***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.**

**Usar um novo repositório!**

**Entrega final: 26 de novembro via GitHub. Prova oral sobre o código ainda a ser definida.**

**Entrega parcial: 19 de novembro via GitHub. Apenas Diagrama de Classes, interfaces e classes abstratas no branch general.**

**Requisitos Gerais**

***Todos os atributos e funções membros devem estar relacionados a classe***

* Pelo menos 4 atributos

Atributos da classe Computador:

protected static int numComputadoresLigados = 0;

protected int HD;

protected int RAM;

protected float processadorGHz;

protected String processadorModelo;

protected boolean OnOff;

protected Programa[] programasPadroesInstalados = new Programa[3];

* Pelo menos 4 funções membros sem incluir get e set

Funções da classe computador:

public boolean ligar() {

if (!isOn()) {

System.out.println("Ligando o computador.");

this.OnOff = true;

Computador.numComputadoresLigados++;

System.out.println("Iniciando SO.");

this.iniciaSO();

return true;

} else {

System.out.println("O seu computador ja esta ligado.");

return false;

}

}

public boolean desligar() {

if (isOn()) {

System.out.println("Finalizando SO.");

System.out.println("Desligando o computador.");

this.OnOff = false;

Computador.numComputadoresLigados--;

return true;

} else {

System.out.println("Voce nao precisa desligar um computador que ja esta desligado.");

return false;

}

}

public void reiniciar() {

if (isOn()) {

this.desligar();

this.ligar();

} else {

System.out.println("Nao faz sentido reiniciar um computador que esta desligado.");

}

}

public static void mostrarComputadores(ArrayList<Computador> computadores){

int i = 1;

for (Computador computador : computadores) {

System.out.println(i + " - " + computador + ((computador.isOn()) ? " Ligado" : " Desligado"));

i++;

}

}

* Diagrama UML completo (obrigatório salvar também o png do diagrama no gitHub)

**Requisitos de implementação**

* 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 (interface) >> Beretta93R
* Ao menos três interfaces. A terceira interface deve ser uma interface que liga duas hierarquias como no exemplo da interface **corredor** (Figura 1).

Interfaces: Device, Jogavel, Periferico e Programa.

* Usar a interface **Comparable** e sobrescrever o método **compareTo** em pelo menos uma hierarquia

Interface Comparable implementada em Arquivo, metodo compareTo sobrescrito:

@Override

public int compareTo(Arquivo o) {

return this.nome.compareTo(o.nome);

}

* Sobrescrever **equals** para de Object

equals sobrescrito em Aplicativo:

@Override

public boolean equals(Object obj) {

if (obj == null) {

return false;

}

if (getClass() != obj.getClass()) {

return false;

}

final Aplicativo other = (Aplicativo) obj;

if (!Objects.equals(this.nomeAplicativo, other.nomeAplicativo)) {

return false;

}

return true;

}

* 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>
* Sempre usar o super para o máximo de reaproveitamento de código

super utilizado em todos os construtores, nas classes ligar e desligar de PC, MAC e SmartPhone.

Exemplo da implementação do super na classe PC:

@Override

public boolean ligar() {

boolean ligou = super.ligar();

if (ligou) {

System.out.println("Sistema " + this + " carregado com sucesso.");

} else {

System.out.println("Houve um problema na inicialização do seu Sistema Operacional.");

}

return ligou;

}

@Override

public boolean desligar() {

boolean desligou = super.desligar();

if (desligou) {

System.out.println("Seu Sistema Operacional " + this + " foi finalizado com maestria.");

System.out.println("Seu computador foi desligado com sucesso.");

} else {

System.out.println("Ocorreu algum erro durante o desligamento de seu computador.");

}

return desligou;

}

* Atributos static e const static

Atributo static de Computador:

protected static int numComputadoresLigados = 0;

Ele é incrementado na função ligar e decrementado na função desligar, ambas da classe computador:

@Override

public boolean ligar() {

if (!isOn()) {

System.out.println("Ligando o " + this + ".");

this.OnOff = true;

Computador.numComputadoresLigados++;

System.out.println("Iniciando SO.");

this.iniciaSO();

return true;

} else {

System.out.println("O seu " + this + " ja esta ligado.");

return false;

}

}

@Override

public boolean desligar() {

if (isOn()) {

System.out.println("Finalizando SO.");

System.out.println("Desligando o computador.");

this.OnOff = false;

Computador.numComputadoresLigados--;

return true;

} else {

System.out.println("Voce nao precisa desligar um computador que ja esta desligado.");

return false;

}

}

Atributo const(final) static :

public static final String SISTEMA\_OPERACIONAL = "Android";

Ele é utilizado na função toString de SmartPhone:

@Override

public String toString() {

return SmartPhone.SISTEMA\_OPERACIONAL + " " + versaoAndroid.toString();

}

* Método static

Metodo static para login em dropbox:

public static boolean login(Dropbox drop, String usuario, String senha) {

if (drop.usuario.equals(usuario) && drop.senha.equals(senha)) {

System.out.println("Login Realizado com sucesso.");

return true;

} else {

System.out.println("Usuario ou senha errados.");

return false;

