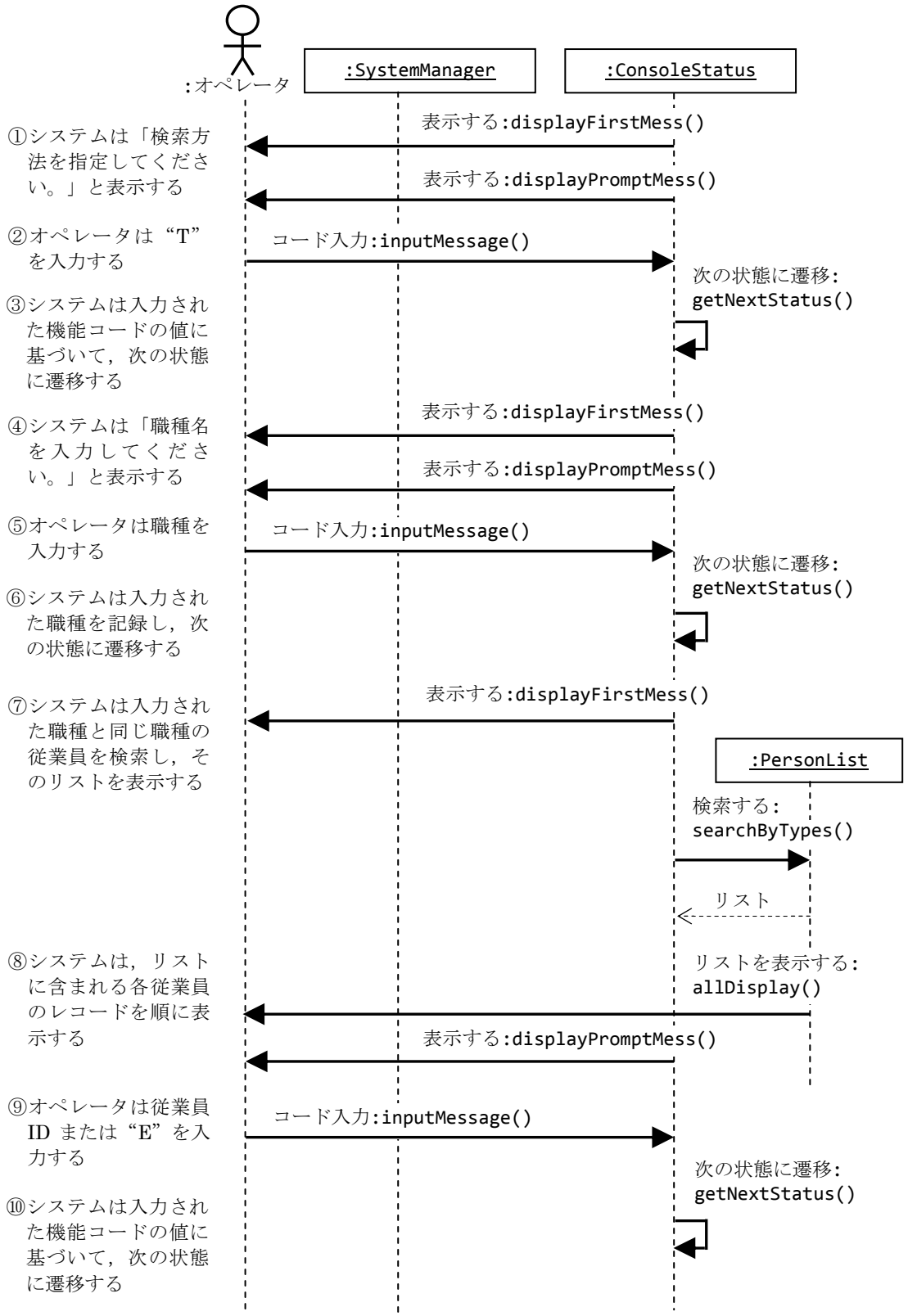
４．シーケンス図

４．１．「従業員情報を検索する」のシーケンス図

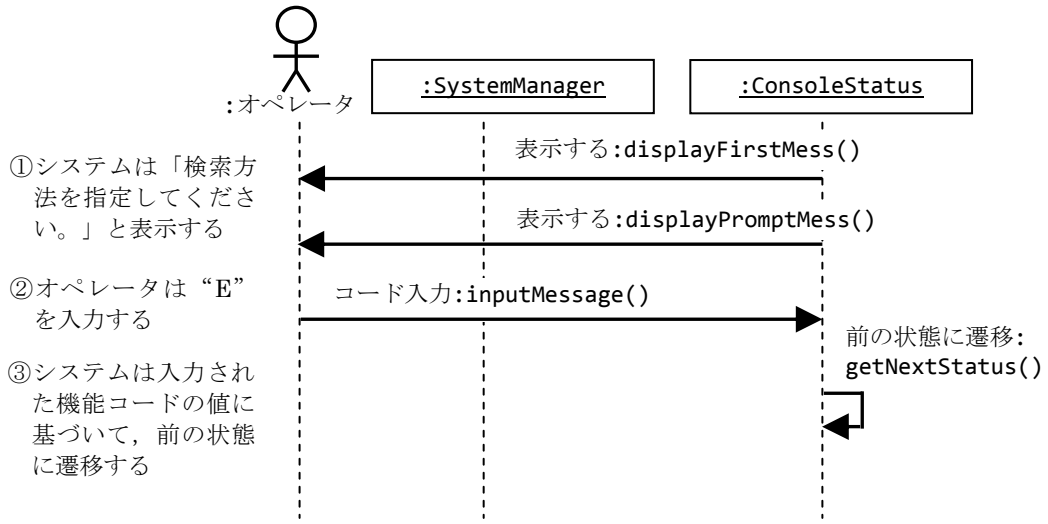
４．１．１．「氏名から検索」が選択された場合



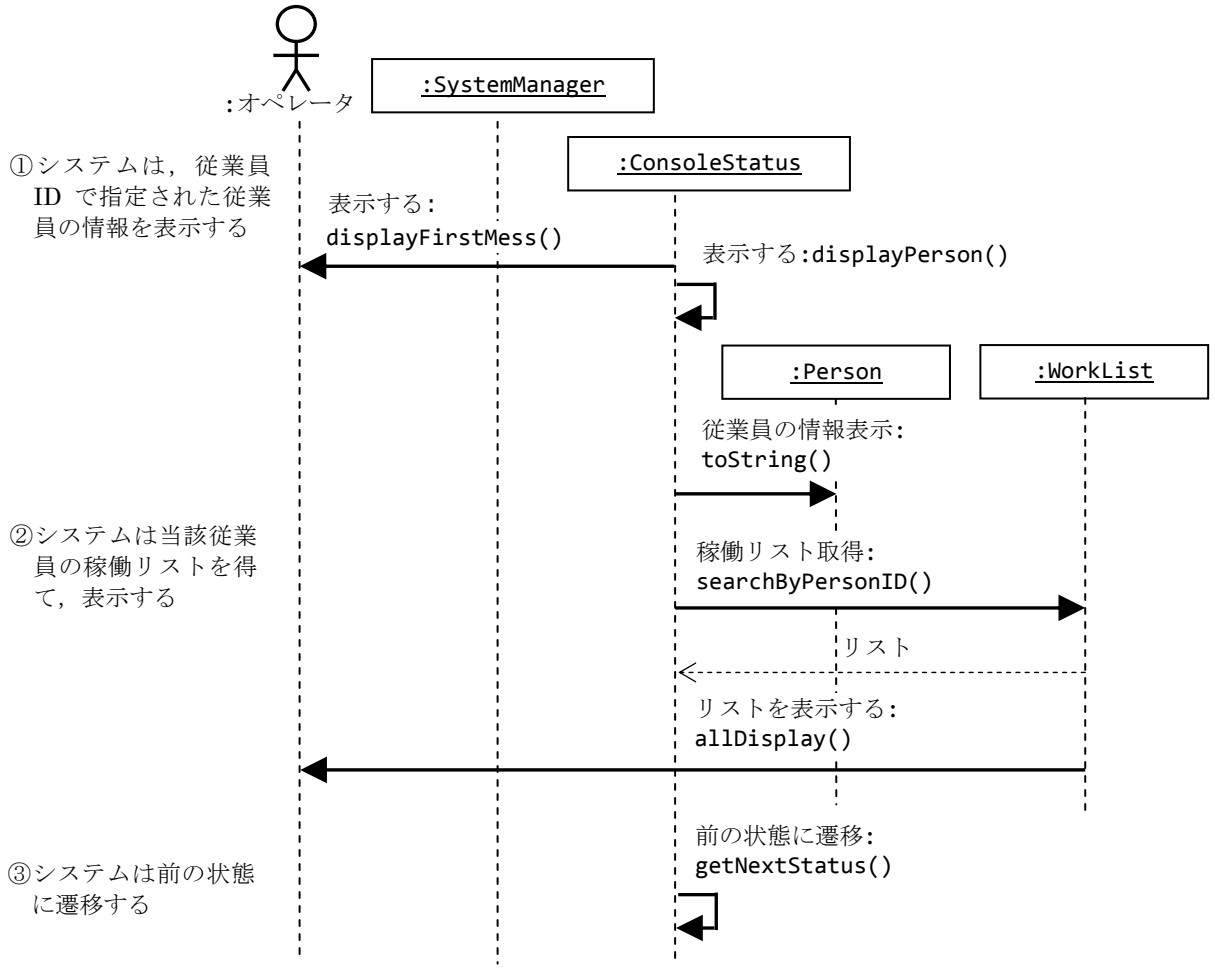
４．１．２．「職種から検索」が選択された場合



４．１．３．「従業員検索終了」が選択された場合

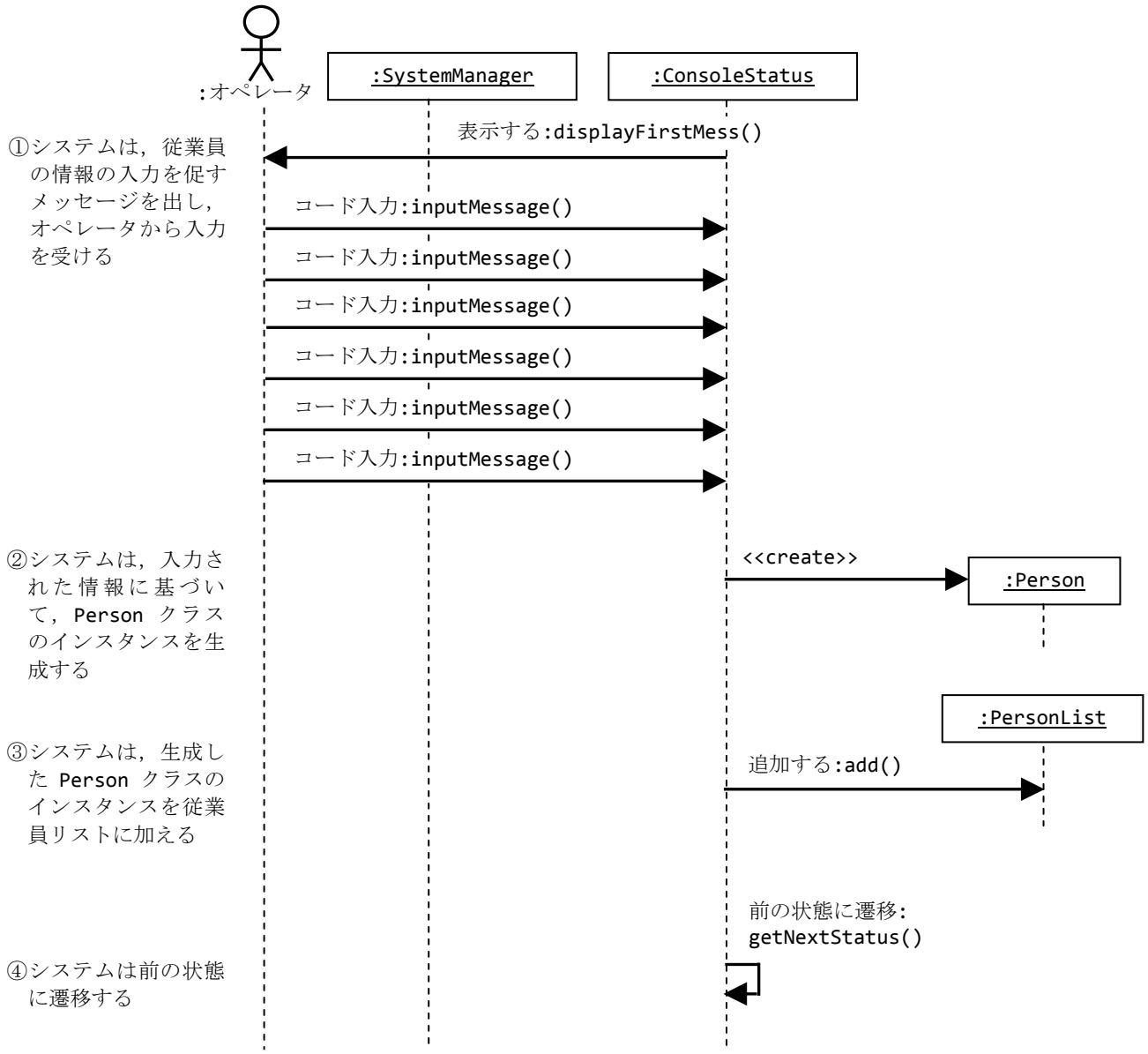


４．１．４．検索結果一覧で「従業員 ID」が入力された場合

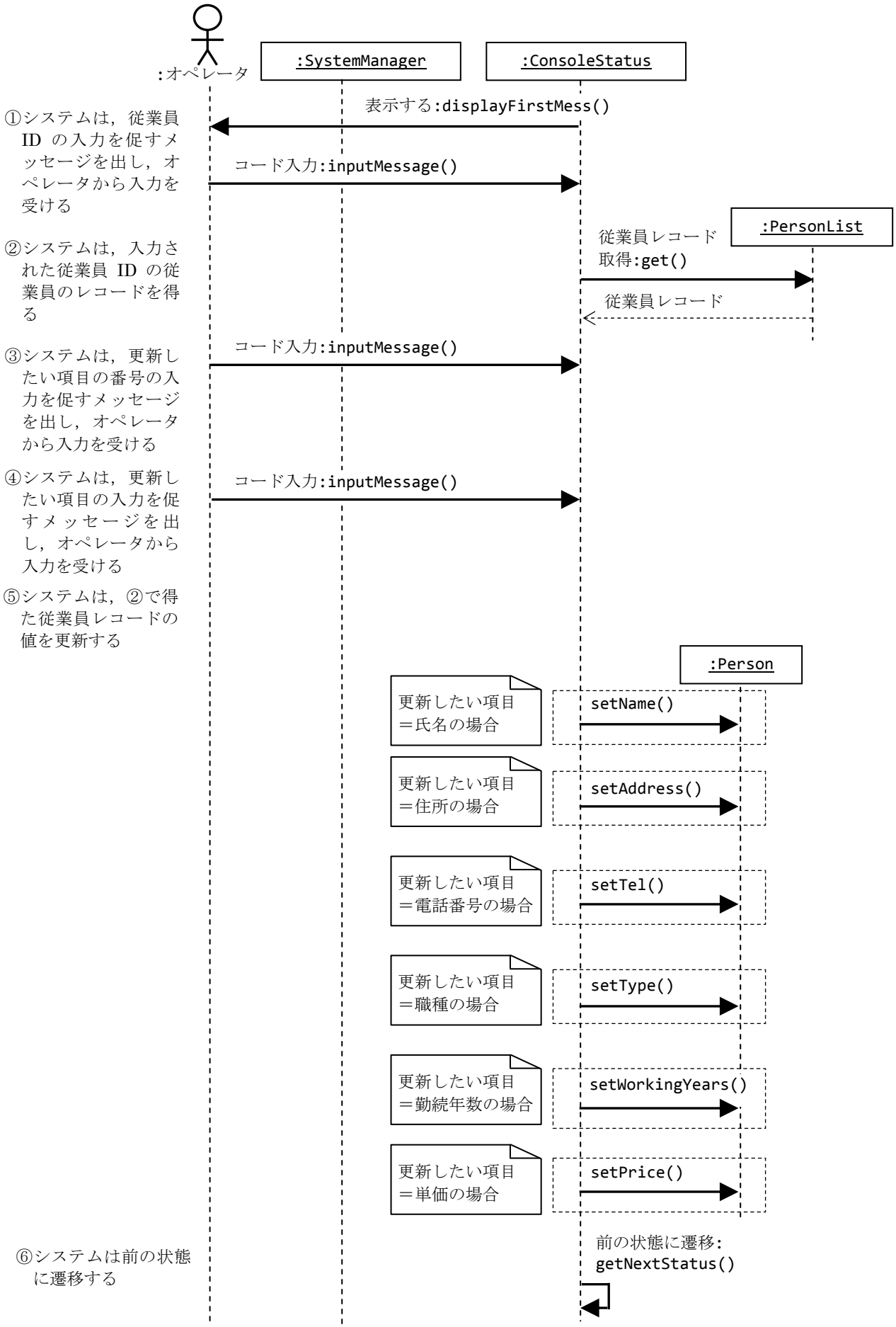


４．２．「従業員情報を管理する」のシーケンス図

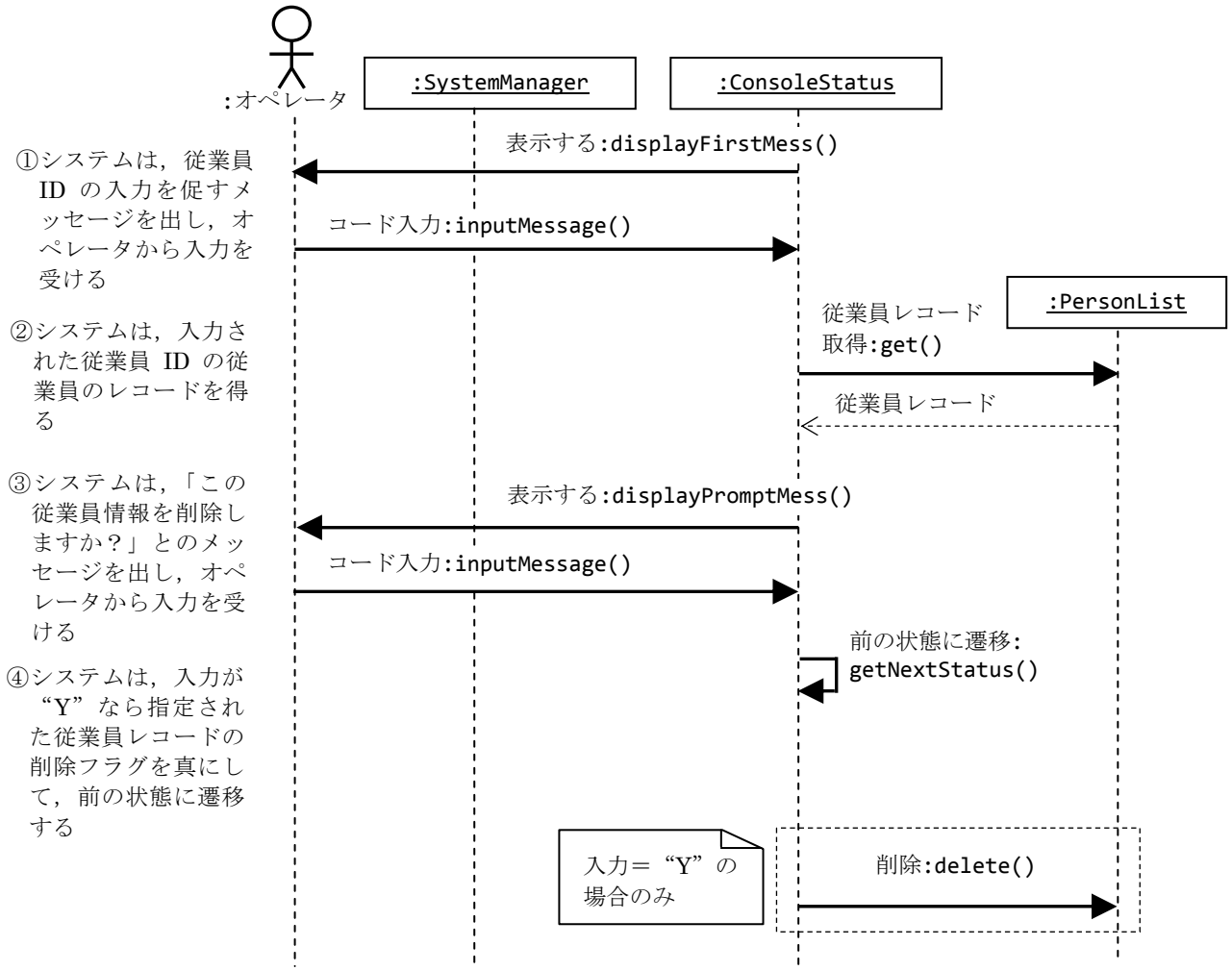
４．２．１．「追加」のコードが選択された場合



４．２．２．「更新」のコードが選択された場合

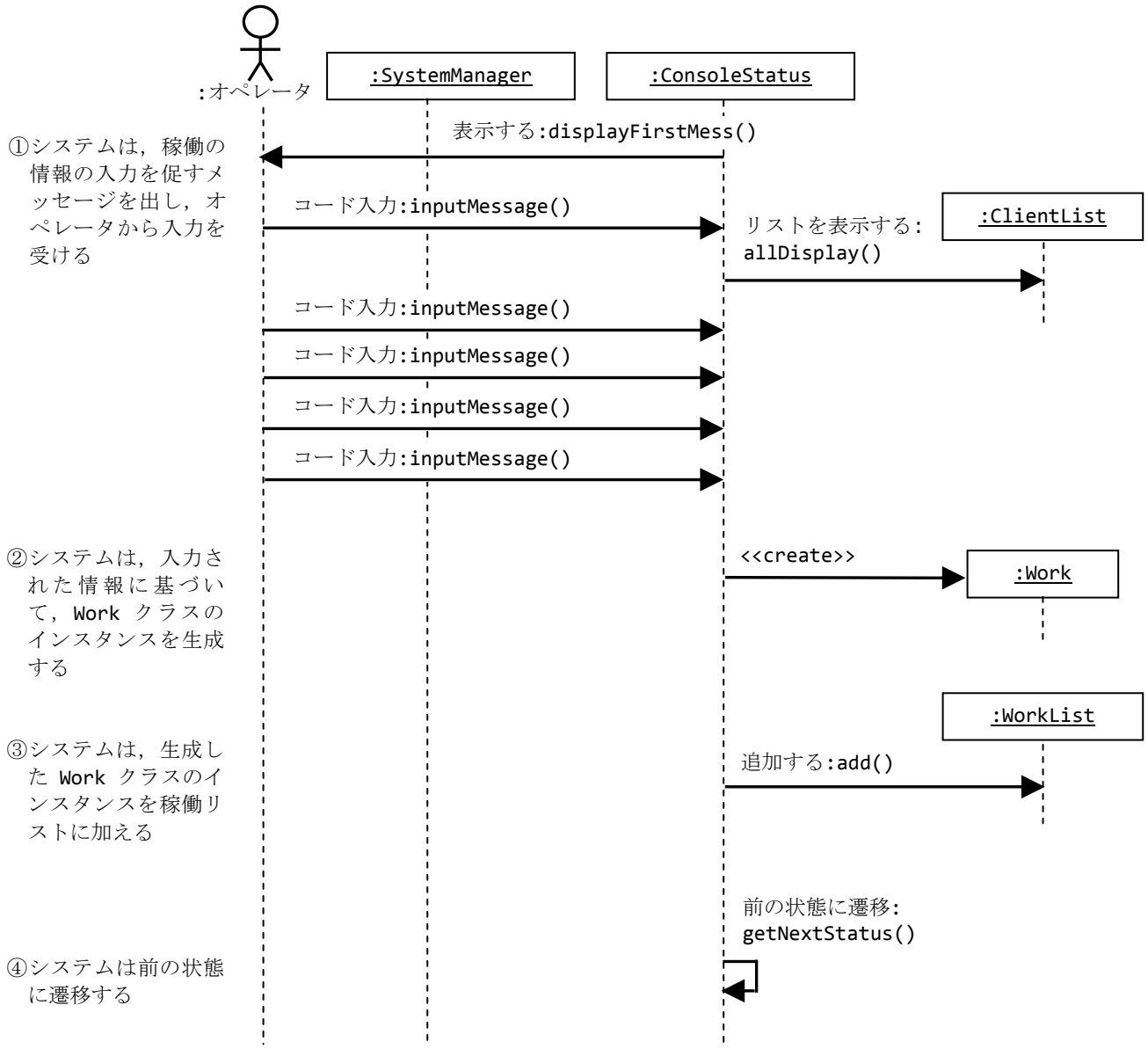


４．２．３．「削除」のコードが選択された場合

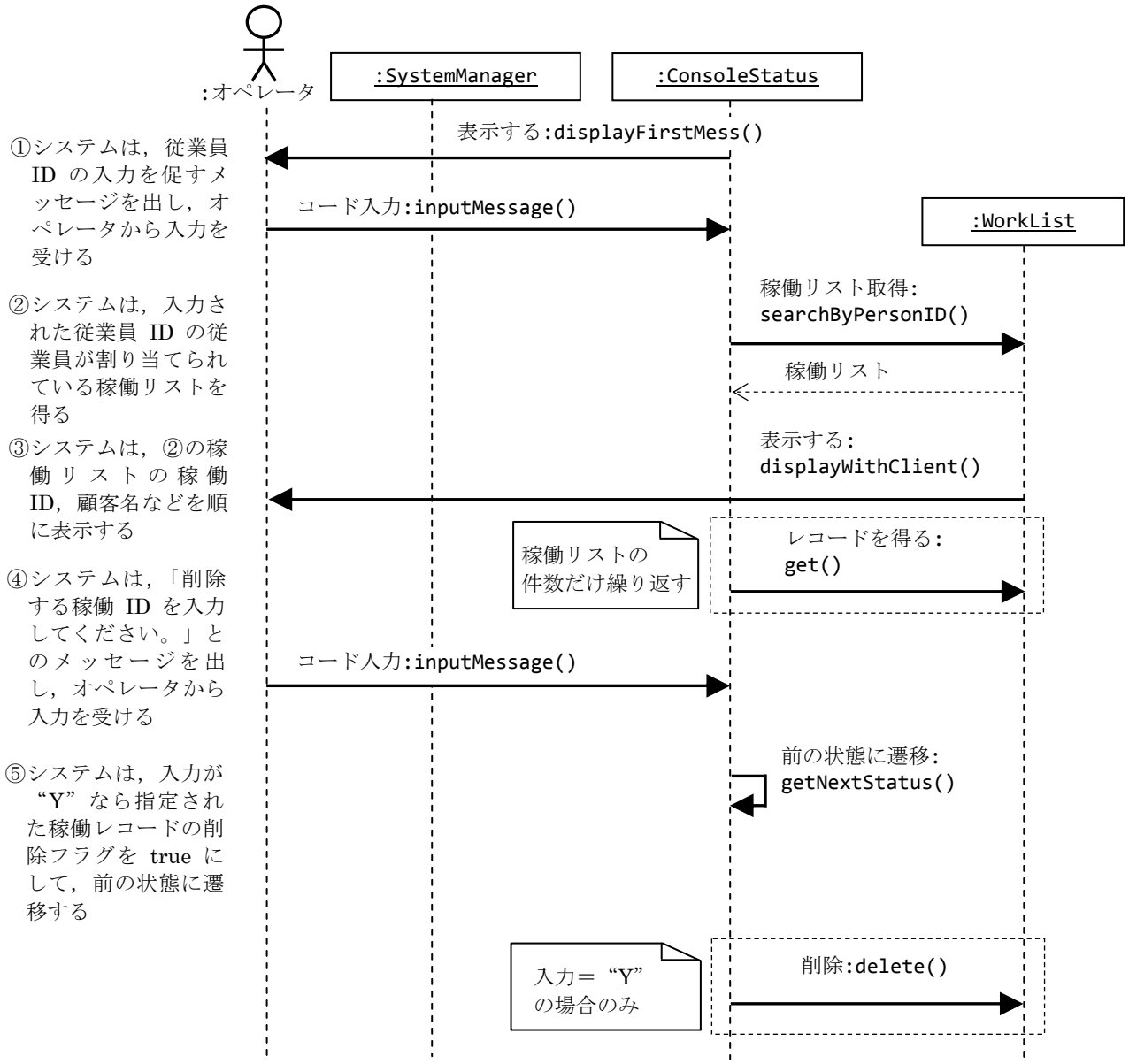


４．３．「稼働状況を管理する」のシーケンス図

４．３．１．「追加」のコードが選択された場合



４．３．２．「削除」のコードが選択された場合



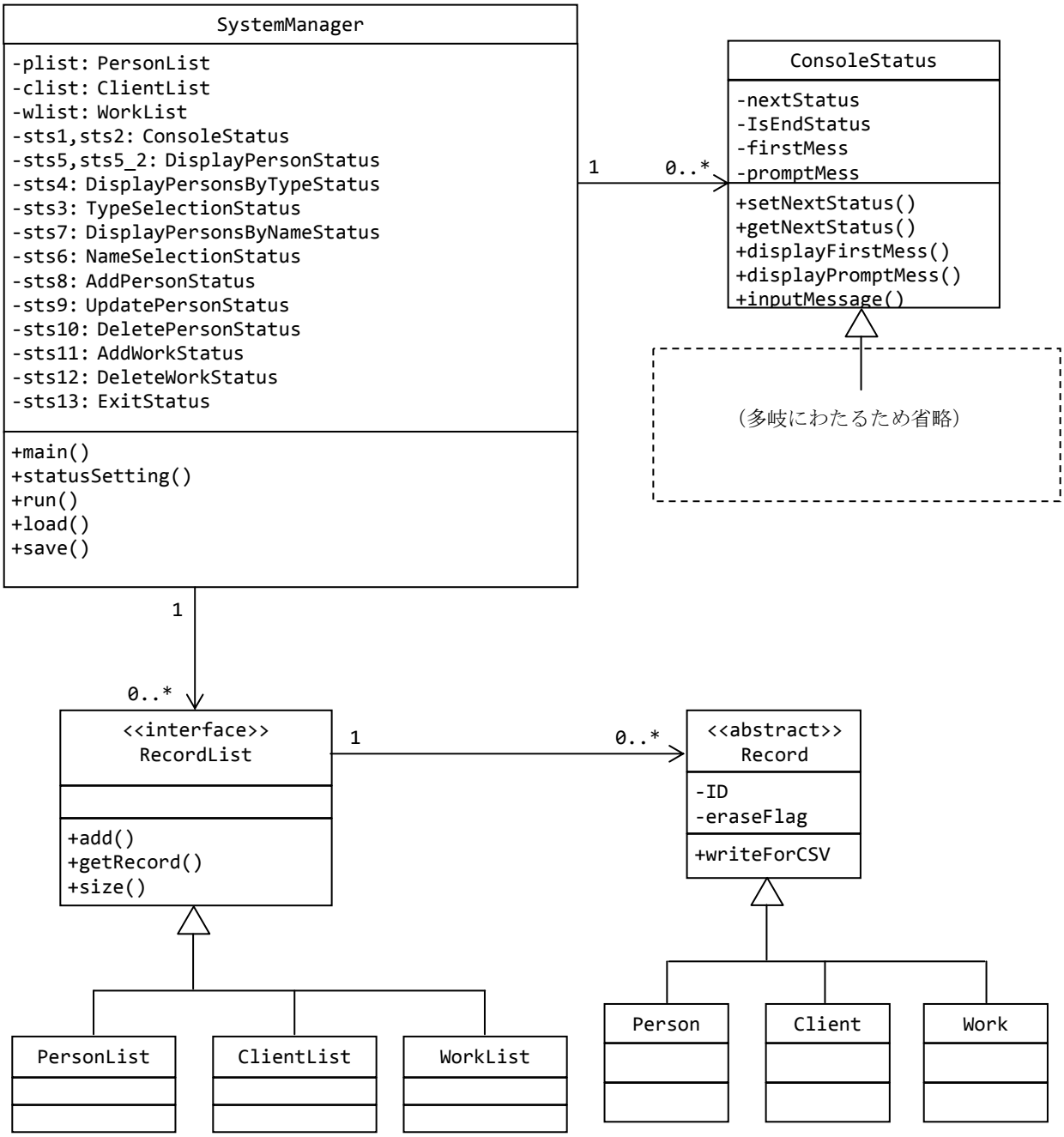
５．クラス図

５．１．クラスの概要

SystemManager	従業員派遣管理メインクラス
Record	マスタファイルから取り出すレコードを表す抽象クラス
Person	従業員マスタファイルのレコードを表す。
Client	顧客マスタファイルのレコードを表す。
Work	稼働マスタファイルのレコードを表す。
RecordList	レコードをまとめたリストを表すインタフェース
PersonList	従業員マスタファイルのレコードをまとめたリストを表す。
ClientList	顧客マスタファイルのレコードをまとめたリストを表す。
WorkList	稼働マスタファイルのレコードをまとめたリストを表す。
FileLoader	マスタファイルの読込み処理を管理する。
FileSaver	マスタファイルの書込み処理を管理する。
ConsoleStatus	システムの状態を表す。また，ユーザとのやりとりやデータの入出力を行う。
DisplayPersonStatus, DisplayPersonsByTypeStatus, TypeSelectionStatus, DisplayPersonsByNameStatus, NameSelectionStatus, AddPersonStatus, UpdatePersonStatus, DeletePersonStatus, AddWorkStatus, DeleteWorkStatus, ExitStatus	ConsoleStatus のサブクラスであり，システムの各状態を表す。また，各状態でのユーザとのやりとりやデータの入出力を行う。

５．２．システムのクラス図

各クラスのコンストラクタ，フィールドを得るメソッド（get…），及びフィールドに値を設定するメソッド（set…）は省略している。



６．メソッド一覧

各クラスのコンストラクタ，フィールドを得るメソッド（get…），及びフィールドに値を設定するメソッド（set…）は省略している。また，スーパークラスから継承したメソッドも省略している。

６．１．クラス SystemManager

クラス SystemManager メソッド一覧

NO.	メソッド名	処理名
1	main	メインメソッド
2	statusSetting	システムの状態の設定
3	run	メインルーチン
4	load	マスタファイルの読込み
5	save	マスタファイルへの書込み

６．１．１．メインメソッド

書式	static void main(String[] args)
パラメータ	コマンドラインの引数
戻り値	なし
処理概要	load，run，save を呼び出す。

６．１．２．システムの状態の設定

書式	void statusSetting()
パラメータ	なし
戻り値	なし
処理概要	システムの各状態における出力メッセージを設定し，各状態が遷移しうる次の状態を設定する。

６．１．３．メインルーチン

書式	void run()
パラメータ	なし
戻り値	なし
処理概要	システムの状態を初期状態とし，オペレータのコマンドに応じて，次の状態に遷移する。

６．１．４．マスタファイルの読込み

書式	void load()
パラメータ	なし
戻り値	なし
処理概要	従業員，顧客，稼働のマスタファイルを読み込む。

６．１．５．マスタファイルへの書込み

書式	void save()
パラメータ	なし
戻り値	なし
処理概要	従業員，顧客，稼働のマスタファイルに書き込む。

６．２．クラス Record

クラス Record メソッド一覧

NO.	メソッド名	処理名
1	writeForCSV	CSV 形式のテキストファイル出力

６．２．１．CSV 形式のテキストファイル出力

書式	abstract String writeForCSV()
パラメータ	なし
