

Faculdade de Ciências Exatas e da Engenharia

Licenciatura em Engenharia Informática

2º Ano

Projeto 2:

“*App Store*”

Programação Orientada por Objetos



Alunos:

Érica Filipa Dias Cunha nº 2033617

Jocelyne Estefanía Pestana Teles nº 2036617

Daniel Orlando de Barros nº 2037717

Funchal, 29 de março de 2019

Indice

# 1 – Introdução

O presente relatório está orientado como meio de explicação do procedimento realizado para a composição do código requerido no segundo projeto relativo ao uso do *Java* como linguagem de programação. O ambiente de desenvolvimento utilizado para este trabalho é o *NetBeans.*

O objetivo é desenvolver uma aplicação que seja capaz de gerir uma “*App Store*”, possibilitando a introdução e a observação de informação que seja de interesse ao utilizador.

Nas partes seguintes do relatório explicar-se-á mais detalhadamente os objetivos específicos que são necessários para o funcionamento da *App Store*, a explicação de como foi feito o código e uma análise geral sobre o resultado final obtido.

# 2 – Descrição geral da App Store

## 2.1 – Componentes gerais da App Store:

* **Utilizadores:** São praticamente o elemento com maior atividade na *App Store,* pois são capazes de comprar *apps,* avaliá-las e comentar sobre elas (só depois de tê-las comprado). São caraterizados por um nome, a idade e um número de ID único.

Também estes têm a possibilidade de subscrever a algumas das aplicações pelo tempo de um ano com contrato renovável.

Existem dois tipos de utilizadores, os normais e os *premium*. Estes últimos só pagam 40% das *apps*. Os dois tipos de utilizadores têm direito a uma *app* gratuita por semana de entre as que possuem algum desconto.

Sempre que um utilizador angaria outro tem direito a 5% de desconto na próxima compra.

* **Programadores:** Outro elemento chave para o funcionamento da *App Store,* são os criadores das *apps,* pelo que cada uma delas está associada a um destes programadores. Os programadores são diferenciados por um número único (ID) e possuem uma lista de *apps* que foram desenvolvidas por eles, além de uma avaliação obtida dos utilizadores que compraram as *apps* da sua autoria.
* ***Apps*:** São os componentes da *App Store* que servem como elemento base para o funcionamento completo da mesma e são os que tem mais características e considerações como se seguem:
  + Podem ser compradas (ação que tem data de realização e preço). A App Store deve permitir compras simultâneas, ou seja, comprar uma lista de *apps* dando o valor total a pagar;
  + As cinco *apps* menos vendidas têm um desconto sobre os seus preços de 15% na semana seguinte;
  + Cada *app* está relacionada a uma categoria e uma delas tem 5% cada mês. As apps têm como categorias possíveis as seguintes:
    - *Games;*
    - *Business;*
    - *Education;*
    - *Lifestyle;*
    - *Entertainment;*
    - *Utilities;*
    - *Travel;*
    - *Health & Fitness*.

## 2.2 – Funcionalidades da App Store:

O que deve ser possível fazer na aplicação:

* Registar novos utilizadores, programadores (com os seus ID gerados automaticamente) e novas *apps* com as informações pertinentes.
* Gerar listas de todos os utilizadores mostrando a sua informação (nome, idade e ID).
* Comprar e subscrever às *apps*.
* Listar apps por categoria, classificação dos utilizadores (mostrando a sua média), nome (ordem alfabética) e por número de vendas. Assim como também, em cada app seja possível listar os comentários e as classificações (se houver).
* Fazer uma listagem de utilizadores segundo o valor em compras que realizaram.
* Imprimir o valor que a *App Store* ganhou com a venda total das suas aplicações, assim como o valor obtido por cada programador com as suas *apps* compradas pelos utilizadores.
* Sair da aplicação.
* Inserir informação através do teclado e de um ficheiro de texto. Assim como a possibilidade de criar um ficheiro texto com toda a informação da *App Store*.

# 3 – Desenvolvimento

## 3.1 – Classes:

### 3.1.1 – Main:

PODE MUDAR:

O *Main* é a classe principal que contém o método *main (),* onde é escrito o código que será executado no programa, não incluindo a criação de instâncias. É uma classe essencial para o programa funcionar e o primeiro a ser invocado pelo intérprete de *Java*.

Como atributo, esta classe contém o data1da classe *Data* que apresenta o dia, mês e ano atuais para a *App Store.*

O método que possui é o *main ()* (referido anteriormente), tendo nele o código que inicializa toda a app para poder ser vista na consola, assim como funcionalidades que escolhe o mês e o dia a apresentar na consola de forma visual. Também inicializa o output mostrando primeiro o menu principal.

Também, como fim de testar a aplicação, inserem-se apps para testar o funcionamento do programa através deste meio.

### 3.1.2 – AppStore:

É uma classe que contém a maior parte dos métodos, sendo nela possível implementar muitas das tarefas da aplicação.

Para esta classe foram usadas as *packages* de *java.util* para algumas utilidades como a classe Scanner, por exemplo, que é usada em muitos métodos que serão explicados a seguir, e a package *java.io*, para operações entre *input* e *output*. Utilizou-se a declaração de *import* para poder chamar a esta package e ser utilizada ao longo desta classe.

Como atributos tem-se vários *ArrayList* que contém objetos como listaUtilizadores da classe *Utilizador,* que contém todos os utilizadores que estão presentes na *App Store,* listaProgramadores da classe *Programador,* que também guarda todos os programadores disponíveis,e listaTodasApps da classe *App,* que lista todas as *apps* existentes até o momento. Também existem como atributo a lista de *Strings* listaCategorias, que contém as categorias mencionadas em pontos anteriores do relatório, e as listas de classe *Integer* listaIDsUtilizadores, que possui os números de identificação dos utilizadores existentes, listaIDsProgramadores, também contendo números de identificação dos programadores existentes.

Outros atributos são a data da classe *Data,* o random da classe *Random* (também pertencente a package mencionada anteriormente), a categoriaDoMes da classe *String* (sendo esta uma variável estática pois pode ser partilhada pelas instâncias da classe permitindo guardar informação sobre as mesmas) e o valorTotalAppStore que é uma variável do tipo *double*.

O construtor desta classe contém a inicialização de todas as listas anteriormente mencionadas. A listaCategorias é preenchida com palavras que representam cada categoria separadas por vírgulas e por meio do método *addAll*, criando uma lista de *Arrays*. As restantes listas são inicializadas como *ArrayList* vazios. As variáveis data e random são inicializadas como novas instanciações das suas respetivas classes, a categoriaDoMes começa como um “vazio” ou *null* e o valorTotalAppStore começa com o valor zero (0). Também possui uma inicialização de um ArrayList com duas opções: “Sim” ou “Não” (representados com as letras S e N), as quais confirmam a decisão do utilizador consoante a estas duas opções.

Como métodos declarados nesta classe temos os seguintes:

O método menuPrincipal () é um método que tem um retorno do tipo void e considera como exceção o IOException, que mostra por uma serie de Strings o menu principal (o que deve aparecer ao inicializar a App Store) que mostra diversas opções, tendo um visual como é mostrado no Anexo 6.1.

A seguir, utilizando um switch, coloca-se cada opção como um case e cada uma leva ao método que lhe corresponde a essa ação, como entrar aos submenus da App Store (menu Registo, menu Listagem e menu Montante Recebido), comprar, subscrever e avaliar aplicação, que são as ações que se podem realizar a partir de aqui. Outras ações são dirigidas para avançar de dia ou de ano (para observar câmbios nas listagem de apps como as cinco menos vendidas e a categoria do mês, assim como o passo de validez de uma subscrição).

Se não se confirmar nenhum dos cases anteriores, executa-se o método opcaoErrada () e a seguir volta a executar menuPrincipal ().

O método menuRegisto () mostra o menu que inicializa quando o utilizador escolheu a opção “1 – Registar” do menu principal. É um método com um retorno do tipo void e considera a exceção IOException. Este menu pode ser visto no Anexo 6.2.

Funciona de forma bastante semelhante ao menuPrincipal () com switch de 5 cases. Os três primeiros contêm métodos que criam novos programadores, novas apps e novos utilizadores. No quarto case permite-se carregar ficheiros de texto e no quinto voltar para o menu principal.

Caso que não se confirme uma destas opções, executa-se o método opcaoErrada () e volta a executar o menuRegisto ().

Para criar um novo programador utiliza-se o método novoProgramador () que tem como tipo de retorno void e um acesso do tipo private. Neste método gera-se um ID (um número aleatório) que representará a o novo programador e o introduz na listaIDsProgramadores o número de identificação novo e à listaProgramadores os dados do novo programador. No output aparecerá como resultado algo como o exemplo que se mostra no Anexo 6.3.

No fim deste procedimento invoca o método voltarMenuRegisto ().

Outro método que é responsável por criar novos elementos para a App Store é o método novaApp () de acesso private e tem como tipo de retorno o void. O método inclui um if-else. Se a lista de programadores estiver vazia mostra a mensagem “Ainda não existem programadores!”. Caso contrário utiliza-se um método da classe Scanner e se pede ao utilizador uma serie de informações como “Insira o nome da app: “, que recebe uma String que representa o nome, “Insira o preço: “, que é um valor do tipo double, “A lista de categorias disponíveis é: “ (mostra toda a listagem das categorias), “Insira a categoria: “, que espera uma resposta do tipo String e também utiliza um método para verificar se a opção escrita é uma que está contida na lista de categorias disponíveis. Depois mostra a lista de Ids dos programadores disponíveis e pede que “Insira o ID do programador: “, sendo um valor do tipo inteiro o que o programa espera receber do utilizador.

Também é incluído um booleano que serve para identificar as apps nas que o utilizador se pode subscrever. Este booleano começa sendo igual a false até que se detete um “Sim” (S) como uma indicação de que tal app pode ter subscrições. A indicação de que uma app é subscrevível ou não é por meio de Strings. No caso de que o booleano mude para true, a consola irá a mostrar a mensagem “É subscrevível? Sim”, mas se ainda manter o seu estado em false, será escrito a frase “É subscrevível? Não”. No final tem-se como resultado o que se mostra no exemplo de Anexo 6.4.

A nova app é criada e inserida na listaTodasApps. No fim, executa-se o método *voltarMenuRegisto ().*

Como possíveis erros detetados neste método, existe os do tipo IOException que é propagada (throws) pelo código e outra exceção que é tratada com um try/catch que é do tipo InputMismatchException para considerar o caso de introduzir letras em vez de números quando é pedido (por exemplo, quando é solicitado um número de identificação).

O método novoUtilizador () permite criar novos utilizadores a partir dos dados que são solicitados. É um método de acesso private e com um tipo de retorno void. Este método pede ao utilizador dados como o seu nome, idade e se quere ser utilizador premium (respondendo com um S (Sim) ou um N (Não)), sendo isto último verificado com o método verificarOpcao (). Através de um booleano, define-se a caraterística premium com um false desde o início até que se confirme um “Sim” (S) na resposta do utilizador, passando a ser true.

Depois, com um if, verifica-se se a lista de utilizadores não está vazia para assim validar o caso de que o novo utilizador foi convidado por outro existente à App Store. Se o novo utilizador contesta afirmativamente ante a pergunta “O utilizador foi convidado por outro? (S/N)” então, pede o número de identificação do utilizador que o convidou. Por isso verifica-se que a lista não esteja vazia, pois caso contrário, não haveria possibilidade de haver convidados. No fim, aparecem todos os dados do novo utilizador da forma como aparece no Anexo 6.5.

Existem exceções como o IOException (tratado com um throws), e o InputMismatchException (tratado com um try/catch).

Na aplicação é possível fazer registos através de ficheiros com recurso ao método *registoPorFicheiros ()* com um tipo de retorno *void* e de acesso *private*. Utiliza o *switch* para executar diferentes métodos dependendo da escolha do utilizador. O primeiro dos *cases* permite adicionar os ficheiros com os programadores através do método *adicionarFicheirosProgramadores ()* e depois volta ao menu de registo. O segundo faz o mesmo mas com as *apps* com recurso ao método *adicionarFicheirosApps ()* finalizando com o retorno ao menu de registo. Se o caso for que o utilizador selecionasse a opção 3, o método a executar será o *adicionarFicheirosUtilizadores ()* que como o seu nome indica, introduz o ficheiro texto com os novos utilizadores e volta para o menu de registo. Se o utilizador escolhesse a opção quatro, a aplicação adiciona todos os ficheiros num instante, chamando a todos os métodos anteriormente mencionados. A opção 5 serve para voltar ao menu principal e caso o utilizador não colocar um número de opção válido (de 1 até 5), executar-se-á o método de *opcaoErrada ()* e voltara inicializar o menu atual.

Para explicar melhor o que faz cada uma das primeiras quatro opções anteriores, serão explicados os métodos que os compõe, começando pelo método *adicionarFicheirosApps ()*, que é um método cujo retorno é do tipo void e o seu acesso é do tipo *public.* O método utiliza outro da classe *FileReader,* o qual abre uma *stream* de leitura para o ficheiro de texto chamado “Apps.txt” e inicializa outro método da classe *BufferedReader* para tornar mais eficiente a leitura do ficheiro, reduzindo o número de acessos aos ficheiros e lendo por cada linha com o método *readLine ().* Também são declaradas as variáveis do tipo *String* nome e categoria, o preco do tipo *double, “*subscrevivel” do tipo *booleano* e o id que é do tipo *int*. A seguir o programa vai ler do ficheiro por cada linha e vai retirando os dados segundo o seu tipo.

No fim, o ficheiro é fechado com o método *close ()*. As exceções também são consideradas, neste caso é unicamente a do *IOException* que mostra a mensagem “Ocorreu um erro no carregamento do ficheiro das apps!”. As apps ingressadas por meio dos ficheiros são inseridas na listaTodasApps e os seus respetivos programadores são colocados na listaIDsProgramadores e listaProgramadores.

O método *adicionarFicheirosProgramadores ()* é semelhante ao anterior inclusive no tipo de retorno, de acesso e a exceção considerada (com a mensagem “Ocorreu um erro no carregamento do ficheiro das apps!”). Utilizam-se os métodos de classe FileReader e BufferedReader para abrir e ler o ficheiro de texto “Programadores.txt”.

Os programadores inseridos por ficheiros são listados na listaIDsProgramadores (o seus próprios IDs unicamente) e na listaProgramadores (com os seus dados completos).

O outro método deste tipo é o adicionarFicheirosUtilizadores () que, semelhante aos métodos anteriores possui um tipo de retorno void e um acesso public. Tal e como em adicionarFicheirosApps (), inicializa-se as variáveis que mostram como informação dos utilizadores o nome (do tipo String), a idade e o id (do tipo int), o premium (do tipo boolean). Depois, utilizando o método readLine () lê-se linha a linha o ficheiro e segundo o tipo de valor, são passados para a consola como output com as informações de cada utilizador. Os novos utilizadores são inseridos na listaIDsUtilizadores (o seu número de identificação unicamente) e na listaUtilizadores com todos os seus dados.

Contém o mesmo tipo de exceção dos anteriores métodos. No Anexo 6.6 encontra-se um exemplo global.

Passando para outro método também do estilo menu e é o *menuListagem ()*, que tem um retorno do tipo *void* e de acesso *private*. Mostra um menu quando é selecionada opção “2 – Listagem” a partir do menu principal, onde as opções são referentes a listagens de *apps*, utilizadores, programadores e avaliações. O menu é feito novamente a base de *cases,* em que as duas primeiras opções farão a apertura de novos menus através dos métodos *menuListagemApps ()* (caso o utilizador escolha a primeira opção) e *menuListagemUtilizadores ()* (caso o utilizador selecionou a segunda opção). Os outros dois cases seguintes abrem listas como a definida pelo método *listarProgramadores ()* (no case 3) e *listarAvaliacoes ()* (no case 4). Como opção 5 temos o método que permite voltar ao menu principal. Caso não se verifique alguma destas opções, invoca-se o método *opcaoErrada ()* e volta a iniciar o *menuListagem ().* No *Anexo 6.7* pode observar-se o menu em questão.

Os métodos mais imediatos dos que foram mencionados anteriormente no menu Listagem são o *listarProgramadores ()* e o *listarAvaliacoes ()*.

Começa-se por explicar o primeiro deles que é um de acesso *public* e tipo de retorno *void,* o *listarProgramadores ()* permite mostrar os programadores existentes, considerando também que a lista pode estar vazia e mostrando a mensagem “Ainda não existem programadores!”. Caso contrário, o programa fará um percorrido à listaProgramadores e irá a verificar se cada programador tem ou não apps. Se um programador não tem apps, mostra no output o seu ID e uma mensagem que indica “Este programador não possui apps!”, mas se este em efeito possuir apps então mostrará o seu ID e o estado da sua avaliação. Se a avaliação for igual a 0.0, então o programa escreverá a seguinte frase “Avaliação: Ainda sem avaliação!”, mas se o programador se possui uma avaliação não nula, mostrará a frase “Avaliação: (valor atual da sua avaliação) estrelas”. Também imprime-se uma lista com as apps que programou cada programador. O Anexo 6.8 mostra um exemplo desta lista.

O segundo método é igual ao primeiro em termos de acesso e de tipo de retorno. O método listarAvaliacoes () realiza um if-else para determinar primeiro se a listaTodasApps está vazia ou não. Caso esteja vazia, mostra a mensagem “Ainda não existem apps!”, mas se o caso for o contrário o método pedirá por meio do Scanner e de um output que o utilizador escolha uma das apps para visualizar a sua classificação e os comentários dados pelos utilizadores. Para isso, faz-se um percorrido na listaTodasApps (já confirmando que esta não for vazia) e obtém-se o unicamente o nome de cada uma delas para ser mostrado ao utilizador. Depois de que ele insere uma resposta, compara-se através do método verificarOpcao (que recebe como parâmetros o nome inserido pelo utilizador e a listaNomesApps) para ver se a escolha é válida. Se a escolha é válida, então serão mostrados o nome, a classificação e os comentários referentes à app escolhida. No caso em que não houver comentários, mostra-se uma mensagem de “Não há comentários”. É mostrado no Anexo 6.9 um exemplo desta lista.

Nos dois métodos mencionados anteriormente consideram-se exceções do tipo *IOException*, propagados no código através do *throws*.

Havendo explicado os métodos que são executados diretamente ao serem escolhidos a partir do menu Listagem, passa-se a explicar outro método que define um menu diferente que é o método *menuListagemApps (),* invocado quando o utilizador escolhe a opção “1 – Apps”. O menu, uma vez mais, é tratado com cases como nos menus anteriores. Neste caso, cada opção inicializa um método diretamente (não abrem mais menus). No *Anexo 6.10* vê-se como está composto o menu.

Explicando os métodos que fazem funcionar cada opção, começa-se pelo de *listarPorCategoria ()* que é de acesso *public* e de retorno do tipo *void*. Este método verifica se a listaTodasApps encontra-se vazia ou não. Caso esteja vazia lança uma mensagem que diz “Ainda não existem apps!”. Mas de ser verificado o contrário, percorre-se a listaCategorias e imprime-se cada categoria. Depois, através de um booleano de nome existeAlgumaAppNestaCategoria que começa sendo false. Logo percorre-se a listaTodasApps e por meio de um equals verifica-se se existe alguma app de uma dada categoria para ser inserida em forma de listagem na sua respetiva categoria da lista que é pretendida mostrar. No caso de não verificar *apps* existentes numa dada categoria mostra a informação “- Não existem apps nesta categoria”. Este método considera exceções do tipo *IOException*. Ver *Anexo 6.11.*

O método *listarPorCategoriaSeletiva ()* é igual ao anterior com a diferença em que o utilizador escolhe a categoria que quere ver e se mostra unicamente a lista com as apps dessa categoria. Ver *Anexo 6.12*.

O método *listarPorClassificacao ()* é de acesso *public* e de tipo de retorno *void*. Este método percorre a listaTodasApps e, caso não esteja vazia, realiza uma ordenação (*sort*) por ordem crescente das classificações dadas pelos utilizadores. Mostra o nome da app e a media da sua classificação. Ver *Anexo 6.13*.

No caso de *listarPorClassificacaoSeletiva ()* faz o mesmo mas por intervalos e tomando em conta a decisão do utilizador sobre qual é o rango que quere ver. As listas estão classificadas em “Sem classificação”, “1 a 2 estrelas”, “2 a 3 estrelas”, “3 a 4 estrelas” e “Mais de 4 estrelas”. Havendo confirmado que existem *apps,* realizam-se cases para definir segundo a opção escolhida o que se irá a listar. As apps são comparadas a nível de classificação em cada *case* com o intervalo dado e se se confirma que estão dentro do mesmo, então são mostradas na mesma (dependendo da escolha do usuário). Este método considera o *IOException* e o trata com um *throws*. Ver *Anexo 6.14.*

O método *listarPorNome ()* de acesso *public* e tipo de retorno *void* faz uma lista das apps em ordem alfabética, verificando primeiro se existem *apps* e criando um *ArrayList* com os elementos de listaTodasApps e usando um sort para ordenar como é pretendido na nova lista. Ver *Anexo 6.15.*

O método *listarPorNumVendas ()* de acesso public e tipo de retorno void ordena as apps numa lista segundo o seu número de vendas em ordem crescente, usando como parâmetro de comparação o número de vendas que obteve cada uma. Utiliza-se um método *sort* para o efeito. Ver *Anexo 6.16*.

Havendo explicado por completo o menu anterior incluindo os seus métodos, passa-se a explicar outro menu que é definido pelo método *menuListagemUtilizadores ()* que é chamado quando é escolhida a opção “2 – Utilizadores” no menuListagem (). Este possui apenas duas opções (além da aquela que serve para voltar para o menu principal). A primeira opção é uma lista organizada de utilizadores (Anexo 6.18) e a segunda é uma lista ordenada pelo dinheiro total gasto (Anexo 6.19).

O método capaz de realizar as duas listagens anteriores é o listarUtilizadores que é um de acesso public e tipo de retorno void. Este método recebe como parâmetros o porDinheiroTotalGasto que é do tipo booleano. Se listaUtilizadores estiver vazia então emite uma mensagem que avisa que ainda não há utilizadores. Caso contrário, se fosse o caso de que porDinheiroTotalGasto for um true então imprime a lista de utilizadores desde os que menos gastaram até os que mais gastaram. Caso contrário imprime a lista de utilizadores organizada (ou seja, por ordem de chegada). Em ambos os casos mostram-se as informações de cada utilizador como o nome, a idade, se é premium ou não, a lista de apps compradas (caso possuam, senão mostrará a mensagem “Este utilizador não possui apps”). Também mostra a data da compra e o início da subscrição, para os casos em que tenha comprado ou subscrito a alguma das apps. Outra informação a ser mostrada é o dinheiro gasto por cada utilizador em apps.

Agora, passa-se a explicar os métodos *inicioCompraOuSubscrição (boolean subscrição)*, *comprar (Utilizador utilizador, Scanner scannerDados, boolean carrinhoVazio, ArrayList <App> listaAppsDisponiveis)* e *subscrever (Utilizador utilizador, ArrayList <App> listaAppsDisponiveis, Scanner scannerDados)* que são métodos que podem ser utilizados para comprar e subscrever a *apps* que estejam disponíveis.

O primeiro é aplicado quando o utilizador escolhe a opção “3 – Comprar” ou a opção “4 – Subscrever” do menu principal, diferenciando o seu efeito através do booleano que recebe como parâmetro, sendo false se se quere realizar uma compra e true se se quere subscrever a uma *app*. Verifica se listaTodasApps está vazia e se não o está prossegue a confirmar se existem utilizadores em listaUtilizadores. Depois de ter feito as devidas confirmações e ter verificado que sim existem ambas os elementos, então inicializa um método da classe *Scanner* e pede para inserir o ID do utilizador, verifica se tal ID existe e se o caso for positivo então o utilizador pode realizar compras. Depois prepara uma união de listaAppsDisponiveis com a lista de apps compradas pelo utilizador, logo realiza uma interceção para comparar as apps que estão presentes nas duas listas e caso exista coincidências, estas são apagadas de listaAppsDisponiveis. Verifica-se por meio de um *if-else* se a opção é diferente de subscrição e se é o caso aplica-se o método de *comprar,* caso contrário o método a executar é *subscrever*. Estes métodos consideram os parâmetros utilizador, scannerDados e listaAppsDisponiveis.

No caso de ser uma compra a operação que deseja fazer, o método *comprar* é invocado. Começa por imprimir uma mensagem “Escolha uma das seguintes apps. Todas as apps assinaladas com [G] são elegíveis para obter gratuitamente:”. Percorre a listaAppsDisponiveis e imprime os nomes e os preços das apps que aparecem nesta lista. Como todos os usuários têm direito a uma app gratuita, verifica-se se ainda o utilizador não escolheu uma app então coloca a “[G]” ao lado das apps que estão aptas para ser selecionadas baixo estas condições. Mas se se escolhe uma app sem desconto, a compra é feita normalmente, invocando o método compra para essa app em específico (através de um this). Ver Anexo 6.20.

Também, verifica-se como decisão se o utilizador quere obter a app gratuita ou não, fazendo alterações ao seu preço para que seja igualado a zero. Quando o utilizador escolhe uma app com desconto aparecerá a mensagem "Todos os utilizadores têm direito a uma app gratuita (por semana) de entre aquelas que têm desconto. Esta é uma dessas. Pretende obtê-la gratuitamente ou proceder com a compra? (S/N)", espera a resposta do utilizador e se esta for afirmativa, então através de um setter muda-se o booleano (para true) que indica se foi usada a app gratuita ou não. É aqui onde o preço da app passa a ser 0.0 automaticamente. O método comprar é chamado para essa app em específico.

Este método considera os erros de exceção do tipo IOException tratados com um throws e os erros do tipo InputMismatchException que são tratados com um try/catch mostrando a mensagem "Introdução inválida! Tente novamente!”.

Se o caso for realizar uma subscrição, então o método subscrever é executado. Inicia com uma frase de petição ao utilizador: "Escolha uma das seguintes apps: " e mostra a lista com as apps que são possíveis de ser obtidas por meio de uma subscrição. O preço de subscrição é 70% do preço de compra da aplicação. Este cálculo é feito dentro do método e o seu arredondamento também. Assim, na aparecerão as apps “subscrevíveis” com o seu nome e o seu preço de subscrição. Também é criada uma lista com apps que não foram subscritas ainda pelo utilizador para que ele saiba quais ficam disponíveis para o efeito. Caso o utilizador quera subscrever a uma, insere o nome da app e depois o mesmo é verificado pelo código através de uma comparação com o método verificarOpcao. Ver Anexo 6.21.

