UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
DISCIPLINA DE PROGRAMAÇÃO II

YASMIN L. SALES DA PAZ

PROGRAMAÇÃO II

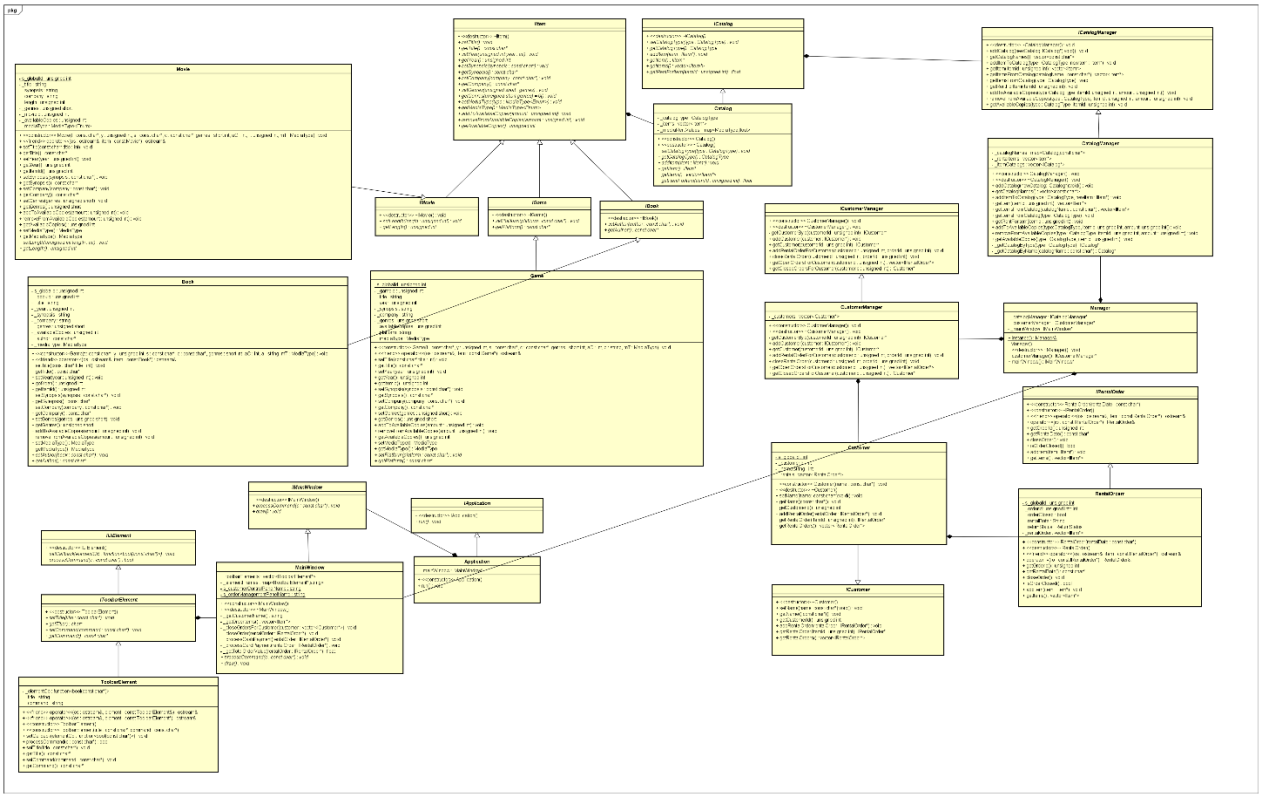
Documento de Requisitos

BELÉM-PA

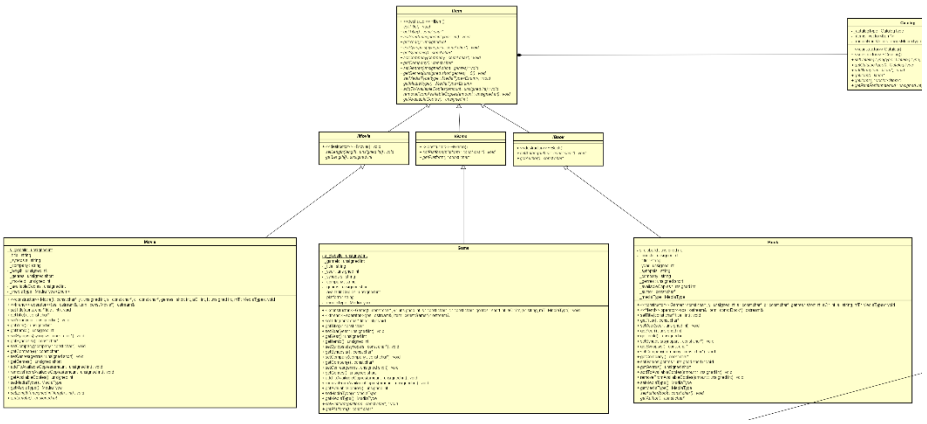
2016

1. Todas as classes concretas devem vir de interfaces ou classes abstratas. Pelo menos três hierarquias de classes. Uma das hierarquias deve ter três níveis.

