```
from abc import ABC, abstractmethod

# Clase abstracta que define la interfaz para los manejadores.
class SupportHandler(ABC):
    def __init__(self):
        self._next_handler = None

    def set_next(self, handler):
        """Establece el siguiente manejador en la cadena"""
        self._next_handler = handler
        return handler  # Permite encadenamiento

    @abstractmethod
    def handle(self, request):
        """Método abstracto que debe implementar cada manejador concreto"""
        pass

# Manejador de soporte básico.
# Resuelve problemas simples como contraseñas y acceso básico.
class BasicSupport(SupportHandler):
    def handle(self, request):
        if request.priority in ['low', 'medium']:
            print(f"BasicSupport resolviendo: {request.description}")
            request.resolved = True
            request.handler_name = "BasicSupport"
            return request
        else:
            if self._next_handler:
```

```
            print(f"BasicSupport escalando a siguiente nivel...")
            return self._next_handler.handle(request)
        else:
            print(f"No hay manejador disponible para: {request}")
        return request

# Manejador de soporte técnico especializado.
# Resuelve problemas técnicos complejos.
class SpecializedTechnician(SupportHandler):
    def handle(self, request):
        if request.priority in ['medium', 'high']:
            print(f"SpecializedTechnician resolviendo: {request.description}")
            request.resolved = True
            request.handler_name = "SpecializedTechnician"
            return request
        else:
            if self._next_handler:
                print(f"SpecializedTechnician escalando a siguiente nivel...")
                return self._next_handler.handle(request)
            else:
                print(f"No hay manejador disponible para: {request}")
            return request

# Manejador de gerente de soporte.
# Resuelve problemas críticos o no resueltos por otros niveles.
class SupportManager(SupportHandler):
    def handle(self, request):
        print(f"SupportManager resolviendo: {request.description}")
```

```
        request.resolved = True
        request.handler_name = "SupportManager"
        if self._next_handler:
            return self._next_handler.handle(request)
        return request

# Representa una solicitud de soporte técnico.
# Contiene información sobre el problema a resolver.
class SupportRequest:
    def __init__(self, request_id, description, priority):
        self.request_id = request_id
        self.description = description
        self.priority = priority  # Prioridad: 'low', 'medium', 'high', 'critical'
        self.resolved = False
        self.handler_name = None

    def __str__(self):
        return (f"ID: {self.request_id} | "
                f"Prioridad: {self.priority} | "
                f"Descripción: {self.description} | "
                f"Resuelto: {self.resolved}")

