

Multiple Inheritance



In this chapter you will learn the concept of **multiple inheritance** which is an advanced form of inheritance of structure and behaviour.

Multiple Inheritance

So far inheritance has been modelled in such a way that one parent can have many children, but each child can only have one parent. This form of inheritance is also known as **single inheritance**. While this mechanism is extremely powerful, it has two distinct disadvantages.

- Hierarchies can become very complex and grow either too deep or too wide or both. This can cause a so-called 'Yo-Yo' effect, which becomes apparent when trying to find certain attributes in a hierarchy.
- Sometimes, it is more natural that a single class inherits properties from more than one class and artificial constructs have to be modelled which do not reflect the actual appearance.

The latter point is an advanced version of inheritance and is called **multiple inheritance**.

Have a look at the familiar Sloopy hierarchy below. At the sub-sub-class level, we have hovering and skateboarding boys as well as singing and flying girls. It was possible to model all these activities using single inheritance.

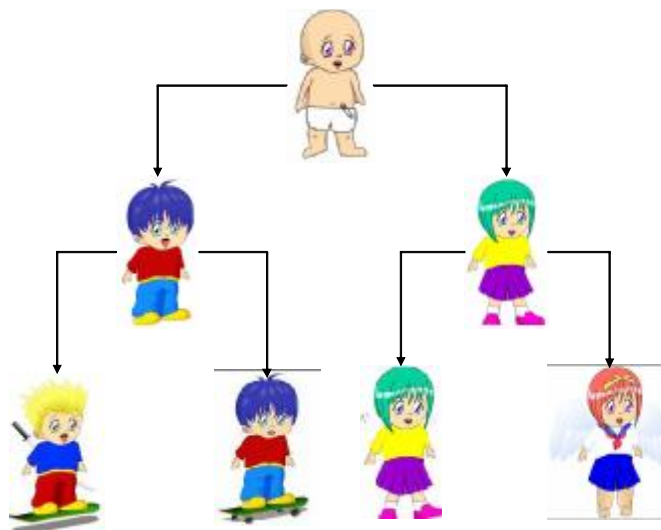


Figure 1: Single Inheritance

If we want to create a new class of skating Sloopies which can also fly, we have to create a sub-class under Kojak **or** two other classes, one for male Sloopies and another for female Sloopies. **Multiple inheritance** provides an alternative approach where the new class can inherit information from the two relevant classes. This is shown in the figure below.

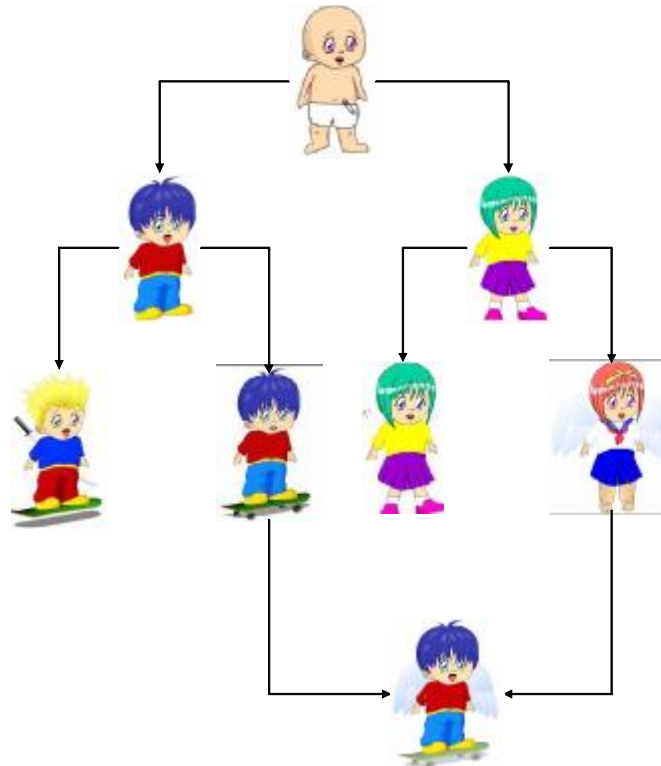


Figure 2: Multiple Inheritance

If you look at this diagram closely you may find some inconsistencies. For example the Sloopie at the bottom has both a skateboard and wings which have been inherited from the *SkatingSloopie* and *FlyingSloopie*. However, the representation is of a boy Sloopie and not a girl Sloopie. In fact the skating flying Sloopie has actually inherited the attributes of both the boy and of the girl. You can see that this causes conflict. See the discussion at the end of the lesson for more on this.