As classes Movie, Game, Book, Catalog, CatalogManager, Customer, CustomerManager, RentalOrder, ToolbarElement, MainWindow e Application vêm de classes puramente abstratas. Com exceção de Manager, todas as classes concretas herdam de uma classe abstrata.



Hierarquia principal: IItem -> IMovie/IGame/IBook -> Movie/Game/Book



2. Em todas as classes: construtor de cópia, operatores<< e +=, e construtor default. Fazer o máximo de reaproveitamento de código usando static_cast

O operador << foi implementado apenas para classes de dados que continham informações relevantes para o uso do programa. O operador += foi implementado apenas para a classe que descreve um pedido de locação, sendo responsável por unificar a lista de itens dos dois pedidos e seus devidos valores. Dynamic casts foram usados para repassar ao operador << o tipo correto do objeto a ser impresso, evitando a necessidade de um código específico para impressão de cada tipo diferente de objeto.

Operatores <<:

```
std::ostream& operator<<(std::ostream& os, const Book* item)
{
   os << item->getTitle() << " by " << item->getAuthor() <<", " <<
       item->getCompany() << " (" << item->getYear() << ")  " << std::endl
       << "Synopsis: " << item->getSynopsis();
   return os;
}

std::ostream& operator<<(std::ostream& os, const Game* item)
{
   os << item->getTitle() << " (" << item->getYear() << "), " <<
       item->getCompany() << ", for " << item->getPlatform() << std::endl
       << "Synopsis: " << item->getSynopsis();
   return os;
}

std::ostream& operator<<(std::ostream& os, const Movie* item)
{
   os << item->getTitle() << " (" << item->getYear() << "), " <<
       item->getCompany() << ", " << item->getLength() << " min." << std::endl
       << "Synopsis: " << item->getSynopsis() << std::endl;
   return os;
}

std::ostream& operator<<(std::ostream& os, const ToolbarElement& element)
{
  return os << "[" << element.getCommand() << "] " << element.getTitle();
}

std::ostream& operator<<(std::ostream& os, const ToolbarElement* element)
{
  return os << "[" << element->getCommand() << "] " << element->getTitle();
}

std::ostream& operator<<(std::ostream& os, const RentalOrder* o)
{
  os << "Rental order date: " << o->getRentalDate() << std::endl
     << "Items rented:" << std::endl;

  std::vector<IItem*> items = o->getItems();
  for (unsigned int i = 0; i < items.size(); i++)
  {
    IItem* item = items.at(i);
    if (dynamic_cast<Movie*>(item) != nullptr)
    {
      os << std::endl
```

```
        << dynamic_cast<Movie*>(items.at(i)) << std::endl;
    }
    else if (dynamic_cast<Game*>(item) != nullptr)
    {
      os << dynamic_cast<Game*>(items.at(i)) << std::endl << std::endl;
    }
   else if (dynamic_cast<Book*>(item) != nullptr)
   {
       os << dynamic_cast<Book*>(items.at(i)) << std::endl << std::endl;
   }
  }
   return os;
}
```

Operador +=:

```
RentalOrder& RentalOrder::operator+=(const RentalOrder& ro)
{
   _rentalOrder = ro._rentalOrder;
   _orderId = s_globalId;
   s_globalId++;
   for (int i = 0; i < _rentalOrder.size(); i++)
   {
       _rentalOrder.push_back(ro._rentalOrder.at(i));
   }
   return *this;
}
```

3. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas.

- IItem -> IMovie/IGame/IBook -> Movie/Game/Book
- ICatalog-> Catalog
- IRentalOrder-> RentalOrder
- ICustomer-> Customer
- IApplication-> Application
- ICustomerManager-> CustomerManager
- ICatalogManager-> CatalogManager
- IUElement-> IToolbarElement-> ToolbarElement
- IMainwindow-> MainWindow

4. Atributos static e const static em todas as hierarquias de classe

Rental Order:

```
private:
    static unsigned int s_globalId;
```

```
    unsigned int _orderId;
    float _orderRent;
    bool _orderClosed;
    std::string _rentalDate;
    ReturnStatus _returnStatus;

    std::vector<IItem*> _rentalOrder;
```

Customer:

```
private:
  static unsigned int s_globalId;

  unsigned int _customerId;
  std::string _name;

  std::vector<IRentalOrder*> _rentals;
```

