Importante: Para cada item abaixo deve ser copiado trechos do código que cumprem o requisito e explicado, se não for aparente, o porquê o requisito é cumprido. Sejam bem explícitos. Deve ser indicado também o arquivo da classe em que está o trecho do código. Eu avaliarei o código do Github a partir desse documento para confirmá-lo e também para detectar possíveis erros. **Quem não seguir o que está indicado aqui, não terá o projeto avaliado e perderá a atividade.**

Criar repositórios desde o início do desenvolvimento. Primeiro commit deve estar com todas as classes declaradas sem implementação.

Primeira versão do código completa dia 01 de dezembro.

Versão final do código dia 08 de dezembro.

No dia da apresentação deve ser mostrado o código rodando para vários casos de teste.

Será avaliado o quanto o projeto está elaborado. Trabalhos muito simples com métodos apenas mostrando mensagens, serão penalizados.

Requisitos de implementação para C++

1. Todas as classes concretas devem vir de classes abstratas. Pelo menos três hierarquias de classes. Uma das hierarquias deve ter três níveis. Exemplo: Personagem (abstract) >> Ciborgue (Abstract) >> Robocop; Class Arma (Abstract) >> Beretta93R
2. Em todas as classes: **construtor de cópia**, **operatores<< e +=,** e **construtor default**. Fazer o máximo de reaproveitamento de código usando **static_cast**
3. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas: Exemplo Servico >> ServicoStream >> (Netflix, HBOStream, AmazonPrime, NowTv). Em uma pesquisa de 10 segundos: http://www.tomsguide.com/us/pictures-story/620-top-online-streaming-video.html
4. Atributos static e const static em todas as hierarquias de classe
5. Método static em todas as hierarquias de classe
6. Construtores em todas as classes, e três para todas as classes da hierarquia principal. Sempre validar os dados em todas as classes
7. Vector em todas em todas as hierarquias de classe

8. ENUM na hierarquia principal

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;

10. Usar o rand. Nota: deve ser usado conforme o contexto do projeto. Se for usado em um método genérico sem relação com a classe e apenas para cumpri-lo, esse requisito será desconsiderado. http://en.cppreference.com/w/cpp/numeric/random/rand

11. No main o usuário deve fazer entrada via teclado e interagir com a aplicação

Requisitos de implementação para Java

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. Exemplo: Personagem >> Ciborgue >> Robocop; Class Arma

```
public class Produto {

public abstract class Disco extends Produto {

public class DiscoCompacto extends Disco {
```

(interface) >> Beretta93R

Produto é a classe abstrata,na qual a classe Disco herda e consequentemente a classe DiscoCompacto herda também(Arquivos Produto.java,Disco.java e DiscoCompacto.java)

2. Ao menos três interfaces. A terceira interface deve ser uma interface que liga duas hierarquias como no exemplo da interface **corredor** (Figura 1).

Interface Autenticavel,Interface Comprar e Interface Menu,a Interface Menu e a Interface Autenticavel,são implementadas em Usuario,Cliente,Moderador e Administrador(Arquivos Autenticavel.java,Interface.java,Comprar.java,Menu.java)

```
public interface Autenticavel {      public interface Comprar {
                    public interface Menu {

public class Moderador extends Usuario implements Menu, Autenticavel {

public class Administrador extends Usuario implements Menu, Autenticavel {

public class Cliente extends Usuario implements Menu, Comprar, Autenticavel {
```

3. Usar a interface **Comparable** e sobrescrever o método **compareTo** em pelo menos uma hierarquia(Não feita)

4. Sobrescrever **equals** para de Object

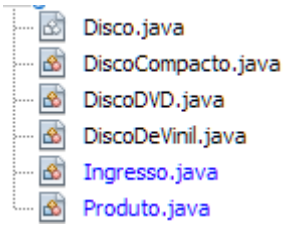
Arquivo Cliente.java

```
public boolean procurarbanda(ArrayList<Banda> listabandas) {
    String aux;
    aux = JOptionPane.showInputDialog(null, "Insira a banda que deseja procurar ");
    for (Banda banda : listabandas) {
        if (Banda.getNome().equalsIgnoreCase(aux)) {
            return true;
```

