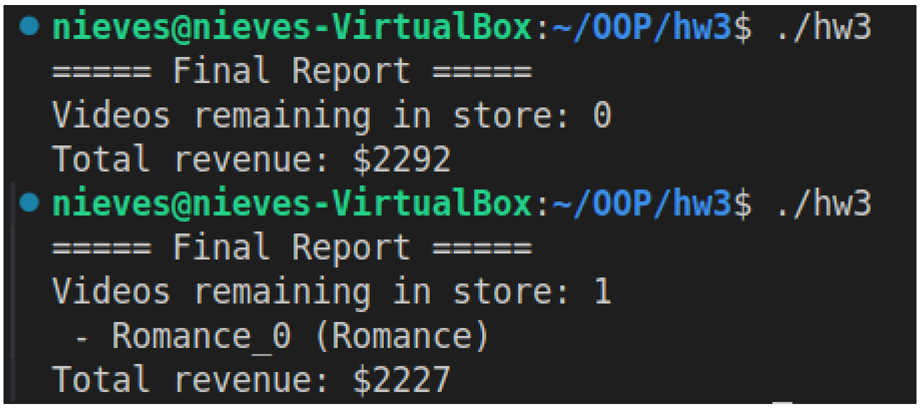
執行步驟:

安裝 g++編譯器後，直接在終端機下 make 指令，會編譯出 hw3 的執行檔，輸入./hw3 即可看到執行結果

執行結果如下:

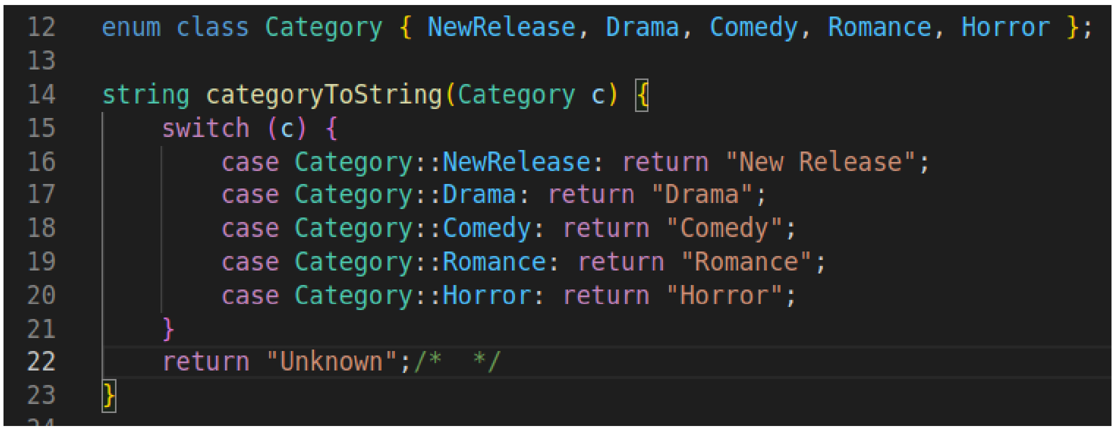


```
nieves@nieves-VirtualBox:~/OOP/hw3$ ./hw3
===== Final Report =====
Videos remaining in store: 0
Total revenue: $2292
nieves@nieves-VirtualBox:~/OOP/hw3$ ./hw3
===== Final Report =====
Videos remaining in store: 1
  - Romance_0 (Romance)
Total revenue: $2227
```