Book:

```
private:

    static unsigned int s_globalId;

    unsigned int _year;
    unsigned int _bookId;
    unsigned short _genres;
    unsigned short _pages;

    std::string _title;
    std::string _author;
    std::string _company;
    std::string _synopsis;

    unsigned int _availableCopies;


    MediaType _mediaType;
```

Game:

```
private:
    static unsigned int s_globalId;

    unsigned int _year;
    unsigned int _gameId;
    unsigned short _genres;

     std::string _title;
     std::string _platform;
     std::string _company;
     std::string _synopsis;

     unsigned int _availableCopies;

     MediaType _mediaType;
```

Movie

```
private:
    static unsigned int s_globalId;

    unsigned int _movieId;
    unsigned int _length;
     unsigned int _year;
     unsigned short _genres;

    std::string _title;
    std::string _synopsis;
    std::string _company;

    unsigned int _availableCopies;

    MediaType _mediaType;
```

5. Método static em todas as hierarquias de classe

O método estático retorna a instância global de Manager de modo a oferecer um ponto de acesso global ao objeto e garantir que apenas uma instância da classe seja criada.

```
class Manager
{
public:
  ~Manager();

  static Manager& instance()
  {
    static Manager mgr;
    return mgr;
  }

  ICatalogManager* catalogManager() const;

  ICustomerManager* customerManager() const;

  IMainWindow* mainWindow() const;

  Manager(const Manager& m) = delete;
  void operator=(const Manager& m) = delete;

private:
  Manager();

  ICatalogManager* _catalogManager;
  ICustomerManager* _customerManager;
  IMainWindow* _mainWindow;



};
```

6. Construtores em todas as classes, e três para todas as classes da hierarquia principal. Sempre validar os dados em todas as classes.

```
ToolbarElement::ToolbarElement()
{
  _title = "No title";
  _command = "";
  _elementCb = nullptr;

 }

Manager::Manager()
{
  _catalogManager = new CatalogManager();
  _customerManager = new CustomerManager();
  _mainWindow = new MainWindow();

}

CatalogManager::CatalogManager()
{
  _catalogNames.insert(std::pair<CatalogType, const char*>(CatalogType::Games,
"Games"));
  _catalogNames.insert(std::pair<CatalogType, const char*>(CatalogType::Movies,
"Movies"));
  _catalogNames.insert(std::pair<CatalogType, const char*>(CatalogType::Books,
"Books"));

}

CustomerManager::CustomerManager()
{


}

RentalOrder::RentalOrder(const char* rentalDate)
{
  assert(rentalDate && std::strlen(rentalDate) != 0);

    _orderId = s_globalId;

    _rentalDate = rentalDate;
    _returnStatus = Pending;
    _orderClosed = false;

    s_globalId++;

}

Application::Application()
{

}

Movie::Movie(const char* title, unsigned int year, const char* synopsis, const
char* company, unsigned short genres, unsigned int availableCopies, unsigned int
length, MediaType mediaType)
{
  assert(title);
  assert(synopsis);
```

```
      assert(company);

      _movieId = s_globalId;

      _year = year;
      _length = length;
      _genres = genres;

      _title = title;
      _synopsis = synopsis;
      _company = company;

      _availableCopies = availableCopies;
      _mediaType = mediaType;

      s_globalId++;
    }

Book::Book(const char* title, unsigned int year, const char* synopsis, const char*
company, unsigned short genres, unsigned int availableCopies, const char* author,
MediaType mediaType)
{
       assert(title);
       assert(synopsis);
       assert(company);
       assert(author);

       _bookId = s_globalId;

       _year = year;
       _genres = genres;

       _title = title;
       _author = author;
       _synopsis = synopsis;
       _company = company;

       _availableCopies = availableCopies;
       _mediaType = mediaType;

       s_globalId++;

}

Book::Book() = delete;
Book::Book(const Book& b) = delete;
void operator=(const Book& b) = delete;

Movie::Movie()                 = delete;
Movie::Movie(const Movie& m)   = delete;
void operator=(const Movie& m) = delete;

Game::Game() = delete;
Game::Game(const Game& g) = delete;
void operator=(const Game& g) = delete;

Catalog()                          = delete;
Catalog(const Catalog& c)          = delete;
void operator=(const Catalog& c) = delete;

RentalOrder() = delete;
RentalOrder(const RentalOrder& ro) = delete;
```

```
void operator=(const RentalOrder& ro) = delete;

Game::Game(const char* title, unsigned int year, const char* synopsis, const char*
company, unsigned short genres, unsigned int availableCopies, const char* platform,
MediaType mediaType)
{
	assert(title);
	assert(synopsis);
	assert(company);
	assert(platform);

	_gameId = s_globalId;

	_year = year;
	_genres = genres;

	_title = title;
	_platform = platform;
	_synopsis = synopsis;
	_company = company;

	_availableCopies = availableCopies;
	_mediaType = mediaType;

	s_globalId++;

}

Movie::Movie(const char* title, unsigned int year, const char* synopsis, const char*
company, unsigned short genres, unsigned int availableCopies, unsigned int length,
MediaType mediaType)
{
  assert(title);
  assert(synopsis);
  assert(company);

  _movieId = s_globalId;

  _year = year;
  _length = length;
  _genres = genres;

  _title = title;
  _synopsis = synopsis;
  _company = company;

  _availableCopies = availableCopies;
  _mediaType = mediaType;

  s_globalId++;
}

RentalOrder::RentalOrder(const char* rentalDate)
{
  assert(rentalDate && std::strlen(rentalDate) != 0);

	_orderId = s_globalId;

	_rentalDate = rentalDate;
	_returnStatus = Pending;
	_orderClosed = false;
```

```
        s_globalId++;

}


Customer::Customer(const char* name)
{
  assert(name && std::strlen(name) != 0);

  _name = name;
  _customerId = s_globalId;

  s_globalId++;
}

Catalog::Catalog(CatalogType type)
{
  _catalogType = type;

  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Game_Cartridge,
  1.5f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Game_DVD, 5.0f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Game_BluRay, 9.0f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Movie_VHS, 1.5f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Movie_DVD, 3.5f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Movie_BluRay, 5.0f));

  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Game_Cartridge,
  1.5f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Game_DVD, 5.0f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Game_BluRay, 9.0f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Movie_VHS, 1.5f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Movie_DVD, 3.5f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Movie_BluRay, 5.0f));

  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Book_Paperback,
  2.0f));
  _mediaRentValues.insert(std::pair<MediaType, float>(MediaType::Book_Hardcover,
  5.0f));


}

ToolbarElement::ToolbarElement(const char* title, const char* command)
{
  _title = title;
  _command = command;
  _elementCb = nullptr;
}
```