戻り値	文字列
機能概要	レコードのデータを CSV 形式にした文字列を返す。抽象メソッドである。

６．３．クラス Person

クラス Person メソッド一覧

NO.	メソッド名	処理名
1	setData	従業員のデータ設定
2	toString	従業員データの文字列の返却

６．３．１．従業員のデータ設定

書式	void setData(String record)
パラメータ	record - 従業員データのレコード
戻り値	なし
機能概要	従業員データのレコードを受け取り，情報を設定する。

６．３．２．従業員データの文字列の返却

書式	String toString()
パラメータ	なし
戻り値	文字列
機能概要	従業員データを表示可能な形式にした文字列を返す。

６．４．クラス Client

クラス Client メソッド一覧

NO.	メソッド名	処理名
1	setData	顧客のデータ設定
2	toString	顧客データの文字列の返却

６．４．１．顧客のデータ設定

書式	void setData(String record)
パラメータ	record - 顧客データのレコード
戻り値	なし
機能概要	顧客データのレコードを受け取り，情報を設定する。

６．４．２．顧客データの文字列の返却

書式	String toString()
パラメータ	なし
戻り値	文字列
機能概要	顧客データを表示可能な形式にした文字列を返す。

６．５．クラス Work

クラス Work メソッド一覧

NO.	メソッド名	処理名
1	setData	稼働のデータ設定
2	toString	稼働データの文字列の返却

６．５．１．稼働のデータ設定

書式	void setData(String record)
パラメータ	record - 稼働データのレコード
戻り値	なし
機能概要	稼働データのレコードを受け取り，情報を設定する。

６．５．２．稼働データの文字列の返却

書式	String toString()
パラメータ	なし
戻り値	文字列
機能概要	稼働データを表示可能な形式にした文字列を返す。

６．６．インタフェース RecordList

インタフェース RecordList メソッド一覧

NO.	メソッド名	処理名
1	add	リストへのレコード追加
2	getRecord	レコードの取得
3	size	レコードの個数の取得

６．６．１．リストへのレコード追加

書式	void add(String data)
パラメータ	data - レコード
戻り値	なし
機能概要	レコードをリストに追加する。

６．６．２．レコードの取得

書式	Record getRecord(int idx)
パラメータ	idx - レコードの位置
戻り値	レコード
機能概要	idx 番目のレコードを得る。

６．６．３．レコードの個数の取得

書式	int size()
パラメータ	なし
戻り値	個数
機能概要	リストがもつレコードの個数を返す。

６．７．クラス PersonList

クラス PersonList メソッド一覧

NO.	メソッド名	処理名
1	add	従業員のレコードの追加
2	delete	レコードの削除
3	allDisplay	リストの全レコード出力
4	find	引数の ID と同じ ID のレコードの検索
5	get	引数の ID のレコードの取得
6	searchByName	指定した氏名の従業員を検索
7	searchByTypes	指定した職種の従業員を検索

６．７．１．従業員のレコードの追加

書式	void add(Person p)
パラメータ	p - レコード
戻り値	なし
機能概要	従業員のレコードをリストに追加する。

６．７．２．レコードの削除

書式	boolean delete(int ID)
パラメータ	ID - 従業員 ID
戻り値	真または偽
機能概要	指定したレコードの削除フラグを真にする。削除できた場合は真，そうでない場合は偽を返す。

６．７．３．リストの全レコード出力

書式	void allDisplay()
パラメータ	なし
戻り値	なし
機能概要	リストの全レコードを順に取り出し，Person.toString メソッドが返す文字列を出力する。

６．７．４．引数の ID と同じ ID のレコードの検索

書式	`int find( int ID )`
パラメータ	`ID` - 従業員 ID
戻り値	従業員 ID
機能概要	引数の ID と同じ ID のレコードを検索する。引数の ID をもつレコードがない場合は−1 を返す。

６．７．５．引数の ID のレコードの取得

書式	`Person get( int ID )`
パラメータ	`ID` - 従業員 ID
戻り値	従業員レコード
機能概要	引数の ID をもつレコードを得る。

６．７．６．指定した氏名の従業員を検索

書式	`PersonList searchByName( String name )`
パラメータ	`name` - 従業員氏名
戻り値	従業員レコードを含む部分的なリスト
機能概要	`name` で指定した氏名を自分の氏名に含んでいる従業員レコードを検索する。

６．７．７．指定した職種の従業員を検索

書式	`PersonList searchByTypes( String type )`
パラメータ	`type` - 職種