Depois de ter comprado ou subscrito a alguma app, o utilizador tem direito de avaliá-la. Para isso se criou o método avaliarAplicacao (). O método é de acesso do tipo public e retorno void. Primeiro verifica se as listas listaTodasApps e listaUtilizadores não estejam vazias, pois a inexistência de um e de outro impede que existam avaliações. Quando confirmar que existem apps e utilizadores, então inicializa um Scanner e pede que “Insira o ID do utilizador” para depois verificar se existe um utilizador registado com tal ID inserido. Caso o utilizador não tenha apps mostra a mensagem “Você ainda não possui apps”, mas se o for verificado o contrário então mostrará a mensagem “A lista de apps disponíveis para avaliação é: “ e a lista com o nome das apps que o utilizador possui. Depois de o utilizador introduzir o nome da app que quere avaliar, executa-se o método de verificarOpcao para confirmar a existência dessa app na lista. O método também considera a possibilidade de mudar de avaliação, confirmando se o utilizador avaliou a mesma app antes e mudando a valoração com a nova classificação introduzida pelo utilizador. A seguir, o programa pede para inserir uma pontuação de 1 a 5 e com inteiros, verificando por meio do método verificarOpcaoPontuacao para determinar se a opção é valida segundo os parâmetros decididos. Logo, o programa irá perguntar “Deseja deixar um comentário? (S/N)”. Se o caso for afirmativo, prepara-se uma variável do tipo String chamada comentario e pede-se para inserir o comentário, então é criada uma avaliação com os parâmetros pontuacao, comentario e utilizador. Mas no caso de ter negado o desejo de comentar, simplesmente cria-se a avaliação mas só com os parâmetros pontuacao e utilizador. As avaliações são adicionadas numa lista. No caso dos programadores também se lhes altera uma informação caso alguma das apps que criou recebe uma avaliação. O seu contador de avaliações incrementa de um a um por cada avaliação recebida dos utilizadores pelas apps criadas. O total desse contador serve para no fim, fazer uma média de classificação do programador pelas suas apps, mostrando este valor como uma característica dele próprio. O mesmo acontece com as apps, faz-se uma soma do total de pontuação recebida a dividir pelo número de utilizadores que avaliaram a mesma app. No fim do procedimento volta para o menu principal. Ver Anexo 6.22.

Este último método considera os erros de exceção dos tipos IOException e InputMismatchException.

Outro submenu criado para a App Store é o que está definido no método menuMontanteRecebido () (Ver Anexo 6.23) com tipo de retorno void, acesso private e considerando erros de exceção do tipo IOException. Serve para mostrar um menu em que se pode escolher duas opções nas quais se podem ver o total de ganancias tanto da App Store como dos programadores, além da opção de voltar para o menu principal. Como os anteriores menus, é construído a base de um switch com 3 cases. O case 1 mostra o valor total da App Store (Anexo 6.24) com recurso à variável valorTotalAppStore, utilizando uma String da seguinte forma "O valor total recebido pela AppStore é de: "+valorTotalAppStore+" euros.". No case 2 (Ver Anexo 6.25), verifica-se antes que a listaProgramadores não esteja vazia para logo mostrar uma lista com todos os programadores e o valor recebido por cada um pelas apps compradas pelos utilizadores. Mas se o caso for que a lista de programadores está vazia, imprime uma mensagem de “Ainda não existem programadores!”. O case 3 simplesmente invoca o método menuPrincipal () para fazer aparecer este menu. Caso a escolha não seja nenhuma dessas aparecerá a mensagem de erro correspondente e voltará a inicializar este menu.

O seguinte método a apresentar é o cincoMenosVendidas que recebe como parâmetro um booleano de nome passouUmAno. Retorna um valor do tipo void e é de acesso private. Serve para verificar quais foram as cinco apps menos vendidas da semana anterior, começando por certificar o passo do tempo através do contador de dias e também do passo de um ano, com atenção de que a listaTodasApps não estiver vazia. Faz um percorrido pela lista de utilizadores e dá o direito de obter uma app gratuita cada semana das que são escolhidas através de um booleano chamado usouAppGratuita. Cada vez que passa uma semana, esta variável passa a ser false para todos os utilizadores. Depois faz um percurso à lista de todas as apps e altera o valor das variáveis booleanas menosVendida e temDesconto para false. Este último procedimento também é feito a cada semana. Ver Anexo 6.26.

A seguir, o método verifica se existem menos de cinco apps na App Store. Se o caso é confirmado, então ficam as que existem como as menos vendidas, alterando a cada uma a sua variável menosVendida para true. Em cada início de semana, o número de vendas de cada app é igualado a zero para começar uma nova contagem.

Mas se houver mais de cinco apps, ordenam-se as apps (por ordem crescente) segundo o seu número de vendas através de um sort. Por meio de um if-else verifica se não se passou um ano (a partir da opção de passar um ano no menu principal). No caso em que esta função não foi ativada, seleciona as cinco primeiras apps da lista ordenada que representam as apps menos vendidas da semana e imprime o nome de cada uma, altera o valor da variável menosVendida para true e altera o seu número de vendas para zero, uma vez que já passou uma semana.

No caso de que passa um ano, o procedimento é o mesmo exceto pela parte de que nada é impresso. Percorre os cinco primeiros elementos da lista ordenada de apps menos vendidas e imprimem-se os seus nomes. As variáveis menosVendida e numeroDeVendasSemanal são alteradas para true e 0 respetivamente.

O método categoriaDoMes é de acesso public e tipo de retorno void. Recebe como parâmetro o booleano passouUmAno. O método verifica se o programa foi inicializado ou se já se passou um mês ou se já passou um ano para alterar a categoria. A alteração é feita gerando um número aleatório entre 0 e 7 (cada número é associado a uma categoria através do seu índice na lista), utilizando o random. O número que resultar definirá qual é a categoria do mês. Ver Anexo 6.27.

O método verificarSubscricoes () é um de acesso private e tipo de retorno void. Encarrega-se de verificar o tempo das subscrições feitas pelos utilizadores. Isto é feito fazendo um percurso na lista de utilizadores e verificando se cada um tem subscrição a alguma app. Caso um utilizador não se subscreveu a uma app passa-se a analisar o utilizador seguinte. Mas se o utilizador em questão tem subscrito a uma app é feito uma análise às apps nas quais fez uma subscrição. O programa verifica se o contador de dias de cada subscrição é maior ou não do que 365 dias. Se se passou de 365 dias, então a app é removida da lista e é introduzido um null no seu lugar para não alterar o valor de cada índice para continuar com a análise. Quando chega ao final da lista de subscrições, remove todos os null da lista.

O método getAppPorNome recebe um parâmetro nome de classe String e tem como retorno um objeto da classe App, tendo como tipo de acesso um public. Este método utiliza um ciclo for que percorre uma lista cujo índice é uma variável do tipo int que começa no 0 e não ultrapassa o tamanho (size) do ArrayList. Enquanto é percorrida a lista vai-se comparando um a um a app na posição indicada pelo índice com o nome recebido como parâmetro, através do método equals (nome). Quando se confirma uma igualdade, o método retorna a app dessa posição.

O método getUtilizadorPorID é de acesso private e retorna um valor da classe Utilizador. Recebe um parâmetro id do tipo int. Percorre a lista de IDs de utilizadores e compara com cada um deles o id introduzido. Quando encontrar um igual, retorna esse utilizador. Tem em consideração exceções do tipo IOException tratados com um throws.

O método getProgramadorPorID é de acesso private e retorna um valor da classe Programador. Recebe um parâmetro id do tipo int. Percorre a lista de IDs de programadores e compara com cada um deles o id introduzido. Quando encontrar um igual, retorna esse utilizador. Tem em consideração exceções do tipo IOException tratados com um throws.

O método *voltarMenuRegisto ()* é do tipo de retorno *void* e de acesso *private.* Este método pergunta ao utilizador se quere voltar para o menu registo (opção 1) ou para o menu principal (opção 2). A seguir, utiliza o método da classe Scanner para ler o número que representa a opção que o utilizador quere tomar que é um valor do tipo inteiro. Através de um switch se executa a opção que o utilizador tomou. Caso (case) seja o valor igual a 1, é inicializado o método menuRegisto (), mas se o caso for 2, o método a executar será o menuPrincipal (). Caso contrário destes dois últimos, na consola aparecerá a mensagem de “Opção inválida!” e voltará a executar este mesmo método de voltarMenuRegisto ().

O método voltarMenuListagem () de retorno void e acesso private é muito semelhante ao anterior, só que começa por consultar ao utilizados se deseja voltar para o menu listagem (opção 1) ou para o menu principal (opção 2). Neste caso o método da classe Scanner lê a opção que o utilizador introduz e através de um if-else realiza uma decisão consoante à decisão tomada, sendo que se for 1 o valor lido, vai para o método de menuListagem (), mas se selecionou o 2, vai para o menuPrincipal (). Caso não se verifica nenhuma destas opções, coloca a mensagem de “Opção inválida!” e voltará a executar este mesmo método de voltarMenuListagem ().

O método *voltarMenuMontante ()* é do tipo de retorno *void* e de acesso *private.* Este método pergunta ao utilizador se quere voltar para o menu montante recebido (opção 1) ou para o menu principal (opção 2). Depois, utiliza o método da classe Scanner para ler o número que representa a opção que o utilizador quere tomar que é um valor do tipo int. Com recurso a um switch se executa a opção que o utilizador tomou. Caso seja o valor igual a 1, é inicializado o método menuMontanteRecebido (), mas se o caso for 2, o método a executar será o menuPrincipal (). Caso contrário destes dois últimos, na consola aparecerá a mensagem de “Opção inválida!” e voltará a executar este mesmo método de voltarMenuRegisto ().

Os três últimos métodos anteriores consideram os erros do tipo IOException e são tratados com o throws.

O método de acesso *private* e de retorno do tipo *void* (não retorna nada) *opcaoErrada ()* lança uma mensagem de erro como *output* que diz “A opção assinalada não é válida!”.

O método *verificarOpcaoPontuacao* tem como parâmetro o scannerDados da classe *Scanner*, é de acesso *public* e tipo de retorno *int*. Recebe o que inseriu o utilizador e verifica por meio de uma lista com os valores válidos para realizar classificações às apps comparando com cada um para ver se é um valor válido para inserir. Quando encontrar uma igualdade entre o valor inserido e um valor da lista, realiza-se um break e retorna o valor da pontuação dada. Mas se chega ao fim da lista e não encontra nenhuma coincidência então mostra a mensagem “A pontuação inserida não é válida. Insira novamente: “, voltando a inicializar este ciclo.

O método verificarOpcao tem como parâmetros o scannerDados da classe Scanner e a listaStrings que é um ArrayList <?> (de tipo não especificado). Este método é de acesso *public* e tipo de retorno *String*. O que faz este método é basicamente receber uma lista e um valor inserido pelo utilizador e verifica se tem alguma coincidência entre algum elemento da lista e o que foi inserido. Caso se confirme uma coincidência, retorna o valor da opção mas se for o contrário, envia uma mensagem de erro "A opção inserida é inválida. Insira novamente:" e volta a começar o mesmo ciclo de verificação.

Os métodos verificaOpcaoID e verificaOpcaoNomeApp fazem o mesmo que os dois métodos anteriores mas recebendo as listas correspondentes e o valor inserido pelo utilizador para comparar.

O método *getData ()* da classe *Data* e de acesso do tipo *public,* ou seja, pode ser acedida por outras classes. É um *getter* que simplesmente tem como valor de retorno a data atual da *App Store*.

O método *getOpcao ()* retorna um valor do tipo inteiro e é de acesso *private,* isto é, só pode ser acedido pela própria classe. Neste método utiliza-se a classe *Scanner* para ler a opção que o utilizador introduz na consola. Retorna o valor da opção.

A seguir de este método, são declarados *getters* para as listas de IDs tanto de programadores como de utilizadores, a lista de todas as *apps* e a de todos os utilizadores, retornando essas mesmas listas no seu getter correspondente.

Por último está o método de *addValorAppstore* que recebe como parâmetro o *novoValor* que é do tipo double. Cada vez que é realizado um pagamento, este valor atualiza.

### 3.1.3 – App:

É uma classe que define cada uma das instâncias deste tipo, ou seja, define cada app pelos mesmos tipos de atributos que serão especificados a seguir.

Tem como atributos de classe *String* o nome e a categoria, de tipo *double* o preco e a classificacao, de tipo *int* os atributos numeroDeVendas e numeroDeVendasSemanal, de tipo booleano a menosVendida, o subscrevivel e o temDesconto, de classe *Programador* o programador e uma lista de objetos da classe *Avaliacao* (*ArrayString <Avaliacao>*) de nome listaAvaliacoes.

Esta classe tem um construtor App que recebe como parâmetros o nome, o preço, o programador e a decisão de ser “subcrevível” ou não. Estas variáveis são instanciadas no inicio do programa. Também são inicializadas outras variáveis como o numeroDeVendas e o numeroDeVendasSemanal, sendo estas igualadas a 0 por serem inteiros, e classificacao é igualada a 0.0 por ser um *double*. Outra variável que é inicializada é menosVendida que começa por ter como valor o *false*, igualmente para a variável temDesconto. No caso de listaAvaliacoes, começa por ser uma ArrayList vazio.

A seguir tem-se um caso de *Overloading* com os métodos *comprar (Utilizador utilizador, AppStore appstore, ArrayList<App> listaAppsDisponiveis)* e *comprar(Utilizador utilizador, AppStore appstore, ArrayList<App> listaAppsDisponiveis, App app, double precoFinal)*, pois ambos têm o mesmo nome e parâmetros diferentes.

O primeiro método *comprar* serve para calcular o preço de cada *app* com descontos ou sem eles. O método é de acesso *public* e retorno do tipo *void*. O procedimento a tomar é igualar o valor da app em questão com a variável precoFinal. Depois faz a seguinte conta:

**precoFinal = ( Preço da app + (100 - desconto) ) / 100**

**Sendo que o desconto é igual a 0 quando é existente (o precoFinal acaba por ser o mesmo que o preço original da *app*) ou, caso exista, altera o preço original diminuindo-o segundo o valor do desconto. O valor calculado é arredondado. No final chama o método *comprar (utilizador, appstore, listaAppsDisponiveis, app, precoFinal)*.**

**O segundo método *comprar* que tem o mesmo tipo de acesso e de retorno que o anterior, permite a criação de objetos da classe *Compra* e o ato de continuar a compra ou não. Primeiro inicializa as variáveis que guardaram o valor do dia, do mês e do ano da data em que a compra foi realizada, criando assim um objeto da classe *Data* com estes dados. Depois cria um objeto da classe Compra com os dados até agora (app, data e preço). O método consulta ao utilizador se quere continuar com a compra ou não, sendo que cada opção é representada com os números 1 e 2, respetivamente. A seguir, utilizando um *switch* com dois *cases*, definem-se as duas ações. No *case* 1, adiciona a compra anterior no “carrinho de compra” e apaga a *app* que representa essa compra da lista de *apps* que estão para ser compradas ainda. Se esta lista for vazia, então imprime a mensagem** “Não existem mais apps disponíveis! Redirecionando para o pagamento… ” e invoca o método pagar. O case 2 adiciona a compra no carrinho e invoca o método pagar. Se a opção não for nenhuma dessas, imprime a mensagem “Opção inválida!”.

O método subscrever recebe como parâmetros o utilizador da classe Utilizador e appstore da classe AppStore, é de acesso public e tipo de retorno void. Contém variáveis que recebem o dia, mês e ano da data da subscrição, assim como o contador de dias para saber a validade da subscrição. Cria objetos da classe Data e Subscrição com os dados obtidos e invoca o método pagar.

O método verificarDesconto é de tipo de retorno int e de acesso private. Por meio de este método definem-se os descontos com recurso a ifs. Inicializa uma variável do tipo inteiro com o nome de desconto. Se é uma app que está entre as cinco menos vendidas, receberá um true da sua variável menosVendida e dará um desconto de 15% (que para efeitos de cálculo é colocado como 15 para realizar a conta de precoFinal) e modifica a variável temDesconto para true. Se a app pertence à categoria do mês, então recebe 5% de desconto, modificando a variável temDesconto para true. Outro desconto é verificado quando um utilizador angaria mais utilizadores, sendo que por cada um tem mais 5% de desconto. Por isso realiza-se um cálculo de multiplicação entre os dois valores. Um último desconto a considerar é a do utilizador premium, que tem um valor de 60%. Também o código verifica o caso de que os descontos ou a soma deles ultrapassarem os 100%, então iguala para 100%, ficando a app grátis.

Criam-se umas interfaces de comparação que realizam as devidas comparações entre o número de vendas, o número de vendas semanais e a classificação recebida entre as apps. Estas interfaces ajudam a métodos como o que foi mencionado anteriormente.

Existem dois setters com parâmetros do tipo booleano, acesso do tipo public e tipo de retorno void. Estes são o setMenosVendida e o setTemDesconto que permitem modificar os valores das variáveis menosVendida e temDesconto.

Um método getter usado nesta classe e retorna um valor da classe Programador e tem acesso do tipo public. Este método é chamado getProgramador.

Outros getters são getCategoria e getNome, ambos com tipo de retorno String, getNumVendas e getNumeroDeVendasSemanal do tipo int, getClassificacao e getPreco do tipo double, getMenosVendida, getTemDesconto e getSubscrevivel do tipo booleano.

Outro modificador com tipo de retorno void e parâmetro int numeroDeVendasSemanal é o setNumeroDeVendasSemanal.

Os métodos addNumeroDeVendas e addNumeroDeVendasSemanal vão adicionando 1 por cada venda obtida numa semana ou em total.

### 3.1.4 – Utilizador:

A classe Utilizador tem como atributos as variáveis nome (do tipo String e correspondente ao nome do utilizador), idade, idUtilizador e numUtilizadoresAngariados (ambas do tipo int e que representam a idade, o ID do utilizador e o número de utilizadores angariados pelo mesmo, respetivamente), as variáveis booleanas premium e usouAppGratuita (que indicam se o utilizador é premium ou não e se este já usou a promoção semanal da app gratuita, respetivamente) e também os *ArrayLists* correspondentes à lista de apps do utilizador, à sua lista de subscrições ativas e à sua lista de compras feitas até então. Todos os atributos desta classe são *private*, de modo a respeitar o princípio do encapsulamento.

O construtor desta classe recebe como parâmetros as variáveis do nome, idade e o booleano que indica se este é premium ou não. Tanto estas três variáveis como os outros atributos desta classe são inicializados no construtor.

O primeiro método desta classe é o método *gerarIDUtilizador*, que trata de gerar o ID de um utilizador aquando da sua criação. Estabeleceu-se que os utilizadores têm um ID entre 10000 e 19999, pelo que este método gera um número dentro deste intervalo e verifica, através da lista (neste caso, *ArrayList*) que contém os IDs já utilizados por outros utilizadores, se já existe algum utilizador com o ID que foi gerado. Caso isto aconteça, é gerado um novo ID. Caso contrário, este valor gerado é atribuído ao utilizador em questão como o seu ID.

Esta classe contém os métodos correspondentes ao pagamento tanto da compra como da subscrição.

Inicialmente, o método que trata do pagamento da compra percorre o carrinho do utilizador em questão e imprime os nomes das apps e os respetivos preços associados a cada compra (objeto da classe Compra), uma vez que cada compra corresponde a uma aquisição individual. Os valores dos preços são somados e assim obtém-se o valor total a pagar pelo utilizador que também é imprimido. De seguida, é perguntado ao utilizador se ele pretende confirmar o pagamento, sendo isto feito através de um Scanner que receberá uma String. O método *verificarOpcaoConfirmar* verifica se o valor inserido no *input* corresponde a “S” ou “N”, sendo estas opções correspondentes a uma resposta afirmativa e a uma resposta negativa, respetivamente. Caso o utilizador insira outro valor qualquer, o programa exibe uma mensagem de erro. No caso de uma resposta negativa, o programa volta para o menu principal, enquanto no caso de uma resposta afirmativa (“S”) o programa percorre mais uma vez o carrinho e adiciona as apps de cada compra na lista de apps do utilizador em questão e os respetivos valores a pagar pelas mesmas tanto na variável correspondente ao dinheiro ganho pelo programador como na variável do dinheiro gasto pelo utilizador e no valor total da *App Store*. Além disso, é incrementado em 1 os números de vendas totais e semanais dessas mesmas apps. De seguida, o conteúdo do carrinho é copiado para a lista de compras do utilizador em questão, uma vez que esta lista guarda todas as compras alguma vez efetuadas pelo utilizador de modo a guardar os dados das mesmas. Por fim, o conteúdo do carrinho é apagado para que este possa ser reutilizado numa próxima compra deste utilizador, é exibida uma mensagem de confirmação e o programa volta ao menu principal.

Por outro lado, o método que trata do pagamento da subscrição, que recebe o objeto correspondente à subscrição em questão, simplesmente calcula o valor correspondente a 70% do preço da app desta mesma subscrição (estabeleceu-se que cada subscrição por um período de 1 ano custa 70% do preço da app sem incluir os descontos ativos naquele momento) e apresenta esse valor ao utilizador através de uma impressão. Tal como no método anterior, é perguntado ao utilizador se pretende ou não confirmar o pagamento, recorrendo mais uma vez ao método *verificarOpcaoConfirmar*. Em caso de resposta negativa, volta ao menu principal, enquanto em caso de resposta afirmativa, adiciona a subscrição em questão à lista de subscrições (ativas) do utilizador, a app correspondente a essa subscrição na sua lista de apps e o valor total a pagar nas variáveis do dinheiro total gasto pelo mesmo, do dinheiro total recebido pelo programador e do valor total recebido pela AppStore como um todo.

De seguida, implementou-se nesta classe uma interface “*Comparator*” denominada *COMPARAR\_POR\_DINHEIRO\_GASTO* que, tal como o nome indica, serve para comparar os utilizadores pelo dinheiro que gastaram e então é utilizado na listagem dos mesmos de acordo com estes valores (listagem por dinheiro gasto). Nesta interface, a variação do método compare para variáveis do tipo Double (uma vez que a variável do dinheiro gasto é deste tipo) é sobreposto (*override*) por um método que recebe dois utilizadores e compara os seus dinheiros totais.

Os métodos apresentados no final desta classe correspondem a métodos modificadores, seletores e incrementadores (métodos que adicionam certos valores aos atributos desta classe).

O primeiro método é um destes métodos incrementadores. Este método trata simplesmente de incrementar em 1 o atributo correspondente ao número de utilizadores angariados de um determinado utilizador (*numUtilizadoresAngariados*).

Os métodos *getIDUtilizador*, *getIdade*, *getNome* e *getPremium* são métodos seletores que devolvem os atributos do ID, da idade, do nome e da variável booleana premium do utilizador em questão.

O método *getCarrinho* é um método seletor que devolve a variável carrinho que, tal como já se referiu anteriormente, corresponde à lista temporária de compras (objetos da classe *Compra*) aquando de uma determinada compra do utilizador em questão.

O método *getNumUtilizadoresAngariados* corresponde a um método seletor que devolve o número de utilizadores angariados pelo utilizador.

Por outro lado, o método *setNumUtilizadoresAngariados* é um método modificador que recebe um determinado número para o qual a variável do número de utilizadores angariados de um utilizador será alterada.

O método *getListaSubscricoes* trata-se de um método seletor que devolve a lista de subscrições de um determinado utilizador.

O método *getUsouAppGratuita* é um método seletor que devolve o atributo usouAppGratuita do utilizador em questão, sendo que esta variável indica se o utilizador já utilizou a promoção da app gratuita na semana corrente.

Por fim, o método *getListaCompras* é um método seletor que trata de retornar a lista de compras do utilizador em questão, ou seja, a lista de todas as compras alguma vez efetuadas pelo mesmo.

### 3.1.5 – Programador:

Esta classe *Programador* tem como atributos idProgramador do tipo int que será gerado aleatoriamente, avaliacao do tipo *double* que corresponde a média das avaliações das apps desenvolvidas pelo programador, dinheiroTotalRecebido do tipo *double* que é a soma de todas as apps desenvolvidas pelo programador que foram vendida ou subscritas, um *ArrayList* listaApps contém objetos da classe App e numAvaliacoesRecebidas do tipo int que será o somatório do números de avaliações que suas apps desenvolvidas receberam.

O construtor desta classe contém a inicialização de todos os atributos referidos em cima.

O método *gerarIDProgramador (AppStore appstore)* que permite a geração do ID do programador alectoriamente, definiu-se que o ID do programador iniciaria em 10000 até 19999 e com o auxilio da biblioteca Random gerou-se o ID mas com as seguintes condições:

* Se a lista de IDs já utilizados pelos programadores está vazia então iguala o idProgramador do programador à variável numeroaleatorio*,* gerando assim o ID de programador.
* O ID do programador (idProgramador) deverá ser único por isso, caso a lista de programadores não seja vazia então percorre a lista de IDs já utilizados pelos programadores e caso seja encontrado um ID nesta lista que seja igual ao ID gerado para o novo programador numeroaleatorio, o método é reinicializado de modo a que seja gerado um novo ID. Caso contrário quando a lista tenha sido percorrida na sua totalidade e nenhum dos IDs corresponde ao ID gerado, o ID deste programador é igualado à variável *numeroaleatorio*.

No fim do método é retornado o valor do atributo *idProgramador*.

O método *addAppProgramador (App appAdicionada)* adiciona app desenvolvida pelo programador a lista de apps que o mesmo contém. O *addDinheiroTotalRecebido (double novoValor)* é o método que adiciona o novo valor recebido pela venda ou subscrição de uma aplicação ao valor do atributo dinheiroTotalRecebido*.* Tal como o método anterior, *addNumAvaliacoesRecebidas ()*, este método incrementa quando uma nova avaliação é realizada ao atributo *numAvaliacoesRecebidas*.

Os métodos seletores nesta classe são *getIDProgramador (), getListaAppsProgramador (), getDinheiroTotalRecebido (), getAvaliacao ()* e *getNumAvaliacoesRecebidas ().* Esta classe contêm um único método modificador *setAvaliacao (double novaAvaliacao)* que recebe como parâmetro a novaAvaliacaodo tipo *double* que modifica o atributo avalicao pois iguala avalicao a novaAvalicao, assim modificando.

### 3.1.6 – Compra:

A classe Compra contêm os seguintes atributos: app que é do tipo *App*, valorTotal do tipo *double* e dataCompra do tipo *Data*.

O atributo app é aplicação que estará a ser comprada, o valorTotal é o valor a pagar pela compra já com os descontos incluídos e o objeto da classe *Data* correspondente ao atributo dataCompra em que é realizada a compra.

O construtor desta classe contém como parâmetros app, valorTotal e dataCompra dos mesmos tipos referidos em cima respetivamente. A inicialização dos atributos ocorre no construtor.