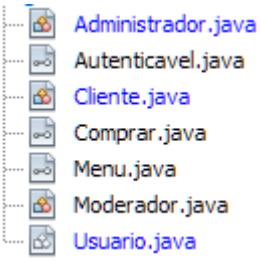
5. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas: Exemplo Servico >> ServicoStream >> (Netflix, HBOStream, AmazonPrime, NowTv). Em uma pesquisa de 10 segundos: http://www.tomsguide.com/us/pictures-story/620-top-online-streaming-video.html

Produto<--Disco<--DiscoCompacto,DiscoDVD,DiscoVinil

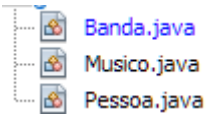
Produto<--Ingresso



Usuario<--Administrador,Cliente,Moderador



Pessoa<--Musico,Banda



6. Sempre usar o super para o máximo de reaproveitamento de código em todas as classes

Classe Administrador(Arquivo Administrador.java)

Puxando ambas funções da classe usuario

```
@Override
public void imprimirlistabandas() {
    super.imprimirlistabandas();
}

@Override
public void imprimirlistashows() {
    super.imprimirlistashows();

}
```

Classe Cliente(Arquivo Cliente.java)

Puxando construtor da classe usuario

```
public Cliente(locaisdevenda venda, int senha) {
    super(senha);
    this.venda = venda;
}
```

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

Classe Moderador(Arquivo Moderador.java)

```
private static final int SENHAFINAL = 4500;
```

Classe Ingresso(Arquivo Ingresso.java)

```
private static final int IDVALIDACAO;
```

8. Método static em todas as hierarquias de classe

Metodos em classes Usuario,Administrador

```
public static void adicionarshow(Show show,Ingresso ingresso) {
    show.setAtributos(ingresso);
    listashows.add(show);

    }
public static void adicionarbanda(Banda nova) {
    nova.setAtributos();
    listabandas.add(nova);
    }
```

Metodos de validação na classe Banda(Arquivo Banda.java)

```java
public static boolean validar_nome2(String arg3) {
    return arg3.matches("[a-zA-Z]{1,10}(\\s)[a-zA-Z]{1,10}");
}

public void imprimirdados() {
    JOptionPane.showMessageDialog(null, " " + this.toString());

}
```

9. Três construtores para todas as classes, sendo um deles o construtor de cópia. Sempre validar os dados em todas as classes

Classe Banda(Arquivo Banda.java)

```java
public Banda() {
    this.integrantes = 0;
    this.quantidadedediscos = 0;
    this.genero = "";
    this.gravadora = "";
    Banda.nome = "";
}

public Banda(int integrantes, int quantidadedediscos, String genero, String gravadora, String nome) {
    this.integrantes = integrantes;
    this.quantidadedediscos = quantidadedediscos;
    this.genero = genero;
    this.gravadora = gravadora;
    Banda.nome = nome;
}

public Banda(Banda copia) {
    this.integrantes = copia.integrantes;
    this.quantidadedediscos = copia.quantidadedediscos;
    this.genero = copia.genero;
    this.gravadora = copia.gravadora;
    Banda.nome = copia.nome;
}
```

Classe Show(Arquivo Show.java)

```java
public Show() {
    this.endereco = "";
    this.nome = "";
    this.precoingresso = 0.00;
    this.localparaestacionar = "";
    this.tamanhoestacionamento = 0;
    this.capacidade = 0;
    this.venda = venda;
    this.pagamento = pagamento;
}

public Show(String endereco, String nome, String localparaestacionar, double precoingresso, double
    this.endereco = endereco;
    this.nome = nome;
    this.localparaestacionar = localparaestacionar;
    this.precoingresso = precoingresso;
    this.capacidade = capacidade;
    this.tamanhoestacionamento = tamanhoestacionamento;
    this.venda = venda;
    this.pagamento = pagamento;
}

public Show(Show copia) {
    this.endereco = copia.endereco;
    this.nome = copia.nome;
    this.precoingresso = copia.precoingresso;
    this.localparaestacionar = copia.localparaestacionar;
    this.tamanhoestacionamento = copia.tamanhoestacionamento;
    this.capacidade = copia.capacidade;
    this.venda = copia.venda;
    this.pagamento = copia.pagamento;
```

