Trabalho de POO
Professor Marco Aurélio Domingues
Alunos: Alexandre Herman, Enzo Vignotti, Lucas Alves e Vitor Lorençone

DOCUMENTAÇÃO DO CÓDIGO

PROJETO DO SISTEMA

O sistema é bem simples. Como não fizemos um mecanismo de login, acabamos não utilizando uma classe “secretária” (da maneira que implementamos as outras classes de pessoas, a classe secretária seria muito próxima de um pojo, guardaria apenas atributos como nome e CPF e métodos set e get, porém, como essas informações não foram necessárias em nenhum momento, a classe secretária ficou apenas “de enfeite”, por isso optamos por removê-la) e os objetos das classes de pessoas “médico” e “paciente” são praticamente apenas para guardar informações, com um detalhe de que além dos métodos set e get, possuem métodos de exibição de informações.

O sistema funciona com base em menus e submenus e no controle do fluxo de ações. O programa se inicia com um menu inicial que oferece três opções para o usuário:

1) Acessar o menu da secretaria;
2) Acessar o menu do médico; e
3) Acessar o menu do gerenciador de mensagens.

A implementação desses menus foi feita por meio de classes compostas apenas por métodos. No caso, foram implementadas as classes:

1) Menu inicial: com funções de exibição e controle das opções mencionadas acima;
2) Menu da secretaria: reúne todos os submenus da secretaria (para gerenciamento de pacientes e consultas e para a produção de relatórios), tanto seus métodos de exibição quanto os de controle.
3) Menu do médico: idem;
4) Menu do gerenciador de mensagens: idem;

Todos os menus estão dentro do mesmo package “menus”. Optamos por organizar os menus dessa forma pois entendemos que as funções de cada menu estavam muito bem delimitadas e inchadas, as funções da secretaria, por exemplo, só poderiam ser feitas pela secretaria e todas funções que a secretária faz são fundamentais, o que nos levou a concluir que seria muito improvável uma situação em que apenas um menu de remoção fosse necessário em um contexto fora da secretaria, ou então uma situação em que a secretaria não pudesse ter acesso ao menu de cadastro de pacientes, por exemplo. Então, com base nesse raciocínio, montamos cada menu como uma unidade lógica e funcional que reúne todos os métodos necessários para seu funcionamento.

Além disso, implementamos três classes de documentos: consultas, prontuário do paciente e prontuário de atendimento, que, assim como as classes de pessoas, possuem apenas atributos, métodos set e get e alguns métodos de exibição de informações. Essas classes estão no package “documentos”. Também implementamos uma classe “banco de dados” que armazena objetos das classes de pessoas e das classes de documentos em ArrayLists estáticos e que apresenta uma série de métodos de busca, remoção, adição e exibição de dados. Por fim, foi feita uma classe “gerenciador de mensagens” para agregar os métodos de gerenciamento de mensagens.

PACKAGES E CLASSES

pkg : bancoDados

clss : BancoDeDados

ATRIBUTOS	
Todos os atributos da classe “BancoDeDados” possuem o modificador de acesso “private” e foram definidos como “static”. Como essa classe tem a função de simular um banco de dados, logicamente seus atributos (as listas que guardam as informações da clínica), não podem ser acessados de forma indiscriminada, por isso foram definidos como “private”. Além disso, como se trata de um “banco de dados”, as listas que armazenam as informações devem ser as mesmas para qualquer objeto da classe, não faz sentido cada instância da classe “BancoDeDados” ter um conjunto diferente de informações armazenadas, por isso os atributos foram declarados como “static”. Ainda pensando na integridade das informações armazenadas, essa classe não possui métodos “set”, toda alteração do conteúdo dos atributos deve ser feita por meio dos métodos que adicionam objetos a essas listas.	
`private static ArrayList<Paciente> Pacientes;`	
	>>> guarda todos os pacientes da clínica que foram cadastrados no sistema;
`private static ArrayList<Medico> Medicos;`	
	>>> guarda todos os médicos que atendem na clínica e que foram cadastrados no sistema;
`private static ArrayList<Consulta> Consultas;`	
	>>> guarda todas as consultas agendadas da clínica que foram cadastradas por meio do sistema;
`private static ArrayList<ProntuarioPaciente> ProntuarioPacientes;`	
	>>> guarda os prontuários gerais[1] de cada paciente que foram cadastrados no sistema;
`private static ArrayList<ProntuarioAtendimento> ProntuarioAtendimentos;`	

[1] Mais informações sobre prontuários gerais e prontuários de atendimento em “pkg : documentos”.