Esta classe também contêm métodos seletores para os atributos, *getValorTotal (), getApp ()* e *getDataCompra (),* respetivamente. São necessários estes métodos seletores pois estes atributos, por encapsulamento são *private*, uma das regras fundamentais de programação orientada por objetos.

### 3.1.7 – Data:

A classe *Data*, tal como o nome indica, representa uma determinada data. Os objetos desta classe servem para não só trabalhar com o tempo para fins simulatórios como também para guardar as datas em que ocorreram as compras e as subscrições, tendo uma importância ainda maior para estas últimas, uma vez que estas são temporárias.

Os atributos desta classe são o dia, o mês, o ano e o contadorDias, todas estas variáveis do tipo *int*. Além disso, esta classe também tem como atributo a variável random, pertencente à classe *Random*, utilizada para a obtenção de números aleatórios.

Esta classe possui dois construtores. O primeiro construtor só é chamado uma vez, no início da simulação, simplesmente para fins de inicialização dos atributos. O segundo construtor é chamado nas compras e nas subscrições para fins de criação de um novo objeto da mesma de acordo com a data nesse momento, sendo que este construtor recebe então o dia, o mês, o ano e o contador de dias do objeto correspondente à data atual.

O primeiro método implementado nesta classe é o método *escolherMes* que, tal como o nome indica, escolhe um número aleatório entre 1 e 12 para escolher o mês no início da simulação, através do objeto da classe Random.

O método *escolherDia ()* serve para escolher um dia aleatoriamente de acordo com o mês, uma vez que cada mês possui um número específico de meses. Para tal, é utilizado um switch com os vários meses.

O método *avancarDia* é o método que permite avançar um dia na simulação. Para tal, este método incrementa o contador de dias do objeto da data atual em 1 e de acordo com o mês e o respetivo dia é verificado se a variável do dia será simplesmente incrementada em 1 ou se está passará a ser 1 e o mês será incrementado em 1 (no caso de o dia que estava anteriormente ser o último dia de um determinado mês). No caso do mês 12 (dezembro), se o dia anterior for igual a 31 (último dia desse mês), os valores das variáveis do dia e do mês serão igualados a 1 e a variável do ano será, logicamente, incrementada.

O método *avancarAno* simplesmente tem como função avançar um ano completo na simulação. Como tal, os atributos do dia e do mês mantêm-se intactos enquanto a variável correspondente ao ano é incrementada em 1 e o contador de dias em 365 (número de dias correspondentes a um ano).

O método *mostraData* trata de devolver os atributos do objeto da data (no formato dia/mês/ano) em forma de *String*. Para garantir que o dia e o mês são representados por 2 dígitos, esta função verifica se o dia e o mês são menores que 10 e, caso isso aconteça, acrescenta um 0 antes da apresentação destas variáveis.

No fim desta classe estão definidos os métodos seletores para os atributos da data em si, sendo eles as variáveis correspondentes ao dia, ao mês, ao ano e a variável correspondente ao contador de dias.

### 3.1.8 – Subscrição:

A classe *Subscrição* contêm apenas dois atributos, app que é do tipo *App* e dataSubscricao do tipo *Data*.

O atributo app é aplicação que estará a subscrita pelo utilizador e a dataSubscricao é data quando a subscrição estará a ser realizada.

No construtor desta classe contêm dois parâmetros do mesmo tipo que os atributos e onde estes são inicializados.

Os métodos seletores são *getDataSubscricao ()* e *getApp ()* como já foi referido são necessários fazer estes métodos quando necessitamos em outras classes.

### 3.1.9 – Avaliação:

A classe Avaliacao contêm três atributos, pontuacaodo tipo *int,* comentariodo tipo *String* e utilizador do tipo *Utilizador*.O atributo pontuacaocorresponde a pontuação que será inserida pelo utilizador, o comentario ao comentário que o utilizador terá ou não a possibilidade de deixar ao avaliar a aplicação e o utilizadorindica quando o utilizador que está a avaliar a aplicação.

Pelo facto do utilizador ter possibilidade de optar por deixar um comentário ou não na app que está avaliando (o utilizador só poderá avaliar uma app que tenha comprado ou subscrito) existem nesta classe dois construtores.

O primeiro construtor passou-se como parâmetros pontuacao, comentario e utilizador (do mesmo tipo que os atributos) é utilizado quando o utilizador opte por deixar um comentário acerca da app, caso contrário o segundo construtor só contém dois dos parâmetros, pontuacao e utilizador. Em ambos os construtores inicializam os atributos.

Os métodos *getPontucao ()* e *getComentario ()* são os métodos seletores desta classe.

# 4 – Discussão (Diagrama UML Antigo vs. Diagrama UML final)

Antes de ser começado o trabalho, solicitou-se a criação de um diagrama UML para ter ideias iniciais prévias à realização do código. Como se pode notar nos anexos da secção 8, existe uma visível diferença de dimensão de programa, assim como alterações relativas ao tipo de relação entre as classes criadas.

Em primeiro lugar deve notar-se que o número de classes não mudou. O que mudou foi a ideia de ter como uma classe principal a *AppStore* e como subclasses da mesma a *App*, o *Utilizador* e o *Programador*, que passaram a ser classes associadas a *AppStore*. Esta mudança é devida a que não fazia sentido ter estas últimas três classes como subclasses de *AppStore,* pois nem todos os métodos ou características são herdadas dela mas, o que sim é confirmado é que necessariamente há uma relação de associação.

Outra diferença é que foram criadas mais métodos e mais variáveis em algumas classes que servem para melhorar o funcionamento da *App* *Store* e não foram ideadas num início pois a medida que se realizava o código houve necessidade de métodos extras a aqueles que se tinha pensado.

Também foi realizada uma correção numa das ligações entre classes, nomeadamente entre as classes *Programador* e *App.* É de conhecer que os programadores podem existir na App Store sem ter criado alguma *app*, ou seja, os programadores podem existir sem *apps* mas, no caso das apps não se pode verificar essa independência de forma reciproca, isto é, as *apps* não podem existir sem os seus respetivos programadores. A correção simplesmente foi uma troca de sentido da seta que denota a agregação composta. Antes estava a sair de *Programador* e apontando para *App* e agora é ao contrário.

Outra mudança foi em relação à classe *Avaliação.* Antes era considerada como uma utilização em comum entre *App* e *Utilizador*, mas depois foi decidido coloca-la como uma classe associada a *Utilizador*.

# 5 – Conclusão

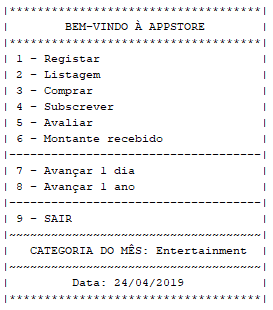
Como conclusão pretende-se reforçar que o trabalho demonstrou que a labor de programar é feita progressivamente, desde a planificação inicial com um diagrama UML até a parte de analisar o que é preciso fazer para que certa ação resulte como é intencionado.

Também foi reforçada a ideia da utilidade das classes e o desempenho no código em termos de organização e facilidade de localizar os elementos de um programa.

Em termos de trabalho, o avance foi progressivo sem a existência de problemas graves, considerando também que representa uma ajuda para afixar conhecimentos sobre a programação em linguagem Java para trabalhos futuros.

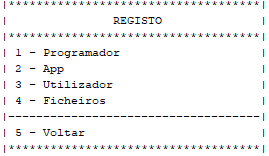
# 6 – Anexo (Imagens de output)

Menu Principal (Anexo 6.1

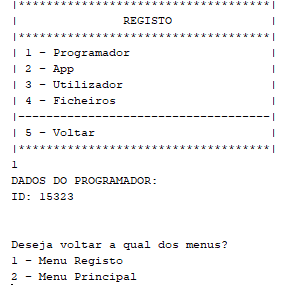


# 

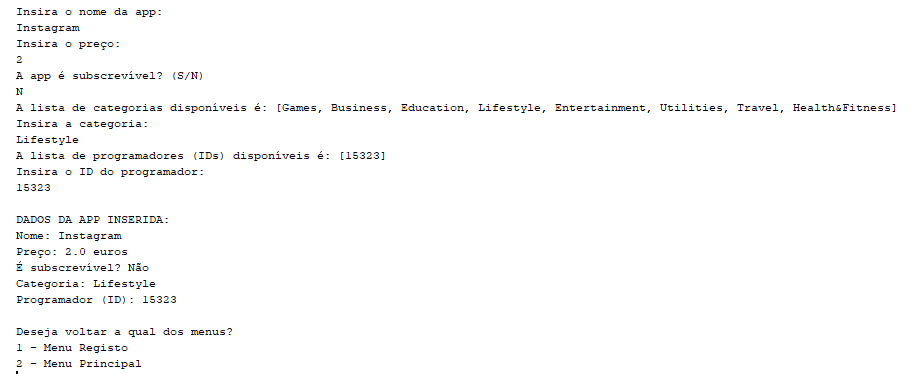
Menu Registo (Anexo 6.2)

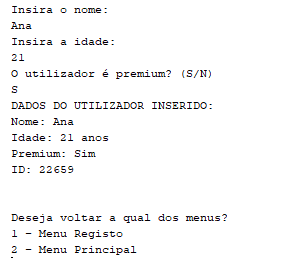


Registo de um novo programador (Anexo 6.3)

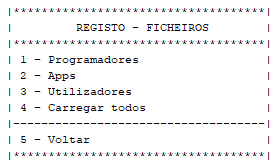


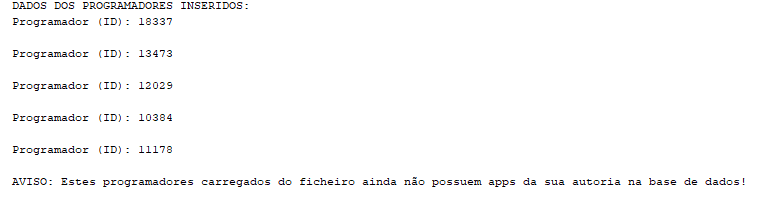
Registar nova app (Anexo 6.4)

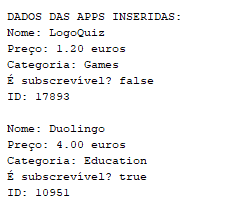


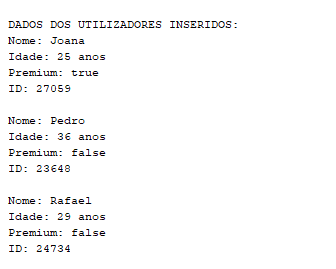
Registar novo utilizador (Anexo 6.5)

Inserção de ficheiros (Anexo 6.6)

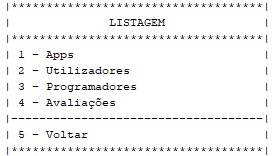




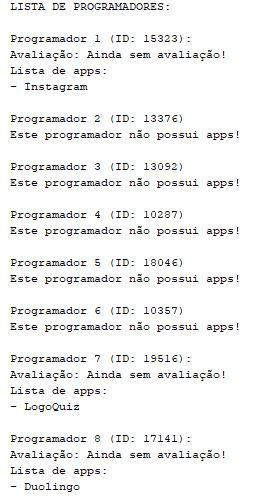




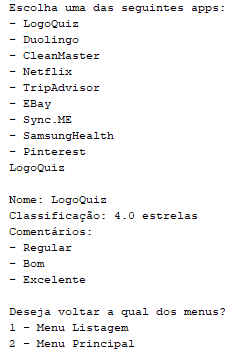
Menu Listagem (Anexo 6.7)



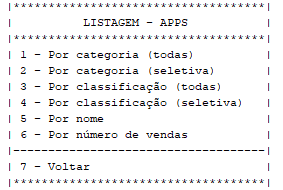
Listagem de Programadores (Anexo 6.8)



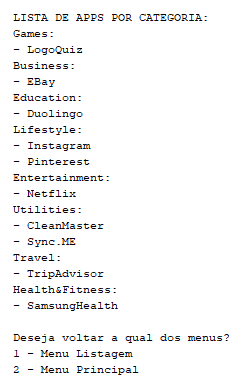
Listagem de Avaliações (Anexo 6.9)



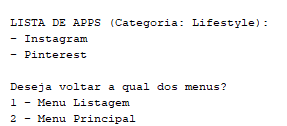
Menu Listagem – Apps (Anexo 6.10)



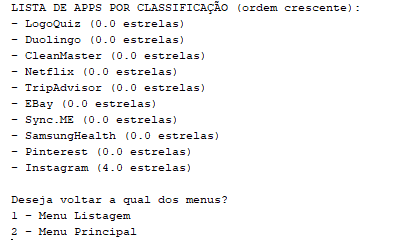
Listagem por categoria (todas) (Anexo 6.11)



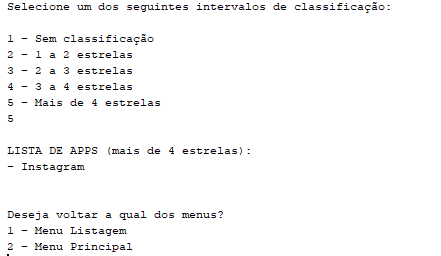
Listar por Categoria (Seletiva) (6.12)



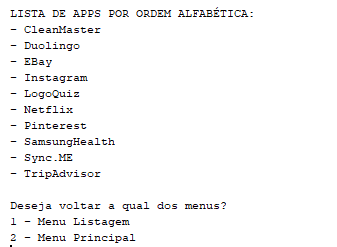
Listar por Classificação (6.13)



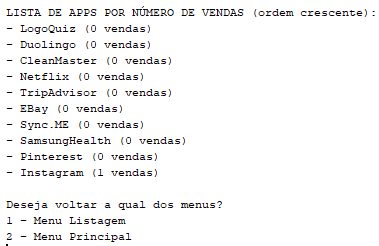
Lista por Classificação (Seletiva) (6.14)



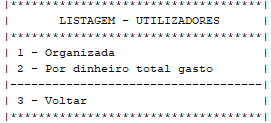
Lista por Nome (6.15)



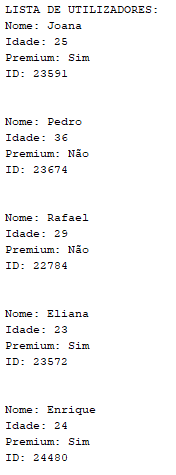
Lista por Número de Vendas (6.16)



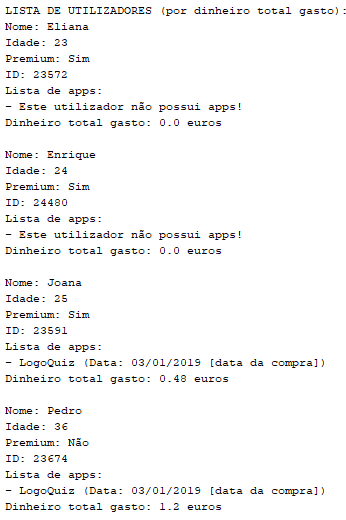
Menu Listagem – Utilizadores (6.17)

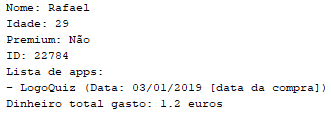


Lista Organizada de Utilizadores (Anexo 6.18)

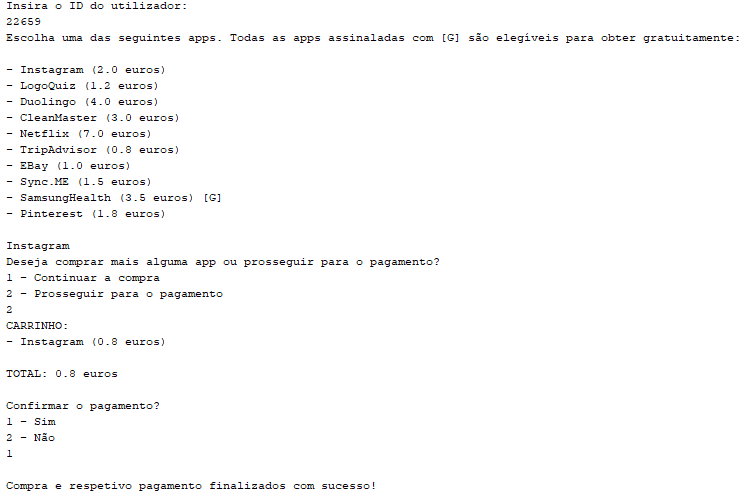


Lista de Utilizadores segundo o dinheiro gasto (Anexo 6.19)

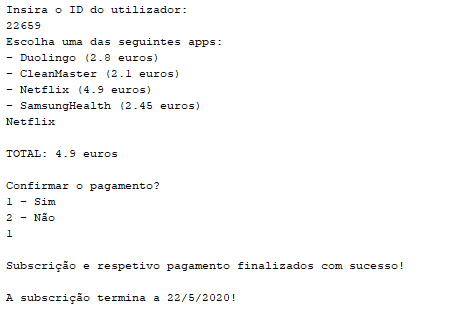




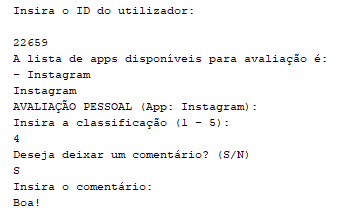
Exemplo de comprar (6.20)



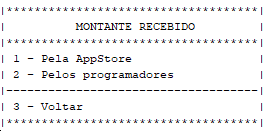
Exemplo de Subscrever (6.21)



Exemplo de Avaliar (6.22)



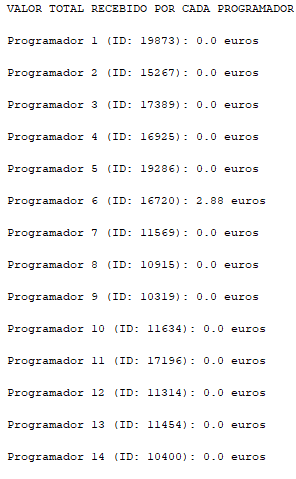
Menu Montante Recebido (6.23)



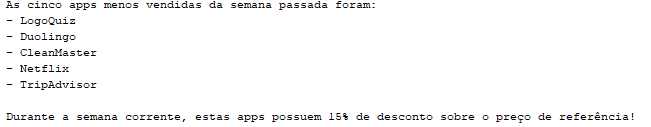
Valor total da App Store (6.24)



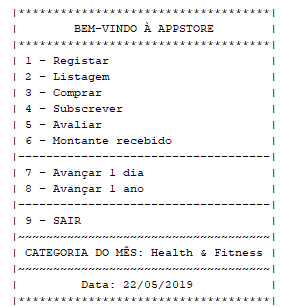
Valor total recebido por cada programador (6.25)



Mostrar as cinco apps menos vendidas (Anexo 6.26)



Mostrar a categoria do mês (Anexo 6.27)



# 7 – Anexo (Código implementado)

**APPSTORE:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

**import** java**.**io**.\*;**

**import** java**.**util**.\*;**

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class AppStore **{**

/\*\*

\* **@param** args the command line arguments

\*/

private ArrayList**<**Utilizador**>** listaUtilizadores**;** //Lista de utilizadores (objetos da classe Utilizador)

private ArrayList**<**Programador**>** listaProgramadores**;** //Lista de programadores (objetos da classe Programador)

private ArrayList**<**String**>** listaCategorias**;** //Lista de categorias (lista de Strings)

private ArrayList**<**App**>** listaTodasApps**;** //Lista de apps (objetos da classe App)

private Data data**;** //Data atual (objeto da classe data)

private ArrayList**<**Integer**>** listaIDsUtilizadores**;** //Lista de IDs já utilizados pelos utilizadores existentes

private ArrayList**<**Integer**>** listaIDsProgramadores**;** //Lista de IDs já utilizados pelos programadores existentes

private Random random**;** //Objeto da classe Random, a qual possui um método que permite obter números aleatoriamente dentro de um determinado intervalo (e que é utilizado neste programa)

private String categoriaDoMes**;** //Variável correspondente ao nome da categoria do mês (String)

private double valorTotalAppStore**;** //Variável corresponde ao valor (monetário) total de ganhos da AppStore

private ArrayList**<**String**>** simOuNao**;** //Lista de strings constituída pelas strings "S" e "N" (utilizadas como resposta para perguntas de sim ou não)

public AppStore**(){**

listaCategorias **=** **new** ArrayList**<>();**

listaCategorias**.**addAll**(**Arrays**.**asList**(**"Games" **,** "Business" **,** "Education"**,** "Lifestyle"**,** "Entertainment" **,**"Utilities" **,** "Travel" **,** "Health&Fitness"**));** //Adiciona-se as Strings correspondentes aos nomes das várias categorias à lista de categorias

listaUtilizadores **=** **new** ArrayList**<>();** //Inicialização da lista de utilizadores

listaProgramadores **=** **new** ArrayList**<>();** //Inicialização da lista de programadores

listaIDsUtilizadores **=** **new** ArrayList**<>();** //Inicialização de IDs já utilizados pelos utilizadores existentes

listaIDsProgramadores **=** **new** ArrayList**<>();** //Inicialização de IDs já utilizados pelos programadores existentes

listaTodasApps **=** **new** ArrayList**<>();** //Inicialização da lista de apps

data **=** **new** Data**();** //Inicialização da variável "data", que corresponde à data atual (objeto da classe Data)

random **=** **new** Random**();** //Inicialização da variável "random"

categoriaDoMes **=** **null;**

valorTotalAppStore **=** 0.0**;**

simOuNao **=** **new** ArrayList**<>();**

simOuNao**.**addAll**(**Arrays**.**asList**(**"S"**,**"N"**));** //Adiciona-se à lista anteriormente criada as Strings correspondentes às duas opções que servem de resposta às perguntas de "sim ou não", sendo estas Strings "S" e "N"

**}**

public void menuPrincipal**()** **throws** IOException**{**

String menu **=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //Display correspondente ao menu principal.

menu **+=** "| BEM-VINDO À APPSTORE |\n"**;** //Neste display são apresentadas as várias opções

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //correspondentes às funcionalidades da AppStore,

menu **+=** "| 1 - Registar |\n"**;** //as opções de avanço do tempo (para fins de simulação),

menu **+=** "| 2 - Listagem |\n"**;** //a categoria do mês e ainda a data atual.

menu **+=** "| 3 - Comprar |\n"**;**

menu **+=** "| 4 - Subscrever |\n"**;**

menu **+=** "| 5 - Avaliar |\n"**;**

menu **+=** "| 6 - Montante recebido |\n"**;**

menu **+=** "|------------------------------------|\n"**;**

menu **+=** "| 7 - Avançar 1 dia |\n"**;**

menu **+=** "| 8 - Avançar 1 ano |\n"**;**

menu **+=** "|------------------------------------|\n"**;**

menu **+=** "| 9 - SAIR |\n"**;**

menu **+=** "|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|\n"**;**

**if** **(**categoriaDoMes**.**equals**(**"Games"**))** //Apresentação do nome da categoria do mês

menu **+=** "| CATEGORIA DO MÊS: Games |\n"**;**

**else** **if** **(**categoriaDoMes**.**equals**(**"Business"**))**

menu **+=** "| CATEGORIA DO MÊS: Business |\n"**;**

**else** **if** **(**categoriaDoMes**.**equals**(**"Education"**))**

menu **+=** "| CATEGORIA DO MÊS: Education |\n"**;**

**else** **if** **(**categoriaDoMes**.**equals**(**"Lifestyle"**))**

menu **+=** "| CATEGORIA DO MÊS: Lifestyle |\n"**;**

**else** **if** **(**categoriaDoMes**.**equals**(**"Entertainment"**))**

menu **+=** "| CATEGORIA DO MÊS: Entertainment |\n"**;**

**else** **if** **(**categoriaDoMes**.**equals**(**"Utilities"**))**

menu **+=** "| CATEGORIA DO MÊS: Utilities |\n"**;**

**else** **if** **(**categoriaDoMes**.**equals**(**"Travel"**))**

menu **+=** "| CATEGORIA DO MÊS: Travel |\n"**;**

**else**

menu **+=** "| CATEGORIA DO MÊS: Health & Fitness |\n"**;**

menu **+=** "|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|\n"**;**

menu **+=** "| "**+**data**.**mostraData**()+**" |\n"**;** //O objeto "data" (da classe Data) é constituído pelas variáveis dia, mês e ano. O método mostraData trata de

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //apresentar estas mesmas variáveis na forma "dia/mês/ano".

System**.**out**.**print**(**menu**);** //Apresentação do display do menu principal na consola

**switch(**getOpcao**()){**

**case** 1 **:**

menuRegisto**();**

**break;**

**case** 2 **:**

menuListagem**();**

**break;**

**case** 3 **:**

inicioCompraOuSubscricao**(false);**

**break;**

**case** 4 **:**

inicioCompraOuSubscricao**(true);**

**break;**

**case** 5 **:**

avaliarAplicacao**();**

**break;**

**case** 6 **:**

menuMontanteRecebido**();**

**break;**

**case** 7 **:**

data**.**avancarDia**();**

cincoMenosVendidas**(false);**

categoriaDoMes**(false);**

verificarSubscricoes**();**

menuPrincipal**();**

**break;**

**case** 8 **:**

data**.**avancarAno**();**

cincoMenosVendidas**(true);**

categoriaDoMes**(true);**

verificarSubscricoes**();**

menuPrincipal**();**

**break;**

**case** 9 **:**

System**.**out**.**println**(**"Volte sempre!"**);**

**break;**

**default:**

opcaoErrada**();**

menuPrincipal**();**

**break;**

**}**

**}**

private void menuRegisto**()throws** IOException**{** //Método de apresentação do display do menu de registo e que define o que acontece de acordo com a opção escolhida

String menu **=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //Display correspondente ao menu de registos

menu **+=** "| REGISTO |\n"**;** //Neste display são apresentadas as várias opções

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //de registo. Neste caso, o utilizador pode

menu **+=** "| 1 - Programador |\n"**;** //escolher registar um programador, uma app,

menu **+=** "| 2 - App |\n"**;** //um utilizador ou apresentar o menu que apresenta

menu **+=** "| 3 - Utilizador |\n"**;** //as várias opções de registo através de ficheiros.

menu **+=** "| 4 - Ficheiros |\n"**;**

menu **+=** "|------------------------------------|\n"**;**

menu **+=** "| 5 - Voltar |\n"**;**

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;**

System**.**out**.**print**(**menu**);**

**switch(**getOpcao**()){**

**case** 1 **:**

novoProgramador**();**

**break;**

**case** 2 **:**

novaApp**();**

**break;**

**case** 3 **:**

novoUtilizador**();**

**break;**

**case** 4 **:**

registoPorFicheiros**();**

**break;**

**case** 5 **:**

menuPrincipal**();**

**break;**

**default:**

opcaoErrada**();**

menuRegisto**();**

**break;**

**}**

**}**

private void novoProgramador**()throws** IOException**{** //Método de criação de um novo programador (objeto da classe Programador) e respetiva impressão dos seus dados

System**.**out**.**println**(**"DADOS DO PROGRAMADOR:"**);** //Os dados do programador criado são imprimidos (neste caso, é só ID do mesmo)

Programador novoProgramador **=** **new** Programador**();** //Criação e instanciação do novo programador (objeto da classe Programador)

int IDP **=** novoProgramador**.**gerarIDProgramador**(this);** //Geração do ID do novo programador criado

listaIDsProgramadores**.**add**(**IDP**);** //Adiciona o ID do novo programador à lista de IDs já utilizados pelos outros programadores existentes

System**.**out**.**println**(**"ID: "**+**IDP **+** "\n" **);** //Imprime o valor do ID deste programador que foi criado

listaProgramadores**.**add**(**novoProgramador**);** //Adiciona este novo programador (objeto da classe Programador) à lista de programadores

voltarMenuRegisto**();**

**}**

private void novaApp**()throws** IOException**{** //Método de criação de uma nova app (objeto da classe App) e respetiva impressão dos seus dados

**try{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

**if** **(**listaProgramadores**.**isEmpty**())** **{** //Caso a lista de programadores esteja vazia, é imprimida uma mensagem de erro informando que não ainda não existem programadores

System**.**out**.**println**(**"Ainda não existem programadores!"**);**

menuRegisto**();**

**}**

**else** **{**

Scanner scannerDados **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

System**.**out**.**println**(**"Insira o nome da app: "**);**

ArrayList**<**String**>** listaNomesApps **=** **new** ArrayList**<>();** //Criação de um ArrayList de Strings que será preenchido pelos nomes das apps existentes

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps e adiciona o respetivo nome de cada app à lista (neste caso, ArrayList) acima criada

listaNomesApps**.**add**(**listaTodasApps**.**get**(**i**).**getNome**());**

**}**

String nome **=** verificarOpcaoNomeApp**(**scannerDados**,** listaNomesApps**);** //Iguala a variável do tipo String "nome" (que corresponde ao nome da app) ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir uma das Strings presentes na lista de nomes das apps e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então a categoria escolhida, dependendo do que o utilizador inseriu no input.