戻り値	従業員レコードを含む部分的なリスト
機能概要	`type` で指定した職種をもつ従業員レコードを検索する。

６．８．クラス ClientList

クラス `ClientList` メソッド一覧

NO.	メソッド名	処理名
1	`allDisplay`	リストの全レコード出力
2	`find`	引数の ID と同じ ID のレコードの検索
3	`get`	引数の ID のレコードの取得

６．８．１．リストの全レコード出力

書式	`void allDisplay()`
パラメータ	なし
戻り値	なし
機能概要	リストの全レコードを順に取り出し，`Client.toString` メソッドが返す文字列を出力する。

６．８．２．引数の ID と同じ ID のレコードの検索

書式	`int find( int ID )`
パラメータ	`ID` - 顧客 ID
戻り値	顧客 ID
機能概要	引数の ID と同じ ID のレコードを検索する。引数の ID をもつレコードがない場合は−1 を返す。

６．８．３．引数の ID のレコードの取得

書式	Client get(int ID)
パラメータ	ID - 顧客 ID
戻り値	顧客レコード
機能概要	引数の ID をもつレコードを得る。

６．９．クラス WorkList

クラス WorkList メソッド一覧

NO.	メソッド名	処理名
1	add	稼働のレコードの追加
2	delete	レコードの削除
3	allDisplay	リストの全レコード出力
4	displayWithClient	稼働レコードの情報及び顧客名の表示
5	find	引数の ID と同じ ID のレコードの検索
6	get	引数の ID のレコードの取得
7	searchByPersonID	指定した従業員 ID に割り当てられている稼働を検索

６．９．１．稼働のレコードの追加

書式	void add(Work w)
パラメータ	w - レコード
戻り値	なし
機能概要	稼働のレコードをリストに追加する。

６．９．２．レコードの削除

書式	boolean delete(int ID)
パラメータ	ID - 稼働 ID
戻り値	真または偽
機能概要	指定したレコードの削除フラグを真にする。削除できた場合は真，そうでない場合は偽を返す。

６．９．３．リストの全レコード出力

書式	void allDisplay()
パラメータ	なし
戻り値	なし
機能概要	リストの全レコードを順に取り出し，Work.toString メソッドが返す文字列を出力する。

６．９．４．稼働レコードの情報及び顧客名の表示

書式	void displayWithClient()
パラメータ	なし
戻り値	なし
機能概要	顧客リストを検索して顧客名を取り出し，すべての稼働レコードの稼働 ID，稼働開始年月日，稼働終了年月日を顧客名と共に表示する。

６．９．５．引数の ID と同じ ID のレコードの検索

書式	`int find( int ID )`
パラメータ	`ID` - 稼働 ID
戻り値	稼働 ID
機能概要	引数の ID と同じ ID のレコードを検索する。引数の ID をもつレコードがない場合は−1 を返す。

６．９．６．引数の ID のレコードの取得

書式	`Work get( int ID )`
パラメータ	`ID` - 稼働 ID
戻り値	稼働レコード
機能概要	引数の ID をもつレコードを得る。

６．９．７．指定した従業員 ID に割り当てられている稼働を検索

書式	`WorkList searchByPersonID( int pID )`
パラメータ	`pID` - 従業員 ID
戻り値	稼働レコードを含む部分的なリスト
機能概要	`pID` で指定した従業員 ID に割り当てられている稼働レコードを検索し，それらをまとめた部分的なリストを作成して返す。

６．１０．クラス FileLoader

クラス `FileLoader` メソッド一覧

NO.	メソッド名	処理名
1	`read`	マスタファイルのレコードの読込み

６．１０．１．マスタファイルのレコードの読込み

書式	`void read( RecordList rl )`
パラメータ	`rl` - レコードを格納するリスト
戻り値	なし
機能概要	マスタファイルのレコードを順に読み込み，リストに格納する。

６．１１．クラス FileSaver

クラス `FileSaver` メソッド一覧

NO.	メソッド名	処理名
1	`write`	マスタファイルへのレコードの書込み

６．１１．１．マスタファイルへのレコードの書込み

書式	`void write ( RecordList rl )`
パラメータ	`rl` - レコードが格納されているリスト
戻り値	なし
機能概要	リストに格納されたレコードを順に取り出し，マスタファイルへ書き込む。

６．１２．クラス ConsoleStatus

クラス ConsoleStatus メソッド一覧

NO.	メソッド名	処理名
1	setNextStatus	次に遷移する状態及びコマンドの文字列の設定
2	getNextStatus	次に遷移する状態の取得
3	displayFirstMess	最初に出力するメッセージの表示
4	displayPromptMess	次の状態に遷移することを促すためのメッセージの表示
5	inputMessage	操作者からのキー入力受付