因為顧客借書數量跟天數都為 random 處理，所以每次執行結果都可能不一樣

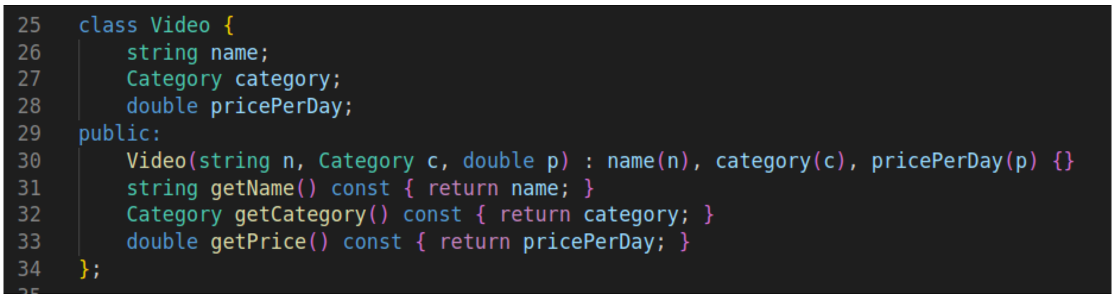
程式介紹:

影片類別



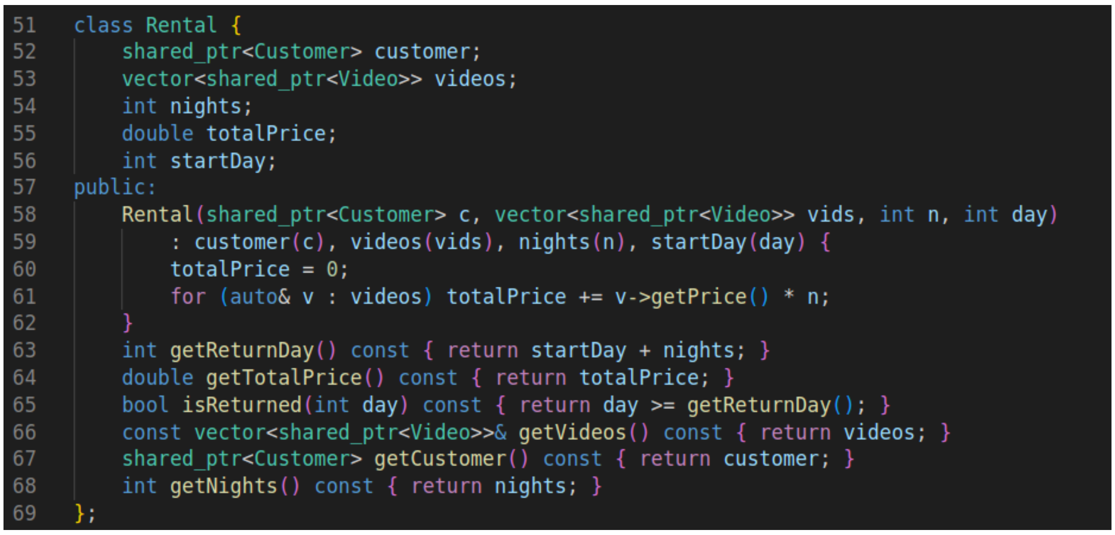
```cpp
enum class Category { NewRelease, Drama, Comedy, Romance, Horror };

string categoryToString(Category c) {
    switch (c) {
        case Category::NewRelease: return "New Release";
        case Category::Drama: return "Drama";
        case Category::Comedy: return "Comedy";
        case Category::Romance: return "Romance";
        case Category::Horror: return "Horror";
    }
    return "Unknown";/*   */
}
```

Video class: 代表一部影片，包含名稱、類別、每日租金。



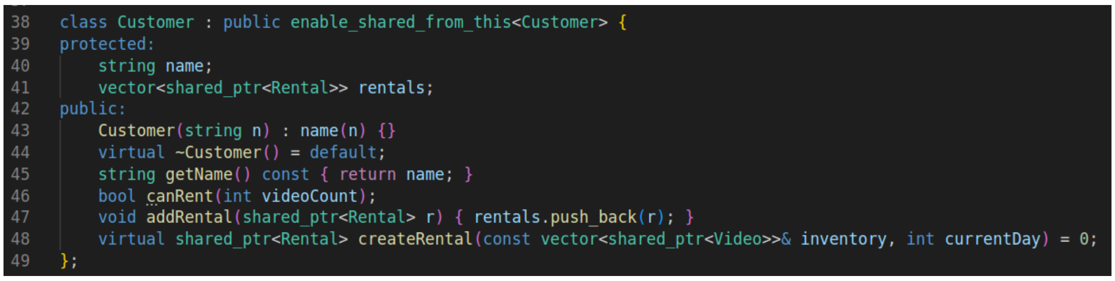
```cpp
class Video {
    string name;
    Category category;
    double pricePerDay;
public:
    Video(string n, Category c, double p) : name(n), category(c), pricePerDay(p) {}
    string getName() const { return name; }
    Category getCategory() const { return category; }
    double getPrice() const { return pricePerDay; }
};
```