Classe Ingresso(Arquivo Ingresso.java)

```
public Ingresso() {
    this.data = "";
    this.IDVALIDACAO = 6829598;
}

public Ingresso(String data, int IDVALIDACAO) {
    this.data = data;
    this.IDVALIDACAO = IDVALIDACAO;
}
public Ingresso(Ingresso copia){
    this.data = copia.data;
    this.nome = copia.nome;
    this.IDVALIDACAO = 6829598;
}
```

Classe Administrador(Arquivo Administrador.java)

```
public Administrador() {

}

public Administrador(Administrador copia) {
    super(copia);
    this.opcao = copia.opcao;
}
public Administrador(int opcao, int senha) {
    super(senha);
    this.opcao = 0;
}
```

Classe Cliente(Arquivo Cliente.java)

```
public Cliente() {
    this.senha = 0;
}

public Cliente(locaisdevenda venda, int senha) {
    super(senha);
    this.venda = venda;
}

public Cliente(Cliente copia) {
    this.senha = copia.senha;
    this.venda = copia.venda;
}
```

Classe Usuario(Arquivo Usuario.java)

```
public Usuario() {

}

public Usuario(int senha) {
   this.senha = 0;
}
public Usuario(Usuario copia){
    this.senha = copia.senha;
}
```

10. ArrayList na hierarquia principal

ArrayList em Usuario(Arquivo Usuario.java)

```
protected int senha;
public static ArrayList<Banda> listabandas = new ArrayList<>();
public static ArrayList<Show> listashows = new ArrayList<>();
public static ArrayList<Cliente> listacliente = new ArrayList<>();
```

11. ENUM na hierarquia principal

ENUM na classe Cliente(Arquivo Cliente.java)

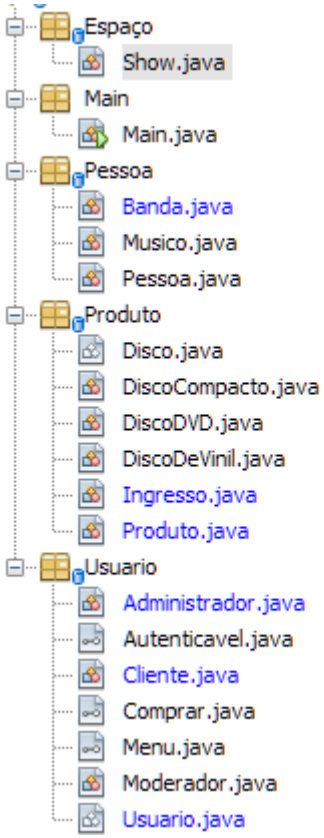
```
private enum formasdepagamento {

    MASTERCARD, AMEX, VISA, DINHEIRO, PAYPAL
}
```

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

Classe Main(Arquivo Main.java)

```
case 0:
    break;
case 1:
    if( adm instanceof Administrador ){adm.menu(adm,banda,show,ingresso);}
    break;
case 2:
    if( moderador instanceof Moderador ){moderador.menu(moderador,cliente);}
    break;
case 3:
    if( cliente instanceof Cliente){cliente.menu(cliente,show,adm,banda);}
    break;
```

13. Dividir o projeto em pacotes



14. Sobrescrever para todas as classes o método toString

Classe Show

```
@Override
public String toString() {
    return "Show\n " + "Endereco = " + endereco + "\n Local = " + nome + "\n Possui estacionamento?
}
}
```

Classe Banda

```
@Override
public String toString() {
    return "Banda\n" + "Nome = " + nome + ",\n Quantida de Discos = " + quantidadedediscos + ",\n Ge
}
```

15. Usar a classe Random do pacote java.util (java.util.Random).
Nota: deve ser usado conforme o contexto do projeto. Se for usado em um método genérico sem relação com a classe e apenas para cumpri-lo, esse requisito será desconsiderado.