7. Vector em todas em todas as hierarquias de classe

Catalog: `std::vector<IItem*> _items;`
Customer: `std::vector<IRentalOrder*> _rentals;`
RentalOrder: `std::vector<IItem*> _rentalOrder;`
`std::vector<IItem*> _getOrderItems();`
MainWindow: `std::vector<IToolbarElement*> _toolbarElements;`
`std::vector<IItem*> _getOrderItems();`

8. Enum na hierarquia principal

```
enum CatalogType
{
  Movies,
  Games,
  Books
};

enum ReturnStatus
{
       Returned,
       Pending,
};


enum GenreType
{
  Action          = 1,
  Adventure       = 2,
  Documentary     = 4,
  Drama           = 8,
  Comedy          = 16,
  Romance         = 32,
  ScienceFiction = 64,
  Horror          = 128
};

enum MediaType
{
  Game_Cartridge,
  Game_DVD,
  Game_BluRay,
  Movie_VHS,
  Movie_DVD,
  Movie_BluRay,
  Book_Hardcover,
  Book_Paperback

};
```

9. Usar o **dynamic cast** e **typeid** no main junto com as classes concretas. Para uma da classe concreta identificada, chamar um método dessa classe e fazer uma ação;

```
std::ostream& operator<<(std::ostream& os, const RentalOrder* o)
{
  os << "Rental order date: " << o->getRentalDate() << std::endl
     << "Items rented:" << std::endl;

  std::vector<IItem*> items = o->getItems();
  for (unsigned int i = 0; i < items.size(); i++)
  {
    IItem* item = items.at(i);
    if (dynamic_cast<Movie*>(item) != nullptr)
    {
      os << std::endl
         << dynamic_cast<Movie*>(items.at(i)) << std::endl;
    }
```

```
          else if (dynamic_cast<Game*>(item) != nullptr)
          {
            os << dynamic_cast<Game*>(items.at(i)) << std::endl << std::endl;
          }
        else if (dynamic_cast<Book*>(item) != nullptr)
        {
              os << dynamic_cast<Book*>(items.at(i)) << std::endl << std::endl;
        }
         }
        return os;

IToolbarElement* checkHistory = new ToolbarElement("Check Customer History", "H");
  std::function<bool(const char*)> checkHistoryCb = [this](const char* c)
  {
    // Registered customer to retrieve the history.
    ICustomer* customer = nullptr;
    std::string customerName;

    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    while (true)
    {
      customerName = _getCustomerName();
      if (customerName.empty())
      {
        // User aborted.
        return true;
      }
      try
      {
        customer = Manager::instance().customerManager()-
>getCustomer(customerName.c_str());
      }
      catch (...)
      {
        std::cin.clear();
        std::cout << std::endl
                  << "Invalid user name." << std::endl;
        continue;
      }

      break;
    }

    std::cout << std::endl
                    << FRAME_SEPARATOR "Open rental orders for customer " <<
       customer->getName() << "." << std::endl;
    std::vector<IRentalOrder*> openOrders = Manager::instance().customerManager()-
    >getOpenOrdersForCustomer(customer->getCustomerId());
    if (openOrders.size() > 0)
    {
      for (unsigned int i = 0; i < openOrders.size(); i++)
      {
        std::cout << std::endl
                        << "[" << i + 1 << "] " << dynamic_cast<const
       RentalOrder*>(openOrders.at(i)) << std::endl;
          }
          }
          else
          {
            std::cout << std::endl
                      << "  This customer has no open rental orders." << std::endl;
          }
```

```
    std::cout << std::endl
                  << FRAME_SEPARATOR "Closed rental orders for customer " <<
customer->getName() << "." << std::endl;
        std::vector<const IRentalOrder*> closedOrders =
Manager::instance().customerManager()->getClosedOrdersForCustomer(customer-
>getCustomerId());
        if (closedOrders.size() > 0)
        {
          for (unsigned int i = 0; i < closedOrders.size(); i++)
          {
            std::cout << std::endl
                      << "[" << i + 1 << "] " << dynamic_cast<const
RentalOrder*>(closedOrders.at(i)) << std::endl;
          }
        }
        else
        {
          std::cout << std::endl
                    << "  This customer has no closed orders." << std::endl;
        }

        std::cout << std::endl
                  << "Press any key to continue...";
        _getch();

        return false;
      };
      checkHistory->setCallback(checkHistoryCb);
      _toolbarElements.push_back(checkHistory);
      _elementFrames.insert(std::pair<const IToolbarElement*,
    std::string>(checkHistory, s_customerControlPanelName));
```

10. Usar o rand. O usuário deve fazer entrada via teclado e interagir com a aplicação.

Gera números aleatórios para determinar o sucesso ou fracasso de uma transação de pagamento em cartão. A probabilidade de ocorrer falha no processamento do cartão de crédito/débito é de 25%.

```
std::random_device rd;
std::mt19937 eng(rd());
std::uniform_int_distribution<> distr(0, 100);
int checkoutProb = distr(eng);

// 75% chances of success.
if (checkoutProb <= 75)
{
  order->closeOrder();

  std::cout << std::endl << std::endl
    << "Payment complete." << std::endl << std::endl
    << "Press any key to continue...";
  _getch();
}
else
{
  std::cout << std::endl << std::endl
```

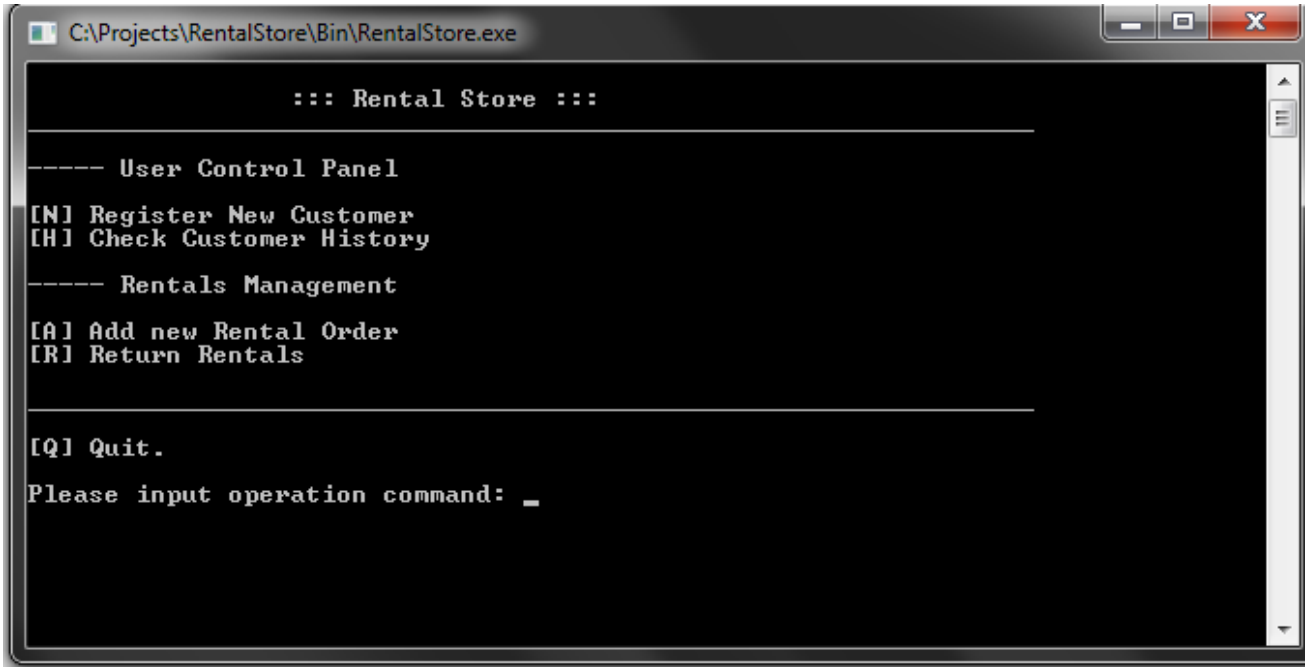
```
            << "Error: Transaction declined by bank. Please try another card." <<
        std::endl << std::endl
            << "Press any key to continue...";
          _getch();
        }
```