Rental class: 代表一次租借行為，紀錄顧客、影片清單、租幾晚、總金額與起租日。

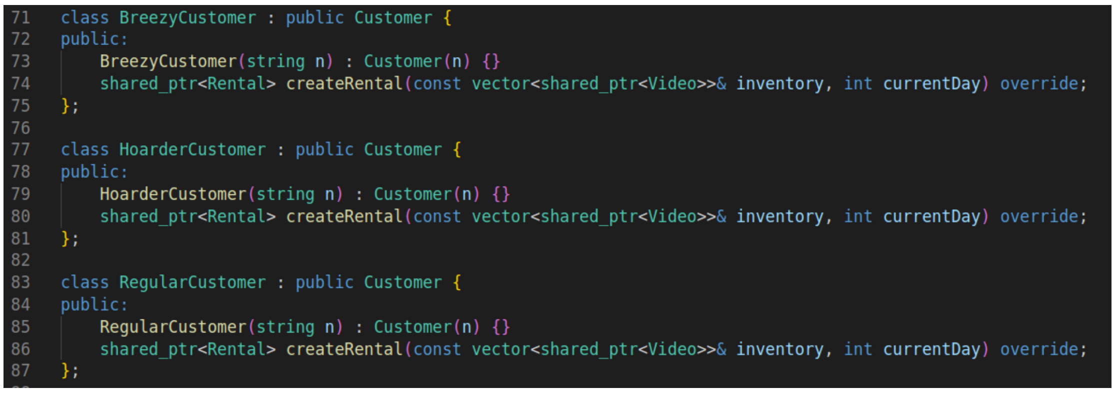


```cpp
class Rental {
    shared_ptr<Customer> customer;
    vector<shared_ptr<Video>> videos;
    int nights;
    double totalPrice;
    int startDay;
public:
    Rental(shared_ptr<Customer> c, vector<shared_ptr<Video>> vids, int n, int day)
        : customer(c), videos(vids), nights(n), startDay(day) {
        totalPrice = 0;
        for (auto& v : videos) totalPrice += v->getPrice() * n;
    }
    int getReturnDay() const { return startDay + nights; }
    double getTotalPrice() const { return totalPrice; }
    bool isReturned(int day) const { return day >= getReturnDay(); }
    const vector<shared_ptr<Video>>& getVideos() const { return videos; }
    shared_ptr<Customer> getCustomer() const { return customer; }
    int getNights() const { return nights; }
};
```