	>>> guarda os prontuários de atendimento que foram cadastrados no sistema;

MÉTODOS	
`public void adicionarPacientes(Paciente pac){}`	

	>>> Recebe um objeto “Paciente” e o adiciona na lista “Pacientes” do banco de dados;
	
	>>> Cria um objeto da classe “ProntuarioPaciente”, que representa o prontuário de um paciente, para o paciente “pac” e o adiciona à lista “ProntuariosPacientes” pois todo paciente possui um único prontuário geral e não existe prontuário geral que não seja de algum paciente
	
	>>> O método não retorna nenhum valor.

`public void adicionarMedico(Medico med){}`	
	>>> Recebe um objeto “Medico” e o adiciona à lista “Medicos”.
`public void adicionarConsulta(Consulta con){}`	
	>>> Recebe um objeto “Consulta” e o adiciona à lista “Consultas”.
`public void adicionarProntuarioAtendimento(ProntuarioAtendimento pat){}`	
	>>> Recebe um prontuário de atendimento e o adiciona à lista “ProntuariosAtendimentos”.
`public Paciente buscarPaciente(String nome){}`	
	>>> Busca um paciente na lista “Pacientes” por meio do nome do paciente e faz isso utilizando um laço “for” que itera a lista comparando o nome que foi fornecido com os “nomes” de cada objeto da lista. Quando o match ocorre, a função retorna o objeto paciente. Caso a busca não dê match, é retornado o valor null; >>> Caso a lista de pacientes esteja vazia, é printada uma mensagem na tela informando que a lista de pacientes está vazia e o método retorna null.
`public Paciente buscarPaciente(Paciente pac){}`	
	>>> Faz o mesmo processo que o outro método “buscarPaciente”, porém recebe um objeto “paciente” como parâmetro.
`public ArrayList<Consulta> buscarConsultas(){}`	
	>>> A função simplesmente retorna a lista de consultas que está armazenada no banco de dados;
`public Medico buscarMedico(String nome){}`	

	>>> Faz o mesmo que o método “buscarPaciente”, porém para os médicos.
`public Consulta buscarConsulta(int ID){}`	
	>>> Faz o mesmo que o método “buscarPaciente”, porém para encontrar consultas por meio do seu “ID”.
`public ArrayList<Consulta> buscarConsultaPaciente(Paciente PAC){}`	
	>>> Busca todas as consultas de um paciente e as retorna em uma lista, recebendo um objeto “Paciente” como parâmetro; >>> Também informa caso o paciente não possua consultas cadastradas
`public ProntuarioPaciente buscarProntuarioPaciente(String nomePaciente){}`	
	>>> Busca o prontuário geral de um paciente por meio do seu nome e o retorna; >>> Retorna null caso não encontre o prontuário;
`public ProntuarioAtendimento buscarProntuarioAtendimento(int ID){}`	
	>>> Busca um prontuário de atendimento específico por meio do seu “ID”; >>> Retorna null caso não encontre nada;
`public boolean removerPaciente(String nome){}`	

	>>> Recebe o nome de um “Paciente”, percorre a lista de Pacientes procurando pelo paciente com o nome recebido e, caso encontre, o remove da lista;
	>>> Printa uma mensagem de erro caso a lista de Pacientes esteja vazia;
	>>> Retorna true se o paciente foi removido e false em qualquer outro cenário;

