First Class ADT Pattern

Encapsulation is a tool for **managing complexity** and to **separate aspects of a problem**.

Two views of a design: the **external behavior** (interface) and the **internal details** (implementation).

ADT: Abstract Data Type - a set of values and operations on these values [1]

FIRST-CLASS ADT : "a first-class citizen [..] is an entity which supports all the operations generally available to other entities. These operations typically include being passed as an argument, returned from a function, modified, and assigned to a variable." [2]

Exercise

Write a program that manages Order and Customers

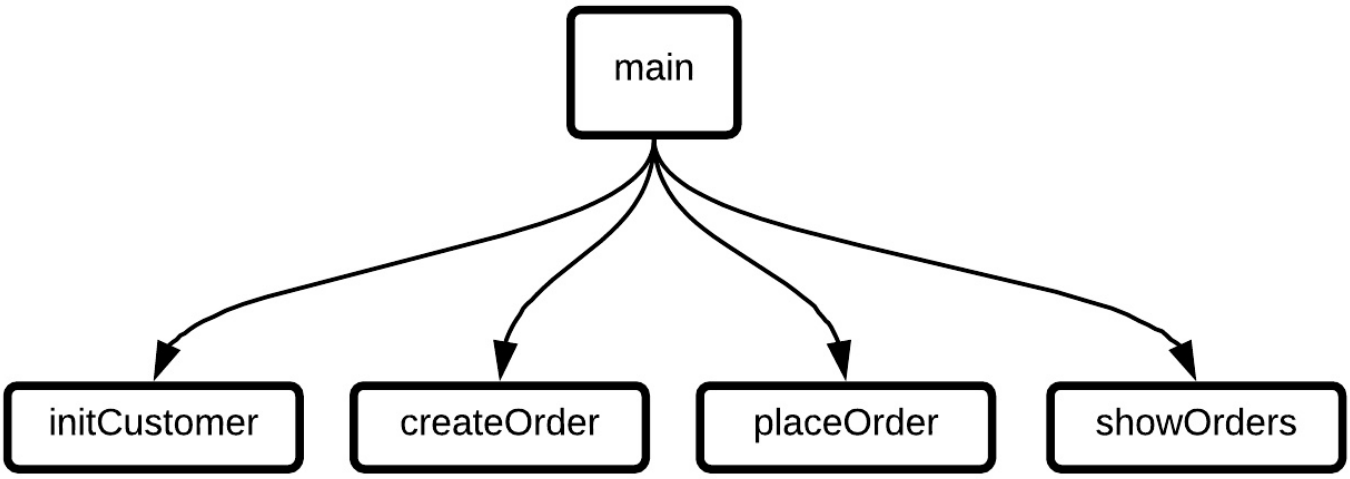
data:
- order: idOrder, price
- customer: name, number of orders, list of orders.

functions:
- `initCustomer()` initializes a Customer
- `createOrder()` creates an Order
- `placeOrder()` adds an *order* to the *list of orders* contained in a Customer
- `showOrders()` shows all the Orders in a Customer

---

functions:

- `initCustomer()` initializes a Customer
- `createOrder()` creates an Order
- `placeOrder()` adds an *order* to the *list of orders* contained in a Customer
- `showOrders()` shows all the Orders in a Customer



A typical `main` :

```
int main(){
    // Create 2 customers and 4 orders
    // Show that customers contain NO order, for now.
    // ...

    // add orders to customers
    // ...

    // Show that customers contain order, now!
    // ...

    getchar();
    return 0;
}
```

A typical `main` :

```c
// include...
int main(){
    Customer aCustomer_1, aCustomer_2;
    Order anOrder_1, anOrder_2, anOrder_3, anOrder_4;

    initCustomer(&aCustomer_1,"Luca");

    // attenzione: avrei potuto scrivere anche
    // aCustomer_1.name="Luca";
    // aCustomer_1.nOfOrders=0;
    // accedendo direttamente ai campi della struttura
    // i dati NON sono incapsulati

    initCustomer(&aCustomer_2,"Franco");
    printf("\n\n");
    showOrders(&aCustomer_1);
    showOrders(&aCustomer_2);

    anOrder_1=createOrder(1,100);
    anOrder_2=createOrder(2,200);
    anOrder_3=createOrder(3,300);
    anOrder_4=createOrder(4,400);

    placeOrder(&aCustomer_1, anOrder_1);
    placeOrder(&aCustomer_1, anOrder_2);
    placeOrder(&aCustomer_2, anOrder_3);
    placeOrder(&aCustomer_2, anOrder_4);

    printf("\n\n");
    showOrders(&aCustomer_1);
    showOrders(&aCustomer_2);

    getchar();
    return 0;
}
```

Exercise (hint)

3 files: main.c, customer.c, customer.h

The data definition (Order, Customer) are in the interface customer.h

Customer contains a list (a static array) of orders

Solve the exercise!

https://codeboard.io/projects/139782

(Solution in https://github.com/DIEE-PAIS-code/placeOrder_ADT_01_c)

PROBLEMS

Lack of information hiding:

- details of the module customer are unveiled to the client (see `customer.h`)
- if we change the implementation (i.e. linked list instead of array) we must inform the client
- the client has access to the data structure

SOLUTION

The C standard [C 1999] allows us to declare objects of incomplete types in a context where there sizes aren't needed.
We can specify **a pointer to an incomplete type**.