System**.**out**.**println**(**"Insira o preço: "**);**

double preco **=** scannerDados**.**nextDouble**();** //Iguala a variável do tipo double "preco" (que corresponde ao preço da app) ao input do teclado

System**.**out**.**println**(**"A app é subscrevível? (S/N)"**);**

String opcaoSubscrevivel **=** verificarOpcao**(**scannerDados**,** simOuNao**);** //Iguala a variável do tipo String "opcaoSubscrevivel" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir uma das Strings "S" ou "N" e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então uma dessas duas Strings, dependendo do que o utilizador inseriu no input.

System**.**out**.**println**(**"A lista de categorias disponíveis é: " **+** listaCategorias **+** "\nInsira a categoria:"**);**

String categoria **=** verificarOpcao**(**scannerDados**,** listaCategorias**);** //Iguala a variável do tipo String "categoria" (que corresponde à categoria da app) ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir uma das Strings presentes na lista de categorias e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então a categoria escolhida, dependendo do que o utilizador inseriu no input.

System**.**out**.**println**(**"A lista de programadores (IDs) disponíveis é: " **+** listaIDsProgramadores **+** "\nInsira o ID do programador:" **);** //Apresenta uma lista com os IDs dos programadores já existentes

int id **=** verificarOpcaoID**(**scannerDados**,** listaIDsProgramadores**);** //Iguala a variável do tipo int "ID" (que corresponde ao ID do programador da app) ao valor retornado pelo método verificarOpcaoID, que neste caso verifica se o utilizador está a inserir um dos IDs presentes na lista de programadores e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então o ID do programador escolhido, dependendo do que o utilizador inseriu no input.

System**.**out**.**println**(**"\n" **+**"DADOS DA APP INSERIDA:"**);** //Os dados da app inserida são imprimidos

System**.**out**.**println**(**"Nome: "**+**nome**);**

System**.**out**.**println**(**"Preço: "**+**preco**+** " euros"**);**

boolean subscrevivel **=** **false;**

**if** **(**opcaoSubscrevivel**.**equals**(**"S"**))** **{**

System**.**out**.**println**(**"É subscrevível? Sim"**);**

subscrevivel **=** **true;**

**}**

**else** **{**

System**.**out**.**println**(**"É subscrevível? Não"**);**

**}**

System**.**out**.**println**(**"Categoria: "**+**categoria**);**

System**.**out**.**println**(**"Programador (ID): "**+**id**);**

Programador programador **=** getProgramadorPorID**(**id**);** //Cria um novo objeto da classe "Programador", que neste caso corresponderá ao programador (objeto) correspondente ao ID inserido, obtido através do método getProgramadorPorID.

App novaApp **=** **new** App**(**nome**,** preco**,** categoria**,** programador**,** subscrevivel**);** //Cria a nova app (objeto da classe App) com os respetivos dados inseridos

listaTodasApps**.**add**(**novaApp**);** //Adiciona a nova app (objeto da classe App) à lista de apps existentes

programador**.**addAppProgramador**(**novaApp**);** //Adiciona a nova app à lista de apps do programador em questão

voltarMenuRegisto**();**

**}**

**}**

**catch(**InputMismatchException ime**){** //Caso tenha ocorrido algum erro em que o utilizador inseriu letras em vez de números ou números em vez de letras, aparece a mensagem de erro abaixo e este método é reinicializado.

System**.**out**.**print**(**"Introdução inválida! O processo de criação de uma app irá ser reinicializado... \n"**);**

novaApp**();**

**}**

**}**

private void novoUtilizador**()throws** IOException**{** //Método de criação de um novo utilizador (objeto da classe Utilizador) e respetiva impressão dos seus dados

**try{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

Scanner scannerDados **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

System**.**out**.**println**(**"Insira o nome: "**);**

String nome **=** scannerDados**.**next**();** //Iguala a variável do tipo String "nome" (que corresponde ao nome do utilizador) ao input do teclado

System**.**out**.**println**(**"Insira a idade: "**);**

int idade **=** scannerDados**.**nextInt**();** //Iguala a variável do tipo int "idade" (que corresponde à idade do utilizador) ao input do teclado

System**.**out**.**println**(**"O utilizador é premium? (S/N)"**);**

String opcaoPremium **=** verificarOpcao**(**scannerDados**,** simOuNao**);** //Iguala a variável do tipo String "opcaoPremium" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir uma das Strings "S" ou "N" e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então uma dessas duas Strings, dependendo do que o utilizador inseriu no input.

boolean premium **=** **false;**

**if** **(**opcaoPremium**.**equals**(**"S"**))** **{** //Se a variável "opcaoPremium" corresponde a "S", a variável premium passa a ser true, significando que o utilizador que está a ser inserido é premium. Caso contrário, esta variável continua false, significando que o utilizador NÃO é premium.

premium **=** **true;**

**}**

**if** **(!**listaUtilizadores**.**isEmpty**())** **{** //Se a lista de utilizadores está vazia, o código abaixo não chega a ser executado, uma vez que este trata de perguntar ao utilizador se ele foi convidado por outro utilizador, não fazendo sentido perguntar isso se não há outros utilizadores.

System**.**out**.**println**(**"O utilizador foi convidado por outro? (S/N)"**);**

String opcaoConvidado **=** verificarOpcao**(**scannerDados**,** simOuNao**);** //Iguala a variável do tipo String "opcaoPremium" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir uma das Strings "S" ou "N" e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então uma dessas duas Strings, dependendo do que o utilizador inseriu no input.

**if** **(**opcaoConvidado**.**equals**(**"S"**))** **{** //Se a variável "opcaoConvidado" corresponde a "S":

System**.**out**.**println**(**"Indique o ID do utilizador que o convidou: "**);**

int id **=** verificarOpcaoID**(**scannerDados**,** listaIDsUtilizadores**);** //Iguala a variável do tipo int "ID" (que corresponde ao ID do utilizador que convidou este novo utilizador) ao valor retornado pelo método verificarOpcaoID, que neste caso verifica se o utilizador está a inserir um dos IDs presentes na lista de utilizadores e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então o ID do utilizador escolhido, dependendo do que o utilizador inseriu no input.

Utilizador utilizador **=** getUtilizadorPorID**(**id**);** //Cria um novo objeto da classe "Utilizador", que neste caso corresponderá ao utilizador (objeto) correspondente ao ID inserido, obtido através do método getUtilizadorPorID.

utilizador**.**addUtilizadorAngariado**();** //Incrementa o número de utilizadores angariados por esse utilizador em 1

**}**

**}**

System**.**out**.**println**(**"DADOS DO UTILIZADOR INSERIDO:"**);** //Os dados do utilizador inserido são imprimidos

System**.**out**.**println**(**"Nome: "**+**nome**);**

System**.**out**.**println**(**"Idade: "**+**idade**+** " anos"**);**

**if** **(**premium**)** **{**

System**.**out**.**println**(**"Premium: Sim"**);** **}**

**else** **{**

System**.**out**.**println**(**"Premium: Não"**);** **}**

Utilizador novoUtilizador **=** **new** Utilizador**(**nome**,** idade**,** premium**);** //Cria o novo utilizador (objeto da classe Utilizador) com os respetivos dados inseridos

int ID **=** novoUtilizador**.**gerarIDUtilizador**(this);** //Geração do ID do novo utilizador criado

listaIDsUtilizadores**.**add**(**ID**);** //Adiciona o ID do novo utilizador à lista de IDs já utilizados pelos outros utilizadores existentes

System**.**out**.**println**(**"ID: "**+**ID **+** "\n"**);**

listaUtilizadores**.**add**(**novoUtilizador**);** //Adiciona este novo utilizador (objeto da classe Utilizador) à lista de utilizadores

voltarMenuRegisto**();**

**}**

**catch** **(**InputMismatchException ime**)** **{** //Caso tenha ocorrido algum erro em que o utilizador inseriu letras em vez de números ou números em vez de letras, aparece a mensagem de erro abaixo e este método é reinicializado.

System**.**out**.**print**(**"Introdução inválida! O processo de criação de um novo utilizador irá ser reinicializado... \n"**);**

novoUtilizador**();**

**}**

**}**

private void registoPorFicheiros**()** **throws** IOException**{**

String menu **=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //Display correspondente ao menu de registo através de ficheiros

menu **+=** "| REGISTO - FICHEIROS |\n"**;** //Neste display são apresentadas as várias opções de

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //registo por ficheiros. Neste caso, o utilizador pode

menu **+=** "| 1 - Programadores |\n"**;** //carregar e registar programadores, apps, utilizadores

menu **+=** "| 2 - Apps |\n"**;** //ou então todos estes três em simultâneo.

menu **+=** "| 3 - Utilizadores |\n"**;**

menu **+=** "| 4 - Carregar todos |\n"**;**

menu **+=** "|------------------------------------|\n"**;**

menu **+=** "| 5 - Voltar |\n"**;**

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;**

System**.**out**.**print**(**menu**);**

**switch(**getOpcao**()){**

**case** 1 **:**

adicionarFicheirosProgramadores**();**

voltarMenuRegisto**();**

**break;**

**case** 2 **:**

adicionarFicheirosApps**();**

voltarMenuRegisto**();**

**break;**

**case** 3 **:**

adicionarFicheirosUtilizadores**();**

voltarMenuRegisto**();**

**break;**

**case** 4 **:**

adicionarFicheirosProgramadores**();**

adicionarFicheirosApps**();**

adicionarFicheirosUtilizadores**();**

voltarMenuRegisto**();**

**break;**

**case** 5 **:**

menuPrincipal**();**

**break;**

**default:**

opcaoErrada**();**

registoPorFicheiros**();**

**break;**

**}**

**}**

private void adicionarFicheirosApps**()** **throws** IOException **{** //Método de criação de novas apps (objetos da classe App) através do carregamento de ficheiro e respetiva impressão dos seus dados

**try** **{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

FileReader inStream **=** **new** FileReader**(**"Apps.txt"**);** //Aberuta de uma stream para o ficheiro "Apps.txt", a classe FileReader permite abrir uma stream de leitura de carateres

BufferedReader bR **=** **new** BufferedReader**(**inStream**);** //Criação de um buffer para aumento da eficiênica, pois é mais de uma linha que ser lida

String linha **=** bR**.**readLine**();** //Leitura da linha

String nome**,** categoria**;** //Declaração das variáveis "nome" e "categoria" do tipo String (corresponde ao nome da app e sua categoria respectivamnete)

double preco**;** //Declaração da variável "preco" do tipo Double (corresponde ao preço da app)

boolean subscrevivel**;** //Declaração da variável "subscrevivel" do tipo Boolean (corresponde se a app é ou não subscrevivel)

int id**;** //Declaração da variável "id" do tipo Integer (corresponde ao id do programador que a desenvolveu)

System**.**out**.**println**(**"DADOS DAS APPS INSERIDAS:"**);**

**while** **(**linha **!=** **null)** **{** //O ciclo while só termina quando a linha for igual null, logo quando não existem linhas

linha **=** bR**.**readLine**();** //Lê a linha seguinte

nome **=** String**.**valueOf**(**linha**);** //Atribuiu a variável "nome" o valor da linha que estará o nome da app

System**.**out**.**println**(**"Nome: " **+** linha**);** //Imprime no display o nome da app

linha **=** bR**.**readLine**();** //Lê a linha seguinte

preco **=** Double**.**parseDouble**(**linha**);** //Atribui a variável "preco" o valor da linha mas convertendo o valor da linha em double

System**.**out**.**println**(**"Preço: " **+** linha **+** " euros"**);** //Imprime no display o preço da app

linha **=** bR**.**readLine**();** //Lê a linha seguinte

categoria **=** String**.**valueOf**(**linha**);** //Atribui a variável "categoria" o valor da linha que conterá a categoria par app

System**.**out**.**println**(**"Categoria: " **+** linha**);** //Imprime no display a categoria da app

linha **=** bR**.**readLine**();** //Lê a linha seguinte

subscrevivel **=** Boolean**.**parseBoolean**(**linha**);** //Atribui a variavel "subscrevivel" o valor da linha mas convertendo o valor da linha em boolean

System**.**out**.**println**(**"É subscrevível? " **+** linha**);** //Imprime no display se aplicação é subscrevível ou não

Programador programador **=** **new** Programador**();** //Cria um novo objeto da classe "Programador"

id**=** programador**.**gerarIDProgramador**(this);** //Geração do ID do novo programador criado

System**.**out**.**println**(**"ID: " **+** id **+** "\n"**);** //Imprime o valor do ID deste programador que foi criado

listaIDsProgramadores**.**add**(**id**);** //Adiciona o ID do novo programador à lista de IDs já utilizados pelos outros programadores existentes

App app **=** **new** App**(**nome**,** preco**,** categoria**,** programador**,** subscrevivel**);** //Cria a nova app (objeto da classe App) com os respetivos dados tirados do ficheiro

programador**.**addAppProgramador**(**app**);** //Adiciona a nova app à lista de apps do programador em questão

listaProgramadores**.**add**(**programador**);** //Adiciona o novo programador (objeto da classe Programador) à lista de programadores existentes

listaTodasApps**.**add**(**app**);** //Adiciona a nova app (objeto da classe App) à lista de apps existentes

linha **=** bR**.**readLine**();** //Lê a linha seguinte

//Uma linha em vazio para separar a informação da app seguinte

**}**

bR**.**close**();** //Fechar a stream

**}**

**catch** **(**IOException ioe**)** **{** //Caso tenha ocorrido um erro no carregamento do ficheiro, é apresentada a mensagem de erro abaixo e o método é reinicializado

System**.**out**.**println**(**"Ocorreu um erro no carregamento do ficheiro das apps!"**);**

menuPrincipal**();**

**}**

**}**

private void adicionarFicheirosProgramadores**()throws** IOException**{** //Método de criação de novos programadores (objetos da classe Programador) através do carregamento de ficheiro e respetiva impressão dos seus dados

**try** **{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

FileReader inStream **=** **new** FileReader**(**"Programadores.txt"**);** //Aberuta de uma stream para o ficheiro "Programadores.txt", a classe FileReader permite abrir uma stream de leitura de carateres

BufferedReader bR **=** **new** BufferedReader**(**inStream**);** //Criação de um buffer para aumento da eficiênica, pois é mais de uma linha que ser lida

String linha **=** bR**.**readLine**();** //Leitura da linha

int quantidade**=**Integer**.**parseInt**(**linha**);** //Declaração da variàvel "quantidade" que terá o valor da linha, após a conversão desse valor para inteiro, pois no ficheiro conterá um único número que corresponde quantos programadores serão criados

**while** **(**linha **!=null){** //O ciclo while só termina quando a linha for igual null, logo quando não existem linhas

linha **=** bR**.**readLine**();** //Lê a linha seguinte

**}**

System**.**out**.**println**(**"DADOS DOS PROGRAMADORES INSERIDOS:"**);**

**for** **(**int i **=** 0 **;** i **<**quantidade**;** i **++){** //Ciclo para a criação dos novos programadores correspondendo a quantidade colocada no ficheiro

Programador novoProgramador **=** **new** Programador**();** //Cria um novo objeto da classe "Programador"

int ID **=** novoProgramador**.**gerarIDProgramador**(this);** //Geração do ID do novo programador criado

listaIDsProgramadores**.**add**(**ID**);** //Adiciona o ID do novo programador à lista de IDs já utilizados pelos outros programadores existentes

System**.**out**.**println**(**"Programador (ID): "**+**ID **+** "\n" **);** //Imprime o valor do ID deste programador que foi criado

listaProgramadores**.**add**(**novoProgramador**);** //Adiciona o novo programador (objeto da classe Programador) à lista de programadores existentes

**}**

System**.**out**.**println**(**"AVISO: Estes programadores carregados do ficheiro ainda não possuem apps da sua autoria na base de dados! \n "**);**

bR**.**close**();** //Fechar a stream

**}**

**catch** **(**IOException ioe**)** **{** //Caso tenha ocorrido um erro no carregamento do ficheiro, é apresentada a mensagem de erro abaixo e o método é reinicializado

System**.**out**.**println**(**"Ocorreu um erro no carregamento do ficheiro dos programadores!"**);**

menuPrincipal**();**

**}**

**}**

private void adicionarFicheirosUtilizadores**()** **throws** IOException **{** //Método de criação de novos utilizadores (objetos da classe Utilizador) através do carregamento de ficheiro e respetiva impressão dos seus dados

**try** **{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

FileReader inStream **=** **new** FileReader**(**"Utilizadores.txt"**);**//Aberuta de uma stream para o ficheiro "Utilizadores.txt", a classe FileReader permite abrir uma stream de leitura de carateres

BufferedReader bR **=** **new** BufferedReader**(**inStream**);** //Criação de um buffer para aumento da eficiênica, pois é mais de uma linha que ser lida

String linha **=** bR**.**readLine**();** //Leitura da linha

String nome**;** //Declaração das variáveis "nome" do tipo String (corresponde ao nome do utilizador)

int idade**;** //Declaração da variável "idade" do tipo Integer (corresponde a idade do utilizador)

boolean premium**;** //Declaração da variável "premium" do tipo Boolean (corresponde se o utilizador é ou não premium)

System**.**out**.**println**(**"DADOS DOS UTILIZADORES INSERIDOS:"**);**

**while** **(**linha **!=** **null)** **{** //O ciclo while só termina quando a linha for igual null, logo quando não existem linhas

linha **=** bR**.**readLine**();** //Lê a linha seguinte

nome **=** String**.**valueOf**(**linha**);** //Atribuiu a variável "nome" o valor da linha que estará o nome do utilizador

System**.**out**.**println**(**"Nome: " **+** linha**);** //Imprime no display o nome do utilizador

linha **=** bR**.**readLine**();** //Lê a linha seguinte

idade **=** Integer**.**parseInt**(**linha**);** //Atribui a variável "idade" o valor da linha mas convertendo o valor da linha em integer

System**.**out**.**println**(**"Idade: " **+** linha **+** " anos"**);** //Imprime no display a idade do utilizador

linha **=** bR**.**readLine**();** //Lê a linha seguinte

premium **=** Boolean**.**parseBoolean**(**linha**);** //Atribui a variavel "premium" o valor da linha mas convertendo o valor da linha em boolean

System**.**out**.**println**(**"Premium: " **+** linha**);** //Imprime no display se o utilizador é premium ou não

Utilizador utilizador**=** **new** Utilizador**(**nome**,** idade**,** premium**);** //Cria o novo utilizador (objeto da classe Utilizador) com os respetivos dados tirados do ficheiro

int id**=** utilizador**.**gerarIDUtilizador**(this);** //Geração do ID do novo utilizador criado

System**.**out**.**println**(**"ID: "**+**id **+** "\n" **);** //Imprime o valor do ID deste utilizador que foi criado

listaIDsUtilizadores**.**add**(**id**);** //Adiciona o ID do novo utilizador à lista de IDs já utilizados pelos outros utilizadores existentes

listaUtilizadores**.**add**(**utilizador**);** //Adiciona o novo utilizador (objeto da classe Utilizador) à lista de utilizadores existentes

linha **=** bR**.**readLine**();** //Lê a linha seguinte

//Uma linha em vazio para separar a informação do utilizador seguinte

**}**

bR**.**close**();** //Fechar a stream

**}**

**catch** **(**IOException ioe**)** **{** //Caso tenha ocorrido um erro no carregamento do ficheiro, é apresentada a mensagem de erro abaixo e o método é reinicializado

System**.**out**.**println**(**"Ocorreu um erro no carregamento do ficheiro dos utilizadores!"**);**

menuPrincipal**();**

**}**

**}**

private void menuListagem**()throws** IOException**{**

String menu **=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //Display correspondente ao menu da listagem

menu **+=** "| LISTAGEM |\n"**;** //Neste display são apresentadas as várias opções de

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //listagem. Neste caso, o utilizador pode escolher

menu **+=** "| 1 - Apps |\n"**;** //ir para o menu de seleção do tipo de listagem das apps, ir para o

menu **+=** "| 2 - Utilizadores |\n"**;** //menu de seleção do tipo de listagem de utilizadores, listar os

menu **+=** "| 3 - Programadores |\n"**;** //programadores ou então listar as avaliações de uma determinada app a escolher.

menu **+=** "| 4 - Avaliações |\n"**;**

menu **+=** "|------------------------------------|\n"**;**

menu **+=** "| 5 - Voltar |\n"**;**

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;**

System**.**out**.**print**(**menu**);**

**switch(**getOpcao**()){**

**case** 1 **:**

menuListagemApps**();**

voltarMenuListagem**();**

**break;**

**case** 2 **:**

menuListagemUtilizadores**();**

voltarMenuListagem**();**

**break;**

**case** 3 **:**

listarProgramadores**();**

voltarMenuListagem**();**

**break;**

**case** 4 **:**

listarAvaliacoes**();**

voltarMenuListagem**();**

**break;**

**case** 5 **:**

menuPrincipal**();**

**break;**

**default:**

opcaoErrada**();**

menuListagem**();**

**break;**

**}**

**}**

private void listarProgramadores**()throws** IOException**{** //Método que lista os programadores e seus respetivos IDs

System**.**out**.**println**(**"LISTA DE PROGRAMADORES:\n"**);**

**if(**listaProgramadores**.**isEmpty**()){** //Caso a lista de programadores esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem programadores!"**);**

**}**

**else** **{**

**for** **(**int i **=** 0**;** i **<** listaProgramadores**.**size**();** i **++){** //Percorre a lista de programadores. Caso a lista de apps desse programador esteja vazia, é apresentada uma mensagem que indica que este não possui apps, depois de ser apresentado o programador e o seu respetivo ID.

**if** **(**listaProgramadores**.**get**(**i**).**getListaAppsProgramador**().**isEmpty**()){**

System**.**out**.**println**(**"Programador " **+** **(**i**+**1**)** **+** " (ID: " **+**listaProgramadores**.**get**(**i**).**getIDProgramador**()** **+** ")"**);** //Imprime o valor (+1) do índice do programador (1, 2, 3...) de acordo com a ordem dos mesmos na lista de programadores e de seguida imprime o seu respetivo ID.

System**.**out**.**println**(**"Este programador não possui apps! \n"**);**

**}**

**else{**

System**.**out**.**println**(**"Programador " **+** **(**i**+**1**)** **+** " (ID: " **+** listaProgramadores**.**get**(**i**).**getIDProgramador**()** **+** "):"**);** //Imprime o valor (+1) do índice do programador (1, 2, 3...) de acordo com a ordem dos mesmos na lista de programadores e de seguida imprime o seu respetivo ID.

**if** **(**listaProgramadores**.**get**(**i**).**getAvaliacao**()** **==** 0.0**)** **{** //Caso a avaliação do respetivo programador seja "0.0", significando que ainda nenhuma das suas apps foi avaliada (uma vez que as classificações vão de a 1 a 5), é imprimida uma mensagem que indica que este programador ainda não tem uma avaliação.

System**.**out**.**println**(**"Avaliação: Ainda sem avaliação!"**);**

**}**

**else** **{** //Caso contrário, ou seja, caso o programador tenha uma avaliação, é imprimida a sua respetiva avaliação

System**.**out**.**println**(**"Avaliação: " **+** listaProgramadores**.**get**(**i**).**getAvaliacao**()** **+** " estrelas"**);**

**}**

System**.**out**.**println**(**"Lista de apps: "**);**

**for** **(**int j **=** 0**;** j **<**listaProgramadores**.**get**(**i**).**getListaAppsProgramador**().**size**();** j**++){** //Percorre a lista de apps deste programador e imprime o nome das mesmas, linha por linha.

System**.**out**.**println**(**"- " **+**listaProgramadores**.**get**(**i**).**getListaAppsProgramador**().**get**(**j**).**getNome**()** **);**

**}**

System**.**out**.**print**(**"\n"**);**

**}**

**}**

**}**

**}**

private void listarAvaliacoes**()** **throws** IOException**{** //Método que lista as avaliações das várias apps

**if** **(**listaTodasApps**.**isEmpty**())** **{** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

**}**

**else** **{**

Scanner scannerDados **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

System**.**out**.**println**(**"Escolha uma das seguintes apps: "**);**

ArrayList**<**String**>** listaNomesApps **=** **new** ArrayList**<>();** //Criação de um ArrayList de Strings que será preenchido pelos nomes das apps existentes

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps e adiciona o respetivo nome de cada app à lista (neste caso, ArrayList) acima criada

listaNomesApps**.**add**(**listaTodasApps**.**get**(**i**).**getNome**());**

System**.**out**.**println**(**"- " **+** listaNomesApps**.**get**(**i**)** **);** //Imprime o nome da respetiva app

**}**

String nome **=** verificarOpcao**(**scannerDados**,** listaNomesApps**);** //Iguala a variável do tipo String "nome" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir um dos nomes presentes na lista de nomes das apps e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então o nome da app escolhida, dependendo do que o utilizador inseriu no input.