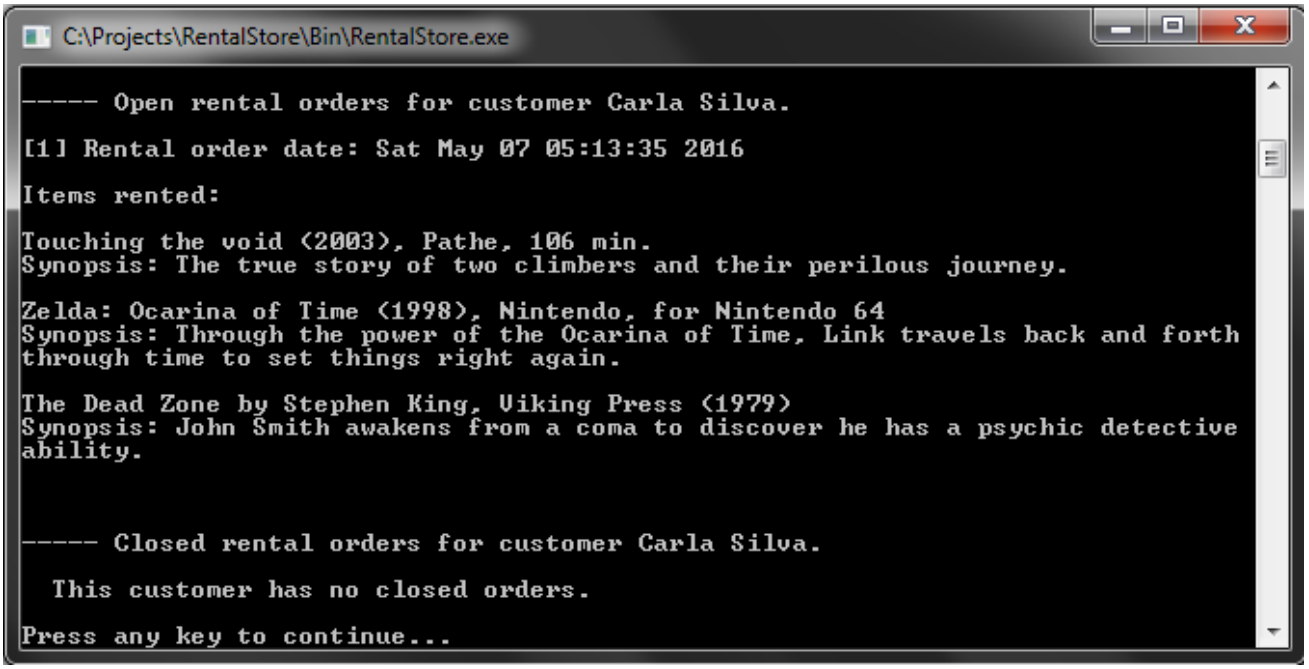
11. Na main o usuário deve fazer entrada via teclado e interagir com a aplicação.