`public boolean removerProntuarioPaciente(int ID){}`	
	>>> Similar ao “removerPaciente”, mas para os prontuários gerais. >>> Faz a busca por meio do “ID”.
`public boolean removerProntuarioAtendimento(int ID){}`	
	>>> Similar ao “removerPaciente”, mas para os prontuários de atendimento. >>> Faz a busca por meio do “ID”.
`public boolean removerConsulta(int ID){}`	
	>>> Similar ao “removerPaciente”, mas para os prontuários de atendimento. >>> Faz a busca por meio do “ID”.
`public void mostrarMedicos(){}`	

	>>> Printa o id e o nome de cada médico cadastrado no sistema em uma espécie de tabela;
`public void mostrarPacientes(){}`	
	>>> Printa o id e o nome de cada paciente cadastrado no sistema em uma espécie de tabela;

pkg : Documentos

Aqui em documentos, foi tomada a decisão de modelar dois tipos de prontuário, um geral e único para cada paciente, que contém as informações do paciente e seu histórico de atendimentos, e prontuários de atendimento para cada atendimento, que contém todas as informações do atendimento. Foi feita a opção de implementar dessa forma com a organização e a recuperação das informações em mente. Sempre que um prontuário de atendimento é criado, ele é adicionado tanto com banco de dados na lista “Prontuários” quando ao prontuário do paciente atendido na lista “históricoAtendimentos”, facilitando a busca de prontuários de atendimento de um paciente específico (não será necessário buscar entre todos os prontuários de todos os pacientes, basta procurar na lista com os prontuários daquele paciente específico).

classe : Consulta

ATRIBUTOS	
Os atributos dessa classe guardam as informações básicas de qualquer consulta; Todos atributos possuem o modificador de acesso “private” para garantir a segurança do acesso e da modificação das informações	
`private static int contadorID;`	
	>>> É um atributo static que é incrementado toda vez que um novo objeto “consulta” é criado para garantir que cada consulta tenha um ID único.

`private int id`	
	>>> Guarda o ID da consulta;
`private String data;`	
	>>> guarda todas as consultas agendadas da clínica que foram cadastradas por meio do sistema;
`private String horario;`	
	>>> guarda os prontuários gerais[2] de cada paciente que foram cadastrados no sistema;
`private Medico medico;`	
	>>> Guarda o objeto “Medico” do medico que realizou a consulta.
`private Paciente paciente;`	
	>>> Guarda o objeto “Paciente” do paciente marcou a consulta.
`private String tipoConsulta;`	
	>>> Guarda o tipo de consulta, que pode ser normal ou retorno;
`private String duracao;`	
	>>> Guarda a duração da consulta, 1h se for normal e 30min se for retorno; >>> É definida pelo sistema de acordo com o valor repassado para o campo “tipo de consulta” no momento de criação do objeto consulta;

MÉTODOS	
`public Consulta(){}`	
	>>> Método construtor de consulta que apenas preenche o campo ID sem

[2] Mais informações sobre prontuários gerais e prontuários de atendimento em “pkg : documentos”.

	settar o valor de mais nenhum atributo;
`public Consulta(String data, String horario, Medico medico, Paciente paciente, String tipoConsulta)`	
	>>> Método construtor que setta o valor de todos os atributos de acordo com o que foi passado como parâmetro e de acordo com as regras de negócio da clínica (duração das consultas);
`public void mostrarConsulta(){}`	
	>>> Exibe os dados da consulta em uma espécie de tabela;
`public void resumoConsulta(){}`	
	>>> Exibe os mesmos dados que “mostrarConsulta”, mas em linha;
`métodos setters e getters`	

classe : ProntuarioAtendimento

ATRIBUTOS
Os atributos dessa classe guardam as informações básicas de qualquer Prontuário; Todos atributos possuem o modificador de acesso “private” para garantir a segurança do acesso e da modificação das informações
`private static int contadorID;`
>>> É um atributo static que é incrementado toda vez que um novo objeto é criado para garantir que cada consulta tenha um ID único.
`private int id`
>>> Guarda o ID da consulta;