抽象 Customer 提供基本屬性與介面，三種子類別實作不同的租片邏輯。



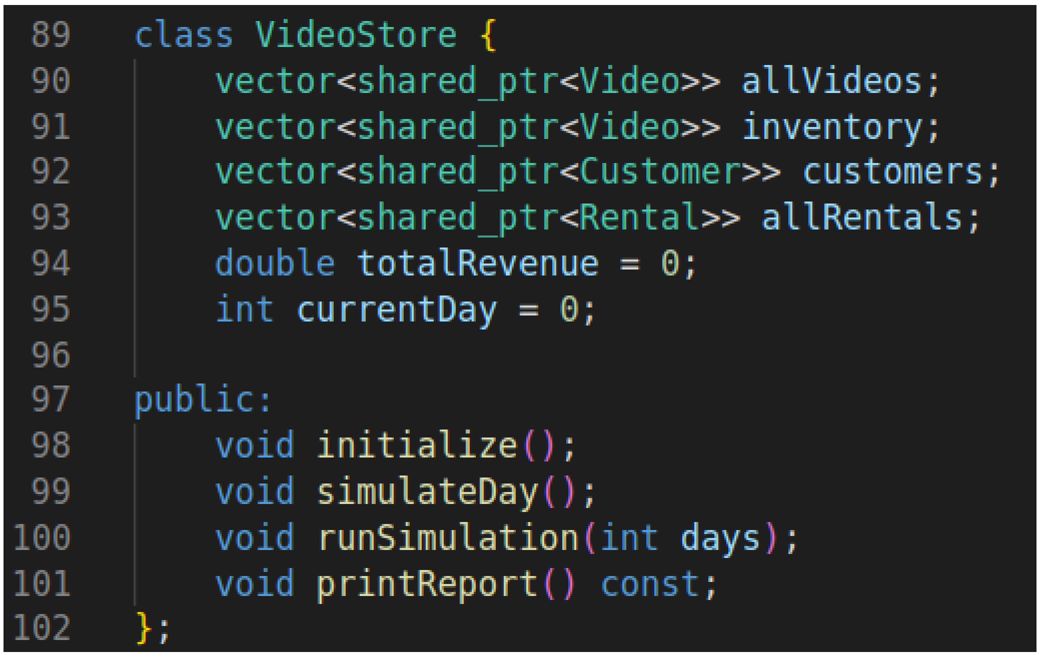
```cpp
class Customer : public enable_shared_from_this<Customer> {
protected:
    string name;
    vector<shared_ptr<Rental>> rentals;
public:
    Customer(string n) : name(n) {}
    virtual ~Customer() = default;
    string getName() const { return name; }
    bool canRent(int videoCount);
    void addRental(shared_ptr<Rental> r) { rentals.push_back(r); }
    virtual shared_ptr<Rental> createRental(const vector<shared_ptr<Video>>& inventory, int currentDay) = 0;
};
```



```cpp
class BreezyCustomer : public Customer {
public:
    BreezyCustomer(string n) : Customer(n) {}
    shared_ptr<Rental> createRental(const vector<shared_ptr<Video>>& inventory, int currentDay) override;
};

class HoarderCustomer : public Customer {
public:
    HoarderCustomer(string n) : Customer(n) {}
    shared_ptr<Rental> createRental(const vector<shared_ptr<Video>>& inventory, int currentDay) override;
};

class RegularCustomer : public Customer {
public:
    RegularCustomer(string n) : Customer(n) {}
    shared_ptr<Rental> createRental(const vector<shared_ptr<Video>>& inventory, int currentDay) override;
};
```

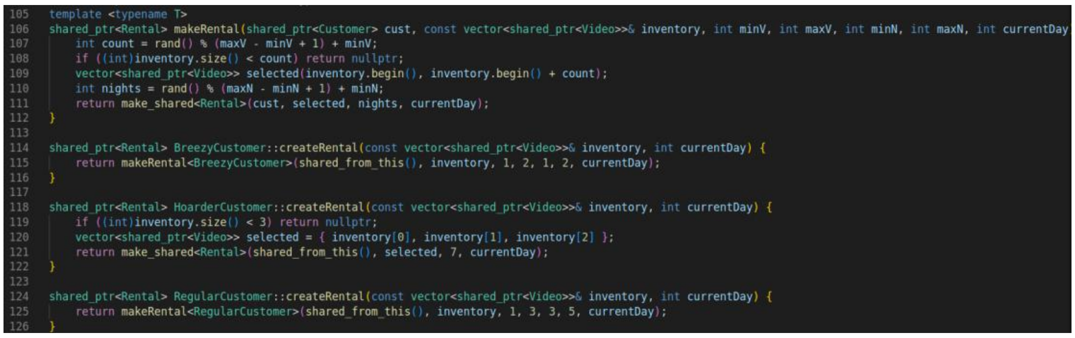
videoStore class: 負責初始化影片與顧客、模擬每日租片與歸還、計算營收與輸

