Inheritance of Instances and Classes

Inheritance is the process of creating a new class with the characteristics of an existing class, along with additional characteristics unique to the new class. In this section we will deal with the **inheritance** of **structure**.

Inheritance

So far, classes and objects seem like little more than just a fancy way to bundle entities together. However, there is one important ingredient to object-orientation that completely sets it apart: **inheritance**. More often than not, objects have to be modelled that are not the same, but are **similar** to each other. For example, take a car, a tractor and a racing car. They all have an engine, 4 wheels, a petrol tank, a steering wheel, and so on. However given the clear differences in the vehicles you would not model them as the same class; instead, they would be modelled as three separate classes. Inheritance overcomes that nuisance and allows the creation of similar but related classes.

Inheritance is the ability to begin with an existing class and use it as a starting point for creating another class. When you inherit from a class, the new class (called the **child class**) maintains a connection with the original class (the **parent class**). This connection assures that any change in the 'parent' is automatically also made in all its 'children' as well.

Let's have a look at our Sloopy family below. So far we have avoided modelling the differences. For example, we have deliberately ignored the fact that Nina wears a skirt and the boys wear trousers. While we can model them in different classes *Boys* and *Girls* it would be more elegant to model the attributes they share in the same class. Inheritance does exactly that.

synergy learning gentleware just model
© Copyright Synergy Learning & Gentleware AG 2006 All Rights Reserved

Figure 1: Sloopies before Inheritance

Let's assume we would have a generic Sloopy – one that has the properties of every Sloopy ever created, no matter whether it is male, female, young, old or even a Sloopy with wings. We call it **Kojak**. Kojak contains the very basic properties which are inherited by every single Sloopy.

Figure 2: Generic Sloopy Kojak

Kojak has 2 arms, 2 legs, a head with 2 eyes, 2 ears, a mouth, a nose and so on. The optional DEFAULT key word in pseudo code expresses this.

```
CLASS Sloopy
ATTRIBUTES:
    name: TEXT
    arms: NUMBER DEFAULT 2
    legs: NUMBER DEFAULT 2
    head: NUMBER DEFAULT 1
    eyes: NUMBER DEFAULT 2
    eye_colour: COLOUR
    ears: NUMBER DEFAULT 2
    mouth: NUMBER DEFAULT 1
    nose: NUMBER DEFAULT 1
```

Every Sloopy will **share** those attributes. When creating a first **sub-class**, that is a class that inherits all properties from its **super-class**, all instances are passed on to

© Copyright Synergy Learning & Gentleware AG 2006 All Rights Reserved

the next generation. Now we can re-define our existing male Sloopies in a new class to represent boys: *SloopyBoy*

```
CLASS SloopyBoy INHERITS Sloopy
ATTRIBUTES:
    jumper: COLOUR
    trousers COLOUR
    shoes: COLOUR
```

Each *SloopyBoy* now has 12 properties, 9 inherited from the parent class *Sloopy* (indicated by the keyword INHERITS) and 3 newly defined ones, specific to boys.

As expected UML fully supports the modelling of inheritance. It connects the parent class with a child class through a link called **generalisation**, indicated by a solid arrow with an unfilled triangular head. For instance, *animal* is a generalization of *bird* because every bird is an animal, and there are animals which are not birds (dogs, for instance).

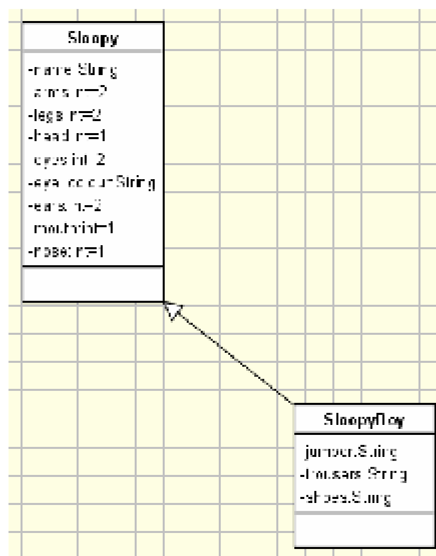

Figure 3: Generic Sloopy Kojak

The critical learner would have already spotted that girls also wear shoes and are not limited to wearing skirts and blouses. There are now two ways of modelling this.

Either the properties are duplicated in a *SloopyGirl* class or a new class is introduced which is called, say, *DressedSloopy*, which inherits properties from Sloopy and passes them on to *BoySloopy* and *GirlSloopy* respectively. That is, inheritance also works for multiple generations as well. You can inherit from a **parent** to create a **child**. Then you can inherit from the child to create a **grandchild**. Any changes made to either the parent or child is reflected in the grandchild.

synergy learning gentleware just model
© Copyright Synergy Learning & Gentleware AG 2006 All Rights Reserved

Abstraction

Inheritance is used to derive specific objects from other more generic or **abstract** objects. In the Sloopy example below we have a generic object called Sloopy. We then have two different types of characters (boys and girls) which all share some generic properties (legs, arms, etc) but also have some unique properties (skirts and trousers). At the next level down we have specific instances of each of the character types which inherit the attributes and behaviours of their parent type. This arrangement is called **abstraction**.

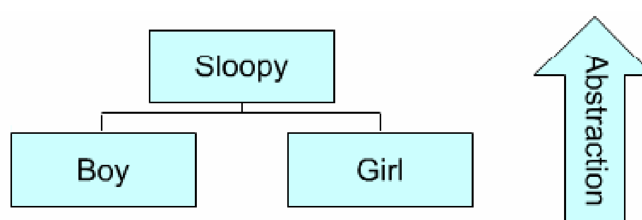

Figure 4: Abstraction

For example, Nina and **Angel** (another female Sloopy) are both girls but Angel has wings whereas Nina does not. The higher the level in the hierarchy, the more abstract the description, while the further down the hierarchy we go, the more specific the description becomes.

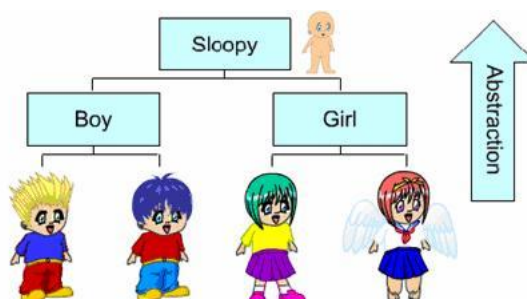

Figure 5: Abstraction: Sloopies after Inheritance

The tree-like arrangement of classes is called a **class hierarchy**.

Inheritance is the process of creating a new class with the characteristics of an existing class, along with additional characteristics unique to the new class. In this section we have dealt with the **inheritance** of **structure**. The next section will deal with the **inheritance** of **behaviour**.

synergy learning
gentleware just model
© Copyright Synergy Learning & Gentleware AG 2006 All Rights Reserved

© Copyright Synergy Learning & Gentleware AG 2006 All Rights Reserved