６．１２．１．次に遷移する状態及びコマンドの文字列の設定

書式	void setNextStatus(String s, ConsoleStatus c)
パラメータ	s - 次の状態に遷移するためのコマンドの文字列 c - 次の状態
戻り値	なし
	次に遷移する状態と，その状態に対応するコマンドの文字列を設定する。

６．１２．２．次に遷移する状態の取得

書式	ConsoleStatus getNextStatus(String s)
パラメータ	s - コマンドの文字列
戻り値	次の状態
機能概要	次に遷移する状態を取り出す。

６．１２．３．最初に出力するメッセージの表示

書式	void displayFirstMess()
パラメータ	なし
戻り値	なし
機能概要	最初に出力するメッセージを表示する。

６．１２．４．次の状態に遷移することを促すためのメッセージの表示

書式	void displayPromptMess()
パラメータ	なし
戻り値	なし
機能概要	次の状態に遷移することを促すためのメッセージを表示する。

６．１２．５．操作者からのキー入力受付

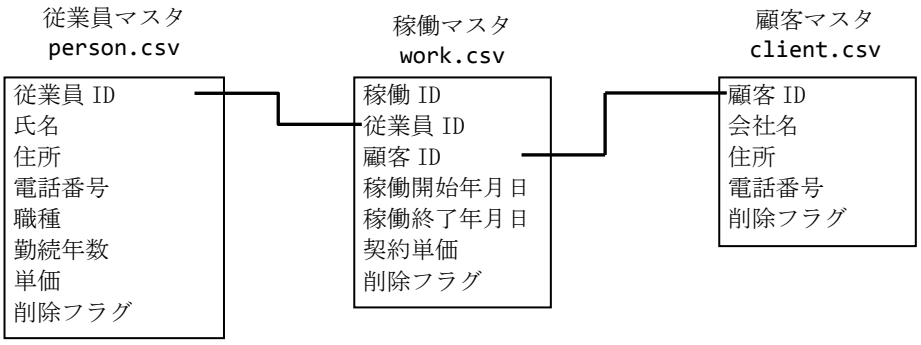
書式	String inputMessage()
パラメータ	なし
戻り値	操作者が入力した文字列
機能概要	操作者からのキー入力を受け付け，その文字列を返す。

※ ConsoleStatus のサブクラスは多岐にわたるため省略しています。

7．ファイル一覧

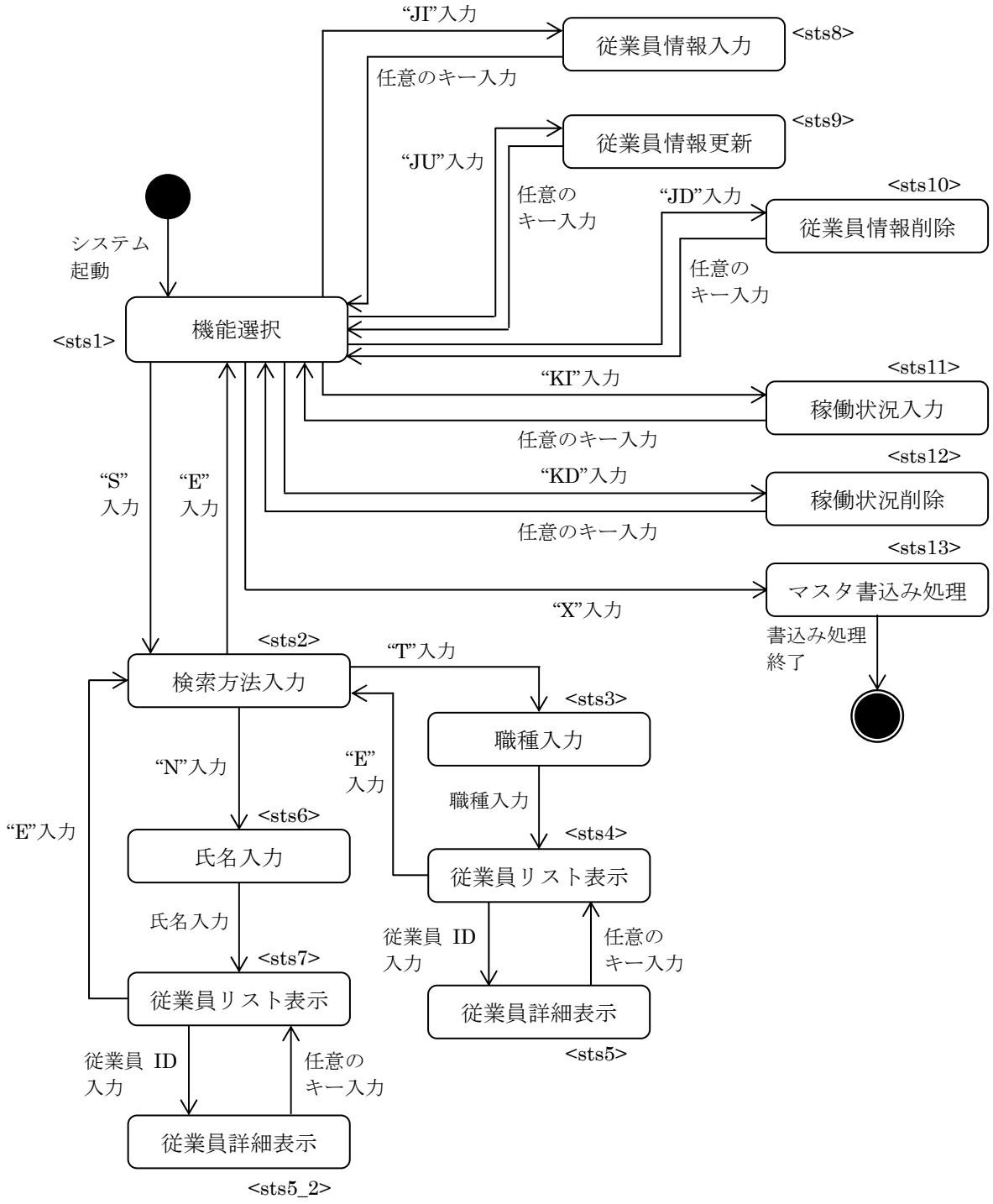
本システムで使用するファイルは，従業員マスタ，稼働マスタ，顧客マスタの三つのテキストファイルがある。それぞれのファイルは CSV 形式であり，レコードごとに改行で区切られ，各レコードの項目間は半角のコンマ（ “,” ）で区切られている。

なお，ファイル中の氏名や会社名などのデータは，半角のコンマを含まないものとする。



８．システムの状態遷移

本システムの状態遷移をステートマシン図で示す。なお，SystemManager クラスがもつ各インスタンス（sts1～sts13 及び sts5_2）がどの状態に対応するかを“<sts n>”という形式で示す。



ConsoleStatus 及びそのサブクラスは，次の状態に遷移するためのコマンドの文字列と次の状態の組を HashMap で保持し，inputMessage メソッドで得たコマンドの文字列に対応する次の状態を，getNextStatus メソッドで返す。次の状態でも同様の処理を繰り返す。

９．画面イメージ

９．１．機能を選択する

```
_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
              従業員派遣管理システム
                  メニュー
  従業員検索(S)
  従業員管理(JI：追加 JU：更新 JD：削除)
  稼働状況管理(KI：追加 KD：削除)
  終了(X)
_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/

どの機能を実行しますか？
[S,JI,JU,JD,KI,KD,X]>
```

９．２．従業員情報を検索する

「従業員検索(S)」選択後，「職種から検索」が選択された場合

```
検索方法を指定してください。
N->氏名から検索        T->職種から検索
E->従業員検索終了(メニューに戻る)
[N,T,E]>T
職種名を入力してください。

[(職種名)]>
```

職種：“プログラマ”を入力

```
ID:0 氏名:池田○実
住所:〒111-1111…… 電話番号:03-3333-1234
職種:プログラマ 勤続年数:10 年 単価:1000 円
ID:2 氏名:塩田○子
住所:〒111-1114…… 電話番号:013-345-6785
職種:プログラマ 勤続年数:12 年 単価:1800 円

E->検索結果一覧終了（検索条件指定に戻る）[(従業員 ID),E]>
```

機能コード：従業員 ID「0」を入力

```
ID:0 氏名:池田○実
住所:〒111-1111…… 電話番号:03-3333-1234
職種:プログラマ 勤続年数:10 年 単価:1000 円
稼働状況-------------------
ID:0 従業員 ID:0 顧客 ID:0 20100401～20100930 契約単価:1200
ID:3 従業員 ID:0 顧客 ID:0 20100101～20101231 契約単価:1000
エンターキーを押すと検索結果一覧に戻ります。
>
```

９．３．従業員を管理する

９．３．１．「追加」のコードが選択された場合

```
氏名を入力してください。>川口〇直
住所を入力してください。>〒111-1111
電話番号を入力してください。>000-000-0001
職種を入力してください。>マネージャ
勤続年数を入力してください。>9
単価を入力してください。>2000
ID:10 で登録されました。
エンターキーを押すとメニューに戻ります。>
```

９．３．２．「更新」のコードが選択された場合

```
従業員 ID を入力してください。
>4
ID:4 氏名:井上〇己
住所:〒111-2223 電話番号:012-345-6789
職種:営業 勤続年数:10 年 単価:1000 円

更新したい項目を入力してください。
1.氏名                  2.住所
3.電話番号              4.職種
5.勤続年数              6.単価

更新する項目の番号を入力してください。
>2

更新後の値を入力してください。
>NewYork
更新しました。
エンターキーを押すとメニューに戻ります。>
```

９．３．３．「削除」のコードが選択された場合

```
従業員 ID を入力してください。
>4
ID:4 氏名:井上〇己
住所:NewYork 電話番号:012-345-6789
職種:営業 勤続年数:10 年 単価:1000 円

この従業員情報を削除しますか？（Y はい  N いいえ）[Y,N]>Y
削除しました。
```

９．４．稼働状況を管理する

９．４．１．「追加」のコードが選択された場合

```
従業員 ID を入力してください。
>2
ID:0 会社名:A 社 住所:〒111-1111…… 電話番号:012-345-6789
ID:1 会社名:B 社 住所:〒111-1112…… 電話番号:013-345-6780
ID:2 会社名:C 社 住所:〒111-1113…… 電話番号:013-345-6781
ID:3 会社名:D 社 住所:〒111-1114…… 電話番号:013-345-6782
ID:4 会社名:E 社 住所:〒111-1115…… 電話番号:013-345-6783
ID:5 会社名:F 社 住所:〒111-1116…… 電話番号:013-345-6784
ID:6 会社名:G 社 住所:〒111-1117…… 電話番号:013-345-6785
ID:7 会社名:H 社 住所:〒111-1118…… 電話番号:013-345-6786
ID:8 会社名:I 社 住所:〒111-1119…… 電話番号:013-345-6787
ID:9 会社名:J 社 住所:〒111-1120…… 電話番号:013-345-6788
ID:10 会社名:K 社 住所:〒111-1121…… 電話番号:013-345-6770
顧客 ID を入力してください。
>0
稼働開始年月日を入力してください。
>20100404
稼働終了年月日を入力してください。
>20101031
単価を入力してください。
>2000
ID:13 で登録されました。
エンターキーを押すとメニューに戻ります。>
```

９．４．２．「削除」のコードが選択された場合

```
従業員 ID を入力してください。
>2
4        20100404～20101031        A 社
13       20100404～20101031        A 社
削除する稼働 ID を入力してください。
[4,13]>4
この稼働情報を削除しますか？（Y はい  N いいえ）[Y,N]>Y
削除しました。
```