`private String data;`	
	>>> guarda todas as consultas agendadas da clínica que foram cadastradas por meio do sistema;
`private String horario;`	
	>>> guarda os prontuários gerais[3] de cada paciente que foram cadastrados no sistema;
`private Medico medico;`	
	>>> Guarda o objeto “Medico” do medico que realizou a consulta.
`private Paciente paciente;`	
	>>> Guarda o objeto “Paciente” do paciente marcou a consulta.
`private String tipoConsulta;`	
	>>> Guarda o tipo de consulta, que pode ser normal ou retorno;
`private String duracao;`	
	>>> Guarda a duração da consulta, 1h se for normal e 30min se for retorno; >>> É definida pelo sistema de acordo com o valor repassado para o campo “tipo de consulta” no momento de criação do objeto consulta;

MÉTODOS	
`public ProntuarioAtendimento(){}`	
	>>> Método construtor de “ProntuarioAtendimento” que apenas preenche o campo ID sem settar o valor de mais nenhum atributo;
`public ProntuarioAtendimento(String dataAtendimento, Paciente paciente, Medico medico, String sintomas, String diagnostico, String prescricao, String dataRetorno)`	
	>>> Método construtor que setta o valor de todos os atributos de acordo com o que foi passado como parâmetro;

[3] Mais informações sobre prontuários gerais e prontuários de atendimento em “pkg : documentos”.

`public void mostrarProntuarioAtendimento(){}`	
	>>> Exibe os dados do prontuário de atendimento em uma espécie de tabela;
`public void resumoConsulta(){}`	
	>>> Exibe os mesmos dados que “mostrarConsulta”, mas em linha;
`métodos setters e getters`	

classe : ProntuarioPaciente

ATRIBUTOS	
Os atributos dessa classe guardam as informações básicas de qualquer prontuário de um paciente, que é feito para ser guardado no banco de dados, mesmo quando o prontuário oficial ainda não foi disparado pelo médico; O objetivo dessa classe é guardar todos os prontuários que um paciente teve durante suas consultas na clínica; Todos atributos possuem o modificador de acesso “private” para garantir a segurança do acesso e da modificação das informações.	
`private static int contadorID;`	
	>>> É um atributo static que é incrementado toda vez que um novo objeto é criado para garantir que cada consulta tenha um ID único.
`private int id`	
	>>> Guarda o ID do Prontuário;
`private String dataAtendimento;`	
	>>> Guarda a data do atendimento registrado pelo prontuário
`private Paciente paciente;`	
	>>> Recebe o paciente em questão.

`private Medico medico;`	
	>>> Recebe o médico em questão.
`private String sintomas;`	
	>>> Recebe os sintomas do paciente.
`private String diagnostico;`	
	>>> Recebe o diagnóstico da consulta.
`private String prescricao;`	
	>>> Recebe a prescrição médica da consulta.
`private String dataRetorno;`	
	>>> Recebe uma data de retorno para o paciente.

MÉTODOS	
`public ProntuarioPaciente(Paciente PAC){}`	
	>>> Método construtor do Prontuário que seta o valor paciente em questão com o PAC e cria uma lista vazia para o histórico de atenimentos;
`public mostrarProntuarioPaciente(){}`	
	>>> Método que imprime no console a lista do histórico do paciente, para que possa ser visualizada
`métodos setters e getters`	

