オーバーライド

ポリモーフィズムと実行時型

同一クラス内で同名の異なるシグネチャを持つメソッドを定義することができました
このオーバーロードに対し、派生クラスで基底クラスと同じシグネチャを持つメソッドを定義できます
こうすることで、基底クラスのメソッドを派生クラスから隠蔽することができるのです

```
class Kitty {
 public void Write() {
  System.Console.WriteLine("Kitty on your lap");
 }
}

class TokyoMM : Kitty {
 new public void Write() {
  System.Console.WriteLine("Tokyo mew mew");
 }
 static void Main() {
  TokyoMM obj = new TokyoMM();
  obj.Write();
 }
}
```

TokyoMM クラスは Kitty クラスを継承し、Write() メソッドを隠蔽しています
Main() メソッドで TokyoMM クラスのインスタンスを生成し Write() を呼び出しています
TokyoMM のインスタンスはそのクラスの Write() を呼び出すため "Tokyo mew mew" が表示されます

このメソッドの隠蔽は、以前簡単に説明したメンバ変数の隠蔽と同じ考えです
呼び出すメソッドは、インスタンスの型でコンパイル時に決定されるのです

ところが、実は**参照型のキャスト**が行われると話は複雑になります
以前説明したように、派生クラスは**基底クラスのインスタンスを持つ**ため
派生クラスのインスタンスは**基底クラス型に暗黙的にキャストできます**
派生クラスは基底クラスのメンバを持っていることを保証しているため、問題はありません

基底クラスにキャストされたオブジェクトは、派生クラスで新たに追加した機能を隠蔽します
これによって機能は削減されますが、基底クラスだけの共通処理を行うメソッドなどに
アップキャストして渡すことで派生クラスを共通した型で扱うことができるようになります

因みに、基底クラス型オブジェクトを派生クラス型にキャストする場合は明示する必要があります
これは、基底クラス型のインスタンスは派生クラスのインスタンスを持つ保証がないためです

では、次のようなプログラムはどのように動くのでしょうか？

```
abstract class Kitty {
 public void Write() {
  System.Console.WriteLine("Kitty on your lap");
 }
}

class TokyoMM : Kitty {
 new public void Write() {
  System.Console.WriteLine("Tokyo mew mew");
 }
 static void Main() {
  Kitty obj = new TokyoMM();
  obj.Write();
 }
}
```

先ほどとは違い、obj オブジェクトは Kitty 型です
しかし、その実体は TokyoMM 型のインスタンスをキャストしたものです
さて、実行されるメソッドはインスタンスの型か参照型かどちらでしょうか
正解は、コンパイラは静的に型を判断するため "Kitty on your lap" が出力されます
この結果から、Kitty クラスのメソッドが実行されたことがわかります

これは、オブジェクトが実装する固有のメソッドが実行されたためです
このような現象を **アーリーバインディング** と呼びます

しかしこの方法は、コンパイル時に型が判定される静的な方法です
オブジェクトが持つ固有の実装ではなく、インスタンスが持つ実装を動的に選択するには
実行時に動的に呼び出すメソッドを判断する必要があります
このような方法を **レイトバインディング** と呼びます

アップキャストされたオブジェクトの正しい実体を呼び出すには
基底クラスのメソッドを**仮想化**させます
メソッドを仮想化させるには **virtual** 修飾子を指定します
仮想メソッドは、その実体は実行時までわからないことをコンパイラに通知します

基底クラスから継承した仮想メソッドを実体化するには
派生クラスのメソッドの宣言で **override** 修飾子を指定します
override は static や virtual と一緒に宣言することはできません

このように、実行時型でメソッドを呼び出す方法を**オーバーライド**とよび
仮想メソッドを実体化したメソッドを**オーバーライドメソッド**と呼びます

```
class Kitty {
 public virtual void Write() {
  System.Console.WriteLine("Kitty on your lap");
 }
}

class TokyoMM : Kitty {
 public override void Write() {
  System.Console.WriteLine("Tokyo mew mew");
 }
 static void Main() {
  Kitty obj;

  obj = new Kitty();
  obj.Write();

  obj = new TokyoMM();
  obj.Write();
 }
}
```

このプログラムを実行すると、次のような結果になります

Kitty on your lap
Tokyo mew mew

最初の Kitty クラスは基底クラスでありどのクラスも継承していないので
Write() メソッドの実体は Kitty クラスのメソッドであるということはすぐにわかります
そのため、最初の obj.Write() は "Kitty on your lap" を出力します

問題は、この次の Kitty 型にキャストされた TokyoMM 型のオブジェクトです
インスタンスは TokyoMM 型であり、Write() メソッドはオーバーライドされています
obj.Write() は仮想メソッドの実体を実行時に選択して適切なメソッドを呼び出します
これによって、2回目の obj.Write() が "Tokyo mew mew" を出力したのです
オーバーライドは隠蔽と異なりプログラムに高度な多様性を生み出します

派生クラスで仮想メソッドと同一のシグネチャを持つメソッドが宣言された場合
基底クラスの仮想メソッドは隠蔽されてしまいます

```
class A {
 public virtual void M() {
  System.Console.WriteLine("Kitty on your lap");
 }
}

class B : A {
 public override void M() {
  System.Console.WriteLine("Tokyo mew mew");
 }
}

class C : B {
 new public virtual void M() {
  System.Console.WriteLine("Chobits");
 }
}

class D : C {
 public override void M() {
  System.Console.WriteLine("Di Gi Charat");
 }
 static void Main() {
  C objC = new D();
  A objA = new D();

  objC.M();
  objA.M();
 }
}
```

このプログラムを実行すると、次のようになりました

Di Gi Charat
Tokyo mew mew

この結果を見れば、C クラスで A クラスの仮想メソッド M() が隠蔽されていることがわかるでしょう
隠蔽とオーバーライドは曖昧になることはありません

---

virtual

仮想メソッドを宣言し、このメソッドの実装は
実行時までわからないことをコンパイラに通知します
static、abstract、override 修飾子と同時に使用することはできません

override

仮想メソッドをオーバーライドします
非仮想メソッド、及び静的メソッドはオーバーライドできません
new、static、virtual、abstract 修飾子と同時に使用することはできません