従業員派遣管理システム

ソースプログラムリスト

```
/* SystemManager.java
 */

/* SystemManager
 */
public class SystemManager {
    /**
     * フィールド
     */
    private PersonList plist;  // 従業員のリスト
    private ClientList clist;  // 顧客のリスト
    private WorkList wlist;    // 稼働のリスト

    private String pfilename = "person.csv";
    private String cfilename = "client.csv";
    private String wfilename = "work.csv";

    private ConsoleStatus sts1, sts2;
    private DisplayPersonStatus sts5, sts5_2;
    private DisplayPersonsByTypeStatus sts4;
    private TypeSelectionStatus sts3;
    private DisplayPersonsByNameStatus sts7;
    private NameSelectionStatus sts6;
    private AddPersonStatus sts8;
    private UpdatePersonStatus sts9;
    private DeletePersonStatus sts10;
    private AddWorkStatus sts11;
    private DeleteWorkStatus sts12;
    private ExitStatus sts13;

    public static void main( String[] args ) {
        try {
            SystemManager manager = new SystemManager();

            manager.load();
            manager.run();
            manager.save();

        } catch ( Exception e ) {
            e.printStackTrace();
            System.exit( 0 );
        }
    }

    /**
     * コンストラクタ SystemManager
     */
    SystemManager() {

        // 従業員，顧客，稼働の各リストを作成
        this.plist = new PersonList();
        this.clist = new ClientList();
        this.wlist = new WorkList( this.clist );
```

```
    statusSetting();
}

// 状態遷移の設定
/**
 * statusSetting
 */
public void statusSetting() {

    // システム起動時の，機能選択の状態
    sts1 = new ConsoleStatus(
        "_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/¥n" +
        "             従業員派遣管理システム¥n" +
        "                 メニュー¥n" +
        "  従業員検索(S)¥n" +
        "  従業員管理(JI：追加 JU：更新 JD：削除)¥n" +
        "  稼働状況管理(KI：追加 KD：削除)¥n" +
        "  終了(X)¥n" +
        "_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/¥n",
        "どの機能を実行しますか？¥n[S,JI,JU,JD,KI,KD,X]>",
        false
     );
    // 起動時から"S"入力時の状態
    sts2 = new ConsoleStatus(
        "検索方法を指定してください。¥n" +
       "N->氏名から検索       T->職種から検索¥n" +
       "E->従業員検索終了(メニューに戻る)",
       "[N,T,E]>",
        false
     );

    // 従業員 ID を入力すると情報を表示する状態
    sts5 = new DisplayPersonStatus(
        "",
        "エンターキーを押すと検索結果一覧に戻ります。¥n>",
        false,
        wlist
     );
    sts5_2 = new DisplayPersonStatus(
        "",
        "エンターキーを押すと検索結果一覧に戻ります。¥n>",
        false,
        wlist
     );

    // 職種から得た従業者リストを表示し，従業員 ID を入力する状態
    sts4 = new DisplayPersonsByTypeStatus(
        "",
        "E->検索結果一覧終了（検索条件指定に戻る）[(従業員 ID),E]>",
        false,
        plist,
        sts5
     );

    // 職種名を入力する状態
    sts3 = new TypeSelectionStatus(
        "職種名を入力してください。¥n",
        "[(職種名)]>",
        false,
        sts4
```

```
 );

// 氏名から得た従業者リストを表示し，従業員 ID を入力する状態
sts7 = new DisplayPersonsByNameStatus(
    "",
    "E->検索結果一覧終了（検索条件指定に戻る）[(従業員 ID),E]>",
    false,
    plist,
    sts5_2
 );

// 従業員の氏名を入力する状態
sts6 = new NameSelectionStatus(
    "氏名を入力してください。¥n",
    "[(氏名)]>",
    false,
    sts7
 );

// 従業員を追加する状態
sts8 = new AddPersonStatus(
    "",
    "エンターキーを押すとメニューに戻ります。>",
    false,
    plist
 );

// 従業員の情報を更新する状態
sts9 = new UpdatePersonStatus(
    "",
    "更新しました。¥n エンターキーを押すとメニューに戻ります。>",
    false,
    plist
 );

// 従業員を削除する状態
sts10 = new DeletePersonStatus(
    "",
    "この従業員情報を削除しますか？（Y はい  N いいえ）[Y,N]>",
    false,
    plist
 );

// 稼働を追加する状態
sts11 = new AddWorkStatus(
    "",
    "エンターキーを押すとメニューに戻ります。>",
    false,
    clist, wlist
 );

// 稼働を削除する状態
sts12 = new DeleteWorkStatus(
    "",
    "この稼働情報を削除しますか？（Y はい  N いいえ）[Y,N]>",
    false,
    plist, wlist
 );

// システムを終了する状態
```

```
        sts13 = new ExitStatus(
            "",
            "",
            true
         );

        sts1.setNextStatus( "S", sts2 );
        sts1.setNextStatus( "JI", sts8 );
        sts1.setNextStatus( "JU", sts9 );
        sts1.setNextStatus( "JD", sts10 );
        sts1.setNextStatus( "KI", sts11 );
        sts1.setNextStatus( "KD", sts12 );
        sts1.setNextStatus( "X", sts13 );

        sts2.setNextStatus( "N", sts6 );
        sts2.setNextStatus( "T", sts3 );
        sts2.setNextStatus( "E", sts1 );

        sts4.setNextStatus( "E", sts2 );

        sts5.setNextStatus( " ", sts4 );
        sts5_2.setNextStatus( " ", sts7 );

        sts7.setNextStatus( "E", sts2 );

        sts8.setNextStatus( " ", sts1 );

        sts9.setNextStatus( " ", sts1 );

        sts10.setNextStatus( " ", sts1 );

        sts11.setNextStatus( " ", sts1 );

        sts12.setNextStatus( " ", sts1 );
    }

    // システムの起動
    /** run
     * @throws Exception
     */
    public void run() throws Exception {
        // メインルーチン
        ConsoleStatus sts = sts1;
        String cmd;

        while( !sts.getIsEndStatus() ) {
            // 最初に出力するメッセージ
            sts.displayFirstMess();
            // 次の状態に遷移することを促すためのメッセージ
            sts.displayPromptMess();
            // キー入力を受け付ける
            cmd = sts.inputMessage();
            // キー入力されたコマンドによって，
            // 次の状態に遷移する
            sts = sts.getNextStatus( cmd );
        }

        // 終了状態になったら，その旨のメッセージを出力して
        // 終了（保存）処理を行う
        sts.displayFirstMess();
```

```
    }

    // マスタファイルの読込み
    /** load
     * @throws Exception
     */
    public void load() throws Exception {
        // 各 CSV ファイルからレコードを読み取る
        FileLoader pload = new FileLoader( pfilename );
        FileLoader cload = new FileLoader( cfilename );
        FileLoader wload = new FileLoader( wfilename );

        pload.read( plist );
        cload.read( clist );
        wload.read( wlist );
    }

    // マスタファイルの保存
    /** save
     * @throws Exception
     */
    public void save() throws Exception {
        FileSaver psave = new FileSaver( pfilename );
        FileSaver csave = new FileSaver( cfilename );
        FileSaver wsave = new FileSaver( wfilename );

        psave.write( plist );
        csave.write( clist );
        wsave.write( wlist );
    }
}

/* Record.java
 */

/* Record
 */
abstract public class Record {

    /**
     * フィールド
     */
    private int ID;
    private boolean eraseFlag;

    /**
     * コンストラクタ Record
     */
    Record() {
        this.ID = -1;
        this.eraseFlag = false;
    }

    /** コンストラクタ Record
     * @param int ID
     * @param boolean eraseFlag
     */
    Record( int ID, boolean eraseFlag ) {
        this.ID = ID;
        this.eraseFlag = eraseFlag;
```

```
    }

    /** getID
     * @return int
     */
    public int getID() {
        return ID;
    }

    /** setID
     * @param int ID
     */
    public void setID( int ID ) {
        this.ID = ID;
    }

    /** getEraseFlag
     * @return boolean
     */
    public boolean getEraseFlag() {
        return eraseFlag;
    }

    /** setEraseFlag
     * @param boolean eraseFlag
     */
    public void setEraseFlag( boolean eraseFlag ) {
        this.eraseFlag = eraseFlag;
    }

    // 抽象メソッド
    /** writeForCSV
     * 実装内容：CSV ファイルに出力する値を返す
     * @return String
     */
    public abstract String writeForCSV();
}

/* Person.java
 */

/* Person <- Record クラスを継承
 */

public class Person extends Record {
    /**
     * フィールド
     */
    private String name;       // 氏名
    private String address;    // 住所
    private String tel;        // 電話番号
    private String type;       // 職種
    private int workingYears;  // 勤続年数
    private int price;         // 単価

    /** コンストラクタ Person
     * @param int ID
     * @param String name
     * @param String address
     * @param String tel
```

```
 * @param String type
 * @param int workingYears
 * @param int price
 * @param boolean eraseFlag
 */
Person( int ID, String name, String address, String tel,
 String type, int workingYears, int price, boolean eraseFlag ) {
    super( ID, eraseFlag );
    this.name = name;
    this.address = address;
    this.tel = tel;
    this.type = type;
    this.workingYears = workingYears;
    this.price = price;
}

/** コンストラクタ Person
 * @param String record
 * @throws Exception
 */
Person( String record ) throws Exception {
    setData( record );
}

/** setData
 * @param String record
 * @throws Exception
 */
public void setData( String record ) throws Exception {
    String[] n = record.split( "," ); // レコードを","で分割
    try {
        if( n.length != 8 )
         throw new ArrayIndexOutOfBoundsException
           ( "不正なレコードを読み込みました。" );

        setID( Integer.parseInt( n[0] ) );
        name = n[1];
        address = n[2];
        tel = n[3];
        type = n[4];
        workingYears = Integer.parseInt( n[5] );
        price = Integer.parseInt( n[6] );
        if( n[7].equals( "t" ) ) setEraseFlag( true );
        else if( n[7].equals( "f" ) ) setEraseFlag ( false );
        else throw new NumberFormatException();
    } catch( NumberFormatException e ) {
        System.out.println
          ( "数値または削除フラグに変換できない値がレコードに含まれています。" );
        throw e;
    }
}

/** toString
 * @return String
 */
public String toString() {
    String ts = "ID:" + getID() + " 氏名:" + name
                + "¥n 住所:" + address + " 電話番号:" + tel
                + "¥n 職種:" + type
                + " 勤続年数:" + workingYears + "年 "
```

```
                + "単価:" + price + "円";
    return ts;
}

// 抽象メソッド writeForCSV の実装
/** writeForCSV
 * @return String
 */
public String writeForCSV() {
    String s = "" + getID() + "," + name + "," + address +
               "," + tel + "," + type + "," + workingYears +
               "," + price + "," + ( getEraseFlag() ? "t" : "f" );
    return s;
}

/** getName
 * @return String
 */
public String getName() {
    return name;
}

/** getType
 * @return String
 */
public String getType() {
    return type;
}

/** setName
 * @param String name
 */
public void setName( String name ) {
    this.name = name;
}

/** setAddress
 * @param String address
 */
public void setAddress( String address ) {
    this.address = address;
}

/** setTel
 * @param String tel
 */
public void setTel( String tel ) {
    this.tel = tel;
}

/** setType
 * @param String type
 */
public void setType( String type ) {
    this.type = type;
}

/** setWorkingYears
 * @param int workingYears
 */
```

```
    public void setWorkingYears( int workingYears ) {
        this.workingYears = workingYears;
    }

    /** setPrice
     * @param int price
     */
    public void setPrice( int price ) {
        this.price = price;
    }

}

/* Client.java
 */

/* Client <- Record クラスを継承
 */
public class Client extends Record {
    /**
     * フィールド
     */
    private String name;       // 会社名
    private String address;    // 住所
    private String tel;        // 電話番号

    /** コンストラクタ Client
     * @param String record
     * @throws Exception
     */
    Client( String record ) throws Exception {
        setData( record );
    }

    /** setData
     * @param String record
     * @throws Exception
     */
    public void setData( String record ) throws Exception {
        String[] n = record.split( "," ); // レコードを","で分割
        try {
            if( n.length != 5 )
             throw new ArrayIndexOutOfBoundsException
               ( "不正なレコードを読み込みました。" );

            setID( Integer.parseInt( n[0] ) );
            name = n[1];
            address = n[2];
            tel = n[3];
            if( n[4].equals( "t" ) ) setEraseFlag( true );
            else if( n[4].equals( "f" ) ) setEraseFlag ( false );
            else throw new NumberFormatException();
        } catch( NumberFormatException e ) {
            System.out.println
              ( "数値または削除フラグに変換できない値がレコードに含まれています。" );
            throw e;
        }
    }

    /** toString
```

```
     * @return String
     */
    public String toString() {
        String ts = "ID:" + getID() + " 会社名:" + name
                    + " 住所:" + address
                    + " 電話番号:" + tel;
        return ts;
    }

    // 抽象メソッド writeForCSV の実装
    /** writeForCSV
     * @return String
     */
    public String writeForCSV() {
        String s = "" + getID() + "," + name + "," + address +
                    "," + tel + "," + ( getEraseFlag() ? "t" : "f" );
        return s;
    }

    /** getName
     * @return String
     */
    public String getName() {
        return name;
    }

    /** setName
     * @param String name
     */
    public void setName( String name ) {
        this.name = name;
    }

    /** setAddress
     * @param String address
     */
    public void setAddress( String address ) {
        this.address = address;
    }

    /** setTel
     * @param String tel
     */
    public void setTel( String tel ) {
        this.tel = tel;
    }
}

/* Work.java
 */

/* Work <- Record クラスを継承
 */
public class Work extends Record {
    /**
     * フィールド
     */
    private int personID;      // 従業員 ID
    private int clientID;      // 顧客 ID
    private String startTime;  // 稼働開始年月日
```

```
private String endTime;        // 稼働終了年月日
private int price;         // 契約単価

/** コンストラクタ Work
 * @param int ID
 * @param int personID
 * @param int clientID
 * @param String startTime
 * @param String endTime
 * @param int price
 * @param boolean eraseFlag
 */
Work( int ID, int personID, int clientID,
      String startTime, String endTime,
      int price, boolean eraseFlag ) {
    super( ID, eraseFlag );
    this.personID = personID;
    this.clientID = clientID;
    this.startTime = startTime;
    this.endTime = endTime;
    this.price = price;
}

/** コンストラクタ Work
 * @param String record
 * @throws Exception
 */
Work( String record ) throws Exception {
    setData( record );
}

/** setData
 * @param String record
 * @throws Exception
 */
public void setData( String record ) throws Exception {
    String[] n = record.split( "," );     // レコードを","で分割
    try {
        if( n.length != 7 )
         throw new ArrayIndexOutOfBoundsException
           ( "不正なレコードを読み込みました。" );

        setID( Integer.parseInt( n[0] ) );
        personID = Integer.parseInt( n[1] );
        clientID = Integer.parseInt( n[2] );
        startTime = n[3];
        endTime = n[4];
        price = Integer.parseInt( n[5] );
        if( n[6].equals( "t" ) ) setEraseFlag( true );
        else if( n[6].equals( "f" ) ) setEraseFlag ( false );
        else throw new NumberFormatException();
    } catch( NumberFormatException e ) {
        System.out.println
          ( "数値または削除フラグに変換できない値がレコードに含まれています。" );
        throw e;
    }
}

/** toString
 * @return String
```

```
 */
public String toString() {
    String ts = "ID:" + getID() + " 従業員 ID:" + personID
                 + " 顧客 ID:" + clientID
                 + " " + startTime + "〜" + endTime
                 + " 契約単価:" + price;
    return ts;
}

// 抽象メソッド writeForCSV の実装
/** writeForCSV
 * @return String
 */
public String writeForCSV() {
    String s = "" + getID() + "," + personID + "," + clientID +
               "," + startTime + "," + endTime +
               "," + price + "," + ( getEraseFlag() ? "t" : "f" );
    return s;
}

/** getPersonID
 * @return int
 */
public int getPersonID() {
    return personID;
}

/** setPersonID
 * @param int personID
 */
public void setPersonID( int personID ) {
    this.personID = personID;
}

/** getClientID
 * @return int
 */
public int getClientID() {
    return clientID;
}

/** setClientID
 * @param int clientID
 */
public void setClientID( int clientID ) {
    this.clientID = clientID;
}

/** getStartTime
 * @return String
 */
public String getStartTime() {
    return startTime;
}

/** setStartTime
 * @param String startTime
 */
public void setStartTime( String startTime ) {
    this.startTime = startTime;
```

```
    }

    /** getEndTime
     * @return String
     */
    public String getEndTime() {
        return endTime;
    }

    /** setEndTime
     * @param String endTime
     */
    public void setEndTime( String endTime ) {
        this.endTime = endTime;
    }

    /** setPrice
     * @param int price
     */
    public void setPrice( int price ) {
        this.price = price;
    }
}

/* RecordList.java
 */

/* RecordList
 */
interface RecordList {

    // 抽象メソッド
    /** add
     * 実装内容：レコードを追加する
     * @param String data
     * @throws Exception
     */
    public abstract void add( String data ) throws Exception;

    // 抽象メソッド
    /** getRecord
     * 実装内容：指定インデックスのレコードを取得する
     * @param int idx
     * @return Record
     */
    public abstract Record getRecord( int idx );

    // 抽象メソッド
    /** size
     * 実装内容：レコード件数を取得する
     * @return int
     */
    public abstract int size();

}

/* PersonList.java
 */

import java.util.List;
```

```
import java.util.ArrayList;

/* PersonList <- RecordList インタフェースを実装
 */
public class PersonList implements RecordList {

    /**
     * フィールド
     */
    private List<Person> list;

    /**
     * コンストラクタ PersonList
     */
    PersonList() {
        this.list = new ArrayList<Person>();
    }

    /** コンストラクタ PersonList
     * @param List<Person> al
     */
    PersonList( List<Person> al ) {
        this.list = al;
    }

    /** size
     * @return int
     */
    public int size() {
        return list.size();
    }

    /** add
