#1.refactored code from asmt\_1 based on feedback

class Permissions:

def \_\_init\_\_(self, read=False, write=False, execute=False):

self.read = read

self.write = write

self.execute = execute

class User:

def \_\_init\_\_(self, name, permissions):

self.name = name

self.permissions = permissions

self.groups = set()

def add\_to\_group(self, group):

self.groups.add(group)

class Group:

def \_\_init\_\_(self, name, permissions):

self.name = name

self.permissions = permissions

class File:

def \_\_init\_\_(self, name, content, owner, group, permissions):

self.name = name

self.content = content

self.owner = owner

self.group = group

self.permissions = permissions

class Directory:

def \_\_init\_\_(self, name, parent=None):

self.name = name

self.parent = parent

self.sub\_directories = {}

self.files = {}

def add\_directory(self, directory\_name):

new\_directory = Directory(directory\_name, self)

self.sub\_directories[directory\_name] = new\_directory

return new\_directory

def add\_file(self, file\_name, content, owner, group, permissions):

new\_file = File(file\_name, content, owner, group, permissions)

self.files[file\_name] = new\_file

return new\_file

def create\_file\_system():

root\_directory = Directory("/")

user1 = User("user1", Permissions(read=True, write=True, execute=True))

user2 = User("user2", Permissions(read=True, write=False, execute=False))

# Creating groups and adding users to them

group1 = Group("group1", Permissions(read=True, write=True))

group2 = Group("group2", Permissions(read=True, write=False))

user1.add\_to\_group(group1)

user2.add\_to\_group(group2)

# Creating a file and setting permissions

root\_directory.add\_file("file1.txt", "This is the content of file 1", user1, group1, Permissions(read=True, write=True))

root\_directory.add\_file("file2.txt", "This is the content of file 2", user2, group2, Permissions(read=True, write=False))

# Creating a sub-directory

sub\_directory = root\_directory.add\_directory("subdir1")

sub\_directory.add\_file("file3.txt", "This is the content of file 3", user1, group1, Permissions(read=True, write=False))

return root\_directory

# Accessing files and directories via an access path

def access\_path(root\_directory, path, user):

current\_directory = root\_directory

elements = path.strip("/").split("/")

for element in elements:

if element in current\_directory.sub\_directories:

current\_directory = current\_directory.sub\_directories[element]

elif element in current\_directory.files:

file = current\_directory.files[element]

if file.owner.name == user.name or file.group in user.groups or file.permissions.read:

print(f"Content of {element}: {file.content}")

else:

print("You don't have permission to read this file.")

else:

print(f"{element} not found.")

if \_\_name\_\_ == "\_\_main\_\_":

file\_system = create\_file\_system()

# Accessing files and directories using access path for user1

user1 = User("user1", Permissions(read=True, write=True, execute=True))

access\_path(file\_system, "/file1.txt", user1) # Output: Content of file1.txt: This is the content of file 1

access\_path(file\_system, "/subdir1/file3.txt", user1) # Output: Content of file3.txt: This is the content of file 3

access\_path(file\_system, "/file2.txt", user1) # Output: You don't have permission to read this file.

access\_path(file\_system, "/non\_existent\_file.txt", user1) # Output: non\_existent\_file.txt not found.

# Accessing files and directories using access path for user2

user2 = User("user2", Permissions(read=True, write=False, execute=False))

access\_path(file\_system, "/file1.txt", user2) # Output: Content of file1.txt: This is the content of file 1

access\_path(file\_system, "/subdir1/file3.txt", user2) # Output: You don't have permission to read this file.

access\_path(file\_system, "/file2.txt", user2) # Output: Content of file2.txt: This is the content of file 2

access\_path(file\_system, "/non\_existent\_file.txt", user2) # Output: non\_existent\_file.txt not found.

'''

-Introduced Group class, and users can now be added to specific groups.

-The access\_path function now takes a user argument, allowing to check the user's permissions and group membership when accessing files.