Classe Main(Arquivo Main.java)

Gera um numero aleatorio para servir de "versão" do aplicativo

```
Random gerador = new Random();
versao = gerador.nextInt(10);
JOptionPane.showMessageDialog(null, "Bem vindo ao Easy Ticket! (Versão " + versao + " )");
```

16. No main o usuário deve fazer entrada via teclado e interagir com a aplicação. Opcional de bônus: pode ser usada a classe JOptionPane do pacote javax.swing. Vejam: showInputDialog e showMessageDialog.

```
public static void main(String[] args) {
    int opcao = 0;
    Cliente cliente = new Cliente();
    Administrador adm = new Administrador();
    Moderador moderador = new Moderador();
    Show show = new Show();
    Banda banda = new Banda();
    Ingresso ingresso = new Ingresso();
    JOptionPane.showMessageDialog(null, "Bem vindo ao Easy Ticket!");
    do {
        opcao = Integer.valueOf(JOptionPane.showInputDialog(null, "0 - Sair\n1 - Area de administracao\
        switch (opcao) {
            case 0:
                break;
            case 1:
                adm.menu(adm,banda,show,ingresso);
                break;
            case 2:
                moderador.menu(moderador,cliente);
                break;
            case 3:
                cliente.menu(cliente,show,adm,banda);
                break;
        }
    } while (opcao != 0);
    JOptionPane.showMessageDialog(null, "Obrigado, volte sempre!");
}
```

Arquivo Main.java

Bonus JoptionPane

Pane Input Dialog e Message Dialog

```
JOptionPane.showMessageDialog(null, "Bem vindo ao Easy Ticket!");
do {
    opcao = Integer.valueOf(JOptionPane.showInputDialog(null, "0 -
    switch (opcao) {
```

Pane Selection

```
public void compraringresso(Show show, Administrador adm) {
    String aux;
    Object[] options = {"Comprar", "Cancelar"};
    JOptionPane.showOptionDialog(null, "Deseja comprar ingresso para este show?Custa " + show.precoingr
            JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,
            null, options, options[0]);
    adm.imprimirlistashows();
    Object[] possibleValues = {"VISA", "AMEX", "PAYPAL", "DINHEIRO", "MASTERCARD"};
    Object selectedValue = JOptionPane.showInputDialog(null,
            "Escolha uma forma de pagamento", "Input",
            JOptionPane.INFORMATION_MESSAGE, null,
            possibleValues, possibleValues[0]);
    JOptionPane.showMessageDialog(null, "Obrigado pela sua compra!");
}
```

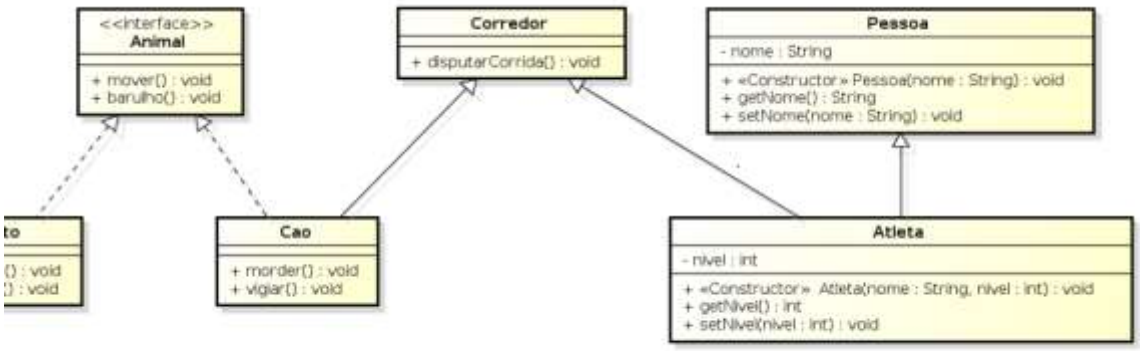


Figura 1 – Interface Corredor conectando duas hierarquias de classe