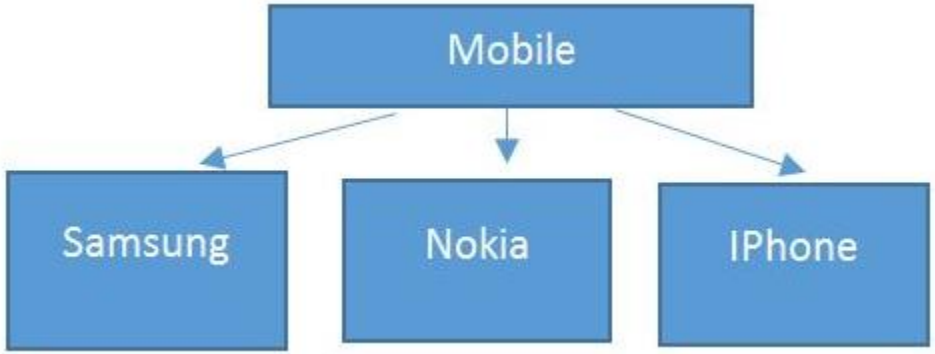
Object-Oriented Programming

Object-oriented programming (OOP) : based on the concept of "objects", which may contain data, in the form of fields, often known as *attributes;* and code, in the form of procedures, often known as *methods.*

Ex:



Super class: Mobile: attribute: functionality of dialing, receiving a call & messaging
Super class can share with all subclass methods (functionality)

ach brand (Samsung, Nokia, IPhone) is sub class can inherent the attributes from the super class as well as they can have their own list of features () attribute and methods.

Object-Oriented Programming

Objects

Any real world entity which can have some characteristics or which can perform some work is called as Object. Mobile is an object

This object is called as an instance.
This objects are differentiated from each other via some identity or its characteristics. This characteristics is given some unique name.