}

}

* Construtores em todas as classes, e dois na hierarquia principal. Sempre validar os dados em todas as classes

Construtores da classe computador:

public Computador() {

this.HD = 500;

this.RAM = 4;

this.processadorGHz = 1.9f;

this.processadorModelo = "Intel core I5-4460T";

this.instalarProgramasPadroes();

}

public Computador(int HD, int RAM, float processadorGHz, String processadorModelo) {

this.setHD(HD);

this.setRAM(RAM);

this.setProcessadorGHz(processadorGHz);

this.setProcessadorModelo(processadorModelo);

this.instalarProgramasPadroes();

}

Contrutores da classe PC:

public PC() {

this.sistemaOperacional = SO.UBUNTU;

this.monitor = new Monitor();

this.teclado = new Teclado();

}

public PC(SO sistemaOperacional, Monitor monitor, Teclado teclado) {

this.sistemaOperacional = sistemaOperacional;

this.monitor = monitor;

this.teclado = teclado;

}

public PC(SO sistemaOperacional, Monitor monitor, Teclado teclado, int HD, int RAM, float processadorGHz, String processadorModelo) {

super(HD, RAM, processadorGHz, processadorModelo);

this.sistemaOperacional = sistemaOperacional;

this.monitor = monitor;

this.teclado = teclado;

}

Construtores da classe SmartPhone:

public SmartPhone() {

this.versaoAndroid = android.IceCreamSandwich;

this.tamanhoDaTela = 5;

}

public SmartPhone(android versaoAndroid, float tamanhoDaTela, int HD, int RAM, float processadorGHz, String processadorModelo) {

super(HD, RAM, processadorGHz, processadorModelo);

this.versaoAndroid = versaoAndroid;

this.tamanhoDaTela = tamanhoDaTela;

}

public SmartPhone(SmartPhone sp) {

this.HD = sp.HD;

this.OnOff = sp.OnOff;

this.RAM = sp.RAM;

for (Aplicativo apkInstalado : sp.apkInstalados) {

this.apkInstalados.add(apkInstalado);

}

this.processadorGHz = sp.processadorGHz;

this.processadorModelo = sp.processadorModelo;

this.tamanhoDaTela = sp.tamanhoDaTela;

this.versaoAndroid = sp.versaoAndroid;

}

* Construtor cópia em uma das hierarquias

Contrutor de cópia em SmartPhone:

public SmartPhone(SmartPhone sp) {

this.HD = sp.HD;

this.OnOff = sp.OnOff;

this.RAM = sp.RAM;

for (Aplicativo apkInstalado : sp.apkInstalados) {

this.apkInstalados.add(apkInstalado);

}

this.processadorGHz = sp.processadorGHz;

this.processadorModelo = sp.processadorModelo;

this.tamanhoDaTela = sp.tamanhoDaTela;

this.versaoAndroid = sp.versaoAndroid;

}

* ArrayList

ArrayList em SmartPhone que armazena os aplicativos instalados no celuar.

protected ArrayList<Aplicativo> apkInstalados = new ArrayList<>();

* ENUM

Enum em PC que armazena os SO que poderao ser instalados no PC:

public enum SO {

UBUNTU, FEDORA, MINT, ZENWALK, WINDOWS\_XP, WINDOWS\_VISTA, WINDOWS\_7, WINDOWS\_8, WINDOWS\_8\_1

}

* 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;
* Dividir o projeto em pacotes

O projeto está dividido em pacotes com os seguintes pacotes:

computador.Computadores

computador.Generalizacoes

computador.Perifericos

computador.Programas

computador.Programas.Aplicativos

* Sobrescrever para todas as classes o método toString
* 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.
* 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.

Interação com o usuario no metodo main, ele pode ligar e desligar os computadores, e executar programas.

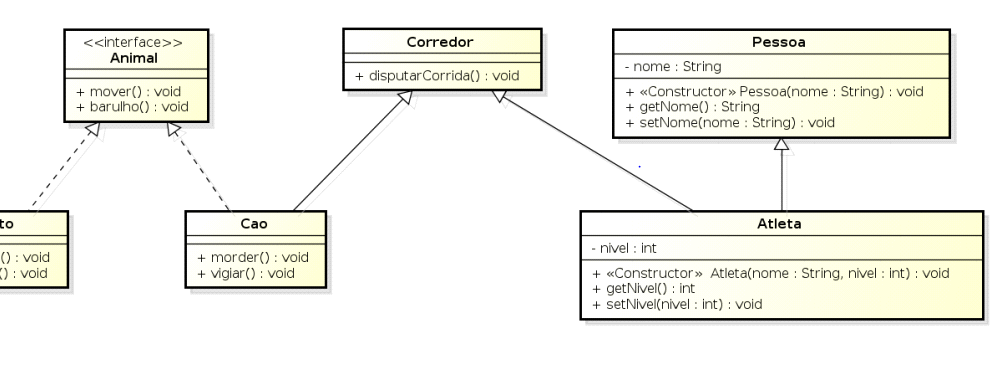


Figura 1 – Interface Corredor conectando duas hierarquias de classe