'''

#2.Use of Encapsulation, Inheritance and Polymorphism

class Permissions:

def \_\_init\_\_(self, read=False, write=False, execute=False):

self.read = read

self.write = write

self.execute = execute

class User:

def \_\_init\_\_(self, name, permissions):

self.name = name

self.permissions = permissions

self.groups = set()

def add\_to\_group(self, group):

self.groups.add(group)

class Group:

def \_\_init\_\_(self, name, permissions):

self.name = name

self.permissions = permissions

class FileSystemElement:

def \_\_init\_\_(self, name, owner, group, permissions):

self.name = name

self.owner = owner

self.group = group

self.permissions = permissions

def access(self, user):

pass

class File(FileSystemElement):

def \_\_init\_\_(self, name, content, owner, group, permissions):

super().\_\_init\_\_(name, owner, group, permissions)

self.content = content

def access(self, user):

if self.owner.name == user.name or self.group in user.groups or self.permissions.read:

print(f"Content of {self.name}: {self.content}")

else:

print("You don't have permission to read this file.")

class Directory(FileSystemElement):

def \_\_init\_\_(self, name, parent=None):

super().\_\_init\_\_(name, None, None, None) # Directories have no owner, group, or permissions

self.parent = parent

self.sub\_directories = {}

self.files = {}

def add\_directory(self, directory\_name):

new\_directory = Directory(directory\_name, self)

self.sub\_directories[directory\_name] = new\_directory

return new\_directory

def add\_file(self, file\_name, content, owner, group, permissions):

new\_file = File(file\_name, content, owner, group, permissions)

self.files[file\_name] = new\_file

return new\_file

def access(self, user):

print(f"Accessing directory: {self.name}")

def create\_file\_system():

root\_directory = Directory("/")

user1 = User("user1", Permissions(read=True, write=True, execute=True))

user2 = User("user2", Permissions(read=True, write=False, execute=False))

# Creating groups and adding users to them

group1 = Group("group1", Permissions(read=True, write=True))

group2 = Group("group2", Permissions(read=True, write=False))

user1.add\_to\_group(group1)

user2.add\_to\_group(group2)

# Creating a file and setting permissions

root\_directory.add\_file("file1.txt", "This is the content of file 1", user1, group1, Permissions(read=True, write=True))

root\_directory.add\_file("file2.txt", "This is the content of file 2", user2, group2, Permissions(read=True, write=False))

# Creating a sub-directory

sub\_directory = root\_directory.add\_directory("subdir1")

sub\_directory.add\_file("file3.txt", "This is the content of file 3", user1, group1, Permissions(read=True, write=False))

return root\_directory

# Accessing files and directories via an access path

def access\_path(root\_directory, path, user):

current\_element = root\_directory

elements = path.strip("/").split("/")

for element in elements:

if element in current\_element.sub\_directories:

current\_element = current\_element.sub\_directories[element]

elif element in current\_element.files:

file = current\_element.files[element]

file.access(user)

else:

print(f"{element} not found.")

if \_\_name\_\_ == "\_\_main\_\_":

file\_system = create\_file\_system()

# Accessing files and directories using access path for user1

user1 = User("user1", Permissions(read=True, write=True, execute=True))

access\_path(file\_system, "/file1.txt", user1) # Output: Content of file1.txt: This is the content of file 1

access\_path(file\_system, "/subdir1/file3.txt", user1) # Output: Content of file3.txt: This is the content of file 3

access\_path(file\_system, "/file2.txt", user1) # Output: You don't have permission to read this file.

access\_path(file\_system, "/non\_existent\_file.txt", user1) # Output: non\_existent\_file.txt not found.

# Accessing files and directories using access path for user2

user2 = User("user2", Permissions(read=True, write=False, execute=False))

access\_path(file\_system, "/file1.txt", user2) # Output: Content of file1.txt: This is the content of file 1

access\_path(file\_system, "/subdir1/file3.txt", user2) # Output: You don't have permission to read this file.

access\_path(file\_system, "/file2.txt", user2) # Output: Content of file2.txt: This is the content of file 2

access\_path(file\_system, "/non\_existent\_file.txt", user2) # Output: non\_existent\_file.txt not found.

'''

-The FileSystemElement class is used to encapsulate common properties and methods of both files and directories.

-The File and Directory classes inherit from FileSystemElement, enabling code reuse and extension of behavior.

-The access method in File and Directory classes demonstrates polymorphism, as it takes different forms based on the object type (file or directory) and the user's permissions.

'''

'''

3.-Incorporate use of Singleton pattern in design.

-Design should be more flexible for future changes.