Samsung, iPhone are instances of object mobile

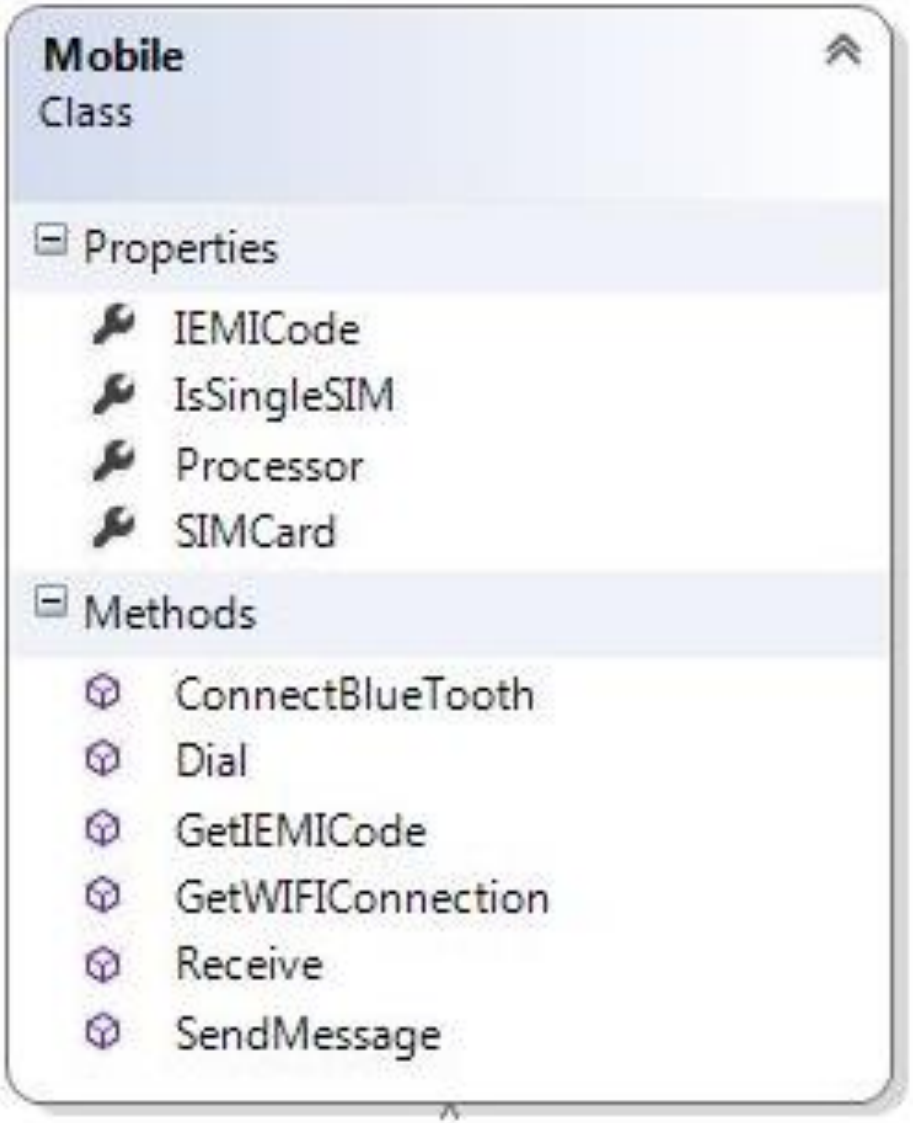
Class

A Class is a plan which describes the object. We call it as a blue print of how the object should be represented.

Mainly a class would consist of a name, attributes & operations.

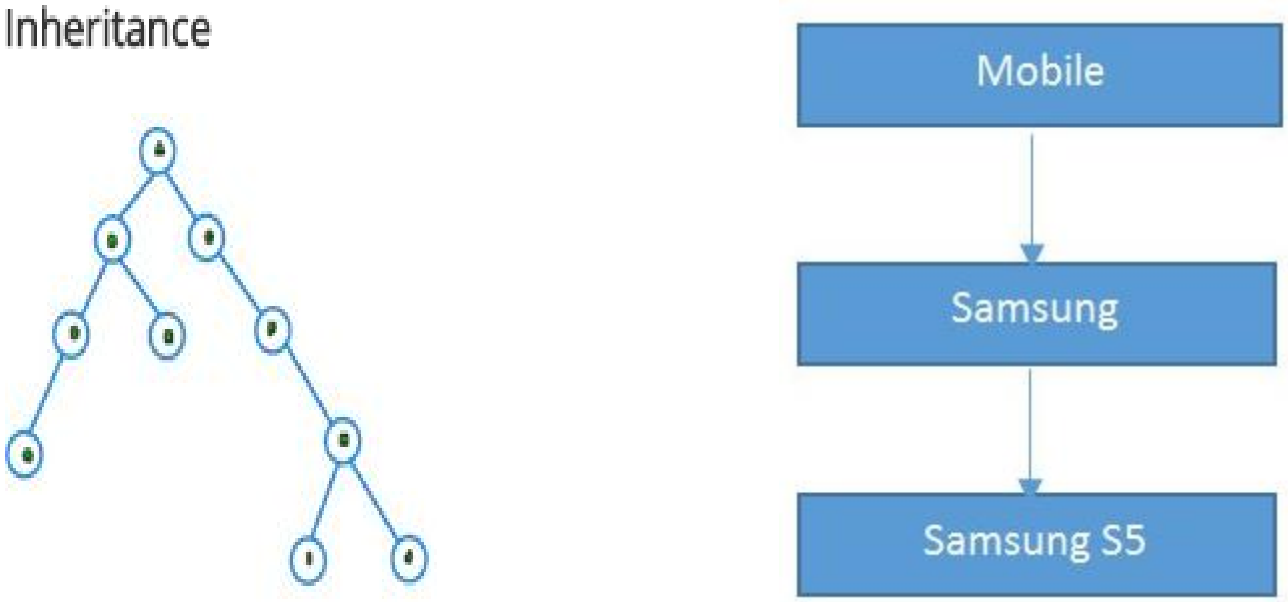
Object-Oriented Programming

EX: class mobile:



Object-Oriented Programming

Inheritance: The transfer of the characteristics of a class to other classes that are derived from it.



Ability to extend the functionality from base entity in new entity belonging to same group. This will help us to reuse the functionality which is defined before.

Object-Oriented Programming

Instance
An instance is a unique copy of a Class that representing an Object. Instance is an individual object of a certain class. An object that belongs to a class, Samsung is an instance of the class mobile.

Instantiation: The creation of an instance of a class.

Method : A special kind of function that is defined in a class definition.

