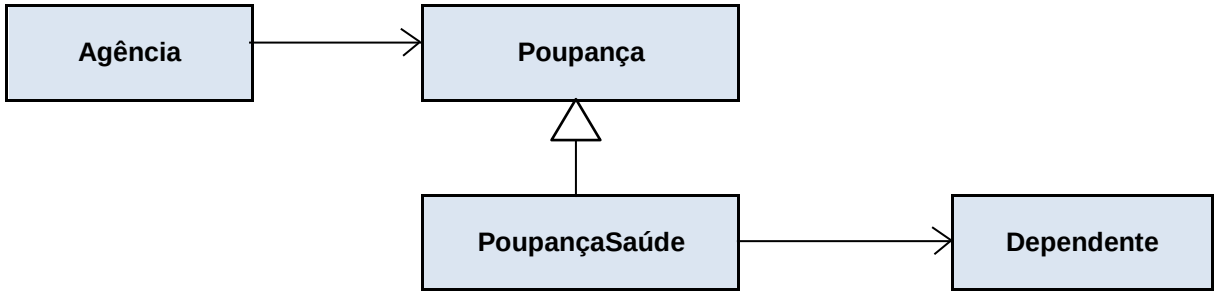
Em grupos de até 2 alunos. O trabalho é extraclasse – **NÃO PODE SER FEITO DURANTE AS AULAS** salvo com autorização do professor.

Simulação (simplificada) do sistema de poupança de um Banco que oferece contas de poupança normais e um tipo especial de poupança que funciona como um plano de saúde.



Classe **Poupanca**

- Os objetos desta classe são contas normais de poupança, que permitem depósitos e retiradas e ganham rendimentos. Esta classe é dada. Os alunos devem copiá-la para seu projeto e não podem alterar nada na sua programação.

```
public class Poupanca{
    private int numero;
    private String cliente; // nome do cliente
    private double saldoLivre;

    public Poupanca(int n, String nome){
        this.numero = n;
        this.cliente = nome;
        this.saldoLivre = 0;
    }

    public void deposita(double valor){
        saldoLivre += valor;
    }

    public boolean retira(double valor){
        if (saldoLivre < valor)
           return false;
        saldoLivre -= valor;
        return true;
    }

    public double creditaRendimento(double taxa){ // a taxa é decimal
        double rendimento = taxa * saldoLivre;
        saldoLivre += rendimento;
        return rendimento;
    }

    public String toString(){
        return "Num: " + numero + "  Cliente: " + cliente +
               "\nSaldo livre: R$ " +  saldoLivre;
    }

    public String getCliente(){return cliente;}
    public int getNumero(){return numero;}
    public double getSaldoLivre(){return saldoLivre;}
}
```

Programe as três classes seguintes. Escolha bons nomes para os membros. Não use acentuação nem "ç". Use o seguinte comentário no início do texto fonte de cada classe:
/ Alunos : xxxxxxxxxxxx e xxxxxxxxxx Trabalho B Lab 1 Prof. Aníbal 2016/2 */**

Classe **Dependente**

Atributos privados (somente estes):
- nome
- parentesco – do tipo **char**, poderá ter valores 'c': cônjuge, 'f': filho(a); 'p': progenitor (pais, avós); 'o': outro

Construtor: um só, com dois parâmetros, para nome e parentesco. Chamar o método *setParentesco*.

Métodos:
+ *setParentesco* – se o parâmetro tiver um parentesco inválido, atribuir 'o'.

+ *traduzParentesco* – retorna o parentesco por extenso.

+ *toString* – retorna numa única linha o nome e o parentesco por extenso.

+ *gets* – apenas se necessário(s) para o funcionamento do sistema.

Classe **PoupancaSaude**

Atributos privados (só estes, além dos herdados):
- saldo vinculado – só pode sofrer retiradas do tipo saúde, ou seja, que irão cobrir despesas com saúde
- saldo financiado – saldo que o cliente deve ao banco por ter este financiado (emprestado) dinheiro para complementar alguma retirada saúde que não teve cobertura pelos outros saldos
- um array de objetos do tipo **Dependente**

Construtor: recebe número da conta e nome do cliente. O array de dependentes deve ser instanciado com tamanho 5, vazio.

Métodos:
+ *contaDependentes* – retorna a quantidade de dependentes existentes no atributo array de dependentes.