     * @param Person p
     */
    public void add( Person p ) {
        for( int idx = 0; idx < list.size(); idx++ ) {
            Person pidx = list.get( idx );
            if( pidx.getID() == p.getID() )    // 同じ ID のレコードがある場合
                return;                        // 何もせず終了
            else if( pidx.getID() > p.getID() ) {
                list.add( idx, p );            // レコードを追加
                return;
            }
        }
        list.add( p );    // リスト末尾にレコードを追加
    }

    /** add
     * @param String data
     * @throws Exception
     */
    public void add( String data ) throws Exception {
        Person p = new Person( data );
        add( p );
    }

    /** getRecord
     * @param int idx
     * @return Record
```

```
 */
public Record getRecord( int idx ) {
    if( idx >= list.size() )
        return null;
    else
        return list.get( idx );
}

/** delete
 * @param int ID
 * @return boolean
 */
public boolean delete( int ID ) {
    int idx;
    Person p;
    if( ( idx = find( ID ) ) == -1 )
        return false;
    else {
        p = get( idx );
        p.setEraseFlag( true );
        return true;
    }
}

/**
 * allDisplay
 */
public void allDisplay() {
    for( Person p : list ) {
        System.out.println( p.toString() );
    }
}

/** find
 * @param int ID
 * @return int
 */
public int find( int ID ) {
    // 引数の ID と同じ ID をもつレコードの位置を検索
    for( int idx = 0; idx < list.size(); idx++ ) {
        Person pidx = list.get( idx );
        if( pidx.getID() == ID )
            return idx;
    }

    return -1;
}

/** get
 * @param int ID
 * @return Person
 */
public Person get( int ID ) {
    Person p;

    // 引数の ID と同じ ID をもつレコードが存在するならば,
    // そのレコードを返す
    int idx;
    if( (idx = find( ID )) != -1 ) {
        p = list.get( idx );
```

```
            // 削除フラグ＝false なら当該レコードを返す
            if( !p.getEraseFlag() )
                return p;
            else
                return null;
        }
        else
            return null;
    }

    /** searchByName
     * @param String name
     * @return PersonList
     */
    public PersonList searchByName( String name ) {

        ArrayList<Person> l = new ArrayList<Person>();

        for( int idx = 0; idx < list.size(); idx++ ) {
            Person p = list.get( idx );
            // idx 番目のレコードの name に引数 name が含まれるか
            // どうかを確認する
            if( p.getName().indexOf( name ) != -1 &&
                !p.getEraseFlag() )
                l.add( p );
        }

        return new PersonList( l );
    }

    /** searchByTypes
     * @param String type
     * @return PersonList
     */
    public PersonList searchByTypes( String type ) {

        ArrayList<Person> l = new ArrayList<Person>();

        for( int idx = 0; idx < list.size(); idx++ ) {
            Person pidx = list.get( idx );
            // idx 番目のレコードの職種 type が引数 type と一致するか
            // どうかを確認する
            if( pidx.getType().equals( type ) &&
                !pidx.getEraseFlag() )
                l.add( pidx );
        }

        return new PersonList( l );
    }
}

/* ClientList.java
 */

import java.util.List;
import java.util.ArrayList;

/* ClientList <- RecordList インタフェースを実装
 */
public class ClientList implements RecordList {
```

```
/**
 * フィールド
 */
private List<Client> list;

/**
 * コンストラクタ ClientList
 */
ClientList() {
    this.list = new ArrayList<Client>();
}

/** コンストラクタ ClientList
 * @param List<Client> al
 */
ClientList( List<Client> al ) {
    this.list = al;
}

/** size
 * @return int
 */
public int size() {
    return list.size();
}

/** add
 * @param String data
 * @throws Exception
 */
public void add( String data ) throws Exception {
    Client p = new Client( data );
    for( int idx = 0; idx < list.size(); idx++ ) {
        Client pidx = list.get( idx );
        if( pidx.getID() == p.getID() )    // 同じ ID のレコードがある場合
            return;                        // 何もせず終了
        else if( pidx.getID() > p.getID() ) {
            list.add( idx, p );            // レコードを追加
            return;
        }
    }
    list.add( p );    // リスト末尾にレコードを追加
}

/** getRecord
 * @param int idx
 * @return Record
 */
public Record getRecord( int idx ) {
    if( idx >= list.size() )
        return null;
    else
        return list.get( idx );
}

/**
 * allDisplay
 */
public void allDisplay() {
```

```
        for( Client p : list ) {
            System.out.println( p.toString() );
        }
    }

    /** find
     * @param int ID
     * @return int
     */
    public int find( int ID ) {
        // 引数の ID と同じ ID をもつレコードの位置を検索
        for( int idx = 0; idx < list.size(); idx++ ) {
            Client pidx = list.get( idx );
            if( pidx.getID() == ID )
                return idx;
        }

        return -1;
    }

    /** get
     * @param int ID
     * @return Client
     */
    public Client get( int ID ) {
        // 引数の ID と同じ ID をもつレコードが存在するならば,
        // そのレコードを返す
        int idx;
        if( (idx = find( ID )) != -1 )
            return list.get( idx );
        else
            return null;
    }
}

/* WorkList.java
 */

import java.util.List;
import java.util.ArrayList;

/* WorkList <- RecordList インタフェースを実装
 */
public class WorkList implements RecordList {

    /**
     * フィールド
     */
    private List<Work> list;
    private ClientList c_list;

    /** コンストラクタ WorkList
     * @param ClientList c_list
     */
    WorkList( ClientList c_list ) {
        this.list = new ArrayList<Work>();
        this.c_list = c_list;
    }

    /** コンストラクタ WorkList
```

```
 * @param List<Work> al
 * @param ClientList c_list
 */
WorkList( List<Work> al, ClientList c_list ) {
    this.list = al;
    this.c_list = c_list;
}

/** size
 * @return int
 */
public int size() {
    return list.size();
}

/** add
 * @param Work w
 */
public void add( Work w ) {
    for( int idx = 0; idx < list.size(); idx++ ) {
        Work widx = list.get( idx );
        if( widx.getID() == w.getID() )    // 同じ ID のレコードがある場合
            return;                        // 何もせず終了
        else if( widx.getID() > w.getID() ) {
            list.add( idx, w );            // レコードを追加
            return;
        }
    }
    list.add( w );    // リスト末尾にレコードを追加
}

/** add
 * @param String data
 * @throws Exception
 */
public void add( String data ) throws Exception {
    Work w = new Work( data );
    add( w );
}

/** getRecord
 * @param int idx
 * @return Record
 */
public Record getRecord( int idx ) {
    if( idx >= list.size() )
        return null;
    else
        return list.get( idx );
}

/** delete
 * @param int ID
 * @return boolean
 */
public boolean delete( int ID ) {
    Work w;
    if( find( ID ) == -1 )
        return false;
    else {
```

```
        w = get( ID );
        w.setEraseFlag( true );
        return true;
    }
}

/**
 * allDisplay
 */
public void allDisplay() {
    for( Work w : list ) {
        System.out.println( w.toString() );
    }
}

// 顧客名とともに ID，稼働開始年月日，稼働終了年月日を出力する
/**
 * displayWithClient
 */
public void displayWithClient() {
    Client c;

    for( Work w : list ) {
        if( !w.getEraseFlag() ) {
            System.out.print( w.getID() + "¥t" );
            System.out.print( w.getStartTime() + "〜" );
            System.out.print( w.getEndTime() + "¥t" );

            c = c_list.get( w.getClientID() );
            if( c != null )
                System.out.println( c.getName() );
            else
                System.out.println( "(顧客名が存在しません。)" );
        }
    }
}

/** find
 * @param int ID
 * @return int
 */
public int find( int ID ) {
    // 引数の ID と同じ ID をもつレコードの位置を検索
    for( int idx = 0; idx < list.size(); idx++ ) {
        Work widx = list.get( idx );
        if( widx.getID() == ID )
            return idx;
    }

    return -1;
}

/** get
 * @param int ID
 * @return Work
 */
public Work get( int ID ) {
    Work w;

    // 引数の ID と同じ ID をもつレコードが存在するならば，
```

```
        // そのレコードを返す
        int idx;
        if( (idx = find( ID )) != -1 ) {
            w = list.get( idx );
            // 削除フラグ=false なら当該レコードを返す
            if( !w.getEraseFlag() )
                return w;
            else
                return null;
        }
        else
            return null;
    }

    /** searchByPersonID
     * @param int pID
     * @return WorkList
     */
    public WorkList searchByPersonID( int pID ) {

        ArrayList<Work> l = new ArrayList<Work>();

        for( int idx = 0; idx < list.size(); idx++ ) {
            Work w = list.get( idx );
            // idx 番目のレコードの personID(従業員 ID)が引数 pID と一致するか
            // どうかを確認する
            if( w.getPersonID() == pID &&
                !w.getEraseFlag() )
                l.add( w );
        }

        return new WorkList( l, c_list );
    }
}

/* ConsoleStatus.java
 */

import java.util.HashMap;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

/* ConsoleStatus
 */
public class ConsoleStatus {
    /**
     * フィールド
     */
    // 次に遷移する状態を記録する HashMap
    private HashMap<String, ConsoleStatus> nextStatus;
    // 終了状態であるかどうかを表す変数
    private boolean IsEndStatus;
    // 状態遷移後，最初に出力するメッセージ
    private String firstMess;
    // 次の状態に遷移することを促すためのメッセージ
    private String promptMess;

    /**
     * コンストラクタ ConsoleStatus
```

```
 * @param String firstMess
 * @param String promptMess
 * @param boolean IsEndStatus
 */
ConsoleStatus( String firstMess, String promptMess, boolean IsEndStatus ) {
    this.nextStatus = new HashMap<String, ConsoleStatus>();
    this.firstMess = firstMess;
    this.promptMess = promptMess;
    this.IsEndStatus = IsEndStatus;
}

// 終了状態かどうかをチェックする
/** getIsEndStatus
 * @return boolean
 */
public boolean getIsEndStatus() {
    return IsEndStatus;
}

// 次の状態をセットする
/** setNextStatus
 * @param String s
 * @param ConsoleStatus c
 */
public void setNextStatus( String s, ConsoleStatus c ) {
    nextStatus.put( s, c );
}

// 次の状態を得る
/** getNextStatus
 * @param String s
 * @return ConsoleStatus
 */
public ConsoleStatus getNextStatus( String s ) {
    // 入力された文字列に対応付けられた次の状態が
    // あるかどうかを判定し，あれば次の状態を返す
    // なければ this(現在の状態)を返す
    ConsoleStatus cs;
    if( (cs = nextStatus.get( s )) != null )
        return cs;
    else
        return this;
}

// 最初に出力するメッセージを表示する
/** displayFirstMess
 * @throws Exception
 */
public void displayFirstMess() throws Exception {
    System.out.println( firstMess );
}

// 次の状態に遷移することを促すためのメッセージの表示
/**
 * displayPromptMess
 */
public void displayPromptMess() {
    System.out.print( promptMess );
}
```

```
    // 操作者からのキー入力を受け付ける
    /** inputMessage
     * @throws IOException
     * @return String
     */
    public String inputMessage() throws IOException {
        String s = null;
        try {

            BufferedReader input
             = new BufferedReader( new InputStreamReader( System.in ) );
            s = input.readLine();

        } catch( IOException e ) {
            System.out.println( "入力中にエラーが発生しました。" );
            throw e;
        }

        return s;
    }
}

/* DisplayPersonStatus.java
 */

/* DisplayPersonStatus
 */
public class DisplayPersonStatus extends ConsoleStatus {

    // フィールド
    private Person pe;
    private WorkList workList; // 稼働状況を得るために用いる稼働のリスト

    /**
     * コンストラクタ DisplayPersonStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     * @param WorkList workList
     */
    DisplayPersonStatus( String firstMess, String promptMess,
                         boolean IsEndStatus, WorkList workList ) {
        super( firstMess, promptMess, IsEndStatus );
        this.pe = null;
        this.workList = workList;
    }

    /** setPersonRecord
     * @param Person pe
     */
    public void setPersonRecord( Person pe ) {
        this.pe = pe;
    }

    /**
     * displayFirstMess
     */
    public void displayFirstMess() {
        displayPerson();
    }
```

```
    // 選択された従業員の ID，氏名などを
    // 表示する処理
    /**
     * displayPerson
     */
    public void displayPerson() {
        // 従業員のレコードの内容を出力
        System.out.println( pe.toString() );

        System.out.println( "稼働状況-------------------" );
        // 当該従業員が割り当てられている
        // 稼働（Work）を表示
        WorkList wl = workList.searchByPersonID( pe.getID() );
        wl.allDisplay();
    }

    // 入力内容が何であろうと，
    // 前の状態に戻る
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
     */
    public ConsoleStatus getNextStatus( String s ) {
        return super.getNextStatus( " " );
    }

}

/* DisplayPersonsByTypeStatus.java
 */

/* DisplayPersonsByTypeStatus
 */
public class DisplayPersonsByTypeStatus extends ConsoleStatus {

    // フィールド
    private String work;
    private PersonList plist;
    private PersonList selectedList;
    private DisplayPersonStatus next;

    /**
     * コンストラクタ DisplayPersonsByTypeStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     * @param PersonList plist
     * @param DisplayPersonStatus next
     */
    DisplayPersonsByTypeStatus( String firstMess, String promptMess,
                        boolean IsEndStatus,
                        PersonList plist, DisplayPersonStatus next ) {
        super( firstMess, promptMess, IsEndStatus );
        this.work = "";
        this.plist = plist;
        this.selectedList = null;
        this.next = next;
    }
```

```
    // 最初に出力するメッセージを表示する
    /** displayFirstMess
     * @throws Exception
     */
    public void displayFirstMess() throws Exception {
        displayList();
        super.displayFirstMess();
    }

    /** setWork
     * @param String work
     */
    public void setWork( String work ) {
        this.work = work;
    }

    // 入力された職種をもつ従業員のレコードだけを
    // 取り出す処理
    /**
     * displayList
     */
    public void displayList() {
        // 入力された職種をもつ従業員のレコードだけを
        // selectedList に取り出す
        selectedList = plist.searchByTypes( work );
        // selectedList の件数＝0 ならば当該職種をもつ
        // 従業員はいないと表示
        if( selectedList.size() <= 0 )
            System.out.println( "従業員が存在しません。" );
        else
            selectedList.allDisplay();
    }

    // 次の状態に遷移することを促すためのメッセージの表示
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
     */
    public ConsoleStatus getNextStatus( String s ) {
        // 数値が入力された場合，その数値と同じ ID をもつ
        // レコードが selectedList にあるかどうか判定し，
        // あればそれを次の状態 DisplayPersonStatus に渡す
        try {
            int i = Integer.parseInt( s );
            Person p = selectedList.get( i );
            if( p == null )
                return this;
            else {
                next.setPersonRecord( p );
                return next;
            }
        } catch( NumberFormatException e ) {
            return super.getNextStatus( s );
        }
    }
}

/* TypeSelectionStatus.java
 */
```

```
import java.io.IOException;

/* TypeSelectionStatus
 */
public class TypeSelectionStatus extends ConsoleStatus {

    // フィールド
    private DisplayPersonsByTypeStatus next;

    /**
     * コンストラクタ TypeSelectionStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     * @param DisplayPersonsByTypeStatus next
     */
    TypeSelectionStatus( String firstMess, String promptMess,
                         boolean IsEndStatus,
                         DisplayPersonsByTypeStatus next ) {
        super( firstMess, promptMess, IsEndStatus );
        this.next = next;
    }

    // 次の状態に遷移することを促すためのメッセージの表示
    /** inputMessage
     * @throws IOException
     * @return String
     */
    public String inputMessage() throws IOException {
        String mess = super.inputMessage();
        next.setWork( mess );

        return mess;
    }

    // このクラスは，次に DisplayPersonsByTypeStatus を呼ぶと
    // 決まっているため，何が入力されても
    // DisplayPersonsByTypeStatus に遷移するようにしている
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
     */
    public ConsoleStatus getNextStatus( String s ) {
        return next;
    }
}

/* DisplayPersonsByNameStatus.java
 */

/* DisplayPersonsByNameStatus
 */
public class DisplayPersonsByNameStatus extends ConsoleStatus {

    // フィールド
    private String name;
    private PersonList plist;
    private PersonList selectedList;
    private DisplayPersonStatus next;
```

```
/**
 * コンストラクタ DisplayPersonsByNameStatus
 * @param String firstMess
 * @param String promptMess
 * @param boolean IsEndStatus
 * @param PersonList plist
 * @param DisplayPersonStatus next
 */
DisplayPersonsByNameStatus( String firstMess, String promptMess,
                    boolean IsEndStatus,
                    PersonList plist, DisplayPersonStatus next ) {
    super( firstMess, promptMess, IsEndStatus );
    this.name = "";
    this.plist = plist;
    this.selectedList = null;
    this.next = next;
}

// 最初に出力するメッセージを表示する
/** displayFirstMess
 * @throws Exception
 */
public void displayFirstMess() throws Exception {
    displayList();
    super.displayFirstMess();
}

// 検索する氏名を登録する
/** setName
 * @param String name
 */
public void setName( String name ) {
    this.name = name;
}

// 入力された氏名の文字列を氏名に含む従業員のレコードだけを
// 取り出す処理
/**
 * displayList
 */
public void displayList() {
    // 入力された氏名に一致または氏名を含む従業員のレコードだけを
    // selectedList に取り出す
    selectedList = plist.searchByName( name );