What is the difference between an Instance and an Object?
An instance is an object in memory. Basically you create object and instantiate them when you are using them.

Creating Classes

The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows

```
class ClassName:
        'Optional class documentation string'

>>>type(ClassName)

>>> ClassName.__doc__
```

Example

Following is the example of a simple Python class :

Creating Classes

You can add, remove, or modify attributes of classes and objects at any time :

```
emp1.age = 7 # Add an 'age' attribute.
emp1.age = 8 # Modify 'age' attribute.
del emp1.age # Delete 'age' attribute.
```

Instead of using the normal statements to access attributes, you can use the following functions :

The **getattr(obj, name[, default])** : to access the attribute of object.
The **hasattr(obj,name)** : to check if an attribute exists or not.
The **setattr(obj,name,value)** : to set an attribute. If attribute does not exist, then it would be created.
The **delattr(obj, name)** : to delete an attribute.

Built-In Class Attributes

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute:

__dict__: Dictionary containing the class's namespace.

__doc__: Class documentation string or none, if undefined.

__name__: Class name.

__module__: Module name in which the class is defined. This attribute is "__main__" in interactive mode.

__bases__: A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Destroying Objects (Garbage Collection)

- Python deletes unneeded objects automatically to free the memory space.
- Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero.
- An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary).
- The object's reference count decreases when it's deleted with *del*, its reference is reassigned, or its reference goes out of scope.
- When an object's reference count reaches zero, Python collects it automatically.

Destroying Objects (Garbage Collection)

EX:

```
class Point:
   def __init( self, x=0, y=0):
      self.x = x
      self.y = y
   def __del__(self):    #   Special method called a destructor
      class_name = self.__class__.__name__
      print (class_name, "destroyed")

pt1 = Point()
pt2 = pt1
pt3 = pt1
print (id(pt1), id(pt2), id(pt3),'\n') # prints the ids of the obejcts
del pt1
del pt2
del pt3
```

Class Inheritance

Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.

The child class inherits the attributes of its parent class,
you can use those attributes as if they were defined in the child class.

A child class can also override data members and methods from the parent.

Override: avoiding duplicated code, and at the same time enhance or customize part of it.

Method overriding is thus a strict part of the inheritance mechanism.
EX:

```
class childStr(str):
        pass  # donothing
>>> b = childStr('Hello World')
>>> b.upper()
```

It behaves like a string class

Class Inheritance

Add customized method to inherited class

Let us add customized method to the child class childStr

```
class childStr(str):
  def check_first_last_same(self):
        # precondition str is not null  len(str) > 0
        return  self[0]  ==  self[-1]
```

Class Inheritance

Overriding Methods

You can always override your parent class methods. One reason for overriding parent's methods is because you may want special or different functionality in your subclass

Ex:

Override method

Override: avoiding duplicated code, and at the same time enhance or customize part of it.

Let us have the following class:

```
class point:
    def __init__(self, xcoord=0, ycoord=0):
        self.x  = xcoord
        self.y = ycoord
>>> p= point(1,2)
>>>p
```

Question1: what is the output of the following code:

```
class Point:
    def __init__(self, xcoord=0, ycoord=0):
        self.x = xcoord
        self.y = ycoord


a = Point(-1, 1)
b = Point(3, 3)
tmp=a
a=b
a.x = 1

print(a.x, a.y, b.x, b.y)
```

Question2: what is the output of the following code:

```
class point:
    def __init__(self, xcoord=0, ycoord=0):
        self.x = xcoord
        self.y = ycoord
    def __repr__(self):
        return 'Point(' +str(self.x)+','+str(self.y)+')'

def increment(x,p):
    x = x + 2
    p.x = p.x + 1
    p.y = p.y + 1

p1=point(1,10)
y=1
print(y,p1)
increment(y,p1)
print(y,p1)
```

Question3: what is the output of the following code:

```
class point:
    def __init__(self, xcoord=0, ycoord=0):
        self.x = xcoord
        self.y = ycoord
    def __repr__(self):
        return 'Point(' +str(self.x)+','+str(self.y)+')'

def riddle(a,b):
    a = b
    a.x = 500
    a.y = 800

p1 = point(1,2)
p2 = point(10,20)
print(p1,p2)
riddle(p1,p2)
print(p1,p2)
```