+ *deposita* – **Sobrescreve** o método de mesmo nome da superclasse. Retém uma parte do valor depositado para acumular ao saldo vinculado, de acordo com a quantidade de dependentes que a conta tem: 15% do valor depositado, se não tem dependente; 20%, se tem até 2 dependentes; 30%, se tem 3 ou 4 dependentes; 50%, se tem 5 dependentes. A parte restante vai ser acumulada no saldo livre.

+ *creditaRendimento* – **Sobrescreve** o método de mesmo nome da superclasse. Aplica a mesma taxa para atualizar o saldo livre e o saldo vinculado. Retorna a soma dos valores creditados.

+ *insereDependente* – recebe, via parâmetro, um objeto do tipo **Dependente**, insere-o no array de dependentes em alguma posição livre e retorna true. Se não houver posições livres no array, retornar false para indicar o insucesso da operação.

+ *buscaDependente* – recebe um nome de pessoa e procura no array de dependentes um objeto com este nome. Se achar, retornar o índice do array onde ele está. Se não achar, retornar o valor 99.

+ *retiraDependente* – recebe um nome de pessoa, procura por ele no array de dependentes, chamando o método anterior para isso, e, se achar, retira-o do array e retorna o objeto retirado. Se não houver dependente com este nome, retornar null.

+ *retiraSaude* – o valor recebido via parâmetro corresponde a uma despesa de saúde e deve ser abatido do saldo vinculado. Se este saldo não for suficiente para absorver toda a despesa, o usuário pode optar por usar o saldo livre, ou um financiamento (empréstimo) do banco, ou ambas as coisas, para completar o valor restante da despesa.

O método deve exibir este valor restante da despesa, além do saldo livre atual da conta e pedir ao cliente que digite o valor que ele quer usar do saldo livre. Validar esta escolha, com repetição, dando a mensagem

Valor muito grande, redigite!

se o usuário digitar valor maior que o saldo livre ou maior que o resto da despesa que ainda está a descoberto. Atualizar devidamente o saldo livre. Se, após isso, ainda

restar valor da despesa de saúde, então o sistema irá tratar como financiamento (empréstimo) e, sobre este restante, aplicar juros de:

5%, se o saldo financiado atual ainda é zero;
10%, se já existe um saldo financiado de até R$ 500,00; e
15%, se o saldo financiado atual é superior a R$ 500,00.

Atualizar o saldo financiado. O método deve retornar o valor da despesa que foi financiado acrescido dos juros, ou zero, se não houve financiamento.

+ *amortizaFinanciamento* – este método será chamado sempre que o cliente desejar pagar parte ou todo o saldo financiado existente na sua poupança saúde. Recebe o valor a ser amortizado como parâmetro. Se o valor recebido é maior que o saldo financiado, retorna 0 e encerra o método. Caso contrário, abate o valor recebido do saldo financiado. Se este saldo fica zerado com a transação, o banco retribui o cliente com um depósito no valor de 5% sobre o valor do pagamento (chamar o método deposita para implementar isso). Ao final, o método deve retornar o valor deste depósito, se ocorreu, ou zero, se o saldo não foi liquidado.

- *ordenaDependentes* – ordena o array de dependentes por ordem alfabética de nome. Atenção para o fato de que este array pode ter posições null intercaladas, devido às operações de inserção e retirada de dependentes.

+ *toString* – retorna os valores de todos os atributos da conta, com títulos adequados. Se a conta tem dependentes, os valores de seus atributos devem vir ao final, um por linha, ordenados em ordem alfabética de nome. Chamar o método anterior para esta ordenação.

Atenção: nenhum método get ou set

Classe **Agencia**

Atributos privados (somente estes):
- array de objetos do tipo **Poupanca**
- contador que mede a quantidade de contas existentes no array

Construtor: só terá um parâmetro, que é a quantidade máxima de contas que o array pode receber. Instanciar o array de contas deixando-o ainda vazio.

Métodos privados:
- *abreConta* – será chamado para inserir uma conta **Poupanca** ou **PoupancaSaude** no atributo array de contas. Se o array estiver lotado, retornar -1 e encerrar o método. Pedir ao usuário para indicar o tipo de conta que ele deseja abrir, digitando no teclado 1, para Poupança simples ou 2, para Poupança Saúde. Se o usuário digitar qualquer outro valor, valerá como opção 2. A seguir, ler do teclado o número da conta e o nome do cliente e instanciar um objeto **Poupanca** ou **PoupancaSaude**, conforme o tipo escolhido pelo usuário. No segundo caso, permitir que o usuário insira um ou mais dependentes, usando uma repetição **while** que, a cada volta, exibe a pergunta *Deseja inserir mais um dependente (s/n)?* que o usuário deve responder com 's' ou 'n'. Parar quando a resposta for diferente de 's' ou não houver mais capacidade para dependentes na conta. Ao final, retornar o número da conta que foi aberta.