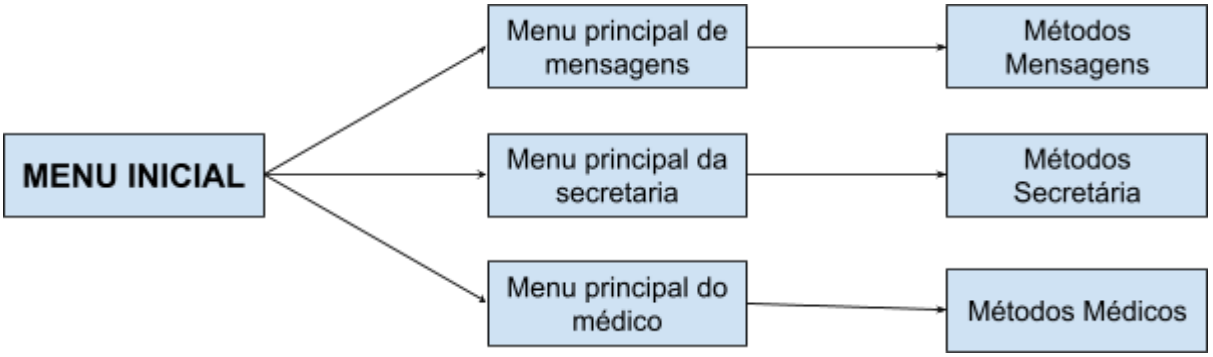
pkg : Menu

A criação do pacote menu tem como finalidade organizar todos os métodos/funções que cada usuária terá na clínica pura saúde. Inicialmente teremos

o menu inicial onde tem como objetivo direcionar o usuário para o seu devido menu correto.

Dentre este menus, teremos:

1. Menu de mensagens;
2. Menu da secretária; e
3. Menu do médico.



- Ilustração sobre sequência de passos para o acesso do menu correto.

classe : MenuInicial

MÉTODOS A finalidade do Menu Inicial é direcionar para outros menus.	
`public menuPrincipal(){}`	
	>>> Método simples de escrita de um menu para futura seleção de tarefas pelo usuário, podendo apontar para o menu da secretária, do médico ou do envio de mensagens;

classe : MenuMensagens

MÉTODOS A finalidade das Mensagens de enviar mensagens SMS ou E-mail para cada paciente antes da sua consulta
`public boolean menuPrincipal(){}`

	>>> Método de um menu principal, para chamar os outros métodos.
`public void gerenciarMensagens(){`	
	>>> Método que imprime todas as mensagens de SMS ou E-mail que foram enviadas neste dia, ou seja, de consultas que acontecerão no próximo dia.
`public void enviarMensagens(){`	
	>>> Método para enviar uma mensagem adicional a um paciente.

classe: MenuSecretaria

MÉTODOS
A finalidade da secretária é gerenciar dados de pacientes e consultas, para isso separamos em três partes: 1. Gerenciamento de pacientes; 2. Gerenciamento de consultas; 3. Gerenciamento de Relatórios. Para cada gestão teremos os seguintes métodos: cadastrar, atualizar e remover. 1. GERENCIAMENTO DE PACIENTES
`public void menuPrincipal(){}`
>>> Método de um menu principal, para chamar os outros métodos.
`public boolean menuGerenciarPacientes(){`
>>> Método de um menu de Gerenciamento de pacientes, para chamar os outros métodos.
`public boolean menuGerenciarConsultas(){`
>>> Método de um menu Gerenciamento de consultas, para chamar os outros métodos.
`public boolean cadastrarPacientes(){}`

>>> Método com finalidade de cadastrar um novo paciente no sistema obtendo suas informações básicas, tais como: nome, data de nascimento, endereço, contato celular, e-mail, tipo de convênio.
`public boolean atualizaPaciente(){}`
>>> Método com finalidade de atualizar os dados de um paciente já cadastrado no sistema. Dentro deste método é possível escolher qual dados a secretaria irá atualizar, sem ter a necessidade de alterar todos os dados, mas sim apenas o essencial.
`public boolean removerPaciente(){}`
>>> Método com a finalidade de remover um paciente do sistema a partir do momento que ocorre o cadastramento do mesmo.
`public boolean gerarRelatorio(){}`
>>> Método de um menu Gerenciamento de Relatórios, para chamar os outros métodos.
`public boolean consultasDiaSeguinteComContato(){}`
>>> Método com a finalidade de realizar a busca por consultas do próximo dia de pacientes que possuem contato (SMS ou E-mail) preenchidos
`public boolean consultasDiaSeguinteSemContato(){}`
>>> Método com a finalidade de realizar a busca por consultas do próximo dia de pacientes que não possuem contato (SMS ou E-mail) preenchidos
`public boolean cadastrarConsulta(){}`
>>> Método com finalidade de cadastrar uma nova consulta no sistema obtendo suas informações básicas.
`public boolean atualizarConsulta(){}`
>>> Método com finalidade de atualizar os dados de uma consulta já cadastrado no sistema.

