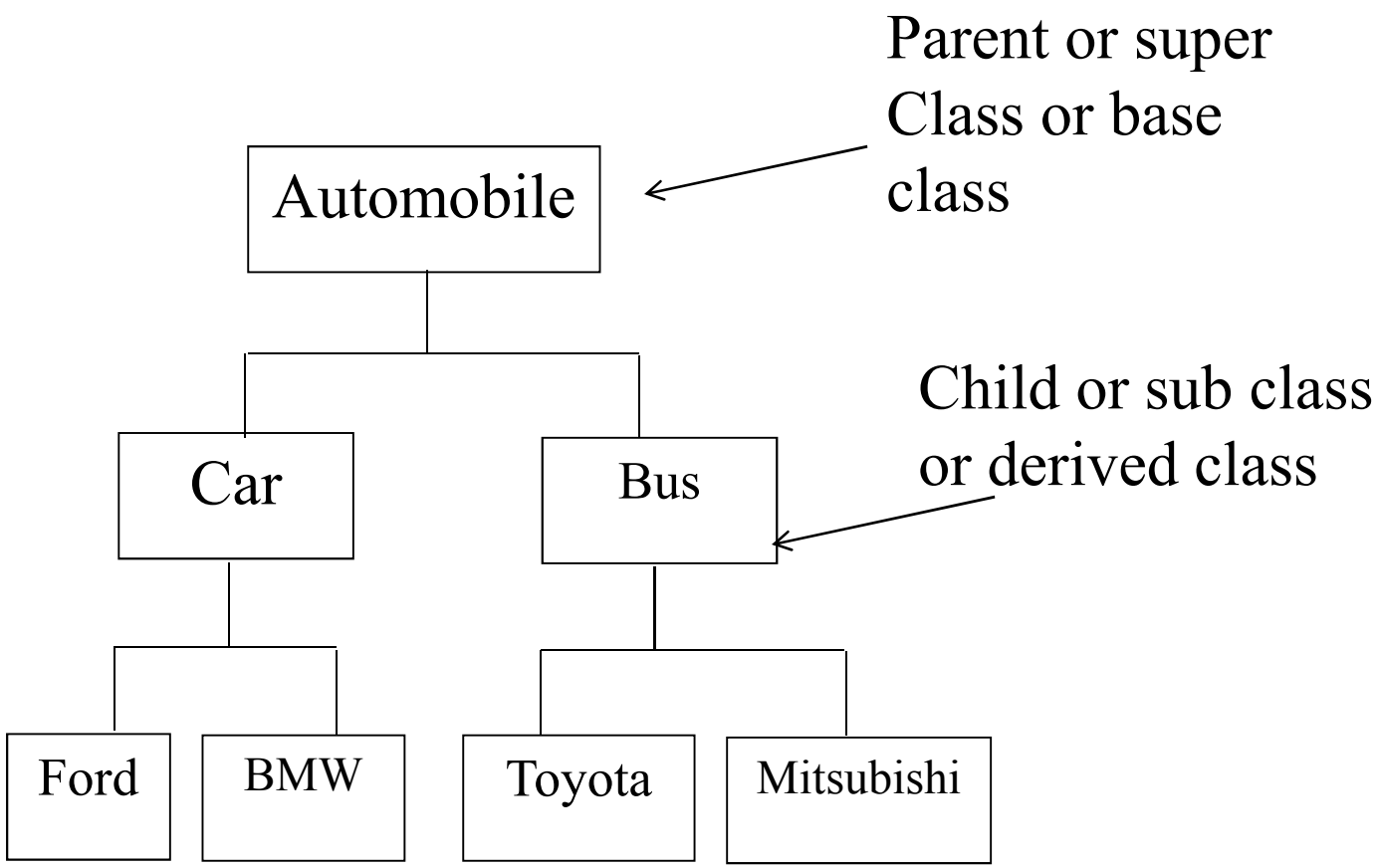
Inheritance

- In C#, inheritance is the property by which the objects of a derived class possess copies of the data members and the member functions of the base class.
- A class that inherits or derives attributes from another class is called the derived class.
- The class from which attributes are derived is called the base class.
- In object-oriented programming, the base class is actually a superclass and the derived class is a subclass.


Parent or super
Class or base
class
Automobile
Child or sub class
or derived class
Car
Bus
Ford
BMW
Toyota
Mitsubishi

Inheritance

- Each instance of the derived class includes the attributes of the base class.

- Any change made to the base class automatically changes the behavior of its derived classes.

- The syntax used in C# for creating derived classes is as follows:

```
<access-specifier> class <base_class>
{
   ...
}
class <derived_class> : <base_class>
{
   ...
}
```

Using Virtual Functions

- When you have a function defined in a class which you want to allow to be implemented by the inherited classes, you can use virtual function.

- The virtual function could be implemented by the inherited classes in their own way and the call to the method is decided at the run time.

Example

```
public class Book
{
        public string title;
        public virtual void ShowDetails()
         {
            Console.WriteLine("Title : " + title);
         }
}
```

Using Abstract Classes

- C# enables you to create abstract classes that are used to provide partial class implementation of an interface.
- Abstract classes contain abstract methods, which can be implemented by the derived class.
- Polymorphism can be implemented by using abstract classes and virtual functions.
- There are certain rules governing the use of an abstraction class:
  - Cannot create an instance of an abstract class.
  - Cannot declare an abstract method outside an abstract class.
  - Cannot be declared sealed.

Using Abstract Methods

- Abstract methods are methods without any body.
- The implementation of an abstract method is done by the derived class.
- When a derived class inherits the abstract method form the abstract class, it must override the abstract methods. This requirement is enforced at compile time, and is also called dynamic polymorphism.
- The syntax for using the abstract method is as follows:

```
[access-modifiers] abstract return-type
method name (parameters]);
```

- The abstract method is declared by adding the abstract modifier to the method.

Example

```
public abstract class Area
{
     public abstract void findArea();
};
```

Using Sealed Classes

- You could restrict users form inheriting the class by sealing the class using the `sealed` keyword.
- The `sealed` keyword tells the compiler that the class is sealed, and therefore, cannot be extended.
- The following is an example of a `sealed` class:

```
sealed class FinalClass
{
   private int x;
   public void Method1()
   {
   }
}
```

- A method can also be sealed and in that case the method cannot be overridden.

Interfaces

- Interfaces define properties, methods, and events, which are known as the members of the interface.
- Interfaces are used when a standard structure of methods is to be followed by the classes, and where classes will implement the functionality.
- Interfaces separate the definition of objects from their implementation so that the objects can evolve without the risk of introducing incompatibility in existing applications.

Interfaces

- Working with interfaces includes interface declaration and implementation of interface by the classes.
- You can declare interfaces using the `interface` keyword.
- Interface statements are public, by default.
- You can declare only methods, functions, and properties in interfaces. You cannot declare a variable in interfaces.
- Interfaces declare methods, which are implemented by classes. A class can inherit from single class but can implement form multiple interfaces.
- A class or a structure that implements interfaces also implements the base interfaces of the inherited interface.