See the code in
https://github.com/DIEE-PAIS-code/ADT_01_c

Using a pointer to an incomplete type

```
typedef struct MyData *MyDataPtr;
```

we can declare in the interface `.h` the alias `MyDataPtr` as a pointer to an unknown struct. It is not a problem if we use only pointers to the **unknown struct MyData**.

The `main` can declare pointers `p` to the unknown struct MyData, and can use them, because **the size needed to store a pointer is independent to the size of the pointed object**. The size of the pointer depends on the architecture of the system in which it is implemented.

- the main can declare variables (i.e. pointers) of MyDataPtr type
- the main can use these pointers (pass te pointer to a function, and so on
- the main can not **dereference** the pointer (i.e., can not use *p) because the main does not have access to the data structure `MyData` , which is stored in `.c`

In this way we can encapsulate the data.

(please reflect on these points and their implications)

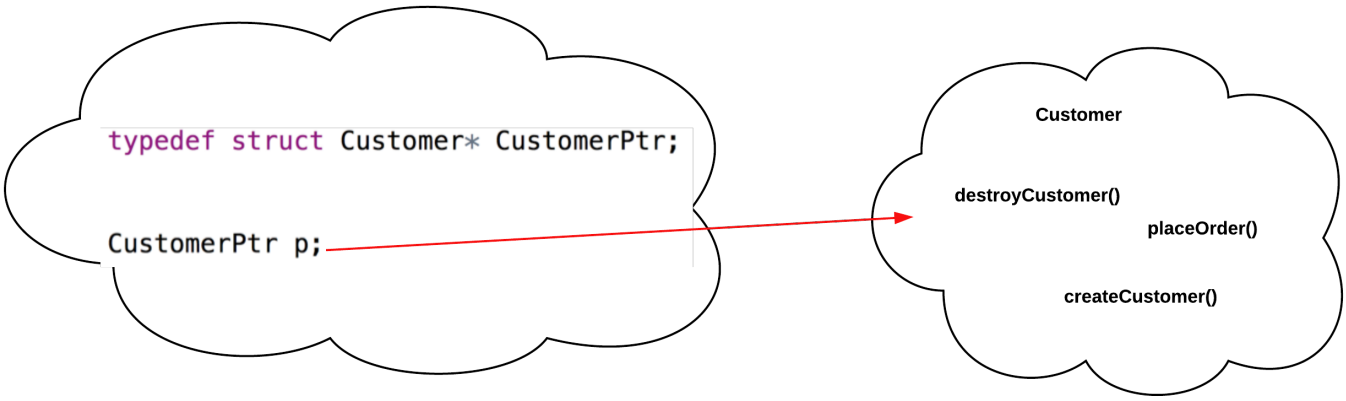
SOLUTION

A pointer to an incomplete type

```
typedef struct Customer* CustomerPtr;
```

provides the possibility to handle the entity Customer, without being able to directly access to the data.

The `main` cannot have direct access to the data, but only to the pointer. So we need also a **complete set of functions to manipulate the data**.



Allocation/deallocation

We **must** allocate and deallocate memory dynamically (**why?**)

```
struct Customer {
    const char* name;
    int nOfOrders;
    Order orders[MAX_NO_OF_ORDERS];
};
void destroyCustomer(CustomerPtr p_customer) {
    /* Perform clean-up of the customer internals, if necessary. */
    free(p_customer);
}
```

NB We must perform clean-up of the customer internals, if necessary.
i.e.

```
struct Customer {
    const char* name;
    int nOfOrders;
    Order * p_linkedOrders;
};
```

Try to solve the problem **before looking at the solution** in https://github.com/DIEE-PAIS-code/placeOrder_ADT_02_c

Other details (not related with ADT pattern)

```
void initCustomer(Customer * p_customer, const char* name){
```

** const** ensures that the function can not change the value pointed to by `name`

Take a look at these differences

```
const char *name;
```

`name` is a modifiable pointer. The value pointed to by `name` is not modifiable.

```
char *const name = &c;
```

Here the pointer `name` is constant.

This is a typical call to the `malloc` function. `malloc` must know how much memory to allocate.

```
Customer * p_customer = malloc(sizeof Customer);
```

Problem: If we change `Customer` into `NewCustomer`, we need to change the code in 2 places!

```
NewCustomer * p_customer = malloc(sizeof NewCustomer);
```

A failure to update both places may have fatal consequences, potentially leaving the code with undefined behavior.

Solution:

```
Customer * p_customer = malloc(sizeof * p_customer);
```

or

```
CustomerPtr p_customer = malloc(sizeof * p_customer);
```

---

1. https://en.wikipedia.org/wiki/Abstract_data_type ↩
2. https://en.wikipedia.org/wiki/First-class_citizen ↩