`public boolean deletarConsulta(){}`
>>> Método com a finalidade de remover uma consulta do sistema.

classe : MenuMédico

MÉTODOS A finalidade do menu médico é gerenciar dados de pacientes e prontuários, para isso separamos em três partes: 1. Prontuários; 2. Ficha do Paciente; 3. Gerenciamento de Relatórios.	
`public boolean menuPrincipal(){}`	
	>>> Método de um menu principal, para chamar os outros métodos.
`public boolean menuProntuariosInicial(){}`	
	>>> Método de um menu de prontuários que direciona para outros métodos
`public boolean menuProntuariosSecundario(String nome){}`	
	>>> Método de um menu de prontuários que direciona para outros métodos
`public boolean menuFichaPaciente(String nome){}`	
	>>> Método de um menu de fichas de pacientes que direciona para outros métodos
`public boolean menuRelatorios(){}`	
	>>> Método de um menu de relatórios que direciona para outros métodos
`public void receita(){}`	
	>>> Método que gera e imprime uma receita médica para um paciente.
`public void atestado(){}`	

	>>> Método que gera e imprime um atestado médica para um paciente.
`public void declaracaoAcompanhamento(){}`	
	>>> Método que gera e imprime uma declaração de acompanhamento médico para um paciente.
`public void clientesNoMes(){}`	
	>>> Método que calcula e imprime o número de clientes atendidos no mês.
`public boolean consultarProntuarioPaciente(String nome){}`	
	>>> Método que procura e imprime no terminal o prontuário de um dado paciente.
`public boolean cadastrarProntuarioAtendimento(String nome){}`	
	>>> Método que cadastra um Prontuário de um paciente e guarda ele no banco de dados.
`public boolean atualizarProntuarioAtendimento(String nome){}`	
	>>> Método que atualiza uma informação específica de um prontuário de atendimento.
`public boolean apagarProntuarioAtendimento(String nome){}`	
	>>> Método que apaga do banco um prontuário de atendimento.
`public boolean adicionarDadosComplementares(String nome){}`	
	>>> Método que adiciona informações extras de um paciente que apenas o médico pode inserir.
`public boolean atualizarDadosComplementares(String nome){}`	
	>>> Método que adiciona atualiza extras de um paciente que apenas o médico pode inserir.

pkg : Pessoas

Objetos que representam os tipos de usuários que a clínica pura saúde terá dentro do seu sistema.

classe : Pacientes

ATRIBUTOS
Os atributos dessa classe guardam as informações básicas e específicas de cada paciente cadastrados no sistema. Todos atributos possuem o modificador de acesso “private” para garantir a segurança do acesso e da modificação das informações **As informações abaixo são acessadas pela secretaria e pelo médico do paciente:**
`private static int contadorID;`
>>> É um atributo static que é incrementado toda vez que um novo objeto “paciente” é criado para garantir que cada paciente tenha um ID único.
`private int id`
>>> Guarda o ID da do paciente;
`private String nome;`
>>> Guarda o nome de cada paciente.
`private String data_nascimento;`
>>> Guarda a data de nascimento do paciente seguindo esta ordem : dia/mês/ano.
`private String endereco;`
>>> Guarda o endereço de onde mora o paciente.

`private String info_contatoCelular;`
>>> Guarda informação de contato telefônico do paciente para finalidade de se for necessário entrar em contato com o mesmo.
`private String info_contatoEmail;`
>>> Guarda informação de contato com paciente através de e-mail, com finalidade de se for necessário entrar em contato com o mesmo.
`private String tipo_convenio;`
>>> Guarda informações de qual tipo de convênio o paciente possui.
As informações abaixo são apenas acessadas pelo médico do paciente, para auxiliar na consulta:
`private String fuma;`
>>> Guarda informação do paciente se ele é/ou não fumante.
`private String bebe;`
>>> Guarda informação do paciente se ele consome ou não bebidas alcoólicas.
`private String nivelColesterol;`
>>> Guarda informação do nível do colesterol do paciente.
`private String diabete;`
>>> Guarda informação do nível de diabetes do paciente.
`private String doencaCardiaca;`
>>> Guarda informação do paciente se ele é/ou não cardíaco.
`private String cirurgias;`
>>> Guarda informação do paciente se ele realizou ou não cirurgias.