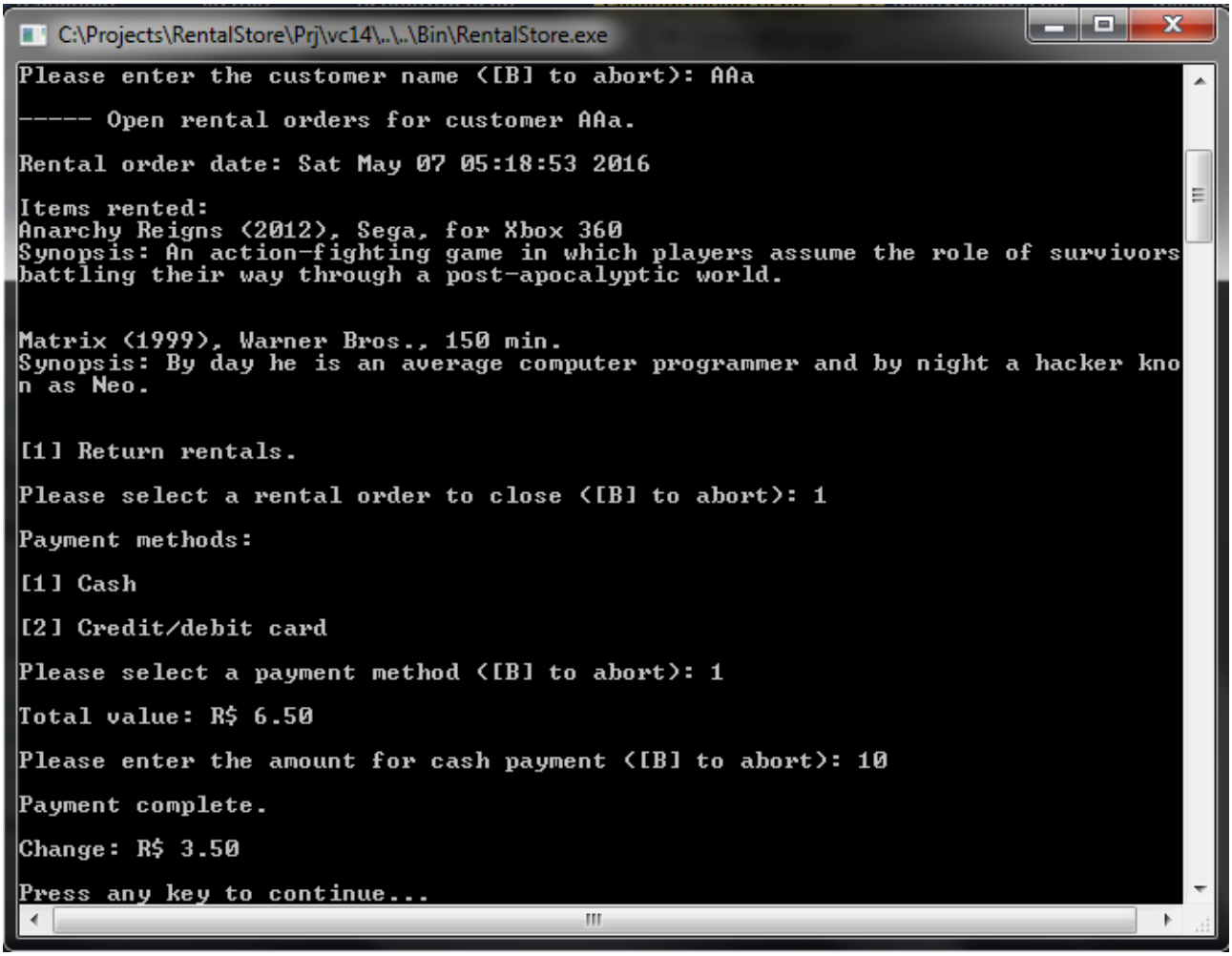
A classe Application é responsável por receber comandos do teclado e redirecioná-los para seus respectivos alvos. Neste código, a main chama o método run em application que recebe entrada via teclado.

```
int main()
{
  Catalog* c1 = new Catalog(CatalogType::Movies);
  c1->addItem(new Movie("Alien", 1979, "In the near future, a commercial spaceship
intercepts a distress signal.", "20th Century Fox", GenreType::Horror |
GenreType::ScienceFiction, 1, 120, MediaType::Movie_BluRay));

  c1->addItem(new Movie("Matrix", 1999, "By day he is an average computer programmer
and by night a hacker known as Neo.", "Warner Bros.", GenreType::Action |
GenreType::ScienceFiction, 3, 150, MediaType::Movie_BluRay));

  c1->addItem(new Movie("A Little Princess", 1992, "Sara is treated as a princess
until, one day, word comes of her father's tragic death.", "Warner Bros.",
GenreType::Drama, 1, 100, MediaType::Movie_VHS));

  c1->addItem(new Movie("Monty Python and the Holy Grail", 1975, "King Arthur and his
knights embark on a search for the Grail.", "EMI Films", GenreType::Comedy, 1, 96,
MediaType::Movie_VHS));

  c1->addItem(new Movie("The Room", 2005, "Can you ever really trust anyone?", "Chloe
Productions", GenreType::Drama, 1, 99, MediaType::Movie_DVD));

  c1->addItem(new Movie("Touching the void", 2003, "The true story of two climbers and
their perilous journey.", "Pathe", GenreType::Documentary, 2, 106,
MediaType::Movie_BluRay));

  c1->addItem(new Movie("Star Wars: The Empire Strikes Back", 1980, "Luke takes Jedi
training with Yoda, while his friends are pursued by Darth Vader.", "20th Century
Fox", GenreType::Adventure | GenreType::ScienceFiction, 6, 124,
MediaType::Movie_BluRay));

c1->addItem(new Movie("The Revenant", 2015, "A frontiersman on a fur trading
expedition in the 1820s fights for survival after being mauled by a bear and left for
dead by members of his own hunting team.", "20th Century Fox", GenreType::Adventure |
GenreType::Drama, 10, 156, MediaType::Movie_BluRay));

  Catalog* c2 = new Catalog(CatalogType::Games);

