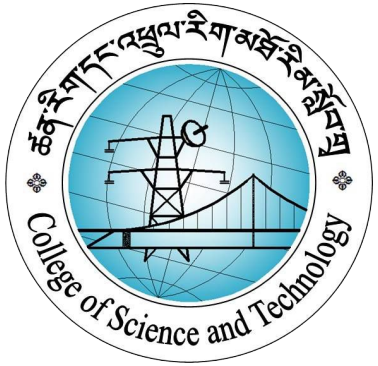


Royal University of Bhutan

Unit V:
Principles of Object Oriented Programming

Programming Methodology (CSF101)

Outline

- SOLID Principle
- Types of Inheritance

SOLID Principles

- SOLID is a set of five object-oriented design principles that can help you write more **maintainable, flexible,** and **scalable** code based on **well-designed, cleanly structured classes**.

SOLID Principle

S

Single Responsibility Principle (SRP)

A class should have only one reason to change, meaning it should have a single, well-defined responsibility.

O

Open/Closed Principle (OCP)

Software entities (e.g., classes, modules) should be open for extension but closed for modification.This promotes the idea of extending functionality without altering existing code.

L

Liskov Substitution Principle (LSP)

Subtypes (derived classes) must be substitutable for their base types (parent classes) without altering the correctness of the program.

I

Interface Segregation Principle (ISP)

Clients should not be forced to depend on interfaces they don't use.This principle encourages the creation of smaller, focused interfaces.

D

Dependency Inversion Principle (DIP)

High-level modules should not depend on low-level modules; both should depend on abstractions. This promotes the decoupling of components through abstractions and interfaces.

1. Single Responsibility Principle (SRP)

- **A class should have only one reason to change.**
- This means that a class should have only one responsibility, as expressed through its methods.
- If a class takes care of more than one task, then you should separate those tasks into separate classes.

2. Open/Closed Principle (OCP)

- **Classes should be open for extension and closed to modification.**
- Means you should be able to extend a class behavior, without modifying it.

3. Liskov Substitution Principle (LSP)

- **Subclasses should be substitutable for their base classes.**
- This means that if you have a base class and a derived class, you should be able to use objects of the derived class wherever objects of the base class are expected, without breaking the program.

4. Interface Segregation Principle (ISP)

- **Many client-specific interfaces are better than one general-purpose interface.**
- Means that it's better to have multiple specific interfaces than a single general-purpose interface.

5. Dependency Inversion Principle (DIP)

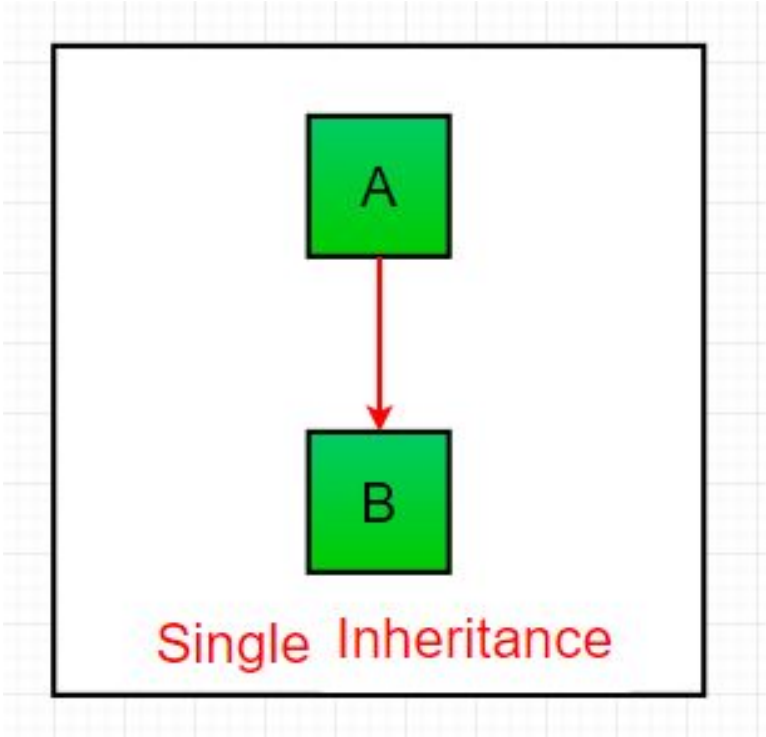
- Classes should depend upon interfaces or abstract classes instead of concrete classes and functions.