'''

class Permissions:

\_instance = None

def \_\_new\_\_(cls, read=False, write=False, execute=False):

if cls.\_instance is None:

cls.\_instance = super().\_\_new\_\_(cls)

cls.\_instance.read = read

cls.\_instance.write = write

cls.\_instance.execute = execute

return cls.\_instance

def update\_permissions(self, read=False, write=False, execute=False):

self.read = read

self.write = write

self.execute = execute

class User:

def \_\_init\_\_(self, name):

self.name = name

self.groups = set()

def add\_to\_group(self, group):

self.groups.add(group)

class Group:

def \_\_init\_\_(self, name):

self.name = name

self.permissions = Permissions()

class FileSystemElement:

def \_\_init\_\_(self, name, owner, group):

self.name = name

self.owner = owner

self.group = group

def access(self, user):

pass

class File(FileSystemElement):

def \_\_init\_\_(self, name, content, owner, group):

super().\_\_init\_\_(name, owner, group)

self.content = content

def access(self, user):

if self.owner.name == user.name or self.group.name in user.groups or self.group.permissions.read:

print(f"Content of {self.name}: {self.content}")

else:

print("You don't have permission to read this file.")

class Directory(FileSystemElement):

def \_\_init\_\_(self, name, parent=None):

super().\_\_init\_\_(name, None, None) # Directories have no owner or group

self.parent = parent

self.sub\_directories = {}

self.files = {}

def add\_directory(self, directory\_name):

new\_directory = Directory(directory\_name, self)

self.sub\_directories[directory\_name] = new\_directory

return new\_directory

def add\_file(self, file\_name, content, owner, group):

new\_file = File(file\_name, content, owner, group)

self.files[file\_name] = new\_file

return new\_file

def access(self, user):

print(f"Accessing directory: {self.name}")

def create\_file\_system():

root\_directory = Directory("/")

user1 = User("user1")

user2 = User("user2")

# Creating groups and adding users to them

group1 = Group("group1")

group2 = Group("group2")

user1.add\_to\_group(group1)

user2.add\_to\_group(group2)

# Updating permissions for groups

group1.permissions.update\_permissions(read=True, write=True)

group2.permissions.update\_permissions(read=True, write=False)

# Creating a file

root\_directory.add\_file("file1.txt", "This is the content of file 1", user1, group1)

root\_directory.add\_file("file2.txt", "This is the content of file 2", user2, group2)

# Creating a sub-directory

sub\_directory = root\_directory.add\_directory("subdir1")

sub\_directory.add\_file("file3.txt", "This is the content of file 3", user1, group1)

return root\_directory

# Accessing files and directories via an access path

def access\_path(root\_directory, path, user):

current\_element = root\_directory

elements = path.strip("/").split("/")

for element in elements:

if element in current\_element.sub\_directories:

current\_element = current\_element.sub\_directories[element]

elif element in current\_element.files:

file = current\_element.files[element]

file.access(user)

else:

print(f"{element} not found.")

if \_\_name\_\_ == "\_\_main\_\_":

file\_system = create\_file\_system()

# Accessing files and directories using access path for user1

user1 = User("user1")

access\_path(file\_system, "/file1.txt", user1) # Output: Content of file1.txt: This is the content of file 1

access\_path(file\_system, "/subdir1/file3.txt", user1) # Output: Content of file3.txt: This is the content of file 3

access\_path(file\_system, "/file2.txt", user1) # Output: You don't have permission to read this file.

access\_path(file\_system, "/non\_existent\_file.txt", user1) # Output: non\_existent\_file.txt not found.

# Accessing files and directories using access path for user2

user2 = User("user2")

access\_path(file\_system, "/file1.txt", user2) # Output: Content of file1.txt: This is the content of file 1

access\_path(file\_system, "/subdir1/file3.txt", user2) # Output: You don't have permission to read this file.

access\_path(file\_system, "/file2.txt", user2) # Output: Content of file2.txt: This is the content of file 2

access\_path(file\_system, "/non\_existent\_file.txt", user2) # Output: non\_existent\_file.txt not found.

'''

-The Permissions class follows the Singleton pattern, ensuring there is only one instance of the permissions object.

-It's more flexible permissions configuration by using the update\_permissions method to modify permissions for groups.

-Users and groups are created with a more simplified approach, and permissions are managed at the group level.

-The FileSystemElement class is updated to handle permissions at the group level and to use the updated Permissions class.

'''