c2->addItem(new Game("Resident Evil 4", 2005, "Leon S. Kennedy, now a federal agent,
is hired to rescue the president's daughter from a sinister cult.", "Capcom",
GenreType::Action | GenreType::Horror, 5, "PlayStation 2", MediaType::Game_DVD));

c2->addItem(new Game("Zelda: Ocarina of Time", 1998, "Through the power of the Ocarina
of Time, Link travels back and forth through time to set things right again.",
"Nintendo", GenreType::Adventure, 3, "Nintendo 64", MediaType::Game_Cartridge));

c2->addItem(new Game("Pokken Tournament", 2016, "A fighting game starring Pokemon. ",
"Bandai Namco", GenreType::Action, 2, "Nintendo WiiU", MediaType::Game_BluRay));
```

```
c2->addItem(new Game("Anarchy Reigns", 2012, "An action-fighting game in which players assume the role of survivors battling their way through a post-apocalyptic world.", "Sega", GenreType::Action, 1, "Xbox 360", MediaType::Game_DVD));

c2->addItem(new Game("Heavy Rain", 2010, "A city on the US east coast is being terrorized by the 'Origami Killer', whose victims are all discovered drowned.", "Sony", GenreType::Drama | GenreType::Horror, 1, "PlayStation 4", MediaType::Game_BluRay));

  Catalog* c3 = new Catalog(CatalogType::Books);

  c3->addItem(new Book("The Dead Zone", 1979, "John Smith awakens from a coma to discover he has a psychic detective ability.", "Viking Press", GenreType::Horror, 4, "Stephen King", MediaType::Book_Hardcover));


  ICatalogManager* cmgr = Manager::instance().catalogManager();
  cmgr->addCatalog(c1);
  cmgr->addCatalog(c2);
  cmgr->addCatalog(c3);

  Application app;
  app.run();

  return 0;
}

void Application::run()
{
  while (true)
  {

    system("cls");

    Manager::instance().mainWindow()->draw();

    std::cout << "[Q] Quit." << std::endl << std::endl;
    std::cout << "Please input operation command: ";
    std::string command;
    std::cin >> command;

    if (strcmp("Q", static_cast<const char*>(command.c_str())) == 0)
    {
      return;
    }

    Manager::instance().mainWindow()->processCommand(static_cast<const char*>(command.c_str()));

}
```