Types of Inheritance

- Types of Inheritance depend upon the **number of child and parent classes involved.**
    1. Single Inheritance
    2. Multiple Inheritance
    3. Multilevel Inheritance
    4. Hierarchical Inheritance
    5. Hybrid Inheritance

Single Inheritance

- Enables a derived class to inherit properties from a single parent class

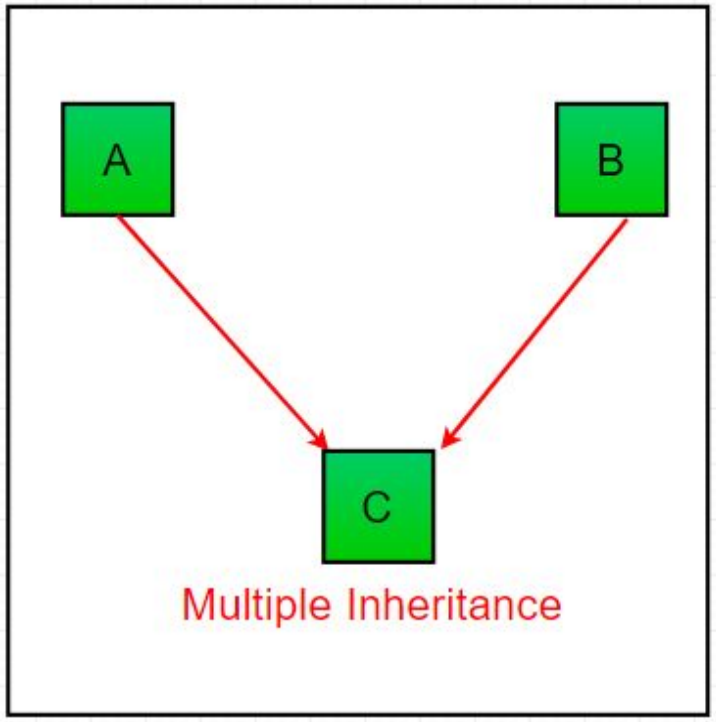



```
class ParentClass:
    # Parent class definition

class ChildClass(ParentClass):
    # Child class definition
```

```python
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class
class Child(Parent):
    def func2(self):
        print("This function is in child class.")

# Driver's code
object = Child()
object.func1()  #This function is in parent class.
object.func2()  #This function is in child class.
```

Multiple Inheritance

- When a class can be derived from more than one base class.

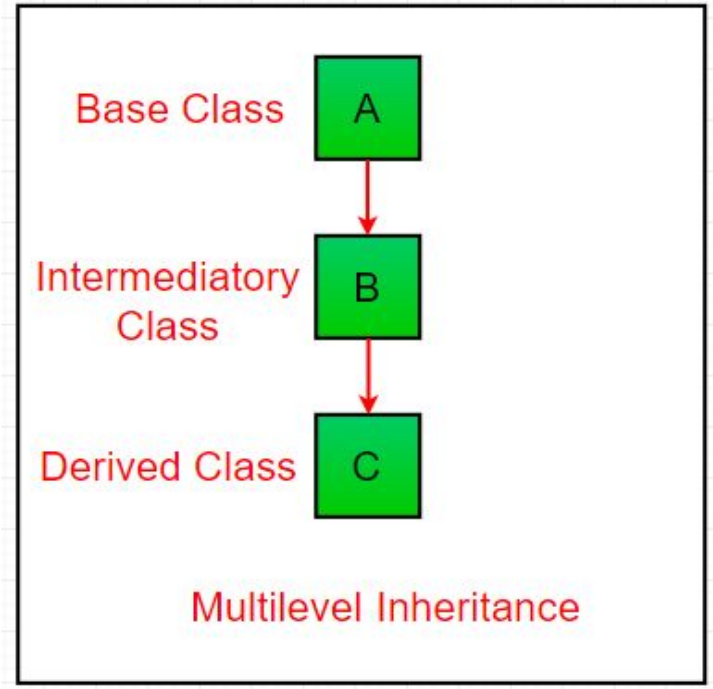



```
class ParentClass1:
    # Parent class 1 definition

class ParentClass2:
    # Parent class 2 definition

class ChildClass(ParentClass1, ParentClass2):
    # Child class definition
```

```python
# Base class1
class Mother:
    mothername = "Dechen"

    def mother(self):
        print(self.mothername)

# Base class2
class Father:
    fathername = "Dorji"

    def father(self):
        print(self.fathername)

# Derived class
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

obj = Son()      #Father : Dorji
obj.parents()    #Mother : Dechen
```