`private String alergias;`
>>> Guarda informação do paciente se ele possui ou não alergias.

MÉTODOS
`public Paciente(String nome, String data_nascimento, String endereco, String info_contatoCelular, String info_contatoEmail, String tipo_convenio){}`
>>> Método construtor um objeto paciente, tendo a finalidade de receber as informações do paciente que está sendo construído no sistema.
`public void setDadosComplementares(String fuma, String bebe, String col, String diab, String card, String cir, String aler){}`
>>> Método que complementa as informações do paciente que foram obtidos na hora do seu cadastramento.
`public void vizualizar_dados(){}`
>>> Exibe os dados do paciente em uma espécie de tabela;
`public void vizualizarDadosCompleto(){}`
>>> Exibe os dados completos do paciente em uma espécie de tabela;
`métodos setters e getters`

classe : Médico

ATRIBUTOS
Os atributos dessa classe guardam as informações básicas e específicas de cada médico cadastrados no sistema. Todos atributos possuem o modificador de acesso “private” para garantir a

segurança do acesso e da modificação das informações
`private static int contadorID;`
>>> É um atributo static que é incrementado toda vez que um novo objeto “médico” é criado para garantir que cada paciente tenha um ID único.
`private int id`
>>> Guarda o ID da do médico;
`private String nome;`
>>> Guarda o nome de cada médico.
`private String CPF;`
>>> Guarda o CPF do médico.
`private ArrayList<Paciente> pacientesAtendidos;`
>>> Guarda uma lista de pacientes atendidos pelo médico.
`private String info_contatoCelular;`
>>> Guarda informação de contato telefônico do paciente para finalidade de se for necessário entrar em contato com o mesmo.

MÉTODOS
`public Medico(String nome, String CPF){}`
>>> Método construtor um objeto médico, tendo a finalidade de receber as informações do médico que está sendo construído no sistema.
`métodos setters e getters`

pkg : gerenciadorMensagens

classe : GerenciadorMensagens

ATRIBUTOS	
Todos os atributos da classe “GerenciadorMensagens” possuem o modificador de acesso “private” e foram definidos como “static”. A classe tem como objetivo simular um gerenciador de mensagens, que consta com uma data definida e métodos para o envio de mensagens propriamente, que são feitos com prints na tela, simulando o envio.	
`private static String dataAtual;`	
	>>> guarda a informação da data atual do gerenciador, sendo “static”, ou seja, o mesmo em cada objeto

MÉTODOS	
`public GerenciadorMensagens(){}`	
	>>> Método construtor que apenas instancia o objeto em memória sem settar o valor de mais nenhum atributo;
`public void enviarMensagens(ArrayList<Consulta> Consultas){`	
	>>> Método para verificar a data e enviar as mensagens de SMS e E-mail
`public void enviarSMS(Paciente paciente, Consulta consulta){`	
	>>> Envia a mensagem de SMS para o paciente.
`public void enviarEmail(Paciente paciente, Consulta consulta){`	
	>>> Envia a mensagem de E-mail para o paciente.
`public boolean proxDia(String d1, String d2){`	
	>>> Função lógica apenas para comparar duas strings “d1” e ”d2”, para verificar se um é o próximo dia da outra.
`métodos setters e getters`	

pkg : principal

classe : Main

MÉTODOS Nossa classe principal para a execução do código e unificação de todas as outras classes.	
`public static void main(String args[]){`	
	>>> Método main do código, definindo informações como pacientes já cadastrados no Banco de Dados, Data atual, dentre outras informações que são usadas apenas para iniciar o código e executar ações nos menus.