App app **=** getAppPorNome**(**nome**);** //Cria um novo objeto da classe App, objeto esse que corresponderá à app escolhida.

System**.**out**.**println**(**"\nNome: " **+** app**.**getNome**());** //Imprime o nome da app escolhida

System**.**out**.**println**(**"Classificação: " **+** app**.**getClassificacao**()** **+** " estrelas"**);** //Imprime a classificação da app escolhida

System**.**out**.**println**(**"Comentários: "**);** //Imprime a lista de comentários da app escolhida

int contadorComentarios **=** 0**;**

**for** **(**int i**=**0 **;** i **<** app**.**getListaAvaliacoes**().**size**();** i **++){** //Percorre a lista de avaliações (constituída por uma classificação ou então uma classificação e um comentário associados)

**if** **(**app**.**getListaAvaliacoes**().**get**(**i**).**getComentario**()** **!=** **null)** **{** //Caso a avaliação em questão (objeto da classe Avaliação) presente na lista contenha um comentário associado:

System**.**out**.**println**(**"- "**+** app**.**getListaAvaliacoes**().**get**(**i**).**getComentario**());** //Imprime o comentário associado a essa mesma avaliação

contadorComentarios**++;** //Por cada comentário encontrado, é incrementado a variável do tipo int correspondente ao número de comentários

**}**

**}**

**if** **(**contadorComentarios **==** 0**)** **{** //Caso não tenham sido encontrados comentários na lista de avaliações, imprime uma mensagem indicando que a app em questão não possui comentários.

System**.**out**.**println**(**"Não há comentários.\n"**);**

**}**

**}**

**}**

private void menuListagemApps**()throws** IOException**{**

String menu **=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //Display correspondente à listagem de apps.

menu **+=** "| LISTAGEM - APPS |\n"**;** //Neste display são apresentadas as várias opções

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //de listagem de apps. Neste caso, o utilizador

menu **+=** "| 1 - Por categoria (todas) |\n"**;** //pode escolher listar todas apps por categoria,

menu **+=** "| 2 - Por categoria (seletiva) |\n"**;** //listar as apps de uma categoria à escolha,

menu **+=** "| 3 - Por classificação (todas) |\n"**;** //listar todas as apps por classificação,

menu **+=** "| 4 - Por classificação (seletiva) |\n"**;** //listar as apps dentro de um intervalo de classificação à escolha,

menu **+=** "| 5 - Por nome |\n"**;** //listar apps por nome ou entã listá-las pelo seu número

menu **+=** "| 6 - Por número de vendas |\n"**;** //de vendas.

menu **+=** "|------------------------------------|\n"**;**

menu **+=** "| 7 - Voltar |\n"**;**

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;**

System**.**out**.**print**(**menu**);**

**switch(**getOpcao**()){**

**case** 1 **:**

listarPorCategoria**();**

voltarMenuListagem**();**

**break;**

**case** 2 **:**

listarPorCategoriaSeletiva**();**

voltarMenuListagem**();**

**break;**

**case** 3 **:**

listarPorClassificacao**();**

voltarMenuListagem**();**

**break;**

**case** 4 **:**

listarPorClassificacaoSeletiva**();**

voltarMenuListagem**();**

**break;**

**case** 5 **:**

listarPorNome**();**

voltarMenuListagem**();**

**break;**

**case** 6 **:**

listarPorNumVendas**();**

voltarMenuListagem**();**

**break;**

**case** 7 **:**

menuPrincipal**();**

**break;**

**default:**

opcaoErrada**();**

menuListagemApps**();**

**break;**

**}**

**}**

private void listarPorCategoria**()throws** IOException**{** //Método de listagem das apps por categoria

**if** **(**listaTodasApps**.**isEmpty**()){** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

**}**

**else{**

System**.**out**.**println**(**"\nLISTA DE APPS POR CATEGORIA:"**);**

**for** **(**int i **=** 0**;** i **<** listaCategorias**.**size**();** i**++)** **{** //Percorre a lista de categorias e imprime o nome da categoria em questão

System**.**out**.**println**(**listaCategorias**.**get**(**i**)+**": "**);**

boolean existemAppsNestaCategoria **=** **false;**

**for** **(**int j **=** 0**;** j **<** listaTodasApps**.**size**();** j**++)** **{** //Percorre a lista de todas as apps

**if** **((**listaCategorias**.**get**(**i**)).**equals**(**listaTodasApps**.**get**(**j**).**getCategoria**())){** //Verifica se a categoria da app em questão corresponde à categoria em análise

System**.**out**.**println**(**"- "**+**listaTodasApps**.**get**(**j**).**getNome**());** //Caso se verifique isso, é imprimido o nome dessa app e o booleano "existeAlgumaAppNestaCategoria" passa a ser verdadeiro.

existemAppsNestaCategoria **=** **true;** **}**

**else** **if** **((**j **==** **(**listaTodasApps**.**size**()** **-** 1**))** **&&** **(**existemAppsNestaCategoria **==** **false))** **{** //Caso a lista de apps já tenha sido percorrida na sua totalidade e o booleano "existeAlgumaAppNestaCategoria" continua a false, significando que esta categoria não tem apps, é imprimida uma mensagem que indica este facto.

System**.**out**.**println**(**"- Não existem apps nesta categoria"**);**

**}**

**}**

**}**

**}**

**}**

private void listarPorCategoriaSeletiva**()throws** IOException**{** //Método de listagem das apps que se encontram dentro de uma determinada categoria selecionável

**if** **(**listaTodasApps**.**isEmpty**()){** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

**}**

**else{**

Scanner scannerDados **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

System**.**out**.**println**(**"A lista de categorias disponíveis é: " **+** listaCategorias **+** "\nInsira a categoria:"**);** //Imprime a lista de categorias

String categoria **=** verificarOpcao**(**scannerDados**,** listaCategorias**);** //Iguala a variável do tipo String "categoria" (que corresponde à categoria da app) ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir uma das Strings presentes na lista de categorias e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então a categoria escolhida, dependendo do que o utilizador inseriu no input.

System**.**out**.**println**(**"\nLISTA DE APPS (Categoria: " **+** categoria **+** "):"**);**

boolean existemAppsNaCategoria **=** **false;**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de todas as apps

**if** **(**listaTodasApps**.**get**(**i**).**getCategoria**().**equals**(**categoria**))** **{** //Verifica se a categoria da app em questão corresponde à categoria escolhida

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**());** //Caso se verifique isso, é imprimido o nome dessa app e o booleano "existemAppsNaCategoria" passa a ser verdadeiro.

existemAppsNaCategoria **=** **true;**

**}**

**else** **if** **((**i **==** **(**listaTodasApps**.**size**()** **-** 1**))** **&&** **(**existemAppsNaCategoria **==** **false))** **{** //Caso a lista de apps já tenha sido percorrida na sua totalidade e o booleano "existemAppsNaCategoria" continua a false, significando que esta categoria não tem apps, é imprimida uma mensagem que indica este facto.

System**.**out**.**println**(**"- Não existem apps nesta categoria!"**);** **}**

**}**

**}**

**}**

// MÉTODO COMPARE:

// Less than zero, if the first value is less than the second;

// Equal to zero, if the two values are equal;

// Greater than zero, if the first value is greater than the second;

private void listarPorClassificacao**(){** //Método de listagem das apps por classificação (ordem crescente)

**if(**listaTodasApps**.**isEmpty**()){** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

**}**

**else{**

Collections**.**sort**(**listaTodasApps**,** App**.**COMPARAR\_POR\_CLASSIFICACAO**);** //Reordena a lista de apps, ordenando-a de acordo com a classificação das apps (por ordem crescente)

System**.**out**.**println**(**"LISTA DE APPS POR CLASSIFICAÇÃO (ordem crescente):"**);**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps (já ordenada por classificação) e imprime o nome das mesmas juntamente com a sua respetiva classificação

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**()+**" ("**+**listaTodasApps**.**get**(**i**).**getClassificacao**()+**" estrelas)"**);**

**}**

**}**

**}**

private void listarPorClassificacaoSeletiva**()throws** IOException**{** //Método de listagem das apps cuja classificação se encontra dentro de um determinado intervalo de classificações selecionável

**if** **(**listaTodasApps**.**isEmpty**()){** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

**}**

**else{**

Scanner scannerDados **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

System**.**out**.**println**(**"Selecione um dos seguintes intervalos de classificação:"**);**

System**.**out**.**println**(**"\n1 - Sem classificação \n2 - 1 a 2 estrelas \n3 - 2 a 3 estrelas \n4 - 3 a 4 estrelas \n5 - Mais de 4 estrelas"**);**

int opcaoIntervalo **=** scannerDados**.**nextInt**();** //A variável "opcaoIntervalo" corresponde à opção selecionada pelo utilizador de acordo com de qual intervalo o mesmo pretende visualizar a lista de apps

boolean existemAppsNesteIntervalo **=** **false;**

**switch(**opcaoIntervalo**){**

**case** 1 **:** //Caso o utilizador selecione a opção 1

System**.**out**.**println**(**"\nLISTA DE APPS (sem classificação):\n"**);**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps

**if** **(**listaTodasApps**.**get**(**i**).**getClassificacao**()** **==** 0.0**)** **{**

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**());** //Imprime todas as apps com uma classificação correspondente a "0.0" (ou seja, aquelas apps que ainda não foram avaliadas e portanto ainda não têm classificação)

existemAppsNesteIntervalo **=** **true;** //Esta variável booleana serve para verificar se existem apps nesta categoria.

**}**

**else** **if** **(**i **==** **(**listaTodasApps**.**size**()** **-** 1**)** **&&** existemAppsNesteIntervalo**)** **{** //Caso não existam apps nesta categoria (facto verificado pela variável booleana " existemAppsNesteIntervalo", é imprimida uma mensagem que indica este facto.

System**.**out**.**println**(**"- Não existem apps cuja classificação se encaixa neste intervalo!"**);**

**}**

**}**

**break;**

**case** 2 **:** //Caso o utilizador selecione a opção 2

System**.**out**.**println**(**"\nLISTA DE APPS (1 a 2 estrelas):\n"**);**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps

**if** **(**listaTodasApps**.**get**(**i**).**getClassificacao**()** **>=** 1 **&&** listaTodasApps**.**get**(**i**).**getClassificacao**()** **<** 2**)** **{**

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**());** //Imprime todas as apps com uma classificação entre 1 (inclusive) e 2

existemAppsNesteIntervalo **=** **true;** //Esta variável booleana serve para verificar se existem apps nesta categoria.

**}**

**else** **if** **(**i **==** **(**listaTodasApps**.**size**()** **-** 1**)** **&&** existemAppsNesteIntervalo**)** **{** //Caso não existam apps nesta categoria (facto verificado pela variável booleana " existemAppsNesteIntervalo", é imprimida uma mensagem que indica este facto.

System**.**out**.**println**(**"- Não existem apps cuja classificação se encaixa neste intervalo!"**);**

**}**

**}**

**break;**

**case** 3 **:** //Caso o utilizador selecione a opção 3

System**.**out**.**println**(**"\nLISTA DE APPS (2 a 3 estrelas):\n"**);**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps

**if** **(**listaTodasApps**.**get**(**i**).**getClassificacao**()** **>=** 2 **&&** listaTodasApps**.**get**(**i**).**getClassificacao**()** **<** 3**)** **{**

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**());** //Imprime todas as apps com uma classificação entre 2 (inclusive) e 3

existemAppsNesteIntervalo **=** **true;** //Esta variável booleana serve para verificar se existem apps nesta categoria.

**}**

**else** **if** **(**i **==** **(**listaTodasApps**.**size**()** **-** 1**)** **&&** existemAppsNesteIntervalo**)** **{** //Caso não existam apps nesta categoria (facto verificado pela variável booleana " existemAppsNesteIntervalo", é imprimida uma mensagem que indica este facto.

System**.**out**.**println**(**"- Não existem apps cuja classificação se encaixa neste intervalo!"**);**

**}**

**}**

**break;**

**case** 4 **:** //Caso o utilizador selecione a opção 4

System**.**out**.**println**(**"\nLISTA DE APPS (3 a 4 estrelas):\n"**);**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps

**if** **(**listaTodasApps**.**get**(**i**).**getClassificacao**()** **>=** 3 **&&** listaTodasApps**.**get**(**i**).**getClassificacao**()** **<** 4**)** **{** //Imprime todas as apps com uma classificação entre 3 (inclusive) e 4

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**());**

existemAppsNesteIntervalo **=** **true;** //Esta variável booleana serve para verificar se existem apps nesta categoria.

**}**

**else** **if** **(**i **==** **(**listaTodasApps**.**size**()** **-** 1**)** **&&** existemAppsNesteIntervalo**)** **{** //Caso não existam apps nesta categoria (facto verificado pela variável booleana " existemAppsNesteIntervalo", é imprimida uma mensagem que indica este facto.

System**.**out**.**println**(**"- Não existem apps cuja classificação se encaixa neste intervalo!"**);**

**}**

**}**

**break;**

**case** 5 **:** //Caso o utilizador selecione a opção 5

System**.**out**.**println**(**"\nLISTA DE APPS (mais de 4 estrelas):"**);**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Perccore a lista de apps

**if** **(**listaTodasApps**.**get**(**i**).**getClassificacao**()** **>=** 4 **&&** listaTodasApps**.**get**(**i**).**getClassificacao**()** **<=** 5**)** **{** //Imprime todas as apps com uma classificação entre 4 e 5

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**()** **+** "\n"**);**

existemAppsNesteIntervalo **=** **true;**

**}**

**else** **if** **(**i **==** **(**listaTodasApps**.**size**()** **-** 1**)** **&&** existemAppsNesteIntervalo**)** **{** //Caso não existam apps nesta categoria (facto verificado pela variável booleana " existemAppsNesteIntervalo", é imprimida uma mensagem que indica este facto.

System**.**out**.**println**(**"- Não existem apps cuja classificação se encaixa neste intervalo!"**);**

**}**

**}**

**break;**

**default** **:** //Caso o utilizador tenha selecionado uma opção inválida

opcaoErrada**();** //Mostra uma mensagem de erro

listarPorClassificacaoSeletiva**();** //Recomeça a execução do método

**break;**

**}**

**}**

**}**

private void listarPorNome**(){** //Método de listagem das apps por nome (ordem alfabética)

**if(**listaTodasApps**.**isEmpty**()){** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

**}**

**else{**

ArrayList**<**String**>** listaNomesApps **=** **new** ArrayList**<>();** //Criação de um ArrayList de Strings que será preenchido pelos nomes das apps existentes

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps e adiciona o respetivo nome de cada app à lista (neste caso, ArrayList) acima criada

listaNomesApps**.**add**(**listaTodasApps**.**get**(**i**).**getNome**());**

**}**

Collections**.**sort**(**listaNomesApps**);** //Reordena a lista de nomes de todas apps de acordo por ordem alfabética

System**.**out**.**println**(**"\nLISTA DE APPS POR ORDEM ALFABÉTICA:"**);**

**for** **(**int j **=** 0**;** j **<** listaNomesApps**.**size**();** j **++)** **{** //Percorre a lista de nomes de todas as apps (já ordenada alfabeticamente) e imprime o nome de cada app, linha por linha

System**.**out**.**println**(**"- " **+** listaNomesApps**.**get**(**j**));** **}**

**}**

**}**

private void listarPorNumVendas**(){** //Método de listagem das apps por número de vendas (ordem crescente)

**if(**listaTodasApps**.**isEmpty**()){** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

**}**

**else{**

Collections**.**sort**(**listaTodasApps**,** App**.**COMPARAR\_POR\_NUM\_VENDAS**);** //Reordena a lista de apps, ordenando-a de acordo com o número de vendas das apps (por ordem crescente)

System**.**out**.**println**(**"\nLISTA DE APPS POR NÚMERO DE VENDAS (ordem crescente):"**);**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps (já ordenada por número de vendas) e imprime o nome das mesmas juntamente com o seu respetivo número de vendas

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**()+**" ("**+**listaTodasApps**.**get**(**i**).**getNumVendas**()+**" vendas)"**);**

**}**

**}**

**}**

private void menuListagemUtilizadores**()throws** IOException**{**

String menu **=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //Display correspondente à listagem de utilizadores.

menu **+=** "| LISTAGEM - UTILIZADORES |\n"**;** //Neste display são apresentadas as duas opções

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //de listagem de utilizadores. A primeira corresponde a

menu **+=** "| 1 - Organizada |\n"**;** //listar os mesmos de uma forma organizada (porém

menu **+=** "| 2 - Por dinheiro total gasto |\n"**;** //desordenada) ou então listá-los por dinheiro total

menu **+=** "|------------------------------------|\n"**;** //gasto na AppStore.

menu **+=** "| 3 - Voltar |\n"**;**

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;**

System**.**out**.**print**(**menu**);**

**switch(**getOpcao**()){**

**case** 1 **:**

listarUtilizadores**(false);**

voltarMenuListagem**();**

**break;**

**case** 2 **:**

listarUtilizadores**(true);**

voltarMenuListagem**();**

**break;**

**case** 3 **:**

menuPrincipal**();**

**break;**

**default:**

opcaoErrada**();**

menuListagemApps**();**

**break;**

**}**

**}**

private void listarUtilizadores**(**boolean porDinheiroTotalGasto**){** //Método que efetua a listagem dos utilizadores de forma organizada (porém desordenada) ou então de forma organizada por dinheiro total gasto na AppStore

**if(**listaUtilizadores**.**isEmpty**()){** //Caso a lista de utilizadores esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem utilizadores!"**);**

**}**

**else{**

**if** **(**porDinheiroTotalGasto**)** **{** //Caso o booleano "porDinheiroTotalGasto" seja verdadeiro, será efetuada a listagem dos utilizadores por dinheiro total gasto

Collections**.**sort**(**listaUtilizadores**,** Utilizador**.**COMPARAR\_POR\_DINHEIRO\_TOTAL\_GASTO**);**

System**.**out**.**println**(**"\nLISTA DE UTILIZADORES (por dinheiro total gasto):"**);**

**}**

**else** **{** //Caso contrário, será efetuada uma listagem organizada (porém desordenada) dos utilizadores

System**.**out**.**println**(**"\nLISTA DE UTILIZADORES:"**);**

**}**

**for** **(**int i **=** 0**;** i **<** listaUtilizadores**.**size**();** i **++){** //Percorre a lista de utilizadores

System**.**out**.**println**(**"Nome: " **+** listaUtilizadores**.**get**(**i**).**getNome**());** //Imprime o nome do utilizador em questão

System**.**out**.**println**(**"Idade: " **+** listaUtilizadores**.**get**(**i**).**getIdade**());** //Imprime a idade do utilizador em questão

**if** **(** listaUtilizadores**.**get**(**i**).**getPremium**()** **==** **true)** **{** //Caso a variável booleana "premium" do utilizador em questão for true, imprime uma mensagem que indica que o utilizador é premium

System**.**out**.**println**(**"Premium: Sim"**);**

**}**

**else** **{** System**.**out**.**println**(**"Premium: Não"**);}** //Caso contrário, imprime que uma mensagem que indica que ele não é premium

System**.**out**.**println**(**"ID: " **+**listaUtilizadores**.**get**(**i**).**getIDUtilizador**());**

**if** **(**porDinheiroTotalGasto**)** **{** //Caso o booleano "porDinheiroTotalGasto" seja verdadeiro, ou seja, está a ser efetuada a listagem dos utilizadores por dinheiro total gasto:

System**.**out**.**println**(**"Lista de apps:"**);**

**if** **(**listaUtilizadores**.**get**(**i**).**getListaApps**().**isEmpty**())** **{** //Caso o utilizador não possua apps, imprime uma mensagem que indica este facto

System**.**out**.**println**(**"- Este utilizador não possui apps!"**);**

**}**

**else** **{**

**for** **(**int j **=** 0**;** j **<** listaUtilizadores**.**get**(**i**).**getListaCompras**().**size**();** j**++)** **{** //Percorre a lista de compras efetuadas pelo utilizador em questão

**if** **(**listaUtilizadores**.**get**(**i**).**getListaCompras**().**isEmpty**())** **{** //Caso esta lista esteja vazia, sai deste ciclo "for"

**break;**

**}**

**else** **{**

System**.**out**.**println**(**"- " **+** listaUtilizadores**.**get**(**i**).**getListaCompras**().**get**(**j**).**getApp**().**getNome**()** **+** " (" **+** listaUtilizadores**.**get**(**i**).**getListaCompras**().**get**(**j**).**getDataCompra**().**mostraData**()** **+** " [data da compra])"**);** //Caso contrário, imprime o nome da app da compra em questão e a data em que foi efetuada essa mesma compra

**}**

**}**

**for** **(**int j **=** 0**;** j **<** listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**size**();** j**++)** **{** //Percorre a lista de subscrições efetuadas pelo utilizador em questão

**if** **(**listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**isEmpty**())** **{** //Caso esta lista esteja vazia, sai deste ciclo "for"

**break;**

**}**

**else** **{**

System**.**out**.**println**(**"- " **+** listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**get**(**j**).**getApp**().**getNome**()** **+** " (" **+** listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**get**(**j**).**getDataSubscricao**().**mostraData**()** **+** " [início da subscrição])"**);** //Caso contrário, imprime o nome da app da subscrição em questão e a data em que foi iniciada essa mesma subscrição

**}**

**}**

**}**

System**.**out**.**println**(**"Dinheiro total gasto: " **+**listaUtilizadores**.**get**(**i**).**getDinheiroTotalGasto**()** **+** " euros\n"**);** //Imprime o dinheiro total gasto pelo utilizador em questão

**}**

**else** **{**

System**.**out**.**println**(**"\n"**);**

**}**

**}**

**}**

**}**

private void inicioCompraOuSubscricao**(**boolean subscricao**)** **throws** IOException**{** //Método inicial do modo compra e do modo subscrição. Neste método, é selecionado o utilizador que irá efetuar a compra e é criada a lista de apps que esse utilizador ainda não possui.

**try{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

**if** **(**listaTodasApps**.**isEmpty**())** **{** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

menuPrincipal**();**

**}**

**else** **if** **(**listaUtilizadores**.**isEmpty**())** **{** //Caso a lista de utilizadores esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem utilizadores!"**);**

menuPrincipal**();**

**}**

**else** **{**

Scanner scannerDados **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

System**.**out**.**println**(**"Insira o ID do utilizador:" **);**

int id **=** verificarOpcaoID**(**scannerDados**,** listaIDsUtilizadores**);** //Iguala a variável do tipo int "ID" (que corresponde ao ID do utilizador que convidou este novo utilizador) ao valor retornado pelo método verificarOpcaoID, que neste caso verifica se o utilizador está a inserir um dos IDs presentes na lista de utilizadores e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então o ID do utilizador escolhido, dependendo do que o utilizador inseriu no input.

Utilizador utilizador **=** getUtilizadorPorID**(**id**);** //Cria um novo objeto da classe "Utilizador", que neste caso corresponderá ao utilizador (objeto) correspondente ao ID inserido, obtido através do método getUtilizadorPorID.

// Prepara a união das listas

ArrayList**<**App**>** listaAppsDisponiveis **=** **new** ArrayList**<>(**listaTodasApps**);** //Inicialização de um ArrayList de apps (classe App) que contém a lista de todas as apps

listaAppsDisponiveis**.**addAll**(**utilizador**.**getListaApps**());** //Adiciona o conteúdo da lista de apps do utilizador em questão à lista criada

// Prepara a interseção das listas

ArrayList**<**App**>** intersecaoListas **=** **new** ArrayList**<>(**listaTodasApps**);** //Inicialização de um ArrayList de apps (classe App) que irá contém a lista de todas as apps

intersecaoListas**.**retainAll**(**utilizador**.**getListaApps**());** //Verifica o que há em comum entre esta lista (que contém todas as apps) e a lista de apps do utilizador em questão. Todos os elementos em comum (apps em comum) entre estas duas listas corresponderão ao conteúdo final desta lista.

// Subtrai a interseção da união

listaAppsDisponiveis**.**removeAll**(**intersecaoListas**);** //Remove-se da lista obtida o conteúdo da interseção (que corresponde às apps que o utilizador já possui), obtendo-se assim uma lista de apps que o utilizador ainda não possui

**if** **(!**subscricao**)**

comprar**(**utilizador**,** scannerDados**,** listaAppsDisponiveis**);**

**else** **{**

subscrever**(**utilizador**,** listaAppsDisponiveis**,** scannerDados**);**

**}**

**}**

**}**

**catch** **(**InputMismatchException ime**){** //Caso tenha ocorrido algum erro em que o utilizador inseriu letras em vez de números ou números em vez de letras, aparece a mensagem de erro abaixo e este método é reinicializado.

System**.**out**.**println**(**"Introdução inválida! Tente novamente! \n"**);**

inicioCompraOuSubscricao**(false);**

**}**

**}**

public void comprar**(**Utilizador utilizador**,** Scanner scannerDados**,** ArrayList **<**App**>** listaAppsDisponiveis**)** **throws** IOException **{** //Método do modo compra. Neste método, é selecionada a app que será comprada/obtida.

**try{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

System**.**out**.**println**(**"Escolha uma das seguintes apps. Todas as apps assinaladas com [G] são elegíveis para obter gratuitamente: "**);**

**for** **(**int i **=** 0**;** i **<** listaAppsDisponiveis**.**size**();** i**++)** **{** //Percorre a lista de apps que o utilizador ainda não possui e que ainda não estão no seu carrinho de compras

System**.**out**.**print**(**"\n- "**+**listaAppsDisponiveis**.**get**(**i**).**getNome**()+**" ("**+**listaAppsDisponiveis**.**get**(**i**).**getPreco**()+**" euros)"**);** //Imprime o nome da app em questão dentro dessa mesma lista e o seu respetivo preço

**if** **((**listaAppsDisponiveis**.**get**(**i**).**getMenosVendida**())** **||** listaAppsDisponiveis**.**get**(**i**).**getCategoria**().**equals**(**categoriaDoMes**))** **{** //Caso a app em questão seja uma das cinco menos vendidas da semana ou então a sua categoria corresponda à categoria do mês (ou seja, possua um desconto):

**if** **(!**utilizador**.**getUsouAppGratuita**())** **{** //Todos os utilizadores têm direito a uma app gratuita por semana de entre aquelas que estão com um desconto. Caso o utilizador ainda não tenha usufruido desse desconto:

System**.**out**.**print**(**" [G]"**);** //Imprime a String "[G] ao lado do nome da app para indicar que esta app é elegível para obter gratuitamente.

listaAppsDisponiveis**.**get**(**i**).**setTemDesconto**(true);** //Muda o valor da variável booleana "temDesconto" da app em questão para verdadeiro, indicando que esta está com um desconto

**}**

**}**

**if** **(**i **==** **(**listaAppsDisponiveis**.**size**()** **-** 1**))** **{**

System**.**out**.**println**(**"\n"**);**

**}**

**}**

ArrayList**<**String**>** listaNomesApps **=** **new** ArrayList**<>();** //Criação de um ArrayList de Strings que será preenchido pelos nomes das apps existentes

**for** **(**int i **=** 0**;** i **<** listaAppsDisponiveis**.**size**();** i**++)** **{** //Percorre a lista de apps que o utilizador ainda não possui e que ainda não estão no seu carrinho de compras e adiciona o respetivo nome de cada uma dessas apps à lista acima criada

listaNomesApps**.**add**(**listaAppsDisponiveis**.**get**(**i**).**getNome**());**

**}**

String nomeApp **=** verificarOpcao**(**scannerDados**,** listaNomesApps**);** //Iguala a variável do tipo String "nomeApp" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir um dos nomes presentes na lista de apps que o utilizador ainda não possui e que ainda não estão no seu carrinho e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então o nome da app escolhida, dependendo do que o utilizador inseriu no input.