- *buscaConta* – recebe um inteiro e procura no atributo array de contas se existe alguma com tal número. Se houver, retornar seu índice no array. Se não houver, retornar -1.

Método público:
+ *menuDeTransacoes* – será chamado para permitir operar o sistema da agência. Exibir, repetidamente, o seguinte menu de opções:

1 – Abre conta
2 – Deposita
3 – Retira
4 – Retira para saúde
5 – Amortiza financiamento
6 – Emite extrato da conta
7 – Credita rendimentos
8 – Insere um dependente
9 – Retira um dependente
10 – Encerra

O usuário deve digitar o número da opção escolhida. Se inválida, deve repetir a escolha, até acertar. Comandar o devido processamento da opção, conforme descrição abaixo, e voltar a exibir o menu. A opção 10 encerra a repetição e o método.

Opção 1: Chamar o método adequado para isso. Exibir uma das duas mensagens, para indicar o sucesso ou não da operação: *Não pode abrir novas contas nesta agência* ou *Conta aberta de número: xx*

Opção 2: Usuário deve digitar o número da conta. Buscar conta com esse número. Se não existe, encerrar com a mensagem *Conta inexistente.* Para conta válida, comandar o depósito do valor que deve ser digitado pelo usuário.

Opção 3: Usuário digita número da conta. Se não existe, encerrar com a mensagem *Conta inexistente.* Comandar a retirada e, se for o caso, emitir a mensagem *Saldo insuficiente.*

Opção 4: Usuário digita número da conta. Encerrar com a mensagem *Conta inexistente ou não é Poupanca Saude*, se uma dessas situações ocorrer. Comandar a retirada e exibir *Valor financiado: R$ xxx*.xx se houver sucesso.

Opção 5: Usuário digita número da conta. Encerrar com uma das duas mensagens, *Conta inexistente* ou *Tipo de conta não aceita esta operação,* conforme o motivo do insucesso. Obter via teclado o valor e comandar a devida amortização. Encerrar com a mensagem *Ganhou desconto-depósito de R$ xxx.xx.*

Opção 6: Usuário digita número da conta. Se não existe, encerrar com a mensagem *Conta inexistente.* Imprimir os dados da conta.

Opção 7: Pedir ao usuário para digitar a taxa de rendimento. Usando esta taxa, comandar o crédito de rendimentos em todas as contas da agência e encerrar com a mensagem *Total creditado em todas as contas: R$ xxxx.xx.* Este valor é a soma de todos os rendimentos creditados.

Opção 8: Usuário digita número da conta. Encerrar com a mensagem *Conta inexistente ou não é Poupanca Saude*, se uma dessas situações de insucesso ocorrer. Instanciar um objeto **Dependente,** lendo dados necessários do teclado, e comandar a inserção deste objeto na conta. Pode ocorrer, ainda, novo insucesso que será traduzido na mensagem *Esta conta não admite mais dependentes.*

Opção 9: Usuário digita número da conta. Encerrar com a mensagem *Conta inexistente ou não é Poupanca Saude*, se uma dessas situações ocorrer. Comandar a retirada do dependente, pedindo ao usuário seu nome via teclado. Pode ocorrer, ainda, novo insucesso que será traduzido na mensagem *Não existe dependente com este nome nesta conta.*

Atenção: nenhum método get ou set.

Entrega:

No dia 1/11/2016 – Apresentar ao professor, em aula, em papel:
a) completar o diagrama de classes que aparece no início do enunciado
b) um diagrama de objetos de uma Agência que tem capacidade de no máximo 4 contas, mas, no momento, tem uma conta do tipo Poupança simples e outra do tipo PoupancaSaude com dependentes. Nomes e valores coerentes a escolha dos alunos.

Até dia 24/11/2016 - Remeter mail para anibalpc53@gmail.com com o assunto:
Lab 1 – Trabalho B de *Fulano* e *Beltrano*,
anexando os arquivos P**oupancaSaude.java, Dependente.java** e **Agencia.java** (só estes arquivos). Não esquecer o seguinte comentário no início do texto fonte de cada classe:

/ Alunos : xxxxxxxxxxxxx e xxxxxxxxxxx Trabalho B Lab 1 Prof. Aníbal 2016/2 */**

Anexar apenas os arquivos .java.

Trabalhos entregues antes da data poderão receber bonificação na nota.