INHERITANCE

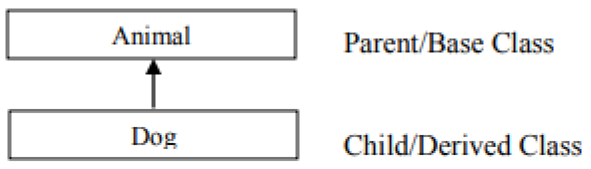
- The mechanism of creating a new class (derived class) from an old one (base class) is called inheritance or derivation.
- The derived class inherits some or all the capabilities of the base class and can also add new features and refinements of its own.
- A child class is a **specialized (restricted)** form of the parent class so it is a **contraction** of the parent class. In the meantime child class are also an **extension** of the parent as they extend the properties associated with parent class.

>>Inheritance is always **transitive** so that a class can inherit features from super classes many levels away.

e.g: Dog is a subclass of Mammal and Mammal is a subclass of Animal so a dog will inherit attributes both from Mammal and Animal.

- The symbolic representation of inheritance is ↑

The direction of the arrow (arrow-head) points towards the parent class meaning that the **derived class refers to the functions and data in the base class** while the base class has no access to the derived class data or functions.
It establishes **<u>"is a kind of"</u>** relationship.



e.g. a dog is a kind of animal

Forms of Inheritance

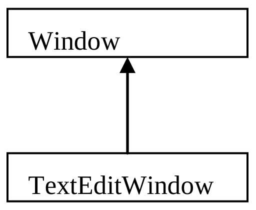
>>Description of categorization of the uses of inheritance.

>>Two or more descriptions may be applicable to a single (same) situation

a. Sub classing for Specialization (Subtyping)

- The most common use and ideal form of inheritance.
- The new class is a specialized form of the parent class but satisfies the specifications of the parent in all relevant respects.
- The principle of substitutability is explicitly upheld.

A *Window* class provides general windowing operations like moving, resizing.
A specialized subclass *TextEditWindow* inherits the window operations.



In addition it provides facilities that allow the window to display and edit textual material.

TextEditWindow satisfies all the properties of a window thus is a **subtype of window** in addition of being a subclass

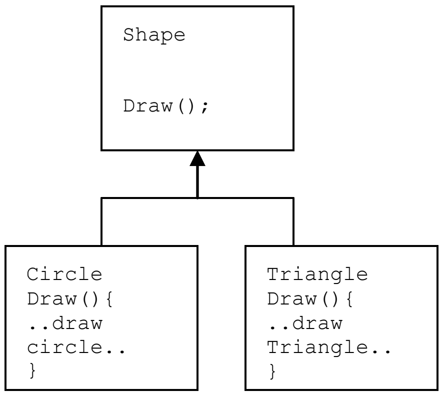
b. Sub classing for Specification

Here the classes maintain a certain *common interface* (*they implement the same methods).*
The parent class contains combination of *implemented operations* and *operations that are deferred to the child class*

There is *no interface change* as the child implements the behavior described but not implemented in the parent

This is a special case of *subclassing for specialization* where subclasses are realization of an *incomplete abstract specification* and they are not refinements of an existing type.

The parent class is called *abstract specification* class as it does not implement actual behavior but only defines them



c. Subclassing for Construction

Here , a class often inherit almost all of its desired functionality from a parent class changing only the names of the methods used to interface to the class or modifying the arguments in certain fashion .

e.g. classes are created to write values to a binary file .
Parent class implements only the ability to write raw binary data.
A subclass is constructed for every structure that is saved using the behavior of the parent class

```
 class storable{  void
writeByte(unsigned char);
};
class storeStruct: public storable{  void
writeByte(myStruct &aStruct);
};
```

d. Subclassing for Generalization

Opposite of *Subclassing for specialization.*
Subclass extends the behavior of parent class to create more general kind of object.
Often applicable when we build on a base of existing classes that we do not wish to or can not modify.
e.g. a class Window is defined for displaying on a simple back and white background

now we can create a subtype *ColoredWindow* that lets the background color to be something else by adding additional field to store the color and overriding the inherited behavior of parent window.

Occurs when the overall design is based *primarily on data values* and only *secondarily on behavior*
e.g. in previous case colored window contains data fields that are not necessary in the simple window case

e. Subclassing for Extension

Subclassing for generalization modifies or expands on the existing functionality of an object whereas *subclassing for extension* adds totally new abilities .

In *subclassing for generalization*, at least one method from the parent must be *overridden* and the functionality is tied to that of the parent .

Extension simply adds new methods to those of the parent and the functionality is less strongly tied to the existing methods of the parent

e.g. a *StringSet* that is specialized for holding string values whose parent class is *Set* class .
StringSet class may provide additional string related methods that are not relevant to the parent class

e.g. "*search_ by_ prefix()*" method that returns a subset of all the elements of the set that begin with certain value

f. Subclassing for Limitation

Occurs when the behavior of the subclass is smaller or more *restrictive* than the behavior of the parent class

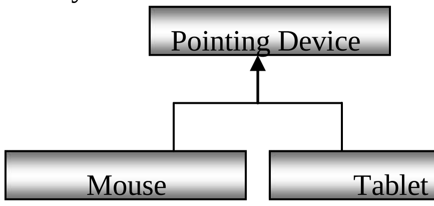
Used when a programmer cannot or should not modify the parent class
e.g. in case of a *deque* double ended queue, elements can be added or removed form either end of it .
The *deque* can be converted into a *stack* (in which elements can be added or removed from only one end) by eliminating the functionality of *deque* .
It doesn‟t follow the rule of substitutability as it builds subclasses that are not subtypes so it should be avoided

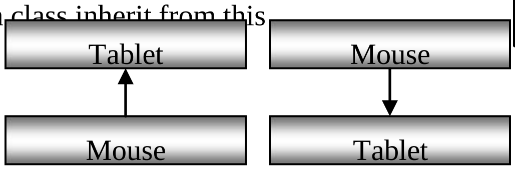
g. Subclassing for Variance

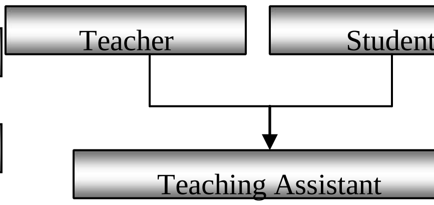
Used when two or more classes have similar implementations but do not seem to possess any hierarchical relationships between concepts represented by the classes

e.g. code required to control a *mouse* and a *tablet* are identical

In this case one of the two classes can be chosen as the parent by inheriting the common code and overriding the de vice specific code A better alternative, factor out the common code into a abstract class, *PointDevice* and have both class inherit from this common parent class







h. Subclassing for Combination

Here, the subclass represents a combination of features from two or more parent classes e.g. a teaching assistant have characteristics of both a teacher and a student
The ability of a class to inherit from two or more parent classes is known as *multiple inheritance*

Differences between subtypes and subclasses

There are important differences between subtypes and subclasses in **supporting reuse**.
>>**Subclasses** allow one to reuse the code inside classes - both instance variable declarations and method definitions. Thus they are useful in supporting code reuse *inside* a class.
>>**Subtyping** on the other hand is useful in supporting reuse externally, giving rise to a form of polymorphism. That is, once a data type is determined to be a subtype of

another, any function or procedure that could be applied to elements of the supertype can also be applied to elements of the subtype.

Notice that the subtype relation depends only on the public interfaces of objects, not their implementations.

>>Property to satisfy for subtype: For each method in the supertype, the subtype must have a corresponding method. (The subtype is allowed to introduce additional, new methods that do not appear in the supertype.)