App app **=** getAppPorNome**(**nomeApp**);** //Cria um novo objeto da classe App, objeto esse que corresponderá à app escolhida.

**if** **(!**utilizador**.**getUsouAppGratuita**())** **{** //Caso o utilizador ainda não tenha usufruido da promoção da app gratuita nessa semana:

**if** **(**app**.**getTemDesconto**())** **{** //Caso a app escolhida tenha sido uma das apps que têm desconto, imprime uma mensagem que indica ao utilizador que é possível ele obter esta app gratuitamente e pergunta se o mesmo pretende que isso aconteça ou se pretende proceder a compra normalmente

System**.**out**.**println**(**"Todos os utilizadores têm direito a uma app gratuita (por semana) de entre aquelas que têm desconto. "

**+** "\nEsta é uma dessas. Pretende obtê-la gratuitamente ou proceder com a compra? (S/N)"**);**

String opcaoGratuita **=** verificarOpcao**(**scannerDados**,** simOuNao**);** //Iguala a variável do tipo String "opcaoGratuita" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir uma das Strings "S" ou "N" e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então uma dessas duas Strings, dependendo do que o utilizador inseriu no input.

**if** **(**opcaoGratuita**.**equals**(**"S"**))** **{** //Caso o String "opcaoGratuita" corresponda ao String "S":

utilizador**.**setUsouAppGratuita**(true);** //Altera o valor do booleano "usouAppGratuita" do utilizador em questão para verdadeiro, indicando assim que este utilizador já utilizou a promoção semanal da app gratuita e fazendo assim com que não o possa fazer até à semana seguinte

double precoFinal **=** 0.0**;** //Cria a variável do tipo double "precoFinal" (correspondente ao preço final que será pago por esta app) e atribui o valor "0.0" à mesma, uma vez que o utilizador não pagará pela obtenção da app escolhida

app**.**comprar**(**utilizador**,** **this,** listaAppsDisponiveis**,** app**,** precoFinal**);** //Chama o método "comprar" para a app escolhida

**}**

**}**

**}**

app**.**comprar**(**utilizador**,** **this,** listaAppsDisponiveis**);** //Chama o método "comprar" para a app escolhida

**}**

**catch** **(**InputMismatchException ime**){** //Caso tenha ocorrido algum erro em que o utilizador inseriu letras em vez de números ou números em vez de letras, aparece a mensagem de erro abaixo e este método é reinicializado.

System**.**out**.**println**(**"Introdução inválida! Tente novamente! \n"**);**

comprar**(**utilizador**,** scannerDados**,** listaAppsDisponiveis**);}**

**}**

private void subscrever**(**Utilizador utilizador**,** ArrayList**<**App**>** listaAppsDisponiveis**,** Scanner scannerDados**)** **throws** IOException**{** //Método que permite a subscrição de um determinado utilizador a uma determinada app "subscrevível" (ou seja, uma app que permite subscrição)

**try{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

System**.**out**.**println**(**"Escolha uma das seguintes apps: "**);**

**for** **(**int i **=** 0**;** i **<** listaAppsDisponiveis**.**size**();** i**++)** **{** //Percorre a lista de apps que o utilizador ainda não possui e que ainda não estão no seu carrinho de compras

**if** **(**listaAppsDisponiveis**.**get**(**i**).**getSubscrevivel**())** **{** //Caso essa app seja "subscrevível" (permita subscrição), é imprimido o seu nome e o preço da subscrição (que corresponde a 70% do preço de compra da app)

double precoSubscricao **=** **(**0.7 **\*** **(**listaAppsDisponiveis**.**get**(**i**).**getPreco**()));** //Cálculo do preço de subscrição

precoSubscricao **=** Math**.**round **(**precoSubscricao **\*** 100.0**)** **/** 100.0**;** //Arrendondamento do preço de subscrição para 2 casas decimais

System**.**out**.**println**(**"- "**+**listaAppsDisponiveis**.**get**(**i**).**getNome**()+**" (" **+** precoSubscricao **+** " euros)"**);**

**}**

**}**

ArrayList**<**String**>** listaNomesAppsSubscreviveis **=** **new** ArrayList**<>();** //Criação de um ArrayList de Strings que será preenchido pelos nomes das apps que permitem subscrição de entre as apps que o utilizador ainda não possui

**for** **(**int i **=** 0**;** i **<** listaAppsDisponiveis**.**size**();** i**++)** **{** //Percorre a lista de apps que o utilizador ainda não possui

**if** **(**listaAppsDisponiveis**.**get**(**i**).**getSubscrevivel**())** **{** //Caso a app em questão seja "subscrevível" (ou seja, permita subscrição), é imprimido o nome da mesma

listaNomesAppsSubscreviveis**.**add**(**listaAppsDisponiveis**.**get**(**i**).**getNome**());**

**}**

**}**

String nomeApp **=** verificarOpcao**(**scannerDados**,** listaNomesAppsSubscreviveis**);** //Iguala a variável do tipo String "nomeApp" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir um dos nomes presentes na lista de apps que o utilizador ainda não possui e são "subscrevíveis" e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então o nome da app escolhida, dependendo do que o utilizador inseriu no input.

App app **=** getAppPorNome**(**nomeApp**);** //Cria um novo objeto da classe App, objeto esse que corresponderá à app escolhida.

app**.**subscrever**(**utilizador**,** **this);** //Chama o método comprar para a app escolhida

**}**

**catch** **(**InputMismatchException ime**){** //Caso tenha ocorrido algum erro em que o utilizador inseriu letras em vez de números ou números em vez de letras, aparece a mensagem de erro abaixo e este método é reinicializado.

System**.**out**.**println**(**"Introdução inválida! Tente novamente! \n"**);**

subscrever**(**utilizador**,** listaAppsDisponiveis**,** scannerDados**);}**

**}**

private void avaliarAplicacao**()** **throws** IOException**{** //Método que permite a avaliação de uma determinada app por um determinado utilizador

**try{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

**if** **(**listaTodasApps**.**isEmpty**())** **{** //Caso a lista de apps esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem apps!"**);**

menuPrincipal**();**

**}**

**else** **if** **(**listaUtilizadores**.**isEmpty**())** **{** //Caso a lista de utilizadores esteja vazia, o programa imprime uma mensagem de erro e a execução deste método assim termina

System**.**out**.**println**(**"Ainda não existem utilizadores!"**);**

menuPrincipal**();**

**}**

**else** **{**

Scanner scannerDados **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

System**.**out**.**println**(**"Insira o ID do utilizador:" **);**

int id **=** verificarOpcaoID**(**scannerDados**,** listaIDsUtilizadores**);** //Iguala a variável do tipo int "ID" (que corresponde ao ID do utilizador que convidou este novo utilizador) ao valor retornado pelo método verificarOpcaoID, que neste caso verifica se o utilizador está a inserir um dos IDs presentes na lista de utilizadores e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então o ID do utilizador escolhido, dependendo do que o utilizador inseriu no input.

Utilizador utilizador **=** getUtilizadorPorID**(**id**);** //Cria um novo objeto da classe "Utilizador", que neste caso corresponderá ao utilizador (objeto) correspondente ao ID inserido, obtido através do método getUtilizadorPorID.

**if** **(**utilizador**.**getListaApps**().**isEmpty**())** **{** //Caso o utilizador ainda não tenha apps, é apresentada uma mensagem de erro e o programa é redirecionado novamente para o menu principal

System**.**out**.**println**(**"Você ainda não possui apps!"**);**

menuPrincipal**();**

**}**

System**.**out**.**println**(**"A lista de apps disponíveis para avaliação é: " **);**

ArrayList**<**String**>** listaNomesApps **=** **new** ArrayList**<>();** //Criação de um ArrayList de Strings que irá conter o nome das apps que o utilizador já comprou e, portanto, podem ser avaliadas pelo mesmo

**for** **(**int i **=** 0**;** i **<** utilizador**.**getListaApps**().**size**();** i**++)** **{** //Percorre a lista de apps do utilizador

listaNomesApps**.**add**(**utilizador**.**getListaApps**().**get**(**i**).**getNome**());** //Adiciona o nome da app em questão à lista anteriormente criada

System**.**out**.**println**(**"- " **+** listaNomesApps**.**get**(**i**));** //Imprime o nome da app em questão

**}**

String nome **=** verificarOpcao**(**scannerDados**,** listaNomesApps**);** //Iguala a variável do tipo String "nome" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir um dos nomes presentes na lista de apps que o utilizador possui e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então o nome da app escolhida, dependendo do que o utilizador inseriu no input.

App app **=** getAppPorNome**(**nome**);** //Cria um novo objeto da classe App, objeto esse que corresponderá à app escolhida.

**for** **(**int i **=** 0**;** i **<** app**.**getListaAvaliacoes**().**size**();** i**++)** **{** //Percorre a lista de avaliações da app em questão

**if** **(**app**.**getListaAvaliacoes**().**get**(**i**).**getUtilizador**().**equals**(**utilizador**))** **{** //Caso seja encontrada uma avaliação que foi criada por este mesmo utilizador, essa avaliação é removida e o utilizador é avisado de que esta nova avaliação irá substituir a antiga.

app**.**getListaAvaliacoes**().**remove**(**i**);**

System**.**out**.**println**(**"A nova avaliação irá substituir a anteriormente submetida!"**);**

**}**

**}**

System**.**out**.**println**(**"AVALIAÇÃO PESSOAL (App: " **+** app**.**getNome**()** **+** "):"**);**

System**.**out**.**println**(**"Insira a classificação (1 - 5): "**);**

int pontuacao **=** verificarOpcaoPontuacao**(**scannerDados**);** //Iguala a variável do tipo int "pontuacao" ao valor retornado pelo método verificarOpcaoPontuacao, que neste caso verifica se o utilizador está a inserir uma das classificações disponíveis (1, 2, 3, 4 ou 5) e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornada então a classificação escolhida, dependendo do que o utilizador inseriu no input.

System**.**out**.**println**(**"Deseja deixar um comentário? (S/N)"**);**

String opcao **=** verificarOpcao**(**scannerDados**,** simOuNao**);** //Iguala a variável do tipo String "opcao" ao valor retornado pelo método verificarOpcao, que neste caso verifica se o utilizador está a inserir uma das Strings "S" ou "N" e, até que isso aconteça, o programa fica num loop, mostrando uma mensagem de erro. Quando isso acontece, é retornado então uma dessas duas Strings, dependendo do que o utilizador inseriu no input.

String comentario**;**

Avaliacao avaliacao**;**

**if** **(**opcao**.**equals**(**"S"**)){** //Se a opção inserida pelo utilizador corresponde a "S", indicando que o mesmo pretende deixar um comentário, é guardado na variável do tipo String "comentario" esse mesmo comentário

System**.**out**.**println**(**"Insira o comentário: "**);**

comentario **=** scannerDados**.**next**();**

avaliacao **=** **new** Avaliacao**(**pontuacao**,** comentario**,** utilizador**);** //Criação de uma nova avaliação (objeto da classe Avaliação) com a respetiva pontuação/classificação atribuida, o comentário e o utilizador que efetuou a avaliação

**}**

**else** **{** //Caso contrário, é criada uma nova avaliação (objeto da classe Avaliação) com a respetiva pontuação/classificação atribuida e o utilizador que efetuou esta avaliação

avaliacao **=** **new** Avaliacao**(**pontuacao**,** utilizador**);**

**}**

app**.**getListaAvaliacoes**().**add**(**avaliacao**);** //Adiciona esta avaliação à lista de avaliações da app em questão

app**.**getProgramador**().**addNumAvaliacoesRecebidas**();** //Incrementa em 1 o número de avaliações recebidas pelo programador da app em questão

double novaPontuacaoProgramador **=** **(**double**)** **((**app**.**getProgramador**().**getAvaliacao**()** **\*** **(**app**.**getProgramador**().**getNumAvaliacoesRecebidas**()** **-** 1**)** **+** pontuacao**)** **/** app**.**getProgramador**().**getNumAvaliacoesRecebidas**());** //Calcula a avaliação atualizada do programador, indo buscar o valor anterior da avaliação do mesmo, multiplicando-o pelo número anterior de avaliações recebidas, somando a nova pontuação recebida e divindo o valor total pelo número atualizado de avaliações recebidas

app**.**getProgramador**().**setAvaliacao**(**novaPontuacaoProgramador**);** //Atualiza a avaliação do programador

int somaPontuacoes **=** 0**;**

int numeroPontuacoes **=** app**.**getListaAvaliacoes**().**size**();**

**for(**int i **=** 0**;** i **<** app**.**getListaAvaliacoes**().**size**();** i**++)** **{** //Percorre a lista de avaliações da app em questão

somaPontuacoes **=** somaPontuacoes **+** app**.**getListaAvaliacoes**().**get**(**i**).**getPontuacao**();** //Soma a classificação da avaliação em questão à variável "somaPontuacoes", que está inicialmente a 0, obtendo-se assim uma soma de todas as classificações atribúidas a essa app

**}**

double mediaPontuacoes **=** **(**double**)** somaPontuacoes **/** numeroPontuacoes**;** //Divide-se a soma de todas as classificações atribuídas a essa app pelo número total dessas mesmas classificações

app**.**setClassificacao**(**mediaPontuacoes**);** //Atualiza o valor da classificação da app com o valor anteriormente calculado

menuPrincipal**();**

**}**

**}**

**catch** **(**InputMismatchException ime**){** //Caso tenha ocorrido algum erro em que o utilizador inseriu letras em vez de números ou números em vez de letras, aparece a mensagem de erro abaixo e este método é reinicializado.

System**.**out**.**println**(**"Introdução inválida! Tente novamente! \n"**);**

avaliarAplicacao**();}**

**}**

private void menuMontanteRecebido**()throws** IOException**{**

String menu **=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //Display correspondente à apresentação de montantes recebidos.

menu **+=** "| MONTANTE RECEBIDO |\n"**;** //Neste display são apresentadas as duas opções de apresentação

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;** //de montantes recebidos. A primeira opção corresponde à

menu **+=** "| 1 - Pela AppStore |\n"**;** //apresentação do valor total recebido pela AppStore enquanto

menu **+=** "| 2 - Pelos programadores |\n"**;** //que a segunda opção corresponde à apresentação do valor

menu **+=** "|------------------------------------|\n"**;** //total (dinheiro) recebido pelo programador de acordo com

menu **+=** "| 3 - Voltar |\n"**;** //as vendas das apps a que lhe pertencem.

menu **+=** "|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n"**;**

System**.**out**.**print**(**menu**);**

**switch(**getOpcao**()){**

**case** 1 **:**

System**.**out**.**println**(**"O valor total recebido pela AppStore é de: "**+**valorTotalAppStore**+**" euros.\n"**);** //Imprime o valor total recebido (dinheiro) pela AppStore

voltarMenuMontante**();**

**break;**

**case** 2 **:**

**if** **(!**listaProgramadores**.**isEmpty**())** **{** //Caso a lista de programadores não esteja vazia:

System**.**out**.**println**(**"VALOR TOTAL RECEBIDO POR CADA PROGRAMADOR:"**);**

**for** **(**int i **=** 0**;** i **<** listaProgramadores**.**size**();** i**++)** **{** //Percorre a lista de programadores

System**.**out**.**println**(**"\nProgramador "**+(**i**+**1**)+**" (ID: "**+**listaProgramadores**.**get**(**i**).**getIDProgramador**()+**"): "**+**listaProgramadores**.**get**(**i**).**getDinheiroTotalRecebido**()** **+** " euros"**);** //Imprime o valor (+1) do índice do programador (1, 2, 3...) de acordo com a ordem dos mesmos na lista de programadores e de seguida imprime o dinheiro total recebido pelo mesmo.

**}**

**}**

**else** **{** System**.**out**.**println**(**"Ainda não existem programadores!"**);** **}** //Caso contrário, é apresentada uma mensagem de erro e é dada a opção de voltar ao menu em que estava ou então voltar ao menu principal.

voltarMenuMontante**();**

**break;**

**case** 3 **:**

menuPrincipal**();**

**break;**

**default:**

opcaoErrada**();**

menuMontanteRecebido**();**

**break;**

**}**

**}**

private void cincoMenosVendidas**(**boolean passouUmAno**)** **{** //Método que verifica quais foram as 5 apps menos vendidas da semana anterior

**if** **(((**data**.**getContadorDias**()** **%** 7 **==** 0**)** **&&** **(**data**.**getContadorDias**()** **!=** 0**)** **&&** **(!**listaTodasApps**.**isEmpty**()))** **||** **(**passouUmAno**))** **{** //Se uma nova semana se está a iniciar e se a lista de apps não está vazia OU então se se passou um ano inteiro de uma só vez (através da opção de avançar 1 ano no menu):

**for** **(**int i **=** 0**;** i **<** listaUtilizadores**.**size**();** i**++)** **{** //Percorre a lista de utilizadores

listaUtilizadores**.**get**(**i**).**setUsouAppGratuita**(false);** //"Reseta" a variável "usouAppGratuita" de todos os utilizadores de modo a que estes possam utilizar a promoção da app gratuita novamente uma vez que uma nova semana se iniciou

**}**

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps

listaTodasApps**.**get**(**i**).**setMenosVendida**(false);** //As variáveis "menosVendida" e "temDesconto" de todas as apps são "resetadas"

listaTodasApps**.**get**(**i**).**setTemDesconto**(false);**

**}**

**if** **(**listaTodasApps**.**size**()** **<=** 5**)** **{** //Se a lista de apps tiver 5 ou menos apps

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++)** **{** //Percorre a lista de apps e altera o valor da variável booleana "menosVendida" da mesma para verdadeiro, indicando assim que esta app é uma das 5 menos vendidas da semana anterior. Também iguala o valor de vendas semanal a 0, uma vez que se iniciou uma nova semana.

listaTodasApps**.**get**(**i**).**setMenosVendida**(true);**

listaTodasApps**.**get**(**i**).**setNumeroDeVendasSemanal**(**0**);**

**}**

**}**

**else** **{**

Collections**.**sort**(**listaTodasApps**,** App**.**COMPARAR\_POR\_NUM\_VENDAS\_SEMANAL**);** //Reordena a lista de apps, ordenando-a de acordo com o número de vendas semanal da semana anterior.

**if** **(!**passouUmAno**)** **{** //Se a opção de avançar 1 ano inteiro não foi utilizada:

System**.**out**.**println**(**"As cinco apps menos vendidas da semana passada foram:"**);**

**for** **(**int i **=** 0**;** i **<** 5**;** i**++)** **{** //Percorre os 5 primeiros elementos da lista reordenada (5 apps menos vendidas)

System**.**out**.**println**(**"- " **+** listaTodasApps**.**get**(**i**).**getNome**());** //Imprime o nome das primeiras 5 apps presentes na lista reordenada de apps, ou seja, imprime o nome das 5 apps menos vendidas.

listaTodasApps**.**get**(**i**).**setMenosVendida**(true);** //Altera o valor da variável booleana "menosVendida" da app em questão para verdadeiro, de modo a estabelecer que esta foi uma das 5 apps menos vendidas da semana anterior

listaTodasApps**.**get**(**i**).**setNumeroDeVendasSemanal**(**0**);** //Iguala o valor de vendas semanal dessa app a 0, uma vez que se iniciou uma nova semana.

**}**

System**.**out**.**println**(**"\nDurante a semana corrente, estas apps possuem 15% de desconto sobre o preço de referência!\n"**);**

**}**

**else** **{** //Caso contrário, o procedimento é exatamente o mesmo excetuando-se o facto de que nada é imprimido.

**for** **(**int i **=** 0**;** i **<** 5**;** i**++)** **{** //Percorre os 5 primeiros elementos da lista reordenada (5 apps menos vendidas)

listaTodasApps**.**get**(**i**).**setMenosVendida**(true);** //Altera o valor da variável booleana "menosVendida" da app em questão para verdadeiro, de modo a estabelecer que esta foi uma das 5 apps menos vendidas da semana anterior

listaTodasApps**.**get**(**i**).**setNumeroDeVendasSemanal**(**0**);** //Iguala o valor de vendas semanal dessa app a 0, uma vez que se iniciou uma nova semana.

**}**

**}**

**}**

**}**

**}**

public void categoriaDoMes**(**boolean passouUmAno**)** **{** //Método que estabelece qual é a categoria do mês

**if** **((**data**.**getDia**()** **==** 1**)** **||** **(**data**.**getContadorDias**()** **==** 0**)** **||** passouUmAno**)** **{** //Se o programa acabou de ser iniciado ou um novo mês começou:

int numeroaleatorio **=** random**.**nextInt**(**8**);** //Iguala a variável "numeroaleatorio" a um número inteiro entre 0 e 7, uma vez que existem 8 categorias

categoriaDoMes **=** listaCategorias**.**get**(**numeroaleatorio**);** //Define a categoria do mês como a categoria presente no índice do número aleatório obtido anteriormente

**}**

**}**

private void verificarSubscricoes**()** **{** //Método que verifica se o tempo das subscrições atualmente ativas já terminou ou não

**for** **(**int i **=** 0**;** i **<** listaUtilizadores**.**size**();** i**++)** **{** //Percorre a lista de utilizadores

**if** **(**listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**isEmpty**())** **{**

i**++;**

**}**

**else** **{**

**for** **(**int j **=** 0**;** j **<** listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**size**();** j**++)** **{** //Percorre a lista de subscrições do utilizador em questão

**if** **(**data**.**getContadorDias**()** **>=** **((**listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**get**(**j**).**getDataSubscricao**().**getContadorDias**())** **+** 365**))** **{** //Se o contador de dias da data atual for maior ou igual à data da subscrição somada com 365 dias, então isso significa que já passou 1 ano desde o início da subscrição

listaUtilizadores**.**get**(**i**).**getListaApps**().**remove**(**listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**get**(**j**).**getApp**());** //Remove a app correspondente à subscrição em questão da lista de apps do utilizador

listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**add**(**j**,** **null);** //Coloca um "null" na posição da lista subscrição a ser removida, de modo a que quando esta seja removida o tamanho da lista de subscrições se mantenha o mesmo e assim o ciclo "for" continue normalmente

listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**remove**(**listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**get**(**j**+**1**));** //Remove a subscrição em questão da lista de subscrições do utilizador, que neste caso passou para o índice seguinte uma vez que se inseriu um "null" no índice de onde ela estava

**}**

**if** **(**j **==** **(**listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**size**()** **-** 1**))** **{** //Quando chega ao fim da lista de subscrições:

listaUtilizadores**.**get**(**i**).**getListaSubscricoes**().**removeAll**(**Collections**.**singleton**(null));** //Remove todas as ocorrências de "null" na lista de subscrições uma vez que estes são redundantes e apenas utilizados para a manuntenção do tamanho da lista para que assim o ciclo "for" funcione de acordo com o esperado

System**.**out**.**println**(**listaUtilizadores**.**get**(**i**).**getListaSubscricoes**());**

**}**

**}**

**}**

**}**

**}**

private App getAppPorNome**(**String nome**){** //Método que permite a obtenção da app (objeto da classe App) através do seu nome

**for** **(**int i **=** 0**;** i **<** listaTodasApps**.**size**();** i**++){** //Percorre a lista de apps

**if(** listaTodasApps**.**get**(**i**).**getNome**().**equals**(**nome**)){**

**return** listaTodasApps**.**get**(**i**);** //Retorna a app em questão assim que encontrar uma app cujo nome é igual ao nome facultado

**}**

**}**

**return** **null;**

**}**

private Utilizador getUtilizadorPorID**(**int id**){** //Método que permite a obtenção do utilizador (objeto da classe Utilizador) através do seu ID

**for** **(**int i **=** 0**;** i **<** listaUtilizadores**.**size**();** i**++){** //Percorre a lista de utilizadores

**if(** listaUtilizadores**.**get**(**i**).**getIDUtilizador**()** **==** id **){**

**return** listaUtilizadores**.**get**(**i**);** //Retorna o utilizador em questão assim que encontrar um utilizador cujo ID é igual ao ID facultado

**}**

**}**

**return** **null;**

**}**

private Programador getProgramadorPorID**(**int id**)throws** IOException**{** //Método que permite a obtenção do programador (objeto da classe Programador) através do seu ID

**for** **(**int i **=** 0**;** i **<** listaProgramadores**.**size**();** i**++){**

**if(** listaProgramadores**.**get**(**i**).**getIDProgramador**()** **==** id **){**

**return** listaProgramadores**.**get**(**i**);** //Retorna o programador em questão assim que encontrar um programador cujo ID é igual ao ID facultado

**}**

**}**

**return** **null;**

**}**

private void voltarMenuRegisto**()throws** IOException**{** //Método para voltar ao menu resgisto ou para o menu principal

System**.**out**.**println**(**"\nDeseja voltar a qual dos menus? \n"**+**"1 - Menu Registo \n" **+** "2 - Menu Principal"**);**

Scanner scannerConfirma **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

int selecao **=** scannerConfirma**.**nextInt**();** //Iguala a variável do tipo Integer "selecao" (que corresponde a opcao) ao input do teclado

**switch** **(**selecao**)** **{**

**case** 1**:** //Caso seja selecionado o número "1" volta para o menuRegisto()

menuRegisto**();**

**break;**

**case** 2**:** //Caso seja selecionado o número "2" volta para o menuPrincipal()

menuPrincipal**();**

**break;**

**default:** //Caso não seja selecionado nenhum destes números apresenta no display a mensagem "Opção inválida!"

System**.**out**.**println**(**"Opção inválida!"**);**

voltarMenuRegisto**();** //Chamando recursivamente esta função até ser selecionado um dos números válidos

**break;**

**}**

**}**

private void voltarMenuListagem**()throws** IOException**{** //Método para voltar ao menu listagem ou para o menu principal