# Función principal que demuestra el uso del patrón Chain of Responsibility.
# Crea la cadena de manejadores y procesa solicitudes.
def main():
    # Crear los manejadores
    basic_support = BasicSupport()
    specialized_tech = SpecializedTechnician()
```

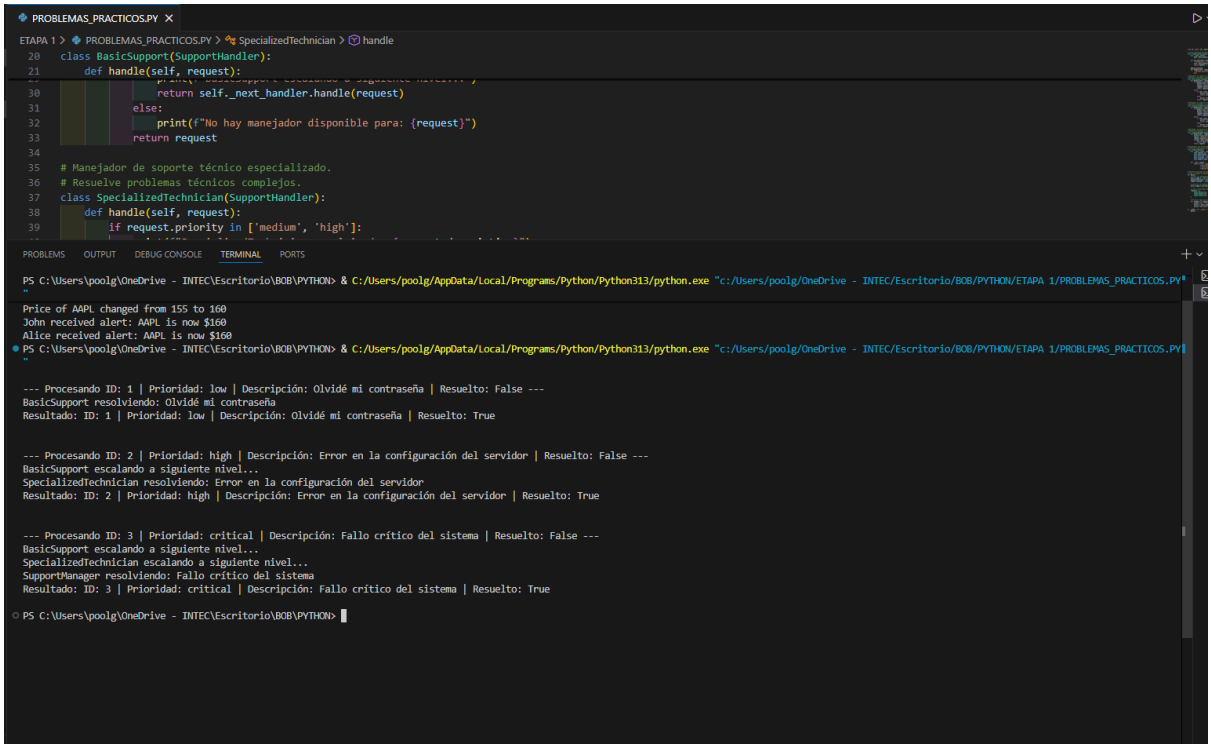
```
    support_manager = SupportManager()

    # Construir la cadena: BasicSupport -> SpecializedTechnician -> SupportManager
    basic_support.set_next(specialized_tech).set_next(support_manager)

    # Crear solicitudes de prueba
    requests = [
        SupportRequest(1, "Olvidé mi contraseña", "low"),
        SupportRequest(2, "Error en la configuración del servidor", "high"),
        SupportRequest(3, "Fallo crítico del sistema", "critical"),
    ]

    # Procesar cada solicitud
    for request in requests:
        print(f"\n--- Procesando {request} ---")
        result = basic_support.handle(request)
        print(f"Resultado: {result}\n")

if __name__ == "__main__":
    main()
```


PROBLEMAS_PRACTICOS.PY
ETAPA 1 > PROBLEMAS_PRACTICOS.PY > SpecializedTechnician > handle
20 class BasicSupport(SupportHandler):
21 def handle(self, request):
30 return self._next_handler.handle(request)
31 else:
32 print(f"No hay manejador disponible para: {request}")
33 return request
34
35 # Manejador de soporte técnico especializado.
36 # Resuelve problemas técnicos complejos.
37 class SpecializedTechnician(SupportHandler):
38 def handle(self, request):
39 if request.priority in ['medium', 'high']:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\poolg\OneDrive - INTEC\Escritorio\BOB\PYTHON> & C:/Users/poolg/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/poolg/OneDrive - INTEC/Escritorio/BOB/PYTHON/ETAPA 1/PROBLEMAS_PRACTICOS.PY"
Price of AAPL changed from 155 to 160
John received alert: AAPL is now $160
Alice received alert: AAPL is now $160
PS C:\Users\poolg\OneDrive - INTEC\Escritorio\BOB\PYTHON> & C:/Users/poolg/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/poolg/OneDrive - INTEC/Escritorio/BOB/PYTHON/ETAPA 1/PROBLEMAS_PRACTICOS.PY"
--- Procesando ID: 1 | Prioridad: low | Descripción: Olvidé mi contraseña | Resuelto: False ---
BasicSupport resolviendo: Olvidé mi contraseña
Resultado: ID: 1 | Prioridad: low | Descripción: Olvidé mi contraseña | Resuelto: True
--- Procesando ID: 2 | Prioridad: high | Descripción: Error en la configuración del servidor | Resuelto: False ---
BasicSupport escalando a siguiente nivel...
SpecializedTechnician resolviendo: Error en la configuración del servidor
Resultado: ID: 2 | Prioridad: high | Descripción: Error en la configuración del servidor | Resuelto: True
--- Procesando ID: 3 | Prioridad: critical | Descripción: Fallo crítico del sistema | Resuelto: False ---
BasicSupport escalando a siguiente nivel...
SpecializedTechnician escalando a siguiente nivel...
SupportManager resolviendo: Fallo crítico del sistema
Resultado: ID: 3 | Prioridad: critical | Descripción: Fallo crítico del sistema | Resuelto: True
PS C:\Users\poolg\OneDrive - INTEC\Escritorio\BOB\PYTHON>