Multilevel Inheritance

- When features of the base class and the derived class are further inherited into the new derived class.

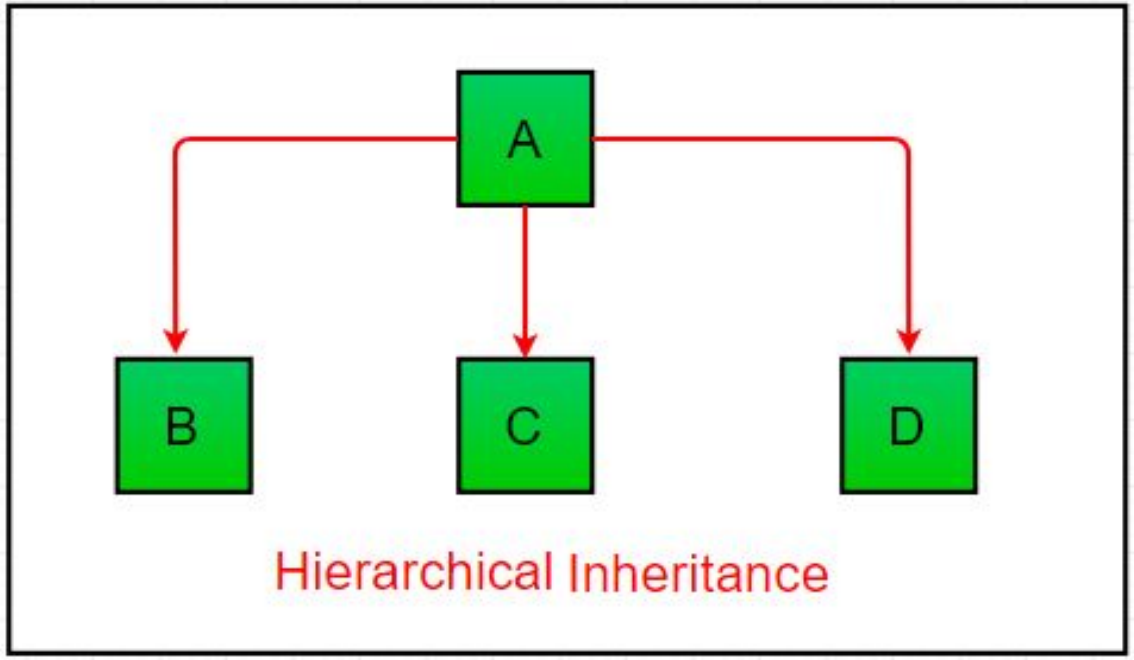



```
class GrandParentClass:
    # Grandparent class definition

class ParentClass(GrandParentClass)
    # Parent class definition

class ChildClass(ParentClass):
    # Child class definition
```

```python
# Base class
class Grandfather:

    def __init__(self, grandfathername):
        self.grandfathername = grandfathername

# Intermediate class
class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername

        # invoking constructor of Grandfather class
        Grandfather.__init__(self, grandfathername)

# Derived class
class Son(Father):
    def __init__(self, sonname, fathername, grandfathername):
        self.sonname = sonname

        # invoking constructor of Father class
        Father.__init__(self, fathername, grandfathername)

    def print_name(self):
        print('Grandfather:', self.grandfathername)
        print("Father:", self.fathername)
        print("Son:", self.sonname)

obj = Son('Karma', 'Dorji', 'Tashi')
obj.print_name()
```

Hierarchical Inheritance

- When more than one derived class is created from a single base class.