In pseudo code the INHERITS keyword is able to accept multiple classes, which has been demonstrated in the pseudo code snippet below.

```

...

CLASS SkatingSloopy INHERITS SloopyBoy
ATTRIBUTES:
METHODS:
    ride(speed) RETURN position

CLASS FlyingSloopy INHERITS SloopyGirl
ATTRIBUTES:
METHODS:
    fly(height, speed) RETURN none
    land() RETURN position

CLASS SkatingFlyingSloopy INHERITS SkatingSloopy, FlyingSloopy
ATTRIBUTES:
METHODS:
    trick(stunt) RETURN none

```

The class *SkatingFlyingSloopy* inherits all the attributes (which have been left out for simplicity) and behaviour from *SkatingSloopy* (*ride* at a certain *speed* arriving at a given *position*) and *FlyingSloopy* (*fly* at a certain *height* and *speed* and *land* also arriving at a given *position*). In addition instances of the new Sloopy class can perform a *trick*, where the input parameter is a certain *stunt*.

The UML specification supports multiple inheritance in a natural way. Generalisation is used in same way as it is used in single inheritance. Below is the UML class diagram representing the above Sloopy example.

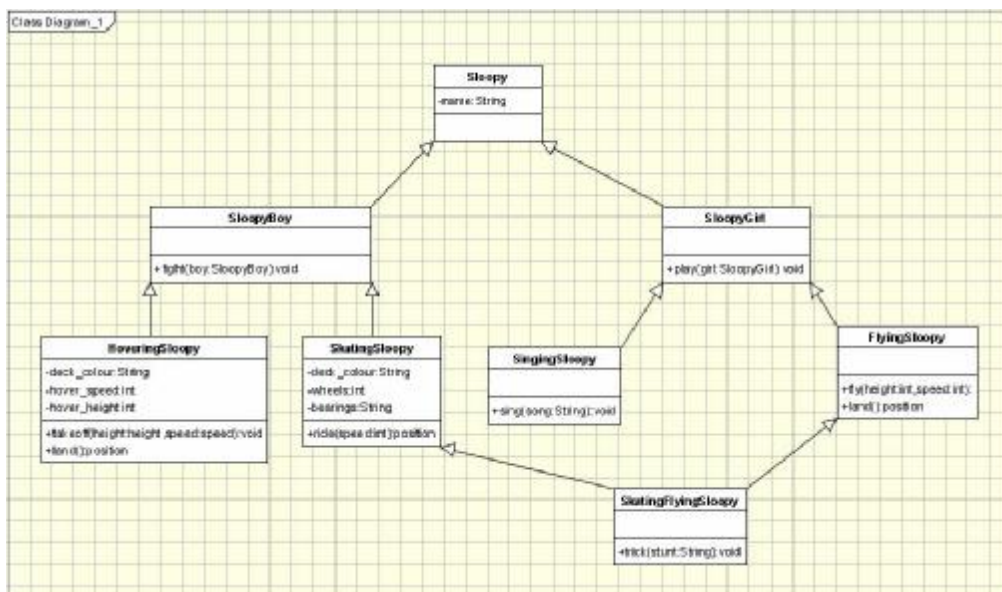


Figure 3: Multiple Inheritance in UML

Discussion

Although multiple inheritance is indeed powerful, it is rather complicated to use and causes many problems both for the modeller and internal mechanisms (things can get messy when two different parents have conflicting attributes with the same name). For example, what happens if both parent classes contain properties with the same name but different data types or methods with the same name but different signatures? For instance, let us assume that the *SloopyBoy* class contains a method called *eat* that takes as a parameter different types of *junk_food*. The *SloopyGirl* class also has an *eat* method, but it only accepts *healthy_food*. Does a class which is derived from both classes – like *SkatingFlyingSloopy* – eat healthy food or junk food? Also, what happens if a property that has been inherited from one class has been overridden by the other class's parent?

Different systems use different conflict resolution mechanisms. Of the two most popular object-oriented programming languages, only C++ provides multiple inheritance. Java provides no direct support, but provides an alternative functionality by using so-called interfaces.

A general rule is to avoid multiple inheritance unless it is absolutely necessary.



In this chapter covered the concepts of **multiple inheritance** which is an advanced form of inheritance of structure and behaviour. We have also discussed the pros and cons of the mechanism.