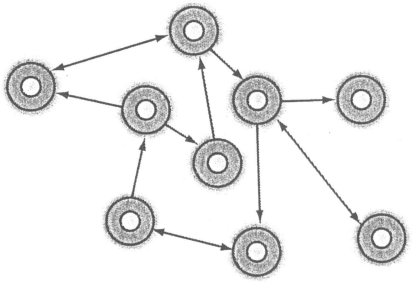
実際のシステムは、多数のオブジェクトから構成されます。そして、オブジェクトが他のオブジェクトにメッセージパッシングすることでシステムの要件機能を実現します。つまり、オブジェクトはメッセージをやり取りすることで協調動作し、目的を達成するのです。

図2-1-4 システム全体でのメッセージパッシング



カプセル化

先ほどの例では、窓口係オブジェクトが口座オブジェクトに対して「預金を引き出す」というメッセージパッシングをすることで、口座オブジェクトの預金を引き出すことができました。

これは、オブジェクト指向における重要な考え方の1つです。**オブジェクト同士は、必ずお互いの操作を通して属性を操作する必要があります。**口座オブジェクトの属性である名義人や預金残高は、口座オブジェクト自身が責任をもって管理をします。窓口係オブジェクトが直接勝手に預金残高を操作することはできません。窓口係が預金残高の増減操作を行いたい場合には、必ず口座オブジェクトの操作を呼び出すことになります。

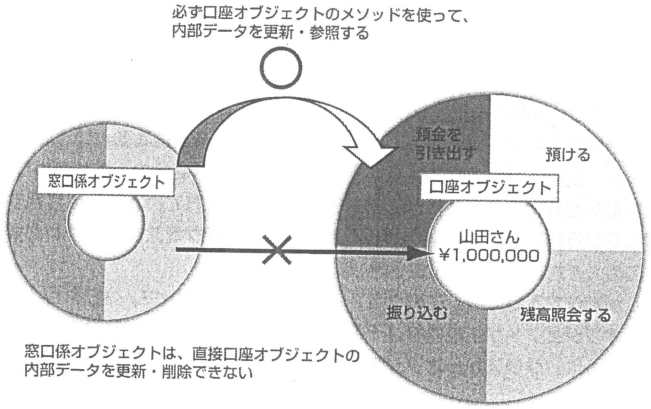
これは一見すると不便なようにも思えるのですが、実は良く考えてみると、窓口係オブジェクトは口座オブジェクトの内部詳細（例えば預金残高）を知らなくてもよいことがわかります。窓口係オブジェクトは「預金を引き出す」という、口座オブジェクトの働きを知ってさえいれば、預金を引き出せるのです。もちろん預金残高が足りない場合は口座オブジェクトがそのことを伝えなければなりません。しかしここでのポイントは、**窓口係オブジェクトが預金残高を気にすることなく、預金を引き出す操作を行うことができる**という点です。

このように、オブジェクト指向ではオブジェクトごとに属性や操作を持ち、責任が分割されています。そのため、外部のオブジェクトからは、**対象のオブジェクトの内部がどうなっているかを知らなくても、公開されている操作を実行して何らかの利益を受けることができます。**

この考え方を**カプセル化**といいます。カプセル化によって、外部のオブジェクトは利用しているオブジェクトの内部構造を知る必要がなくなります。そのため、オブジェクトの内部構造が変更されても外部には影響が及ばなくなります。

薬局で売っている風邪薬のカプセルを思い浮かべてください。カプセルの中身である薬の詳細は知らなくとも、これを飲めば風邪が治るということを知っているのと同様です。

図2-1-5 カプセル化

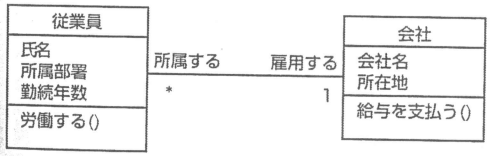


クラス

オブジェクトを**抽象化**した枠組みを**クラス**と呼びます。オブジェクトはクラスを具体化することで作成されます。

「田中さんと佐藤さんは株式会社テクノロジックアートに勤務して、労働の見返りに給料をもらっています。」という内容からクラスを定義し、UMLで表記すると、次のようなモデルになります。

図2-1-6 従業員－会社クラス図



カプセル化
encapsulation。カプセル化によりオブジェクトの詳細を他のオブジェクトから隠蔽することができます。これにより詳細データの扱いを局所化することが可能になります。