出報告



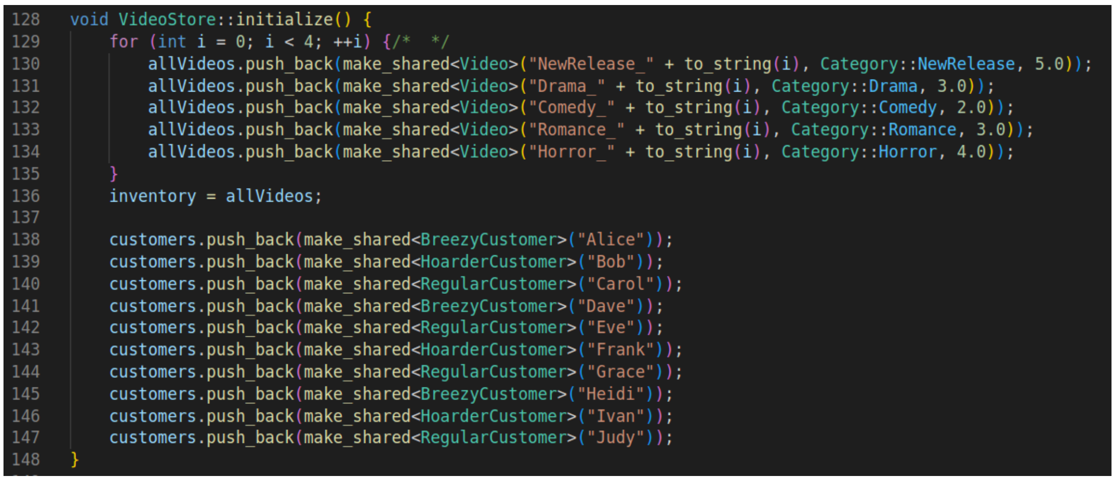
```cpp
class VideoStore {
    vector<shared_ptr<Video>> allVideos;
    vector<shared_ptr<Video>> inventory;
    vector<shared_ptr<Customer>> customers;
    vector<shared_ptr<Rental>> allRentals;
    double totalRevenue = 0;
    int currentDay = 0;

public:
    void initialize();
    void simulateDay();
    void runSimulation(int days);
    void printReport() const;
};
```

設定不同類別顧客的借書天數及借書數量



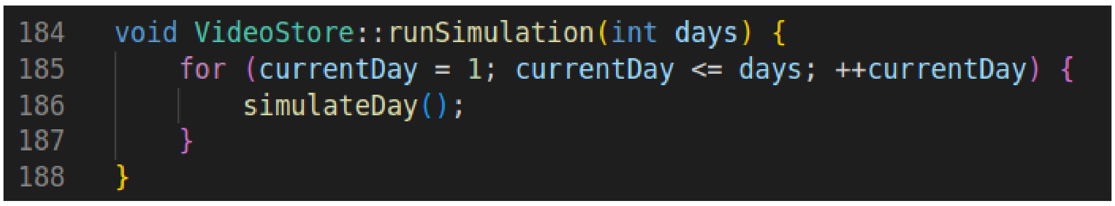
```cpp
template <typename T>
shared_ptr<Rental> makeRental(shared_ptr<Customer> cust, const vector<shared_ptr<Video>>& inventory, int minV, int maxV, int minN, int maxN, int currentDay
    int count = rand() % (maxV - minV + 1) + minV;
    if ((int)inventory.size() < count) return nullptr;
    vector<shared_ptr<Video>> selected(inventory.begin(), inventory.begin() + count);
    int nights = rand() % (maxN - minN + 1) + minN;
    return make_shared<Rental>(cust, selected, nights, currentDay);
}

shared_ptr<Rental> BreezyCustomer::createRental(const vector<shared_ptr<Video>>& inventory, int currentDay) {
    return makeRental<BreezyCustomer>(shared_from_this(), inventory, 1, 2, 1, 2, currentDay);
}

shared_ptr<Rental> HoarderCustomer::createRental(const vector<shared_ptr<Video>>& inventory, int currentDay) {
    if ((int)inventory.size() < 3) return nullptr;
    vector<shared_ptr<Video>> selected = { inventory[0], inventory[1], inventory[2] };
    return make_shared<Rental>(shared_from_this(), selected, 7, currentDay);
}

shared_ptr<Rental> RegularCustomer::createRental(const vector<shared_ptr<Video>>& inventory, int currentDay) {
    return makeRental<RegularCustomer>(shared_from_this(), inventory, 1, 3, 3, 5, currentDay);
}
```