System**.**out**.**println**(**"\nDeseja voltar a qual dos menus? \n"**+**"1 - Menu Listagem \n" **+** "2 - Menu Principal"**);**

Scanner scannerConfirma **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

int selecao **=** scannerConfirma**.**nextInt**();** //Iguala a variável do tipo Integer "selecao" (que corresponde a opcao) ao input do teclado

**switch(**selecao**)** **{**

**case** 1**:** //Caso seja selecionado o número "1" volta para o menuListagem()

menuListagem**();**

**break;**

**case** 2**:** //Caso seja selecionado o número "2" volta para o menuPrincipal()

menuPrincipal**();**

**break;**

**default:** //Caso não seja selecionado nenhum destes números apresenta no display a mensagem "Opção inválida!"

System**.**out**.**println**(**"Opção inválida!"**);**

voltarMenuRegisto**();** //Chamando recursivamente esta função até ser selecionado um dos números válidos

**break;**

**}**

**}**

private void voltarMenuMontante**()throws** IOException**{** //Método para voltar ao menu montagem recebido ou para o menu principal

System**.**out**.**println**(**"\nDeseja voltar a qual dos menus? \n"**+**"1 - Menu Montante Recebido \n" **+** "2 - Menu Principal"**);**

Scanner scannerConfirma **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

int selecao **=** scannerConfirma**.**nextInt**();** //Iguala a variável do tipo Integer "selecao" (que corresponde a opcao) ao input do teclado

**switch** **(**selecao**)** **{**

**case** 1**:** //Caso seja selecionado o número "1" volta para o menuMontateRecebido()

menuMontanteRecebido**();**

**break;**

**case** 2**:** //Caso seja selecionado o número "2" volta para o menuPrincipal()

menuPrincipal**();**

**break;**

**default:** //Caso não seja selecionado nenhum destes números apresenta no display a mensagem "Opção inválida!"

System**.**out**.**println**(**"Opção inválida!"**);**

voltarMenuRegisto**();** //Chamando recursivamente esta função até ser selecionado um dos números válidos

**break;**

**}**

**}**

private void opcaoErrada**(){** //Método que mostra uma mensagem que opção selecionada está incorreta

System**.**out**.**println**(**"A opção assinalada não é válida!"**);**

**}**

public int verificarOpcaoPontuacao**(**Scanner scannerDados**){** //Método que recebe como parâmetro o scanner inserido pelo utilizador para verificar se o número inserido é válido, pois a pontuação só pode variar de 1 a 5

int pontuacao **=** scannerDados**.**nextInt**();** //Iguala a variável do tipo Integer "pontuacao" (que corresponde a pontuacao) ao input do teclado

ArrayList**<**Integer**>** listaPontuacao **=** **new** ArrayList**<>();** //Lista de pontuação (lista de Integers) e sua inicializção

listaPontuacao**.**addAll**(**Arrays**.**asList**(**1**,**2**,**3**,**4**,**5**));** //Adiciona-se as integers correspondentes aos valores das várias pontuações à lista de pontuações

**for** **(**int i **=** 0**;** i **<** listaPontuacao**.**size**();** i**++){** //Ciclo que percorre a lista de pontuações

**if((**listaPontuacao**.**get**(**i**))** **==** pontuacao**)** **{** //Se a pontuação inserida pelo teclado corresponder um pontuação que esteja na lista, então ocorre break e retorna o valor da "pontuacao"

**break;** **}**

**else** **if(**i **==** **(**listaPontuacao**.**size**()-**1**))** **{** //Se chegar ao fim da lista e não ter encontrado essa pontuação igual, então significa que a pontuação foi inserida incorretamente

System**.**out**.**println**(** "A pontuação inserida é inválida. Insira novamente:"**);**

**return** verificarOpcaoPontuacao**(**scannerDados**);** //Retorna para este mesmo método até inserir uma das pontuações possiveís

**}**

**}**

**return** pontuacao**;** //Retorna o valor da pontuação inserida

**}**

public String verificarOpcao**(**Scanner scannerDados**,** ArrayList**<?>** listaStrings**)** **{** //Método que recebe como parâmetro o scanner inserido pelo utilizador e um arraylist chamado listaStrings

String opcao **=** scannerDados**.**next**();** //Iguala a variável do tipo String "opcao" (que corresponde a opcao) ao input do teclado

**for** **(**int i **=** 0**;** i **<** listaStrings**.**size**();** i**++){** //Ciclo for percorre a lista de strings inserida

**if((**listaStrings**.**get**(**i**)).**equals**(**opcao**))** **{** //Caso a opcao inserida seja igual a uma das strings contidas na lista de Strings, então ocorre break e retorna o valor da "opcao"

**break;** **}**

**else** **if(**i **==** **(**listaStrings**.**size**()-**1**))** **{** //Se chegar ao fim da lista e não encontrar num string igual a "opcao" , então significa que a opcao foi inserida incorretamente

System**.**out**.**println**(**"A opção inserida é inválida. Insira novamente:"**);**

**return** verificarOpcao**(**scannerDados**,** listaStrings**);** //Retorna para este mesmo método até inserir uma das strings possiveís

**}**

**}**

**return** opcao**;** //Retorna a opcão inserida

**}**

public int verificarOpcaoID**(**Scanner scannerDados**,** ArrayList**<?>** listaIDs**)** **{** //Método que recebe como parâmetro o scanner inserido pelo utilizador e um arraylist chamado listaIDs

int id **=** scannerDados**.**nextInt**();** //Iguala a variável do tipo Integer "id" (que corresponde a id) ao input do teclado

**for** **(**int i **=** 0**;** i **<** listaIDs**.**size**();** i**++){** //Ciclo for percorre a lista de IDs inserida

**if((**listaIDs**.**get**(**i**)).**equals**(**id**))** **{** //Caso o "id" inserido seja igual a um dos IDs contidos na lista de IDs, então ocorre break e retorna o valor do "id"

**break;** **}**

**else** **if** **(**i **==** **(**listaIDs**.**size**()-**1**))** **{** //Se chegar ao fim da lista e não encontrar num ID igual a "id" , então significa que o "id" foi inserida incorretamente

System**.**out**.**println**(** "O ID inserido é inválido. Insira novamente:"**);**

**return** verificarOpcaoID**(**scannerDados**,** listaIDs**);** **}** //Retorna para este mesmo método até inserir um dos IDs possiveís

**}**

**return** id**;** //Retorna o ID inserido

**}**

public String verificarOpcaoNomeApp**(**Scanner scannerDados**,** ArrayList**<**String**>** listaNomesApps**)** **{** //Método que recebe como parâmetro o scanner inserido pelo utilizador e um arraylist com o nome de todas as apps existentes

String nomeApp **=** scannerDados**.**next**();** //Iguala a variável do tipo String "nomeApp" (que corresponde a nome da app) ao input do teclado

**for** **(**int i **=** 0**;** i **<** listaNomesApps**.**size**();** i**++){** //Ciclo for percorre a lista dos nomes das apps

**if((**listaNomesApps**.**get**(**i**)).**equals**(**nomeApp**))** **{** //Caso o nome (da app) inserido seja igual a um dos nomes das apps já existentes, então aparece uma mensagem de erro e este método é reinicializado (recursividade)

System**.**out**.**println**(**"Já existe uma app com esse nome! Insira outro nome:"**);**

**return** verificarOpcao**(**scannerDados**,** listaNomesApps**);** **}**

**}**

**return** nomeApp**;** //Caso não existam apps com o nome inserido, retorna então esse nome inserido

**}**

public Data getData**(){** //Método seletor para data

**return** data**;**

**}**

private int getOpcao**()** **throws** IOException **{** //Método que obtém a opção que foi introduzida no teclado e devolve a mesma

**try** **{** //Se ocorrer algum erro no código dentro do "try", o programa é redirecionado para o "catch" correspondente ao erro em questão.

Scanner scanneropcao **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

int opcao **=** scanneropcao**.**nextInt**();**

**return** opcao**;**

**}**

**catch** **(**InputMismatchException ime**)** **{** //Caso tenha ocorrido algum erro em que o utilizador inseriu letras em vez de números ou números em vez de letras, aparece a mensagem de erro abaixo e este método é reinicializado.

**return** 100**;** //Retorna 10 pois nenhum dos menus contém uma opção número "100" portanto o programa irá devolver uma mensagem de erro e pedir que o utilizador introduza novamente uma opção

**}**

**}**

public String getCategoriaDoMes**()** **{** //Método seletor para categoriaDoMes

**return** categoriaDoMes**;**

**}**

public ArrayList**<**Integer**>** getListaIDsUtilizadores**()** **{** //Método seletor para a listaIDsUtilizadores

**return** listaIDsUtilizadores**;**

**}**

public ArrayList**<**Integer**>** getListaIDsProgramadores**()** **{** //Método seletor para a listaIDsProgramadores

**return** listaIDsProgramadores**;**

**}**

public ArrayList**<**App**>** getListaTodasApps**()** **{** //Método seletor para a listaTodasApps

**return** listaTodasApps**;**

**}**

public ArrayList**<**Utilizador**>** getListaUtilizadores**()** **{** //Método seletor para a listaUtilizadores

**return** listaUtilizadores**;**

**}**

public void addValorTotalAppStore**(**double novoValor**)** **{** //Método que adiciona novo valor gasto ao valor total de dinheiro da appStore

**this.**valorTotalAppStore **=** valorTotalAppStore **+** novoValor**;**

**}**

**}**

**APP:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

**import** java**.**io**.**IOException**;**

**import** java**.**util**.\*;**

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class App **{**

private String nome**;** //Nome da app

private double preco**;** //Preço da app

private String categoria**;** //Categoria da app

private Programador programador**;** //Programador da app (objeto da classe Programador)

private int numeroDeVendas**;** //Número de vendas total da app

private int numeroDeVendasSemanal**;** //Número de vendas semanal da app

private double classificacao**;** //Classificação (média) da app

private ArrayList**<**Avaliacao**>** listaAvaliacoes**;** //Lista de avaliações (objetos da classe Avaliacao) da app

private boolean menosVendida**;** //Variável booleana que indica se esta app é ou não uma das 5 apps menos vendidas da semana anterior

private boolean subscrevivel**;** //Variável booleana que indica se esta app é ou não "subscrevível" (permite subscrição)

private boolean temDesconto**;** //Variável booleana que indica se esta app tem ou não um desconto durante a semana corrente (ou seja, se pertence à categoria do mês ou se foi uma das 5 menos vendidas da semana anterior)

public App**(**String nome**,** double preco**,** String categoria**,** Programador programador**,** boolean subscrevivel**){**

**this.**nome **=** nome**;**

**this.**preco **=** preco**;**

**this.**categoria **=** categoria**;**

**this.**programador **=** programador**;**

**this.**subscrevivel **=** subscrevivel**;**

numeroDeVendas **=** 0**;**

numeroDeVendasSemanal **=** 0**;**

classificacao **=** 0.0**;**

menosVendida **=** **false;**

listaAvaliacoes **=** **new** ArrayList**<>();**

temDesconto **=** **false;**

**}**

public void comprar**(**Utilizador utilizador**,** AppStore appstore**,** ArrayList**<**App**>** listaAppsDisponiveis**)** **throws** IOException**{** //Método que calcula o preço da app selecionada de acordo com os descontos (existentes ou não)

App app **=** **this;**

double precoFinal **=** app**.**getPreco**();** //Iguala a variável double "precoFinal" ao preço da app em questão

precoFinal **=** **(** precoFinal **\*** **(**100 **-** verificarDesconto**(**app**,** utilizador**,** appstore**))** **)** **/** 100**;** // precoFinal = ( Preço da app + (100 - desconto) ) / 100

precoFinal **=** Math**.**round **(**precoFinal **\*** 100.0**)** **/** 100.0**;** //Arrendonda o valor da variável "precoFinal" para um valor com 2 casas decimais

comprar**(**utilizador**,** appstore**,** listaAppsDisponiveis**,** app**,** precoFinal**);** //Chama o método de

**}**

public void comprar**(**Utilizador utilizador**,** AppStore appstore**,** ArrayList**<**App**>** listaAppsDisponiveis**,** App app**,** double precoFinal**)** **throws** IOException**{** //Método que cria um objeto da classe Compra de acordo com os dados da compra a ser efetuada. Além disso, este método permite continuar ou não a compra.

int diaCompra **=** appstore**.**getData**().**getDia**();** //diaCompra = dia atual (no momento da compra)

int mesCompra **=** appstore**.**getData**().**getMes**();** //mesCompra = mês atual (no momento da compra)

int anoCompra **=** appstore**.**getData**().**getAno**();** //anoCompra = ano atual (no momento da compra)

int contadorDias **=** appstore**.**getData**().**getContadorDias**();** //contadorDias = valor do contador de dias no momento da compra

Data dataCompra **=** **new** Data**(**diaCompra**,** mesCompra**,** anoCompra**,** contadorDias**);** //Cria um novo objeto da classe Data com os dados obtidos anteriormente (correspondentes à data completa no momento da compra)

Compra compra **=** **new** Compra**(**app**,** precoFinal**,** dataCompra**);** //Criação de uma nova compra (objeto da classe Compra) com os dados coletados até então, neste caso a app em questão, o preço final e a data da compra

System**.**out**.**println**(**"Deseja comprar mais alguma app ou prosseguir para o pagamento?"**);**

System**.**out**.**println**(**"1 - Continuar a compra"**);**

System**.**out**.**println**(**"2 - Prosseguir para o pagamento"**);**

Scanner scannerDados **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

int opcao **=** scannerDados**.**nextInt**();** //A variável "opcao" corresponde à opção selecionada pelo utilizador de acordo com a sua intenção (1 se pretende continuar a compra, 2 se pretende finalizar a compra e prosseguir para o pagamento)

**switch** **(**opcao**)** **{**

**case** 1**:** //Caso o utilizador selecione a opção 1, ou seja, pretende prosseguir a compra:

utilizador**.**getCarrinho**().**add**(**compra**);** //Adiciona a compra ao carrinho (lista de compras) do utilizador em questão

listaAppsDisponiveis**.**remove**(this);** //Remove esta app da lista de apps disponíveis para compra

**if(**listaAppsDisponiveis**.**isEmpty**()){** //Caso a lista de apps disponíveis esteja vazia, imprime uma mensagem que indica que não existem mais apps disponíveis

System**.**out**.**println**(**"\nNão existem mais apps disponiveís! Redirecionando para o pagamento... "**);**

utilizador**.**pagar**(**appstore**);** //Chama o método "pagar" para o utilizador em questão

**}**

appstore**.**comprar**(**utilizador**,** scannerDados**,** listaAppsDisponiveis**);** //Chama o método "comprar" para o objeto (único) correspondente à AppStore

**case** 2**:**

utilizador**.**getCarrinho**().**add**(**compra**);** //Adiciona a compra ao carrinho (lista de compras) do utilizador em questão

utilizador**.**pagar**(**appstore**);** //Chama o método "pagar" para o utilizador em questão

**break;**

**default:**

System**.**out**.**println**(**"Opção inválida!"**);**

**break;**

**}**

**}**

public void subscrever**(**Utilizador utilizador**,** AppStore appstore**)** **throws** IOException**{** //Método que obtém a data no momento da subscrição e cria assim um novo objeto da classe Data e uma nova subscrição (objeto da classe subscrição), respetivamente.

Subscricao subscricao**;**

int diaSubscricao **=** appstore**.**getData**().**getDia**();** //diaSubscricao = dia atual (no momento da subscrição)

int mesSubscricao **=** appstore**.**getData**().**getMes**();** //mesSubscricao = mês atual (no momento da subscrição)

int anoSubscricao **=** appstore**.**getData**().**getAno**();** //anoSubscricao = ano atual (no momento da subscrição)

int contadorDias **=** appstore**.**getData**().**getContadorDias**();** //contadorDias = valor do contador de dias no momento da subscrição

Data dataSubscricao **=** **new** Data**(**diaSubscricao**,** mesSubscricao**,** anoSubscricao**,** contadorDias**);** //Cria um novo objeto da classe Data com os dados obtidos anteriormente (correspondentes à data completa no momento da subscrição)

subscricao **=** **new** Subscricao**(**dataSubscricao**,** **this);** //Criação de uma nova subscrição (objeto da classe subscrição) com o objeto da classe Data e o objeto da app em questão

utilizador**.**pagar**(**appstore**,** subscricao**);** //Chama o método "pagar" para o utilizador em questão

**}**

private int verificarDesconto**(**App app**,** Utilizador utilizador**,** AppStore appstore**){** //Descontos: Se for uma das cinco menos vendidas da semana (15%), 1 categoria por mes tem 5% (quando chegar ao final do mes),

int desconto **=** 0**;**

**if** **(**app**.**getMenosVendida**())** **{** //Se a app em questão for uma das cinco menos vendidas da semana anterior, a variável "desconto" é incrementada em 15 (desconto = desconto + 15%)

desconto **=** desconto **+** 15**;**

temDesconto **=** **true;** //Ativa a variável booleana "temDesconto", utilizada para indicar que esta app possui um dos tipos de desconto (ou ambos)

**}**

**if** **(**app**.**getCategoria**().**equals**(**appstore**.**getCategoriaDoMes**()))** **{** //Se a categoria desta app for a categoria do mês, a variável "desconto" é incrementada em 5 (desconto = desconto + 5%)

desconto **=** desconto **+** 5**;**

temDesconto **=** **true;** //Ativa a variável booleana "temDesconto", utilizada para indicar que esta app possui um dos tipos de desconto (ou ambos)

**}**

**if** **(**utilizador**.**getNumUtilizadoresAngariados**()** **!=** 0**)** **{** //Se o número de utilizadores angariados é diferente de 0 (significando que este utilizador angariou no mínimo 1 utilizador), é somado o número de utilizadores angariados multiplicado por 5, uma vez que cada utilizador corresponde a mais 5% de desconto (desconto = desconto + número de utilizadores angariados \* 5%)

desconto **=** desconto **+** **(**utilizador**.**getNumUtilizadoresAngariados**()** **\*** 5**);**

utilizador**.**setNumUtilizadoresAngariados**(**0**);** //"Reseta" o número de utilizadores angariados deste utilizador

**}**

**if** **(**utilizador**.**getPremium**())** **{** //Se o utilizador for premium, a variável "desconto" é incrementada em 60 (desconto = desconto + 60%)

desconto **=** desconto **+** 60**;**

**}**

**if** **(**desconto **>** 100**)** **{** //Se o total do desconto ultrapassar a margem dos 100%, o desconto fica a 100%, evitando assim a obtenção de valores a pagar negativos

desconto **=** 100**;**

**}**

**return** desconto**;**

**}**

public static Comparator**<**App**>** COMPARAR\_POR\_NUM\_VENDAS **=** **new** Comparator**<**App**>()** **{** //Criação da interface de comparação "COMPARAR\_POR\_NUM\_VENDAS" que, tal como o nome indica, trata da comparação das apps de acordo com os seus respetivos números de vendas

@Override

public int compare**(**App a1**,** App a2**)** **{** //Override do método compare da interface "Comparator" para que a comparação feita por esta interface criada seja entre o número de vendas de dois objetos do tipo "App"

**return** a1**.**getNumVendas**()** **-** a2**.**getNumVendas**();** //O método "compare" devolve a diferença entre os dois atributos que está a comparar, neste caso (devido ao override) a diferença entre os números de vendas das apps que estão a ser comparadas. Assim sendo, se o primeiro objeto tiver um número de vendas superior ao segundo objeto, devolve um número positivo. Se tiverem um número de vendas iguais, devolve zero. Já se o segundo objeto for maior que o primeiro objeto, devolve um número negativo. É através da verificação destes valores que o método "sort" das Collections irá ordenar a lista.

**}**

**};**

public static Comparator**<**App**>** COMPARAR\_POR\_CLASSIFICACAO **=** **new** Comparator**<**App**>()** **{** //Criação da interface de comparação "COMPARAR\_POR\_CLASSIFICACAO" que, tal como o nome indica, trata da comparação das apps de acordo com as suas respetivas classificações

@Override

public int compare**(**App a1**,** App a2**)** **{** //Override do método compare da interface "Comparator" para que a comparação feita por esta interface criada seja entre as classificações de dois objetos do tipo "App"

**return** Double**.**compare**(**a1**.**getClassificacao**(),** a2**.**getClassificacao**());** //O método "compare" devolve a diferença entre os dois atributos que está a comparar, neste caso (devido ao override) a diferença entre as classificações das apps que estão a ser comparadas. Assim sendo, se o primeiro objeto tiver um número de vendas superior ao segundo objeto, devolve um número positivo. Se tiverem um número de vendas iguais, devolve zero. Já se o segundo objeto for maior que o primeiro objeto, devolve um número negativo. É através da verificação destes valores que o método "sort" das Collections irá ordenar a lista.

**}**

**};**

public static Comparator**<**App**>** COMPARAR\_POR\_NUM\_VENDAS\_SEMANAL **=** **new** Comparator**<**App**>()** **{** //Criação da interface de comparação "COMPARAR\_POR\_NUM\_VENDAS\_SEMANAL" que, tal como o nome indica, trata da comparação das apps das apps de acordo com os seus respetivos números de vendas semanais

@Override

public int compare**(**App a1**,** App a2**)** **{** //Override do método compare da interface "Comparator" para que a comparação feita por esta interface criada seja entre os números de vendas semanais de dois objetos do tipo "App"

**return** a1**.**getNumeroDeVendasSemanal**()** **-** a2**.**getNumeroDeVendasSemanal**();** //O método "compare" devolve a diferença entre os dois atributos que está a comparar, neste caso (devido ao override) a diferença entre os números de vendas semanais das apps que estão a ser comparadas. Assim sendo, se o primeiro objeto tiver um número de vendas superior ao segundo objeto, devolve um número positivo. Se tiverem um número de vendas iguais, devolve zero. Já se o segundo objeto for maior que o primeiro objeto, devolve um número negativo. É através da verificação destes valores que o método "sort" das Collections irá ordenar a lista.

**}**

**};**

public void setMenosVendida**(**boolean menosVendida**)** **{**

**this.**menosVendida **=** menosVendida**;**

**}**

public void setTemDesconto**(**boolean temDesconto**)** **{**

**this.**temDesconto **=** temDesconto**;**

**}**

public Programador getProgramador**()** **{**

**return** programador**;**

**}**

public void setClassificacao**(**double classificacaoDada**){**

classificacao **=** classificacaoDada**;**

**}**

public ArrayList**<**Avaliacao**>** getListaAvaliacoes**(){**

**return** listaAvaliacoes**;**

**}**

public String getCategoria**()** **{**

**return** categoria**;**

**}**

public String getNome**()** **{**

**return** nome**;**

**}**

public int getNumVendas**()** **{**

**return** numeroDeVendas**;**

**}**

public double getClassificacao**()** **{**

**return** classificacao**;**

**}**

public double getPreco**()** **{**

**return** preco**;**

**}**

public boolean getMenosVendida**()** **{**

**return** menosVendida**;**

**}**

public void addNumeroDeVendas**()** **{**

**this.**numeroDeVendas **=** **this.**numeroDeVendas **+** 1**;**

**}**

public void addNumeroDeVendasSemanal**()** **{**

**this.**numeroDeVendasSemanal **=** **this.**numeroDeVendasSemanal **+** 1**;**

**}**

public int getNumeroDeVendasSemanal**()** **{**

**return** numeroDeVendasSemanal**;**

**}**

public void setNumeroDeVendasSemanal**(**int numeroDeVendasSemanal**)** **{**

**this.**numeroDeVendasSemanal **=** numeroDeVendasSemanal**;**

**}**

public boolean getTemDesconto**()** **{**

**return** temDesconto**;**

**}**

public boolean getSubscrevivel**()** **{**

**return** subscrevivel**;**

**}**

**}**

**UTILIZADOR:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

**import** java**.**io**.**IOException**;**

**import** java**.**util**.\*;**

**import** java**.**util**.**Random**;** // Geração do ID do utilizador (aleatoriamente)

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class Utilizador **{**

private String nome**;** //Nome do utilizador

private int idade**;** //Idade do utilizador

private int idUtilizador**;** //ID do utilizador

private boolean premium**;** //Variável booleana que indica se o utilizador é premium ou não

private int numUtilizadoresAngariados**;** //Número de utilizadores angariados por este utilizador

private ArrayList**<**Compra**>** carrinho**;** //Lista de compras (objetos da classe Compra) do utilizador, a qual é "resetada" após a finalização de cada compra e portanto reutilizada cada vez que se inicia uma nova compra

private double dinheiroTotalGasto**;** //Valor (dinheiro) total gasto pelo utilizador na AppStore

private ArrayList**<**App**>** listaApps**;** //Lista de apps (objetos da classe App) do utilizador

private ArrayList**<**Subscricao**>** listaSubscricoes**;** //Lista de subscrições (objetos da classe

private ArrayList**<**Compra**>** listaCompras**;** //Lista de compras (objetos da classe Compra) do utilizador

private boolean usouAppGratuita**;** //Variável booleana que indica se o utilizador já utilizou a promoção da app gratuita esta semana ou não

public Utilizador**(**String nome**,** int idade**,** boolean premium**){**

**this.**nome **=** nome**;**

**this.**idade **=** idade**;**

**this.**premium **=** premium**;**

idUtilizador **=** 0**;**

numUtilizadoresAngariados **=** 0**;**

carrinho **=** **new** ArrayList**<>();**

dinheiroTotalGasto **=** 0**;**

listaApps **=** **new** ArrayList**<>();**

listaSubscricoes **=** **new** ArrayList**<>();**

listaCompras **=** **new** ArrayList**<>();**

usouAppGratuita **=** **false;**

**}**

public int gerarIDUtilizador**(**AppStore appstore**){** //Método que permite a geração do ID do utilizador

Random random **=** **new** Random**();** //Objeto da classe Random

int numeroaleatorio **=** random**.**nextInt**(**10000**)** **+** 20000**;** //numeroaleatorio = número aleatório entre 20000 e 29999

**if** **(**appstore**.**getListaIDsUtilizadores**().**isEmpty**()){** //Se a lista de IDs já utilizados pelos utilizadores está vazia:

idUtilizador **=** numeroaleatorio**;** //Iguala o ID do utilizador à variável "numeroaleatorio"

**}**

**else** **{**

**for** **(**int j **=** 0**;** j **<** appstore**.**getListaIDsUtilizadores**().**size**();** j**++)** **{** //Percorre a lista de IDs já utilizados pelos utilizadores

**if** **(**appstore**.**getListaIDsUtilizadores**().**get**(**j**)** **==** numeroaleatorio**){** //Caso seja encontrado um ID na lista de IDs já utilizados igual ao ID gerado para o novo utilizador ("numeroaleatorio"), o método é reinicializado de modo a que seja gerado um novo ID