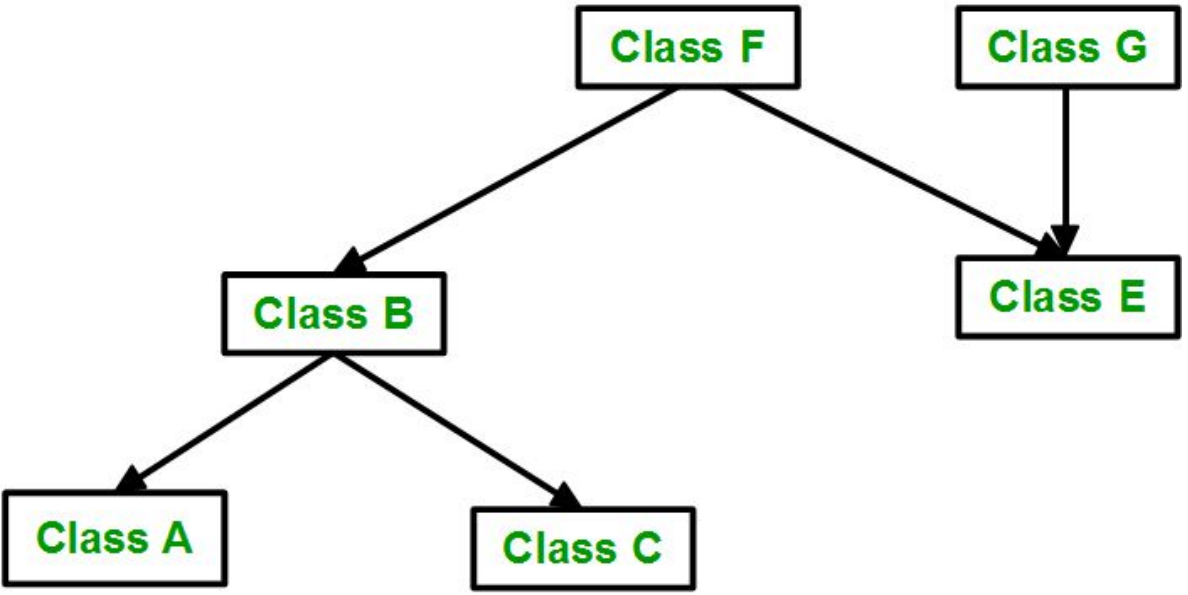



```
class ParentClass:
# Parent class definition


class ChildClass1(ParentClass):
# Child class 1 definition


class ChildClass2(ParentClass):
# Child class 2 definition
```

```python
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class1
class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")

object1 = Child1()
object2 = Child2()
object1.func1() #This function is in parent class
object1.func2() #This function is in child 1.
object2.func1() #This function is in parent class.
object2.func3() #This function is in child 2.
```

Hybrid Inheritance

- Inheritance consisting of multiple types of inheritance.




```
class ParentClass:
# Parent class definition

class ChildClass1(ParentClass):
# Child class 1 definition

class ChildClass2(ParentClass, ChildClass1):
# Child class 2 definition
```

```
class School:
    def func1(self):
        print("This function is in school.")

class Student1(School):
    def func2(self):
        print("This function is in student 1. ")

class Student2(School):
    def func3(self):
        print("This function is in student 2.")

class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")

object = Student3()
object.func1()
object.func2()
object.func4()
```


Thanks!

Reference

GeeksforGeeks. (2024a, March 15). SOLID principles in programming understand with real life examples. GeeksforGeeks.

Python, R. (2023, June 16). SOLID Principles: Improve Object-Oriented Design in Python. https://realpython.com/solid-principles-python/

GeeksforGeeks. (2022, July 7). Types of inheritance Python. GeeksforGeeks. https://www.geeksforgeeks.org/types-of-inheritance-python/