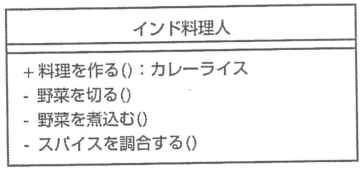
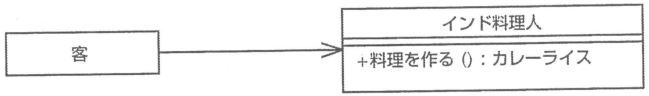
図2-1-9 インド料理人クラス図



このクラス図ではインド料理人をモデル化しています。ここで、インド料理人オブジェクトは、客オブジェクトから次のようにみえます。

図2-1-10 客とインド料理人クラス図



客オブジェクトからは、インド料理人オブジェクトの操作は「料理を作る()」しか見えません。インド料理人オブジェクトは、「料理を作ります。私が作るのはカレーライスですよ。」としか外部オブジェクトに対して公開していないからです。このように、外部に対して公開している部分が、インド料理人クラス（オブジェクト）のインターフェースとなります。

オブジェクト指向では、クラスの属性や操作を内部に隠蔽するか、外部に公開するかは「可視性」という概念を利用して表現します。客は、カレーライスを作る具体的な工程（野菜を切る、スパイスを調合する…）を知りません。また厨房に顔を出して、料理方法や材料に口を出すこともできません。

このように、内部に隠蔽する属性や操作の可視性はプライベート（private）となり、属性名やメソッド名に"-"をつけて表記します。逆に外部に公開する属性や操作の可視性はパブリック（public）となり、属性名や操作名に"+"をつけて表記します。

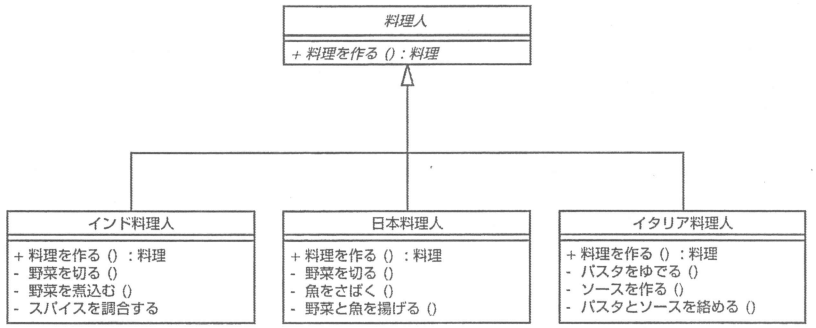
多態性（ポリモーフィズム）

先ほどの例で出てきたインド料理人クラスを汎化して、料理人クラスを作成します。さらに料理人クラスを継承した日本料理人クラスとイタリア料理人クラスを作成します。ここで、料理人クラスは汎化によって抽象度が上がっているため、カレーライスではなく料理を結果として返すことにします。

可視性
属性や振る舞いを内部に隠蔽するのかインターフェースとして外部に公開するのかを表現します。

多態性
同一のメッセージに対して、受け取った側が状況に応じて振る舞いを変えること。多態性は、利用する側の再利用性を促進します。

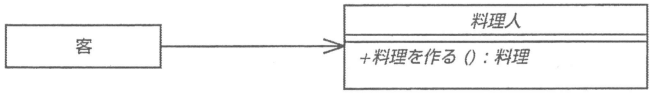
図2-1-11 料理人クラス図



料理人クラスはイタリック表示で記述されています。これは料理人クラスが、インスタンス化されない概念的なクラスである「抽象クラス」であることを表しています。抽象クラスは概念的なクラスなので、実際の操作を行うことができません。つまり、料理人クラスには料理を作るという操作がありますが、実際に料理を作ることはできません（プログラムの処理は書きません）。このイタリック体の操作「料理を作る」は、料理人クラスを継承する派生クラスで、必ず何らかの料理（カレーライスかも知れませんし、天ぷら、パスタかも知れません）を作るように処理の中身が定義される必要があります。

料理人クラスのインターフェースは、図2-1-12のようになります。客クラスはインド料理人クラスや日本料理人クラス、イタリア料理人クラスの詳細を知らなくとも料理人クラスという括りで料理を頼むことができます。

図2-1-12 客と料理人クラス図



客クラスは料理人クラスに対して「料理を作る」というメッセージを送ります。料理人クラスは、その時にインスタンス化されているのがインド料理人クラスか、日本料理人クラスか、イタリア料理人クラスかによって、動的にカレーライスを料理するのか、和食を料理するのか、イタリアンを料理するのかを決定します。これは、料理人がその時々の状況に応じてさまざまな振る舞いをするともいえます。

このように同じ振る舞いを呼び出しても異なる機能を果たすことを「多態性」といいます（英語でポリモーフィズムと呼ばれることもあります）。この多態性は、オブジェクト指向の目指す目標の１つであるアプリケーションの再利用性を向上するための重要な概念です。