    // selectedList の件数＝0 ならば当該職種をもつ
    // 従業員はいないと表示
    if( selectedList.size() <= 0 )
        System.out.println( "従業員が存在しません。" );
    else
        selectedList.allDisplay();
}

// 次の状態に遷移することを促すためのメッセージの表示
/** getNextStatus
 * @param String s
 * @return ConsoleStatus
 */
public ConsoleStatus getNextStatus( String s ) {
    // 数値が入力された場合，その数値と同じ ID をもつ
    // レコードが selectedList にあるかどうか判定し，
```

```
        // あればそれを次の状態 DisplayPersonStatus に渡す
        try {
            int i = Integer.parseInt( s );
            Person p = selectedList.get( i );
            if( p == null )
                return this;
            else {
                next.setPersonRecord( p );
                return next;
            }
        } catch( NumberFormatException e ) {
            return super.getNextStatus( s );
        }
    }
}

/* NameSelectionStatus.java
 */

import java.io.IOException;

/* NameSelectionStatus
 */
public class NameSelectionStatus extends ConsoleStatus {

    // フィールド
    private DisplayPersonsByNameStatus next;

    /**
     * コンストラクタ NameSelectionStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     * @param DisplayPersonsByNameStatus next
     */
    NameSelectionStatus( String firstMess, String promptMess,
                         boolean IsEndStatus,
                         DisplayPersonsByNameStatus next ) {
        super( firstMess, promptMess, IsEndStatus );
        this.next = next;
    }

    // 次の状態に遷移することを促すためのメッセージの表示
    /** inputMessage
     * @throws IOException
     * @return String
     */
    public String inputMessage() throws IOException {
        String mess = super.inputMessage();
        next.setName( mess );

        return mess;
    }

    // このクラスは，次に DisplayPersonsByNameStatus を呼ぶと
    // 決まっているため，何が入力されても
    // DisplayPersonsByNameStatus に遷移するようにしている
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
```

```
     */
    public ConsoleStatus getNextStatus( String s ) {
        return next;
    }
}

/* AddPersonStatus.java
 */

/* AddPersonStatus
 */
public class AddPersonStatus extends ConsoleStatus {

    // フィールド
    private PersonList pl;

    private String[] messages = {
        "氏名を入力してください。>",
        "住所を入力してください。>",
        "電話番号を入力してください。>",
        "職種を入力してください。>",
        "勤続年数を入力してください。>",
        "単価を入力してください。>"
    };
    private String[] data = new String[ 6 ];

    /**
     * コンストラクタ AddPersonStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     * @param PersonList pl
     */
    AddPersonStatus( String firstMess, String promptMess,
                     boolean IsEndStatus, PersonList pl ) {
        super( firstMess, promptMess, IsEndStatus );
        this.pl = pl;
    }

    // 最初に出力するメッセージを表示する
    // このクラスでは従業員のデータの入力処理
    // のみを行う
    /** displayFirstMess
     * @throws Exception
     */
    public void displayFirstMess() throws Exception {
        // messages の各文字列を順に表示しながら
        // キーボードから氏名，住所などを得ていく
        for( int idx = 0; idx < messages.length; idx++ ) {
            System.out.print( messages[ idx ] );
            data[ idx ] = inputMessage();
        }

        try {
            int wy = Integer.parseInt( data[ 4 ] );   // 勤続年数
            int pr = Integer.parseInt( data[ 5 ] );   // 単価

            Person new_p = new Person(
              pl.size(), // 現在の PersonList のレコード数を
                         // 新しいレコードの ID とする
```

```
              data[ 0 ], data[ 1 ], data[ 2 ], data[ 3 ],
              wy, pr, false
            );

            // 新しいレコードを追加
            pl.add( new_p );
            System.out.println( "ID:" + new_p.getID() + "で登録されました。" );
        } catch( NumberFormatException e ) {
            System.out.println( "数値に変換できないデータが入力されています。" );
            System.out.println( "再入力してください。" );
            displayFirstMess();
            return;
        }
    }

    // 次の状態に遷移することを促すためのメッセージの表示
    // このクラスは，初期状態に戻ると決まっているため，何が
    // 入力されても初期状態に遷移するようにしている
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
     */
    public ConsoleStatus getNextStatus( String s ) {
        return super.getNextStatus( " " );
    }
}

/* UpdatePersonStatus.java
 */

import java.io.IOException;

/* UpdatePersonStatus
 */
public class UpdatePersonStatus extends ConsoleStatus {

    // フィールド
    private PersonList pl;

    private String[] messages = {
        "1.氏名¥t¥t¥t2.住所¥n",
        "3.電話番号¥t¥t4.職種¥n",
        "5.勤続年数¥t¥t6.単価¥n"
    };
    private String data;

    /**
     * コンストラクタ UpdatePersonStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     * @param PersonList pl
     */
    UpdatePersonStatus( String firstMess, String promptMess,
                        boolean IsEndStatus, PersonList pl ) {
        super( firstMess, promptMess, IsEndStatus );
        this.pl = pl;
        this.data = "";
    }
```

```
// 最初に出力するメッセージを表示する
// このクラスでは従業員のデータの更新処理
// のみを行う
/** displayFirstMess
 * @throws IOException
 */
public void displayFirstMess() throws IOException {
    int id, no, num;

    // ID の入力
    System.out.print( "従業員 ID を入力してください。¥n>" );
    data = inputMessage();
    try {
        id = Integer.parseInt( data ); // 従業員 ID
    } catch( NumberFormatException e ) {
        System.out.println( "数値に変換できないデータが入力されています。" );
        System.out.println( "再入力してください。" );
        displayFirstMess();
        return;
    }

    Person p = pl.get( id );
    if( p == null ) {
        System.out.println( "指定の ID の従業員は存在しません。" );
        System.out.println( "再入力してください。" );
        displayFirstMess();
        return;
    }

    // 従業員の情報の出力
    System.out.println( p.toString() );

    System.out.println( "¥n 更新したい項目を入力してください。" );
    // messages の各文字列を順に表示する
    for( int idx = 0; idx < messages.length; idx++ )
        System.out.print( messages[ idx ] );

    // 更新する項目の番号の入力
    System.out.print( "¥n 更新する項目の番号を入力してください。¥n>" );
    data = inputMessage();

    try {
        no = Integer.parseInt( data ); // 更新する項目の番号

        // 更新する値の入力
        System.out.print( "¥n 更新後の値を入力してください。¥n>" );
        data = inputMessage();

        if( no == 5 || no == 6 ) {
            num = Integer.parseInt( data ); // 勤続年数または単価
            if( no == 5 )
                p.setWorkingYears( num );
            else
                p.setPrice( num );
        }
        else if( no >= 1 && no <= 4 ) {
            switch( no ) {
                case 1:
                    p.setName( data );   break;
                case 2:
```

```
                    p.setAddress( data );   break;
                case 3:
                    p.setTel( data );   break;
                case 4:
                    p.setType( data );   break;
                default:
                    break;
            }
        }
      } catch( NumberFormatException e ) {
          System.out.println( "数値に変換できないデータが入力されています。" );
          System.out.println( "再入力してください。" );
          displayFirstMess();
          return;
      }
    }

    // 次の状態に遷移することを促すためのメッセージの表示
    // このクラスは，初期状態に戻ると決まっているため，何が
    // 入力されても初期状態に遷移するようにしている
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
     */
    public ConsoleStatus getNextStatus( String s ) {
        return super.getNextStatus( " " );
    }
}

/* DeletePersonStatus.java
 */

/* DeletePersonStatus
 */
public class DeletePersonStatus extends ConsoleStatus {

    // フィールド
    private PersonList pl;
    private int id = -1;
    private String data;

    /**
     * コンストラクタ DeletePersonStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     * @param PersonList pl
     */
    DeletePersonStatus( String firstMess, String promptMess,
                    boolean IsEndStatus, PersonList pl ) {
        super( firstMess, promptMess, IsEndStatus );
        this.pl = pl;
        this.id = -1;
        this.data = "";
    }

    // 最初に出力するメッセージを表示する
    // このクラスでは従業員のデータの削除処理
    // のみを行う
    /** displayFirstMess
```

```
     * @throws Exception
     */
    public void displayFirstMess() throws Exception {
        // ID の入力
        System.out.print( "従業員 ID を入力してください。¥n>" );
        data = inputMessage();
        try {
            id = Integer.parseInt( data ); // 従業員 ID
        } catch( NumberFormatException e ) {
            System.out.println( "数値に変換できないデータが入力されています。" );
            System.out.println( "再入力してください。" );
            displayFirstMess();
            return;
        }

        Person p = pl.get( id );
        if( p == null ) {
            System.out.println( "指定の ID の従業員は存在しません。" );
            System.out.println( "再入力してください。" );
            displayFirstMess();
            return;
        }

        // 従業員の情報の表示
        System.out.println( p.toString() + "¥n" );
    }

    // Y が入力された場合指定されたレコードを削除,
    // N(またはそれ以外)の場合何もせずに
    // 初期状態に遷移するようにしている
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
     */
    public ConsoleStatus getNextStatus( String s ) {
        if( s.equals( "Y" ) ) {
            System.out.println( "削除しました。" );
            pl.delete( id );
        }

        return super.getNextStatus( " " );
    }

}

/* AddWorkStatus.java
 */

/* AddWorkStatus
 */
public class AddWorkStatus extends ConsoleStatus {

    // フィールド
    private ClientList cl;
    private WorkList wl;

    private String[] messages = {
        "従業員 ID を入力してください。¥n>",
        "顧客 ID を入力してください。¥n>",
        "稼働開始年月日を入力してください。¥n>",
```

```
    "稼働終了年月日を入力してください。¥n>",
    "単価を入力してください。¥n>"
};
private String[] data = new String[ 5 ];

/**
 * コンストラクタ AddWorkStatus
 * @param String firstMess
 * @param String promptMess
 * @param boolean IsEndStatus
 * @param ClientList cl
 * @param WorkList wl
 */
AddWorkStatus( String firstMess, String promptMess,
                boolean IsEndStatus, ClientList cl,
                WorkList wl ) {
    super( firstMess, promptMess, IsEndStatus );
    this.cl = cl;
    this.wl = wl;
}

// 最初に出力するメッセージを表示する
// このクラスでは稼働のデータの入力処理
// のみを行う
/** displayFirstMess
 * @throws Exception
 */
public void displayFirstMess() throws Exception {
    // messages の各文字列を順に表示しながら
    // キーボードから氏名，住所などを得ていく
    System.out.print( messages[ 0 ] );
    data[ 0 ] = inputMessage();
    cl.allDisplay();
    System.out.print( messages[ 1 ] );
    data[ 1 ] = inputMessage();
    for( int idx = 2; idx < messages.length; idx++ ) {
        System.out.print( messages[ idx ] );
        data[ idx ] = inputMessage();
    }

    try {
        int pid = Integer.parseInt( data[ 0 ] );    // 従業員 ID
        int cid = Integer.parseInt( data[ 1 ] );    // 顧客 ID
        int pr = Integer.parseInt( data[ 4 ] );     // 契約単価

        Work new_w = new Work(
          wl.size(), // 現在の WorkList のレコード数を
                     // 新しいレコードの ID とする
          pid, cid, data[ 2 ], data[ 3 ],
          pr, false
        );

        // 新しいレコードを追加
        wl.add( new_w );
        System.out.println( "ID:" + new_w.getID() + "で登録されました。" );
    } catch( NumberFormatException e ) {
        System.out.println( "数値に変換できないデータが入力されています。" );
        System.out.println( "再入力してください。" );
        displayFirstMess();
        return;
```

```
        }
    }

    // 次の状態に遷移することを促すためのメッセージの表示
    // このクラスは，初期状態に戻ると決まっているため，何が
    // 入力されても初期状態に遷移するようにしている
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
     */
    public ConsoleStatus getNextStatus( String s ) {
        return super.getNextStatus( " " );
    }
}

/* DeleteWorkStatus.java
 */

import java.lang.NumberFormatException;

/* DeleteWorkStatus
 */
public class DeleteWorkStatus extends ConsoleStatus {

    // フィールド
    private PersonList pl;
    private WorkList wl;
    private int id = -1;
    private String data;
    private WorkList selectedWl = null;

    /**
     * コンストラクタ DeleteWorkStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     * @param PersonList pl
     * @param WorkList wl
     */
    DeleteWorkStatus( String firstMess, String promptMess,
                      boolean IsEndStatus, PersonList pl, WorkList wl ) {
        super( firstMess, promptMess, IsEndStatus );
        this.pl = pl;
        this.wl = wl;
        this.id = -1;
        this.data = "";
        this.selectedWl = null;
    }

    // 最初に出力するメッセージを表示する
    // このクラスでは稼働のデータの削除処理
    // のみを行う
    /** displayFirstMess
     * @throws Exception
     */
    public void displayFirstMess() throws Exception {
        String  comma;
        // ID の入力
        System.out.print( "従業員 ID を入力してください。¥n>" );
        data = inputMessage();
```

```
        try {
            id = Integer.parseInt( data ); // 従業員 ID
        } catch( NumberFormatException e ) {
            System.out.println( "数値に変換できないデータが入力されています。" );
            System.out.println( "再入力してください。" );
            displayFirstMess();
            return;
        }

        Person p = pl.get( id );
        if( p == null ) {
            System.out.println( "指定の ID の従業員は存在しません。" );
            System.out.println( "再入力してください。" );
            displayFirstMess();
            return;
        }

        selectedWl = wl.searchByPersonID( p.getID() );
        // 指定した従業員 ID の従業員が割り当てられている
        // 稼働の表示
        selectedWl.displayWithClient();

        System.out.print( "削除する稼働 ID を入力してください。¥n[" );
        comma = "";
        for( int idx = 0; idx < wl.size(); idx++ ) {
            Work w = selectedWl.get( idx );

            // idx の値と同じ稼働 ID のレコードがない場合，飛ばす
            if( w == null ) continue;
            System.out.print( comma + w.getID() );
            comma = ",";
        }
        System.out.print( "]>" );
        data = inputMessage();
        try {
            id = Integer.parseInt( data ); // 削除する稼働 ID
        } catch( NumberFormatException e ) {
            id = -1;
        }
    }

    // Y が入力された場合指定されたレコードを削除,
    // N(またはそれ以外)の場合何もせずに
    // 初期状態に遷移するようにしている
    /** getNextStatus
     * @param String s
     * @return ConsoleStatus
     */
    public ConsoleStatus getNextStatus( String s ) {
        if( s.equals( "Y" ) ) {
            wl.delete( id );
            System.out.println( "削除しました。" );
        }

        return super.getNextStatus( " " );
    }
}

/* ExitStatus.java
 */
```

```
/* ExitStatus
 */
public class ExitStatus extends ConsoleStatus {

    /* コンストラクタ ExitStatus
     * @param String firstMess
     * @param String promptMess
     * @param boolean IsEndStatus
     */
    ExitStatus( String firstMess, String promptMess, boolean IsEndStatus ) {
        super( firstMess, promptMess, IsEndStatus );
    }

    // システムを終了する特別なメッセージ
    /**
     * displayFirstMess
     */
    public void displayFirstMess() {
        System.out.println( "システムを終了します。" );
    }
}

/* FileLoader.java
 */

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.FileNotFoundException;


/* FileLoader
 */
public class FileLoader {

    /**
     * フィールド
     */
    private String fileName;

    /** コンストラクタ FileLoader
     * @param String fileName
     */
    FileLoader( String fileName ) {
        this.fileName = fileName;
    }

    /** read
     * @param RecordList rl
     * @throws Exception
     */
    public void read( RecordList rl ) throws Exception {
        // ファイル名＝fileName のファイルを読む
        BufferedReader input = null;
        try {
            try {

                input = new BufferedReader( new FileReader( fileName ) );
                String s;
```

```
                while( (s = input.readLine()) != null )
                    rl.add( s );
            } finally {
                if (input != null)
                    input.close();
            }
        } catch( FileNotFoundException e ) {
            System.out.println( fileName + "を開くことができません。" );
            throw e;
        } catch( IOException e ) {
            System.out.println( "ファイル読込み中にエラーが発生しました。" );
            throw e;
        }

    }
}

/* FileSaver.java
 */

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.FileNotFoundException;


/* FileSaver
 */
public class FileSaver {

    /**
     * フィールド
     */
    private String fileName;

    /** コンストラクタ FileSaver
     * @param String fileName
     */
    FileSaver( String fileName ) {
        this.fileName = fileName;
    }

    /** write
     * @param RecordList rl
     * @throws FileNotFoundException, IOException
     */
    public void write( RecordList rl )
     throws FileNotFoundException, IOException {
        // ファイル名＝fileName のファイルに
        // 書き込みを行う
        BufferedWriter output = null;
        try {
            try {

                output = new BufferedWriter( new FileWriter( fileName ) );

                // 引数 RecordList に格納されたレコードを１行ずつ書き込む
                int idx = 0;
                Record r;
                while( idx < rl.size() ) {
```

```
                r = rl.getRecord( idx );
                output.write( r.writeForCSV() );
                output.newLine();     // 改行文字をファイルに出力
                idx++;
            }
        } finally {
            if (output != null)
                output.close();
        }
    } catch( FileNotFoundException e ) {
        System.out.println( fileName + "を開くことができません。" );
        throw e;
    } catch( IOException e ) {
        System.out.println( "ファイル書込み中にエラーが発生しました。" );
        throw e;
    }
  }
}
```