**return** gerarIDUtilizador**(**appstore**);**

**}**

**else** **if** **(**j **==** **(**appstore**.**getListaIDsUtilizadores**().**size**()** **-** 1**)){** //Caso a lista já tenha sido percorrida na sua totalidade e nenhum dos IDs corresponde ao ID gerado, o ID do utilizador é igualado à variável "numeroaleatorio"

idUtilizador **=** numeroaleatorio**;**

**break;**

**}**

**}**

**}**

**return** idUtilizador**;**

**}**

public void pagar**(**AppStore appstore**)** **throws** IOException**{** //Método que efetua o cálculo do valor total a pagar e permite ao utilizador escolher se confirma o pagamento ou pretende cancelar a compra

double valorTotalAPagar **=** 0.0**;**

System**.**out**.**println**(**"CARRINHO:"**);**

**for** **(**int i **=** 0**;** i **<** carrinho**.**size**();** i**++)** **{** //Percorre o carrinho (lista de compras)

**if** **(**carrinho**.**get**(**i**).**getValorTotal**()** **==** 0**)** **{** //Se o preço da app correspondente à compra em questão é 0, é imprimido o nome da app e uma indicação de que esta é grátis

System**.**out**.**println**(**"- " **+** carrinho**.**get**(**i**).**getApp**().**getNome**()** **+** " (Grátis)"**);**

**}**

**else** **{** //Caso contrário, é imprimido o nome da app e o preço da mesma (já com os descontos incluídos)

System**.**out**.**println**(**"- " **+** carrinho**.**get**(**i**).**getApp**().**getNome**()** **+** " (" **+** carrinho**.**get**(**i**).**getValorTotal**()** **+** " euros)"**);**

**}**

valorTotalAPagar **=** valorTotalAPagar **+** carrinho**.**get**(**i**).**getValorTotal**();** //Incrementa a variável "valorTotalAPagar" com o preço da app em questão de modo a se obter no final o total a pagar

**}**

System**.**out**.**println**(**"\nTOTAL: " **+** valorTotalAPagar **+** " euros"**);**

verificarOpcaoConfirmar**(**appstore**);** //Chama o método "verificarOpcaoConfirmar", que permite ao utilizador escolher se confirma o pagamento ou se pretende cancelar a compra

**for** **(**int i **=** 0**;** i **<** carrinho**.**size**();** i**++)** **{** //Percorre o carrinho (lista de compras)

**this.**listaApps**.**add**(**carrinho**.**get**(**i**).**getApp**());** //Adiciona a(s) app(s) comprada(s) à lista de apps do utilizador

carrinho**.**get**(**i**).**getApp**().**addNumeroDeVendas**();** //Incrementa em 1 o número de vendas da app em questão

carrinho**.**get**(**i**).**getApp**().**addNumeroDeVendasSemanal**();** //Incrementa em 1 o número de vendas semanal da app em questão

//Adicionar ao programador o valor total ganho

carrinho**.**get**(**i**).**getApp**().**getProgramador**().**addDinheiroTotalRecebido**(**carrinho**.**get**(**i**).**getValorTotal**());** //Adiciona o valor total pago pelo utilizador ao valor do dinheiro total recebido pelo programador da app em questão

**}**

appstore**.**addValorTotalAppStore**(**valorTotalAPagar**);** //Adiciona o dinheiro total pago pelo utilizador ao valor total da AppStore

**this.**dinheiroTotalGasto **=** dinheiroTotalGasto **+** valorTotalAPagar**;** //Adiciona o dinheiro total pago pelo utilizador nesta compra à variável corresponde ao valor total gasto por si mesmo ("dinheiroTotalGasto")

listaCompras**.**addAll**(**carrinho**);** //Adiciona as compras presentes no carrinho a uma lista de compras do utilizador que acumula todas as compras alguma vez efetuadas pelo mesmo

carrinho**.**clear**();** //Limpa o carrinho de modo a que este possa ser reutilizado na próxima compra

System**.**out**.**println**(**"\nCompra e respetivo pagamento finalizados com sucesso!"**);**

appstore**.**menuPrincipal**();** //Volta para o menu principal

**}**

private void verificarOpcaoConfirmar**(**AppStore appstore**)throws** IOException**{** //Método que permite ao utilizador escolher se confirma o pagamento ou se pretende cancelar a compra/subscrição

Scanner scannerConfirma **=** **new** Scanner**(**System**.**in**);** //Criação de uma variável chamada input com a referência do objeto do tipo Scanner. Esta variável recebe o input do teclado.

System**.**out**.**println**(**"\nConfirmar o pagamento? \n"**+**"1 - Sim \n" **+** "2 - Não"**);**

int selecao **=** scannerConfirma**.**nextInt**();** //A variável "opcao" corresponde à opção selecionada pelo utilizador de acordo com a sua intenção (1 se pretende confirmar o pagamento e 2 se pretende cancelar a compra/subscrição)

**switch** **(**selecao**)** **{**

**case** 1**:** //Caso a opção selecionada seja a 1, termina este método e volta para o método do pagamento

**break;**

**case** 2**:** //Caso a opção selecionada seja a 2, limpa o carrinho (lista de compras) para este ser reutilizado na próxima compra e volta para o menu principal

carrinho**.**clear**();**

appstore**.**menuPrincipal**();**

**break;**

**default:** //Caso a opção selecionada seja outra, imprime uma mensagem que indica que essa opção é inválida

System**.**out**.**println**(**"Opção inválida!"**);**

verificarOpcaoConfirmar**(**appstore**);**

**break;**

**}**

**}**

public void pagar**(**AppStore appstore**,** Subscricao subscricao**)** **throws** IOException **{**

double valorAPagar**;**

valorAPagar **=** 0.7 **\*** **(**subscricao**.**getApp**().**getPreco**());** //valorAPagar = 0,7 \* Preço da app em questão (uma vez que foi estipulado que o valor da subscrição por 1 ano corresponde a 70% do preço original da app)

valorAPagar **=** Math**.**round**(**valorAPagar **\*** 100.0**)** **/** 100.0**;** //Arrendonda o valor da variável "valorAPagar" para um valor com 2 casas decimais

System**.**out**.**println**(**"\nTOTAL: "**+**valorAPagar**+**" euros"**);**

verificarOpcaoConfirmar**(**appstore**);** //Chama o método "verificarOpcaoConfirmar", que permite ao utilizador escolher se confirma o pagamento ou se pretende cancelar a subscrição

**this.**listaSubscricoes**.**add**(**subscricao**);** //Adiciona a subscrição (objeto da classe Subscricao) à lista de subscrições deste utilizador

**this.**listaApps**.**add**(**subscricao**.**getApp**());** //Adiciona a app correspondente a esta subscrição à lista de apps deste utilizador

**this.**dinheiroTotalGasto **=** dinheiroTotalGasto **+** valorAPagar**;** //Adiciona o dinheiro total pago pelo utilizador nesta subscrição à variável corresponde ao valor total gasto por si mesmo ("dinheiroTotalGasto")

subscricao**.**getApp**().**getProgramador**().**addDinheiroTotalRecebido**(**valorAPagar**);** //Adiciona o valor total pago pelo utilizador ao valor do dinheiro total recebido pelo programador da app em questão

appstore**.**addValorTotalAppStore**(**valorAPagar**);** //Adiciona o dinheiro total pago pelo utilizador ao valor total da AppStore

System**.**out**.**println**(**"\nSubscrição e respetivo pagamento finalizados com sucesso!"**);**

System**.**out**.**println**(**"\nA subscrição termina a " **+** subscricao**.**getDataSubscricao**().**getDia**()** **+** "/" **+** subscricao**.**getDataSubscricao**().**getMes**()** **+** "/" **+** **(**subscricao**.**getDataSubscricao**().**getAno**()** **+** 1**)** **+** "!"**);** //Imprime uma mensagem indicando a data em que termina a subscrição (1 ano depois)

appstore**.**menuPrincipal**();** //Volta para o menu principal

**}**

public static Comparator**<**Utilizador**>** COMPARAR\_POR\_DINHEIRO\_TOTAL\_GASTO **=** **new** Comparator**<**Utilizador**>()** **{**

@Override

public int compare**(**Utilizador u1**,** Utilizador u2**)** **{**

**return** Double**.**compare**(**u1**.**getDinheiroTotalGasto**(),** u2**.**getDinheiroTotalGasto**());**

**}**

**};**

public void addUtilizadorAngariado**(){**

numUtilizadoresAngariados **=** numUtilizadoresAngariados **+** 1**;**

**}**

public int getIDUtilizador**()** **{**

**return** idUtilizador**;**

**}**

public int getIdade**()** **{**

**return** idade**;**

**}**

public String getNome**()** **{**

**return** nome**;**

**}**

public boolean getPremium**(){**

**return** premium**;**

**}**

public ArrayList**<**Compra**>** getCarrinho**(){**

**return** carrinho**;**

**}**

public int getNumUtilizadoresAngariados**(){**

**return** numUtilizadoresAngariados**;**

**}**

public int setNumUtilizadoresAngariados**(**int numUtilAngariados**){**

**return** numUtilizadoresAngariados **=** numUtilAngariados**;**

**}**

public double getDinheiroTotalGasto**()** **{**

**return** dinheiroTotalGasto**;**

**}**

public void setDinheiroTotalGasto**(**double DinheiroTotalGasto**)** **{**

**this.**dinheiroTotalGasto **=** DinheiroTotalGasto**;**

**}**

public ArrayList**<**Subscricao**>** getListaSubscricoes**()** **{**

**return** listaSubscricoes**;**

**}**

public ArrayList**<**App**>** getListaApps**()** **{**

**return** listaApps**;**

**}**

public boolean getUsouAppGratuita**()** **{**

**return** usouAppGratuita**;**

**}**

public void setUsouAppGratuita**(**boolean usouAppGratuita**)** **{**

**this.**usouAppGratuita **=** usouAppGratuita**;**

**}**

public ArrayList**<**Compra**>** getListaCompras**()** **{**

**return** listaCompras**;**

**}**

**}**

**DATA:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

**import** java**.**util**.**Random**;**

**import** java**.**util**.\*;**

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class Data **{**

private int dia**;** //Dia atual (1-31)

private int mes**;** //Mês atual (1-12)

private int ano**;** //Ano atual (2019-...)

private int contadorDias**;** //Contador de dias desde o início da simulação

private Random random**;** //Objeto da classe Random

public Data**()** **{**

**this.**dia **=** 1**;**

**this.**mes **=** 1**;**

**this.**ano **=** 2019**;**

**this.**contadorDias **=** 0**;**

random **=** **new** Random**();**

**}**

public Data**(**int dia**,** int mes**,** int ano**,** int contadorDias**)** **{**

**this.**dia **=** dia**;**

**this.**mes **=** mes**;**

**this.**ano **=** ano**;**

**this.**contadorDias **=** contadorDias**;**

**}**

public void escolherMes**()** **{** //Método que escolhe aleatoriamente um mês (entre 1 e 12)

mes **=** random**.**nextInt**(**12**)+**1**;**

**}**

public void escolherDia**()** **{** //Método que escolhe aleatoriamente um dia de acordo com o mês escolhido, respeitando os limites de cada mês

**switch** **(**mes**)** **{**

**case** 1 **:** //janeiro (mês 1)

dia **=** random**.**nextInt**(**31**)+**1**;** //Gera um número entre 1 e 31

**break;**

**case** 2 **:** //fevereiro (mês 2)

dia **=** random**.**nextInt**(**28**)+**1**;** //Gera um número entre 1 e 28

**break;**

**case** 3 **:** //março (mês 3)

dia **=** random**.**nextInt**(**31**)+**1**;** //Gera um número entre 1 e 31

**break;**

**case** 4 **:** //abril (mês 4)

dia **=** random**.**nextInt**(**30**)+**1**;** //Gera um número entre 1 e 30

**break;**

**case** 5 **:** //maio (mês 5)

dia **=** random**.**nextInt**(**31**)+**1**;** //Gera um número entre 1 e 31

**break;**

**case** 6 **:** //junho (mês 6)

dia **=** random**.**nextInt**(**30**)+**1**;** //Gera um número entre 1 e 30

**break;**

**case** 7 **:** //julho (mês 7)

dia **=** random**.**nextInt**(**31**)+**1**;** //Gera um número entre 1 e 31

**break;**

**case** 8 **:** //agosto (mês 8)

dia **=** random**.**nextInt**(**31**)+**1**;** //Gera um número entre 1 e 31

**break;**

**case** 9 **:** //setembro (mês 9)

dia **=** random**.**nextInt**(**30**)+**1**;** //Gera um número entre 1 e 30

**break;**

**case** 10 **:** //outubro (mês 10)

dia **=** random**.**nextInt**(**31**)+**1**;** //Gera um número entre 1 e 31

**break;**

**case** 11 **:** //novembro (mês 11)

dia **=** random**.**nextInt**(**30**)+**1**;** //Gera um número entre 1 e 30

**break;**

**case** 12 **:** //dezembro (mês 12)

dia **=** random**.**nextInt **(**31**)+**1**;** //Gera um número entre 1 e 31

**break;**

**}**

**}**

public void avancarDia**()** **{** //Método que permite que o avanço de um dia quanto à data atual

contadorDias **=** contadorDias **+** 1**;** //Adiciona 1 dia ao contador de dias da simulação

**switch** **(**mes**)** **{**

**case** 1**:**

**case** 3**:**

**case** 5**:**

**case** 7**:**

**case** 8**:**

**case** 10**:** //janeiro, março, maio, julho, agosto e outubro (meses 1, 3, 5, 7, 8 e 10)

**if** **(**dia **==** 31**)** **{** //Se a data atual corresponde ao final do mês, incrementa o mês e o dia é igual a 1

mes **=** mes **+** 1**;**

dia **=** 1**;**

**}**

**else** **{**

dia **=** dia **+** 1**;** //Caso contrário, o dia é incrementado em 1

**}**

**break;**

**case** 2**:** //fevereiro (mês 2)

**if** **(**dia **==** 28**)** **{** //Se a data atual corresponde ao final deste mês, incrementa o mês e o dia é igual a 1

mes **=** mes **+** 1**;**

dia **=** 1**;**

**}**

**else** **{**

dia **=** dia **+** 1**;** //Caso contrário, o dia é incrementado em 1

**}**

**break;**

**case** 12**:** //dezembro (mês 12)

**if** **(**dia **==** 31**)** **{** //Se a data atual corresponde ao final deste mês, incrementa o ano e o mês e o dia são iguais a 1

mes **=** 1**;**

ano **=** ano **+** 1**;**

dia **=** 1**;**

**}**

**else** **{**

dia **=** dia **+** 1**;** //Caso contrário, o dia é incrementado em 1

**}**

**break;**

**default:** //abril, junho, setembro e novembro (meses 4, 6, 9 e 11)

**if** **(**dia **==** 30**)** **{** //Se a data atual corresponde ao final do mês, incrementa o mês e o dia é igual a 1

mes **=** mes **+** 1**;**

dia **=** 1**;**

**}**

**else** **{**

dia **=** dia **+** 1**;** //Caso contrário, o dia é incrementado em 1

**}**

**break;**

**}**

**}**

public void avancarAno**()** **{** //Método que permite um avanço de 1 ano quanto à data atual

contadorDias **=** contadorDias **+** 365**;** //Adiciona 365 dias ao contador de dias da simulação

ano **=** ano **+** 1**;**

**}**

public String mostraData**()** **{** //Mostra a data no formato dia/mês/ano

**if** **(**dia **<** 10 **&&** mes **<** 10**)** **{** //Se o dia e o mês forem menores que 10, adiciona um 0 antes da apresentação dos mesmos

**return** "Data: " **+** "0" **+** dia **+** "/0" **+** mes **+** "/" **+** ano**;**

**}**

**else** **if** **(**dia **<** 10**)** **{** //Se o dia for menor que 10, adiciona um 0 antes da apresentação do mesmo

**return** "Data: " **+** "0" **+** dia **+** "/" **+** mes **+** "/" **+** ano**;**

**}**

**else** **if** **(**mes **<** 10**)** **{** //Se o mês for menor que 10, adiciona um 0 antes da apresentação do mesmo

**return** "Data: " **+** dia **+** "/0" **+** mes **+** "/" **+** ano**;**

**}**

**else** **{** //Caso contrário, apresenta o dia, o mês e o ano tais como eles são, respeitando o formato

**return** "Data: " **+** dia **+** "/" **+** mes **+** "/" **+** ano**;**

**}**

**}**

public int getContadorDias**()** **{**

**return** contadorDias**;**

**}**

public int getDia**()** **{**

**return** dia**;**

**}**

public int getMes**()** **{**

**return** mes**;**

**}**

public int getAno**()** **{**

**return** ano**;**

**}**

**}**

**MAIN:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

**import** java**.**io**.**IOException**;**

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class Main **{**

/\*\*

\* **@param** args the command line arguments

\*/

public static void main**(**String**[]** args**)** **throws** IOException **{**

AppStore appstore **=** **new** AppStore**();** //Cria um novo objeto (único) da classe AppStore, que corresponde à AppStore no seu todo

appstore**.**getData**().**escolherDia**();** //Escolhe aleatoriamente um dia para a data do início da simulação

appstore**.**getData**().**escolherMes**();** //Escolhe aleatoriamente um mês para a data do início da simulação

appstore**.**categoriaDoMes**(false);** //Escolhe aleatoriamente uma categoria do mês para o início da simulação

appstore**.**menuPrincipal**();** //Chama o método que apresenta o menu principal

**}**

**}**

**COMPRA:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class Compra **{**

private App app**;** //App que está a ser/foi comprada

private double valorTotal**;** //Valor total a pagar pela app já com os descontos incluídos

private Data dataCompra**;** //Objeto da classe Data correspondente à data em que é realizada a compra

public Compra**(**App app**,** double valorTotal**,** Data dataCompra**){**

**this.**app **=** app**;**

**this.**valorTotal **=** valorTotal**;**

**this.**dataCompra **=** dataCompra**;**

**}**

public double getValorTotal**()** **{**

**return** valorTotal**;**

**}**

public App getApp**()** **{**

**return** app**;**

**}**

public Data getDataCompra**()** **{**

**return** dataCompra**;**

**}**

**}**

**SUBSCRICAO:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class Subscricao **{**

private Data dataSubscricao**;** //Objeto da classe Data correspondente à data em que é realizada a subscrição

private App app**;** //App que está a ser/foi subscrita

public Subscricao**(**Data dataSubscricao**,** App app**)** **{**

**this.**dataSubscricao **=** dataSubscricao**;**

**this.**app **=** app**;**

**}**

public Data getDataSubscricao**()** **{**

**return** dataSubscricao**;**

**}**

public App getApp**()** **{**

**return** app**;**

**}**

**}**

**PROGRAMADOR:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

**import** java**.**util**.\*;**

**import** java**.**util**.**Random**;** // Geração do ID do utilizador (aleatoriamente)

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class Programador **{**

private int idProgramador**;**

private double avaliacao**;**

private double dinheiroTotalRecebido**;**

private ArrayList**<**App**>** listaApps**;**

private int numAvaliacoesRecebidas**;**

public Programador**(){**

avaliacao **=** 0**;**

idProgramador**=**0**;**

listaApps **=** **new** ArrayList**<**App**>();**

dinheiroTotalRecebido **=** 0**;**

numAvaliacoesRecebidas **=** 0**;**

**}**

public int gerarIDProgramador**(**AppStore appstore**){** //Método que permite a geração do ID do programador

Random random **=** **new** Random**();** //Objeto da classe Random

int numeroaleatorio **=** random**.**nextInt**(**10000**)** **+** 10000**;** //numeroaleatorio = número aleatório entre 20000 e 29999

**if** **(**appstore**.**getListaIDsUtilizadores**().**isEmpty**()){** //Se a lista de IDs já utilizados pelos utilizadores está vazia:

idProgramador **=** numeroaleatorio**;** //Iguala o ID do programador à variável "numeroaleatorio"

**}**

**else** **{**

**for** **(**int j **=** 0**;** j **<** appstore**.**getListaIDsUtilizadores**().**size**();** j**++)** **{** //Percorre a lista de IDs já utilizados pelos programadores

**if** **(**appstore**.**getListaIDsUtilizadores**().**get**(**j**)** **==** numeroaleatorio**){** //Caso seja encontrado um ID nesta lista que seja igual ao ID gerado para o novo programador ("numeroaleatorio"), o método é reinicializado de modo a que seja gerado um novo ID

**return** gerarIDProgramador**(**appstore**);**

**}**

**else** **if** **(**j **==** **(**appstore**.**getListaIDsUtilizadores**().**size**()** **-** 1**)){** //Caso a lista já tenha sido percorrida na sua totalidade e nenhum dos IDs corresponde ao ID gerado, o ID deste programador é igualado à variável "numeroaleatorio"

idProgramador **=** numeroaleatorio**;**

**break;**

**}**

**}**

**}**

**return** idProgramador**;**

**}**

public int getIDProgramador**(){**

**return** idProgramador**;**

**}**

public void addAppProgramador**(**App appAdicionada**){**

listaApps**.**add**(**appAdicionada**);**

**}**

public ArrayList**<**App**>** getListaAppsProgramador**(){**

**return** listaApps**;**

**}**

public void addDinheiroTotalRecebido**(**double novoValor**)** **{**

**this.**dinheiroTotalRecebido **=** dinheiroTotalRecebido **+** novoValor**;**

**}**

public double getDinheiroTotalRecebido**()** **{**

**return** dinheiroTotalRecebido**;**

**}**

public double getAvaliacao**(){**

**return** avaliacao**;**

**}**

public void setAvaliacao **(**double novaAvaliacao**){**

**this.**avaliacao **=** novaAvaliacao**;**

**}**

public void addNumAvaliacoesRecebidas**()** **{**

**this.**numAvaliacoesRecebidas **=** **this.**numAvaliacoesRecebidas **+** 1**;**

**}**

public int getNumAvaliacoesRecebidas**()** **{**

**return** numAvaliacoesRecebidas**;**

**}**

**}**

**AVALIACAO:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package appstore**;**

/\*\*

\*

\* **@author** Daniel Barros, Filipa Dias, Jocelyne Pestana

\*/

public class Avaliacao **{**

int pontuacao**;**

String comentario**;**

Utilizador utilizador**;**

public Avaliacao**(**int pontuacao**,** String comentario**,** Utilizador utilizador**){**

**this.**pontuacao **=** pontuacao**;**

**this.**comentario **=** comentario**;**

**this.**utilizador **=** utilizador**;**

**}**

public Avaliacao**(**int pontuacao**,** Utilizador utilizador**){**

**this.**pontuacao **=** pontuacao**;**

**this.**utilizador **=** utilizador**;**

**}**

public int getPontuacao**(){**

**return** pontuacao**;**

**}**

public String getComentario**(){**

**return** comentario**;**

**}**

public Utilizador getUtilizador**()** **{**

**return** utilizador**;**

**}**

**}**

# 8 – Anexo (Diagramas UML)

Diagrama antigo

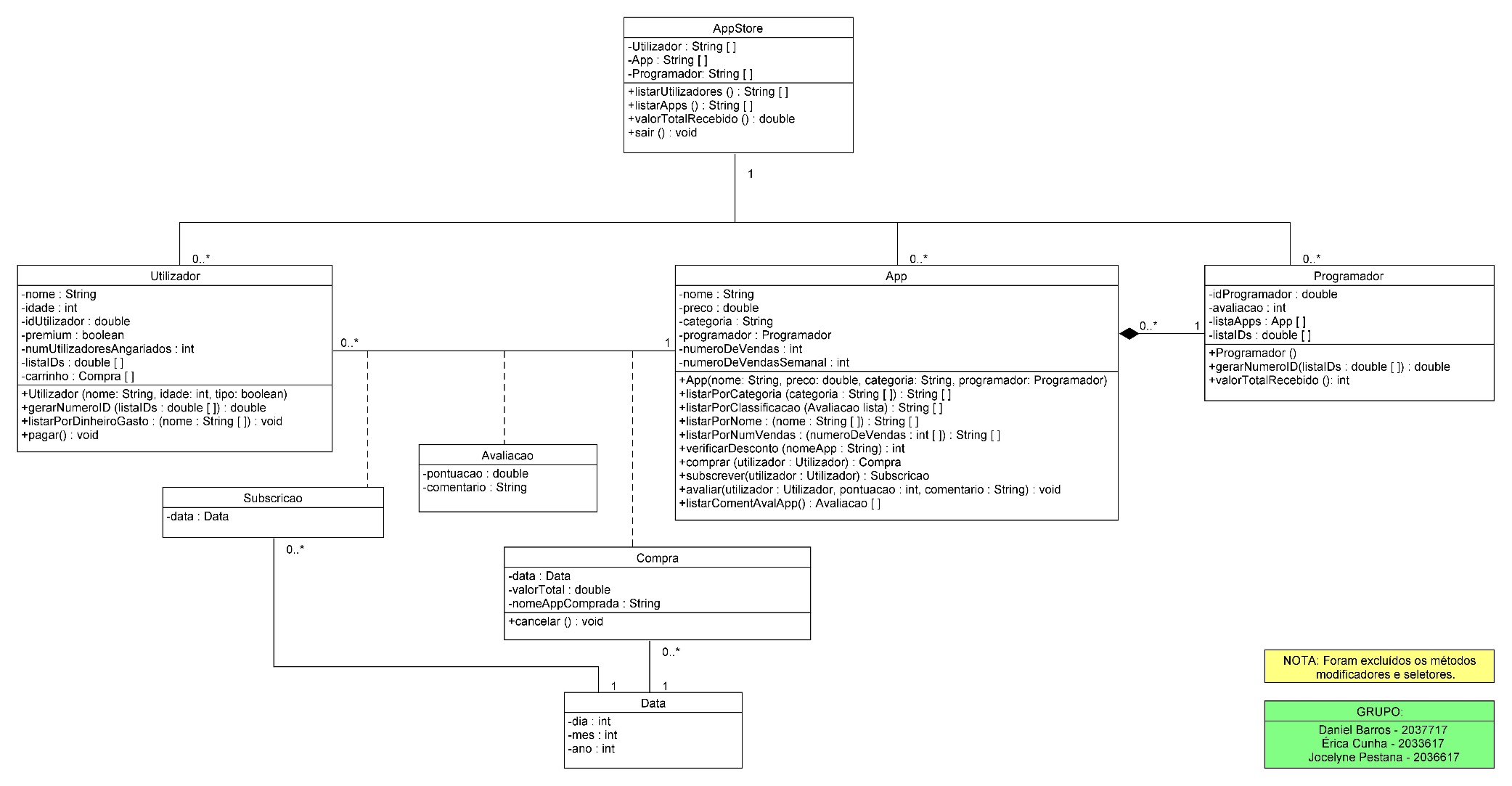


Diagrama final