初始化書籍資訊及顧客資訊



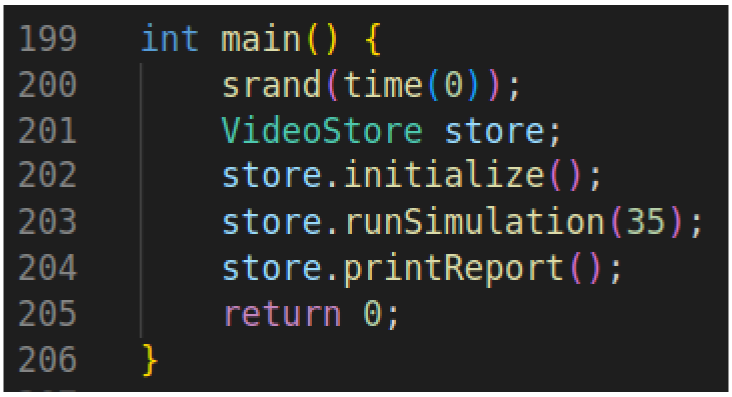
```cpp
void VideoStore::initialize() {
    for (int i = 0; i < 4; ++i) {/*  */
        allVideos.push_back(make_shared<Video>("NewRelease_" + to_string(i), Category::NewRelease, 5.0));
        allVideos.push_back(make_shared<Video>("Drama_" + to_string(i), Category::Drama, 3.0));
        allVideos.push_back(make_shared<Video>("Comedy_" + to_string(i), Category::Comedy, 2.0));
        allVideos.push_back(make_shared<Video>("Romance_" + to_string(i), Category::Romance, 3.0));
        allVideos.push_back(make_shared<Video>("Horror_" + to_string(i), Category::Horror, 4.0));
    }
    inventory = allVideos;

    customers.push_back(make_shared<BreezyCustomer>("Alice"));
    customers.push_back(make_shared<HoarderCustomer>("Bob"));
    customers.push_back(make_shared<RegularCustomer>("Carol"));
    customers.push_back(make_shared<BreezyCustomer>("Dave"));
    customers.push_back(make_shared<RegularCustomer>("Eve"));
    customers.push_back(make_shared<HoarderCustomer>("Frank"));
    customers.push_back(make_shared<RegularCustomer>("Grace"));
    customers.push_back(make_shared<BreezyCustomer>("Heidi"));
    customers.push_back(make_shared<HoarderCustomer>("Ivan"));
    customers.push_back(make_shared<RegularCustomer>("Judy"));
}
```

