HW #5 Christmas presents (Inheritance)

- Your company sends Christmas presents to all their registered customers every year. Of course there are some customers that are more important than others, so the price and quality of the present will not be the same for every customer.

- While developing a new software system for keeping information about customers your project manager thinks it is a good idea to include the information about the yearly Christmas present, too, and automatically generate a list of customers and the presents they get for the secretary every year.

HW #5 (2)

- In the group meeting the public interfaces for the classes to be implemented have been agreed upon. So you already have the public parts of the classes you have to implement plus the code implemented by the rest of the team.

- Your job is now to add the private or protected methods or data objects you need to the classes **Customer**, **OrdinaryCustomer** and **PremiumCustomer** and implement them.

HW #5 (3)

- The **constructor** of each of these classes should print out the number of instances of that class created up to that point. Thus when an instance of **OrdinaryCustomer** is created we might get a printout that says that

This is the 3rd **OrdinaryCustomer**, and 8th **Customer**.

- Similarly, the **destructor** should print how many instances of that class were left after the time of destroying that instance. For example,

Survived 1 **OrdinaryCustomer**, and 1 **Customer**.

HW #5 (4)

- The class **Customer** is meant to be abstract, so objects should only be created from its derived classes. In the Christmas application only the interface of the parent class is used, so the derived classes need to implement different behavior for their own **christmasPresent()** methods.

- Create a new source file **Customer.cpp** for your solution. Add a **main()** program to test your code (the quality of the test cases will be marked, too).

HW #5 (5)

- You find the header file for your code as **Customer.h**. You can also try out your code with the **CustomerList** class implemented in the **CustomerList.cpp** and **CustomerList.h** files.

- The **christmasPresent()** methods you have to implement should produce outputs like these:

HW #5 (6)

Vivian Wang, at Taichung.
This is the 1st OrdinaryCustomer, and 1st Customer.
Please send one bottle Holsten using ordinary present wrapper.

Sam Lee, at Kaohsiung.
This is the 1st PremiumCustomer, and 2nd Customer.
Please send one bottle Bordeaux using premium present wrapper.

John Chen, at Taipei.
This is the 2nd OrdinaryCustomer, and 3rd Customer.
Please send one bottle Ginger Ale using ordinary present wrapper.

Survived 1 OrdinaryCustomer, and 2 Customer.
Survived 0 PremiumCustomer, and 1 Customer.
Survived 0 OrdinaryCustomer, and 0 Customer.

HW #5 (7)

- Listing of **Customer.h** follows:

```
#ifndef CUSTOMER_H
#define CUSTOMER_H
// An abstract class for a Customer.
// Add to this class what is necessary
class Customer {
  protected:
  public:
    // Create a customer with:
    // f: first name
    // s: surname
    // t: town
    Customer (char *f, char *s, char *t);
```

HW #5 (8)

```
// Create a customer copying the properties from another object
// c: a reference to the other Customer object
Customer (Customer &c);

// Release previously allocated memory
virtual ~Customer (void);

// Write a message for the secretary, common part
virtual void christmasPresent (void);
};
```

HW #5 (9)

```
class OrdinaryCustomer : public Customer {
  protected:
  public:
    // Create an ordinary customer with:
    // f: first name
    // s: surname
    // t: town
    // b: favorite beer brand
    OrdinaryCustomer (char *f, char *s, char *t, char *b);

    // Create an ordinary customer copying the properties from another object
    // c: a reference to the other OrdinaryCustomer object
    OrdinaryCustomer (OrdinaryCustomer &c);
```

HW #5 (10)

```
// Release previously allocated memory
virtual ~OrdinaryCustomer (void);

// Write a message for the secretary, particular part
virtual void christmasPresent (void);
};
```

HW #5 (11)

```
class PremiumCustomer : public Customer {
  protected:
  public:
    // Create a premium customer with:
    // f: first name
    // s: surname
    // t: town
    // w: favorite wine brand
    PremiumCustomer (char *f, char *s, char *t, char *w);
```

