import os from typing import Optional, Dict, Any import json import logging from abc import ABC, abstractmethod

class AuthError(Exception): """Custom exception class for authentication errors.""" pass

class AuthMethod(ABC): """Abstract base class for authentication methods."""

@abstractmethod  
def authenticate(self):  
 """Abstract method to perform authentication."""  
 pass

class SPNCertificateAuth(AuthMethod): """Authentication using SPN and certificate."""

def \_\_init\_\_(self, spn: str, certificate\_path: str):  
 if not os.path.exists(certificate\_path):  
 raise AuthError(f"Certificate file not found at {certificate\_path}")  
 self.spn = spn  
 self.certificate\_path = certificate\_path  
  
def authenticate(self):  
 """Authenticate using SPN and certificate."""  
 logging.info(f"Authenticating SPN {self.spn} using certificate at {self.certificate\_path}")  
 # Placeholder for SPN with certificate authentication logic  
 return {"token": "dummy\_spn\_certificate\_token"}

class UserPasswordAuth(AuthMethod): """Authentication using username and password."""

def \_\_init\_\_(self, username: str, password: str):  
 if not username or not password:  
 raise AuthError("Username and password must be provided")  
 self.username = username  
 self.password = password  
  
def authenticate(self):  
 """Authenticate using username and password."""  
 logging.info(f"Authenticating user {self.username} using password")  
 # Placeholder for user-password authentication logic  
 return {"token": "dummy\_user\_password\_token"}

class AuthConfig: """Class to manage authentication configuration."""

def \_\_init\_\_(self, config\_file: Optional[str] = None, config\_dict: Optional[Dict[str, Any]] = None):  
 if config\_file:  
 if not os.path.exists(config\_file):  
 raise AuthError(f"Config file not found at {config\_file}")  
 with open(config\_file, 'r') as f:  
 self.config = json.load(f)  
 elif config\_dict:  
 self.config = config\_dict  
 else:  
 raise ValueError("Either config\_file or config\_dict must be provided")  
  
def get\_auth\_method(self) -> AuthMethod:  
 """Instantiate the appropriate authentication method based on configuration."""  
 method = self.config.get("method")  
 if method == "SPN\_CERTIFICATE":  
 return SPNCertificateAuth(  
 spn=self.config["spn"],  
 certificate\_path=self.config["certificate\_path"]  
 )  
 elif method == "USER\_PASSWORD":  
 return UserPasswordAuth(  
 username=self.config["username"],  
 password=self.config["password"]  
 )  
 else:  
 raise AuthError(f"Unsupported authentication method: {method}")

# **Example usage**

def main(): """Main function to demonstrate authentication module usage.""" logging.basicConfig(level=logging.INFO)

try:  
 # Example configuration using a dictionary  
 config\_dict = {  
 "method": "USER\_PASSWORD",  
 "username": "test\_user",  
 "password": "test\_password"  
 }  
  
 auth\_config = AuthConfig(config\_dict=config\_dict)  
 auth\_method = auth\_config.get\_auth\_method()  
 token = auth\_method.authenticate()  
 logging.info(f"Authentication successful, token: {token}")  
except AuthError as e:  
 logging.error(f"Authentication failed: {e}")  
except Exception as e:  
 logging.error(f"An unexpected error occurred: {e}")

if **name** == "**main**": main()