The content of class can be,

1. Private
2. Protected
3. Public

private, protected or public are called access specifiers or modifiers
access specifiers defines the accessibility of members of class.

1. private members (variables and methods) are accessible within class but cannot accessible outside the class or non members.
   In object oriented application development data hiding is achieved by declaring variables inside class as private.
   Private members are prefix with __
2. default members of class are public, public members are accessible within class and outside the class.
   Public members are not prefix with any underscore

Example:

```
class A:
    def m1(self):
        print("public method")
    def __m2(self):
        print("private method")

def main():
    obj1=A()
    obj1.m1()
    obj1.__m2()

main()
```

Output:

```
public method
Traceback (most recent call last):
  File "C:/Users/user/Desktop/python6pm/py244.py", line 13, in <module>
    main()
  File "C:/Users/user/Desktop/python6pm/py244.py", line 11, in main
    obj1.__m2()
AttributeError: 'A' object has no attribute '__m2'
```

Example:

```
class A:
```

```
    def __init__(self):
        self.x=100 # public
        self.__y=200 # private

def main():
    obj1=A()
    print(obj1.x)
    print(obj1.__y)
main()
```

Output:

```
100
Traceback (most recent call last):
  File "C:/Users/user/Desktop/python6pm/py245.py", line 10, in <module>
    main()
  File "C:/Users/user/Desktop/python6pm/py245.py", line 9, in main
    print(obj1.__y)
AttributeError: 'A' object has no attribute '__y'
>>>
```

Methods written inside class perform 2 operations.

1. **Setter operation**
2. **Getter operation**

An operation which modify the state of object is called setter operation (OR) a method which modify values of object is called setter method

An operation which does not modify the state of object is called getter operation (OR) a method which does not modify values of object is getter method

Example:

```
class Employee:
    def __init__(self):
        self.__empno=None # private
        self.__ename=None
        self.__salary=None
    def set_empno(self,e):
        self.__empno=e
    def set_ename(self,en):
```

```
        self.__ename=en
    def set_salary(self,s):
        self.__salary=s
    def get_empno(self):
        return self.__empno
    def get_ename(self):
        return self.__ename
    def get_salary(self):
        return self.__salary
def main():
    emp1=Employee()
    emp1.set_empno(101)
    emp1.set_ename("ramesh")
    emp1.set_salary(5000)
    print(f'{emp1.get_empno()}\t{emp1.get_ename()}\t{emp1.get_salary()}')
    emp1.set_salary(6000)
    print(f'{emp1.get_empno()}\t{emp1.get_ename()}\t{emp1.get_salary()}')
main()
```

Output:

```
101   ramesh      5000
101   ramesh      6000
>>>
```

Example:

```
class Stack:
    def __init__(self):
        self.__s=[]
    def push(self,value):
        self.__s.append(value)
    def pop(self):
        if len(self.__s)==0:
            return None
        else:
            return self.__s.pop()
def main():
    stack1=Stack()
    stack1.push(10)
    stack1.push(20)
```

```
    stack1.push(30)
    value1=stack1.pop()
    value2=stack1.pop()
    print(value1,value2)
main()
```

Output:

```
30 20
>>>
```

Example:

```
class Student:
    def __init__(self,r,n,c):
        self.__rno=r
        self.__name=n
        self.__course=c
    def set_course(self,c):
        self.__course=c
    def get_rno(self):
        return self.__rno
    def get_name(self):
        return self.__name
    def get_course(self):
        return self.__course

def main():
    stud1=Student(1,"ramesh","java")
    print(f'{stud1.get_rno()}\t{stud1.get_name()}\t{stud1.get_course()}')
    stud1.set_course("python")
    print(f'{stud1.get_rno()}\t{stud1.get_name()}\t{stud1.get_course()}')

main()
```

Output:

```
1       ramesh      java
1       ramesh      python
>>>
```

Class level variables