HW #5 (12)

```
// Create a premium customer copying the properties
// from another object
// c: a reference to the other PremiumCustomer object
PremiumCustomer (PremiumCustomer &c);

// Release previously allocated memory
virtual ~PremiumCustomer (void);

// Write a message for the secretary, particular part
virtual void christmasPresent (void);
};
#endif // CUSTOMER_H
```

A sample main.cpp

```
#include "CustomerList.h"
#include <iostream>
using namespace std;

int main (void) {
  int i = 0;
  CustomerList *list = new CustomerList ();

  list->to (i, new OrdinaryCustomer (“Vivian", “Wang", “Taichung”, "Holsten"));

  ++i;
  list->to (i, new PremiumCustomer (“Sam", “Lee", “Kaohsiung", "Bordeaux"));

  ++i;
  list->to (i, new OrdinaryCustomer (“John", “Chen", “Taipei”, "Ginger Ale"));
```

A sample main.cpp (cont)

```
  // Produce outputs (Write messages for the secretary) – fill in the code here

  // Release previously allocated memory – file in the code here

  delete list;
  return 0;
}
```

CustomerList.h

```
#ifndef CUSTOMER_LIST_H
#define CUSTOMER_LIST_H
#include "Customer.h“
class CustArray {
  Customer **ar;
  int arsize;
  void init (int);
protected:
  void do_resize (int);
  int do_size (void) { return arsize; }
  Customer *do_at (int);
  void do_remove (int);
  void do_to (int, Customer*);
public:
  CustArray (int);
  CustArray (void);
  ~CustArray (void);
};
```

CustomerList.h (cont)

```
class CustomerList : public CustArray {
public:
  // Creates an empty customer list
  CustomerList (void) {}
  // Does nothing
  ~CustomerList (void) {}
  // Returns a copy of the Customer at position pos
  Customer *at (int pos) { return do_at (pos); }
  // Removes the Customer from position pos
  void remove (int pos) { do_remove (pos); }
  // Enters a Customer to position pos
  void to (int pos, Customer *val) { do_to (pos, val); }
  // Returns the current size
  int size (void) { return do_size (); }
  // Resizes the list to sz discarding existing entries
  void resize (int sz) { do_resize (sz); }
};
#endif // CUSTOMER_LIST_H
```

CustomerList.cpp

```
#include "CustomerList.h"
#include <iostream>
using namespace std;

void  CustArray::do_resize (int sz) {
  Customer **tmp = new (Customer*) [sz];
  int i;
  int copynum = (arsize < sz? arsize: sz);

  // Copy existing fields
  for (i = 0; i < copynum; i++) tmp[i] = ar[i];

  // Initialize additional fields (if growing)
  for (i = copynum; i < sz; i++) tmp[i] = NULL;

  delete [ ] ar;
  ar = tmp;
  arsize = sz;
}
```

CustomerList.cpp (2)

```
void CustArray::init (int sz) {
  // assert (sz > 0);
  arsize = sz;
  ar = new (Customer*) [arsize];
  int i;
  for (i = 0; i < arsize; i++) ar[i] = NULL;
}

Customer *CustArray::do_at (int pos) {
  if (pos >= arsize || pos < 0) ; // Index out of bounds
  return ar[pos];
}

void CustArray::do_remove (int pos) {
  if (pos >= arsize || pos < 0) ; // Index out of bounds
  if (ar[pos] != NULL) ar[pos] = NULL;
}
```

CustomerList.cpp (3)

```
void  CustArray::do_to (int pos, Customer *val) {
  if (pos >= arsize || pos < 0) ; // Index out of bounds
  ar[pos] = val;
}

CustArray::CustArray (int sz) { init (sz); }

CustArray::CustArray (void) { init (100); }

CustArray::~CustArray (void) { delete [ ] ar; }
```