C:\Projects\RentalStore\Bin\RentalStore.exe
::: Rental Store :::
----- User Control Panel
[N] Register New Customer
[H] Check Customer History
----- Rentals Management
[A] Add new Rental Order
[R] Return Rentals
[Q] Quit.
Please input operation command:


C:\Projects\RentalStore\Bin\RentalStore.exe
----- Open rental orders for customer Carla Silva.
[1] Rental order date: Sat May 07 05:13:35 2016
Items rented:
Touching the void (2003), Pathe, 106 min.
Synopsis: The true story of two climbers and their perilous journey.
Zelda: Ocarina of Time (1998), Nintendo, for Nintendo 64
Synopsis: Through the power of the Ocarina of Time, Link travels back and forth through time to set things right again.
The Dead Zone by Stephen King, Viking Press (1979)
Synopsis: John Smith awakens from a coma to discover he has a psychic detective ability.
----- Closed rental orders for customer Carla Silva.
This customer has no closed orders.
Press any key to continue...


C:\Projects\RentalStore\Prj\vc14\..\..\Bin\RentalStore.exe
Please enter the customer name ([B] to abort): AAa
----- Open rental orders for customer AAa.
Rental order date: Sat May 07 05:18:53 2016
Items rented:
Anarchy Reigns (2012), Sega, for Xbox 360
Synopsis: An action-fighting game in which players assume the role of survivors battling their way through a post-apocalyptic world.
Matrix (1999), Warner Bros., 150 min.
Synopsis: By day he is an average computer programmer and by night a hacker kno n as Neo.
[1] Return rentals.
Please select a rental order to close ([B] to abort): 1
Payment methods:
[1] Cash
[2] Credit/debit card
Please select a payment method ([B] to abort): 1
Total value: R$ 6.50
Please enter the amount for cash payment ([B] to abort): 10
Payment complete.
Change: R$ 3.50
Press any key to continue...