



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4
**ШАБЛОНИ «SINGLETON»,
«ITERATOR», «PROXY», «STATE»,
«STRATEGY»**

Варіант 14

Виконав:
студент групи ІА – 24:
Любченко І.М

Перевірив:
Мягкий М.Ю

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціонала робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Варіант:

..14 Архіватор (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) - додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.



Хід роботи

1. **Теоретичне введення** У рамках даної роботи було застосовано патерн проектування **Стратегія**, який дозволяє визначати сімейство схожих алгоритмів та розміщувати їх у окремих класах, що дозволяє замінювати один алгоритм на інший під час виконання програми. У цьому випадку патерн застосовувався для створення і розпакування архівів різних форматів, таких як ZIP, RAR і TAR.GZ. Кожен формат архіву має свою стратегію для реалізації операцій створення та розпакування архівів.

```
src > strategies.py > ...
1  class ZipStrategy:
2      def create_archive(self, files):
3          # Логіка для створення ZIP архіву
4          print("Creating ZIP archive...")
5
6      def extract_archive(self, archive):
7          # Логіка для розпакування ZIP архіву
8          print("Extracting ZIP archive...")
9
10 class RarStrategy:
11     def create_archive(self, files):
12         # Логіка для створення RAR архіву
13         print("Creating RAR archive...")
14
15     def extract_archive(self, archive):
16         # Логіка для розпакування RAR архіву
17         print("Extracting RAR archive...")
18
19 class TarGzStrategy:
20     def create_archive(self, files):
21         # Логіка для створення TAR.GZ архіву
22         print("Creating TAR.GZ archive...")
23
24     def extract_archive(self, archive):
25         # Логіка для розпакування TAR.GZ архіву
26         print("Extracting TAR.GZ archive...")
27
```

2. Реалізація патерну Стратегія Використання патерну Стратегія дозволяє зробити алгоритми архівації ізольованими один від одного, що робить код більш гнучким і підтримуваним. Це дозволяє змінювати алгоритм архівації без необхідності змінювати бізнес-логіку програми.

У нашій реалізації були визначені три стратегії для кожного формату архіву:

- **ZipStrategy**: для роботи з ZIP архівами.
- **RarStrategy**: для роботи з RAR архівами.
- **TarGzStrategy**: для роботи з TAR.GZ архівами.

```
src > archiver.py > ...
1  import zipfile
2  import tarfile
3  import rarfile
4
5
6  class Archiver:
7      def __init__(self, archive_type):
8          self.archive_type = archive_type
9          self.archive = None
10
11     def create_archive(self, file_name):
12         if self.archive_type == 'zip':
13             self.archive = zipfile.ZipFile(file_name, 'w', zipfile.ZIP_DEFLATED)
14         elif self.archive_type == 'tar.gz':
15             self.archive = tarfile.open(file_name, 'w:gz')
16         elif self.archive_type == 'rar':
17             raise NotImplementedError("RAR підтримується лише для читання")
18         else:
19             raise ValueError(f"Непідтримуваний тип архіву: {self.archive_type}")
20
21     def add_file(self, file_path):
22         if not self.archive:
23             raise RuntimeError("Архів ще не створено!")
24
25         if self.archive_type == 'zip':
26             self.archive.write(file_path, arcname=file_path.split('/')[-1])
27         elif self.archive_type == 'tar.gz':
28             self.archive.add(file_path)
29
30     def close(self):
31         if self.archive:
32             self.archive.close()
33
```

3. Основна логіка програми Клас **Archiver** працює з обраною стратегією для створення і розпакування архівів. Клас **Archiver** дозволяє змінювати стратегію архівації під час виконання програми, що робить систему гнучкою.

```
src > main.py > ...
1  from gui import ArchiverApp
2  import tkinter as tk
3
4  if __name__ == "__main__":
5      root = tk.Tk()
6      app = ArchiverApp(root)
7      root.mainloop()
8
```

Висновок Реалізувавши патерн **Стратегія**, мені вдалося реалізувати функціональність для створення та розпакування архівів різних форматів. Заміна алгоритмів архівації на льоту значно спростила мені роботу з різними форматами архівів. Це дозволило зробити код більш підтримуваним і зрозумілим.