```cpp
// TestProxy.cpp
//
// (Protection) Proxy pattern using C++ Concepts. This is a very good/elegant
// motivational example to show how it works.
//
// Proxy is an implementation of some well-known functionality in the Access Control
// System (ACS) domain category.
//
// We developed the C++ code by using Proxy prototype code in Python. This code
// is input to design in C++, this is clever use of "design resources".
//
// More info ->
//
//
// Modern Multiparadigm Software Architectures and Design Patterns
// with Examples and Applications in C++, C# and Python Volume I
// Datasim Press 2023, Daniel J.Duffy and Harold Kasperink.
//
// Volume II Ïnteroperability"
//
//     Call C++ from Python and vice versa
//     Call C# from Python and vice versa
//     Call C# from native C++and vice versa
//     (foundations, methods and applications)
//
// In vol II, we can call Python Proxy code from  C++ and vice versa.
//
// (C) Datasim Education BV 2023
//
#include <iostream>

// Level 1
template <typename T>
       concept ICustomer = requires (T& t)
{
       t.valid();
};

template <typename T>
       concept IAccount = requires (T& t, int amount)
{
              t.withdraw(amount);
};


// Level 2
template <ICustomer Customer, IAccount Account>
       struct ProxyClient
{
       Customer _cus;
       Account _acc;

       ProxyClient(Customer customer, Account account) : _cus(customer), _acc(account) {}

       void withdraw(int amount)
       {

              if (!_cus.valid())
```

```
                     throw std::exception("sorry mate, account not legal");

              _acc.withdraw(amount);
       }

       int balance() const
       {
              return _acc.balance();
       }
};

// Level 3
struct MyCustomer
{
       bool _st;
       MyCustomer(bool status) : _st(status) {}
       bool valid() { return _st; }
};

struct MyAccount
{
       int _id;
       int _balance;

       MyAccount(int id, int deposit_amount) : _id(id), _balance(deposit_amount) {}
       void withdraw(int amount)
       {
              _balance -= amount;
       }

       int balance() const
       {
              return _balance;
       }
};

struct PhoneyAccount
{ // Does not satisy the interface

       int _id;
       int _savings;

       PhoneyAccount(int id, int deposit_amount) : _id(id), _savings(deposit_amount) {}
/*     void withdraw(int amount)
       {
              _savings -= amount;
       }
*/

};

int main()
{
       MyCustomer cus(true);
       MyAccount acc(42, 1000);

       //PhoneyAccount acc2(42, 1'000'000);
```

```
	ProxyClient<MyCustomer, MyAccount> client(cus, acc);
	client.withdraw(100);
	std::cout << "Balance 1: " << client.balance() << '\n';

	//ProxyClient<MyCustomer, PhoneyAccount> client2(cus, acc);
	//client2.withdraw(100);
	//

	MyCustomer cus2(false);
	MyAccount acc2(42, 1'000'000);
	ProxyClient<MyCustomer, MyAccount> client2(cus2, acc2);
	try
	{
		client2.withdraw(100);
	}
	catch (std::exception& ex)
	{
		std::cout << ex.what() << '\n';
	}


}

// Python
/*
"""
Proxy pattern example.
There are 7 Proxy styles (remote, protection, cache, synchronisation,
counting, virtual, firewall. See POSA (1996))
"""
from abc import ABCMeta, abstractmethod


NOT_IMPLEMENTED = "You should implement this."


class Account:
	__metaclass__ = ABCMeta

	@abstractmethod
	def withdraw(self, amount : int):
		raise NotImplementedError(NOT_IMPLEMENTED)


class CurrentAccount(Account):
	def withdraw(self, amount : int):
		print("amount withdrawn!")


class Customer:
	def __init__(self, status: bool):
		self.status = status


class ProxyAccount(Account):
	def __init__(self, customer : Customer, acc : Account):
		self.customer = customer
		self.acc = acc
```

```
       def withdraw(self, amount : int):
              if self.customer.status == False:
                     print("Sorry, customer not cleared")
              else:
                     self.acc.withdraw(amount)


customer = Customer(True)
acc = CurrentAccount()
acc = ProxyAccount(customer, acc)
acc.withdraw(100)

customer = Customer(False)
acc = CurrentAccount()
acc = ProxyAccount(customer, acc)
acc.withdraw(7000)



*/
```