每天都會執行先還書，再讓顧客借書的步驟



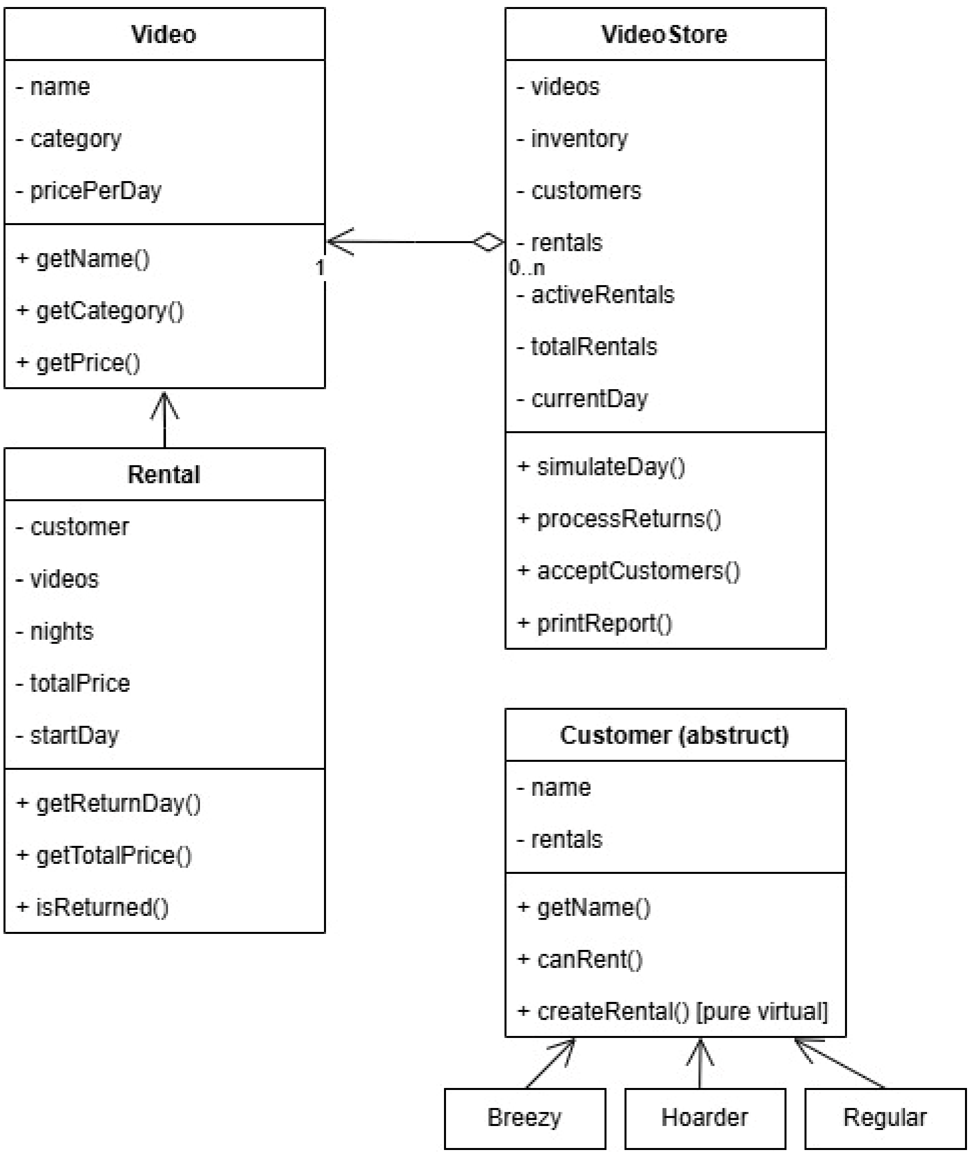
```cpp
void VideoStore::runSimulation(int days) {
    for (currentDay = 1; currentDay <= days; ++currentDay) {
        simulateDay();
    }
}
```

主函式



```cpp
int main() {
    srand(time(0));
    VideoStore store;
    store.initialize();
    store.runSimulation(35);
    store.printReport();
    return 0;
}
```

UML:


Video
- name
- category
- pricePerDay
+ getName()
+ getCategory()
+ getPrice()
VideoStore
- videos
- inventory
- customers
- rentals
- activeRentals
- totalRentals
- currentDay
+ simulateDay()
+ processReturns()
+ acceptCustomers()
+ printReport()
1
0..n
Rental
- customer
- videos
- nights
- totalPrice
- startDay
+ getReturnDay()
+ getTotalPrice()
+ isReturned()
Customer (abstruct)
- name
- rentals
+ getName()
+ canRent()
+ createRental() [pure virtual]
Breezy
Hoarder
Regular