ＵＭＬ解説書

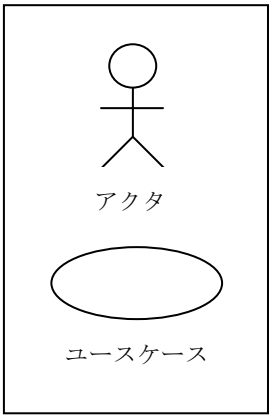
UML（Unified Modeling Language）とは，オブジェクト指向分析・設計においてシステムをモデル化（図式化）する際の記法を規定した言語である。UML で使用される図は 10 種類程（ユースケース図，シーケンス図，コラボレーション図，クラス図など）あり，システム分析・設計の工程に合わせて図を使い分ける。ここではユースケース図，シーケンス図，クラス図，ステートマシン図及びシステムの流れを文章化したイベントフローについて記述する。

１．ユースケース図

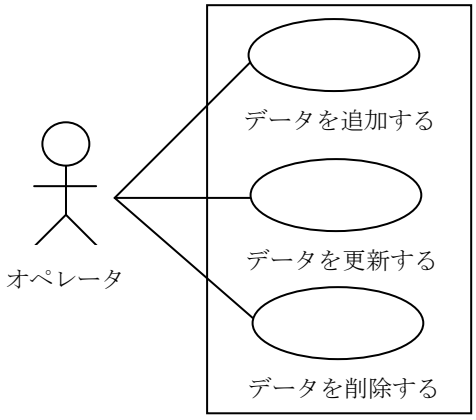
ユースケースは，アクタとシステム間で行われる相互作用と対話（メッセージ）を把握するための表現方法である。アクタとは，システムとやり取りを行うすべての要素（システムを使用するユーザー，システムと通信する別のシステムなど）を表し，ユースケースは，システムが実行する機能を表す。そして，ユースケース図では, アクタとユースケースをコミュニケーションで結び表現する。ユースケース図では，アクタ及びユースケースをそれぞれ＜図 1＞のように表す。

＜図 2＞は「従業員データ登録」のユースケース図の例である。アクタはオペレータであり，ユースケースは「データを追加する」,「データを更新する」,「データを削除する」の各機能である。また，アクタとユースケースをつないでいるのがコミュニケーションである。この例では，オペレータは各機能を使用するため，オペレータと各機能がコミュニケーションで結ばれている。

＜図 1＞　一般図



＜図 2＞　従業員データ登録



２．イベントフロー

イベントフローとは，各ユースケースの詳細を文章化したもので，下記の項目の順に記述する。

（１） 一つの基本フロー（最も重要である正常なケース）

（２） 複数の代替フロー（基本フロー以外の下記のケース）

・基本フローの異形なケース

・特異なケース

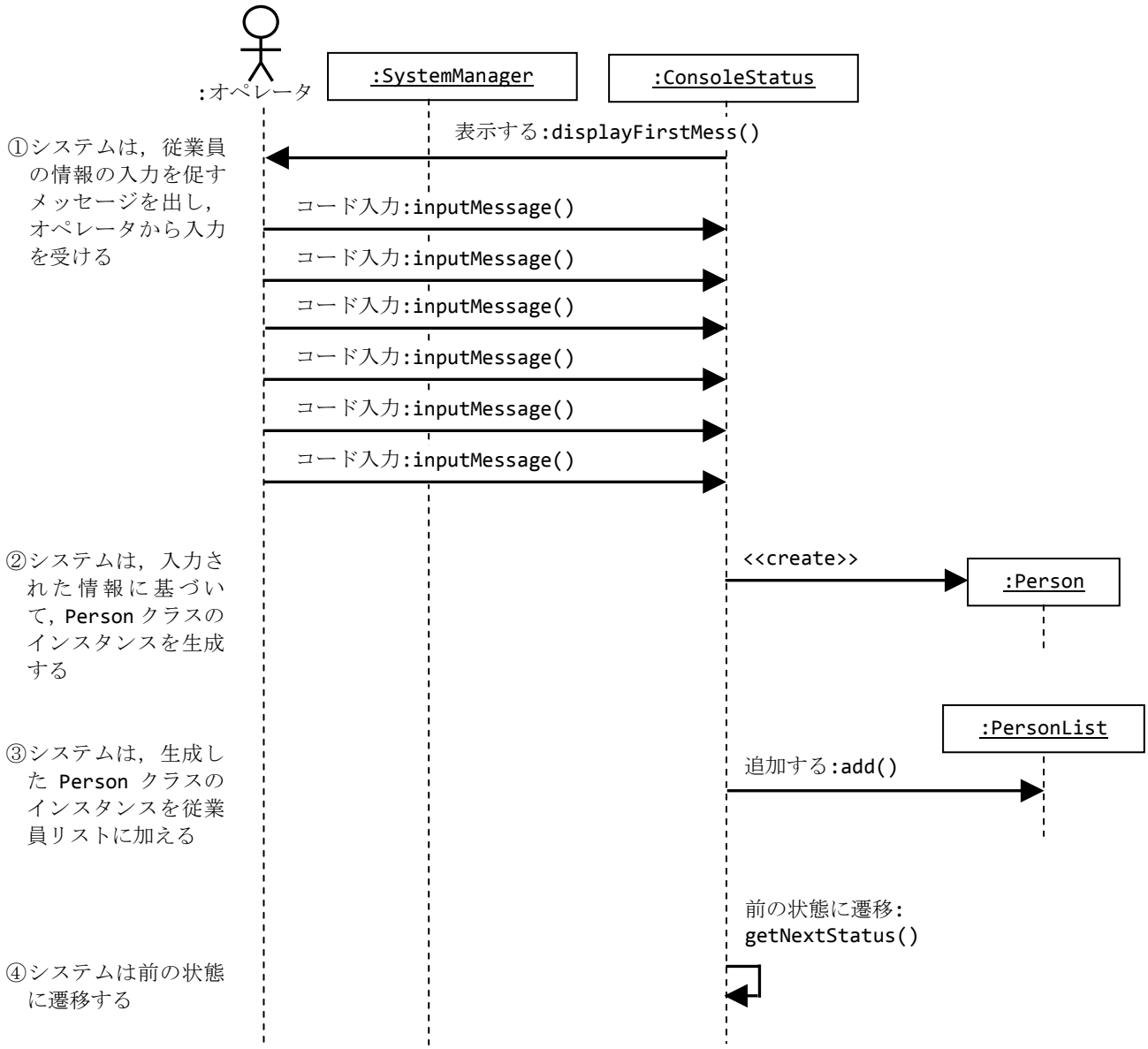
・例外フロー（例外やエラーケース）

３．シーケンス図

シーケンス図は，アクタとシステム間の相互作用を時間的な側面からモデル化するための図である。各動作をステップとして，各ステップを図の左側に，動作の行われる順に上から下へ記述する。各ステップに作用するオブジェクト（システムの機能）を図の上部に左側から記述する。さらに，各オブジェクト間で送受信される対話（メッセージ）を矢印で示し，矢印上部には対話（メッセージ）の内容を記述する。なお，以下に示す記号でコメントを表記することができる。分岐や繰返し処理などがあって図が複雑になる場合に，分岐内容などをコメントで表記することがある。

＜図 3＞は「従業員データ登録」の「データの追加」のユースケースのシーケンス図の例である。

＜図 3＞　従業員データ登録における「データ追加」シーケンス図

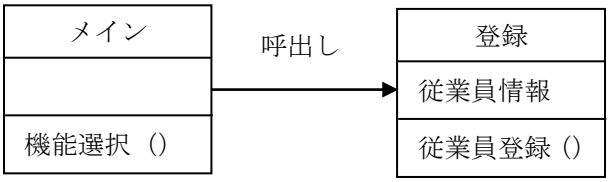


4．クラス図

クラスはオブジェクトを抽象化して表したもので，クラス図は，様々な種類のクラスとそれらのクラス間の関係（リレーションシップ）で構成される。クラスは 3 分割の矩形で表現し，上部にクラス名，中央に属性，下部に操作を記述する。属性，操作は省略することもできる。属性には，属性名，型，初期値などを記述し，操作には，操作名，パラメータ，戻り値の型などを記述する。

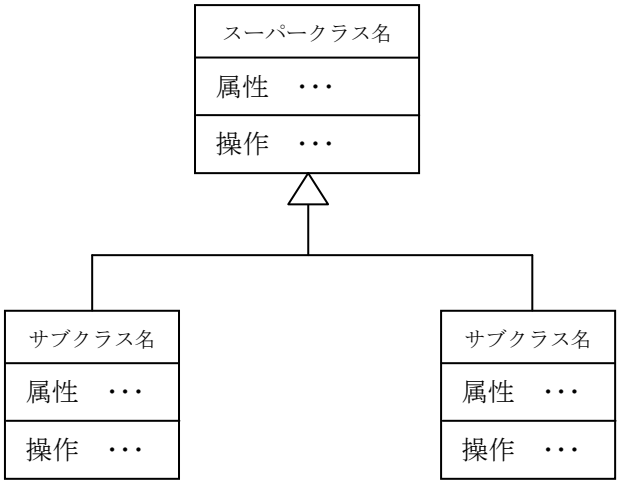
＜図 4＞　一般形

＜図 5＞　従業員データ登録におけるクラス図



クラス間の継承は，図 6 に示すように，スーパークラスとサブクラスとを線で結び，スーパークラス側に三角形を付与することで表す。

＜図 6＞　クラス間の継承の例



５．ステートマシン図

ステートマシン図は，状態遷移図と同様に，システムのとりうる状態と，ある状態から別の状態に遷移する様子を示すものである。状態を角丸四角形，開始状態を黒丸，終了状態を二重線黒丸で表す。また，遷移を矢印で表す。遷移の上側には，状態が遷移するときのイベント，条件及び遷移によってもたらされる効果を“イベント[条件]/効果”という形式で表記できる。

＜図 7＞　ステートマシン図の例