1a) Implemente a classe **Gerente** herdando a classe **Funcionario** (Exercício 1 da lista 4). Além dos dados do funcionário, o gerente tem mais um atributo para armazenar o nome do secretário e os métodos get e set para este atributo. A classe Gerente tem dois construtores: um para criar um gerente sem secretário colocando "Cargo Vago" no atributo nomeSecretario e outro para criar o gerente com secretário.

1b) Implemente uma classe de teste para instanciar gerentes e funcionários **usando a mesma variável (do tipo Funcionario)** como segue:

Leia (do teclado) a quantidade **n** de funcionários (e/ou gerentes). Use um comando de repetição para ler e processar os n funcionários e dentro do laço faça o que segue:

Cada vez, sorteie o tipo (1-gerente 2-funcionário). Leia (do teclado) os dados e se for gerente, leia o nome do secretário e nomeie o secretário. Logo após, leia e valide o código de reajuste, exibindo uma mensagem de erro e lendo novamente, cada vez que for digitado um código inválido. Aplique o reajuste, invocando o método que recebe apenas o código de reajuste.

A cada iteração, faça o que for necessário para obter os dados e totais que devem ser exibidos após a repetição.

Após a leitura e processamento dos **n** funcionários e/ou gerentes, exibir:

- O valor da folha de pagamento (salário total da empresa) após o reajuste.
- O funcionário ou gerente (todos os dados) com o maior salário, após o reajuste.
- O salário médio, após o reajuste, dos funcionários (incluindo os gerentes) do departamento 2

1c) Modifique o exercício anterior, para que não seja necessário ler a quantidade de funcionários. Cada vez perguntar "Quer entrar com dados [S-sim N-não]?" Encerrar a entrada de dados quando for digitado 'N'ou 'n'

```
char resp;
 do {
        resp= Character.toUpperCase(t.leChar("Quer entrar com dados [S-sim N-não] ? "));
 } while (resp != 'S'&& resp != 'N');
```

2) Seja a classe **Avaliacao**:

```
public class Avaliacao {
      private double gA;
      private double gB;
      private double mF;

      public Avaliacao(double gA,double gB){
           setNotaGa (gA);
           setNotaGb (gB);
           calculaMediaFinal();
      }
     public double getGrauA(){
          return gA;
      }

     public double getGrauB(){
          return gB;
      }
     public double getMediaFinal(){
          return mF;
      }

      private void setNotaGa (double nota){
          if (nota>=0 && nota<=10)
              gA = nota;
      }
       private void setNotaGb (double nota){
          if (nota>=0 && nota<=10)
              gB = nota;
      }

      private void calculaMediaFinal(){
          mF = gA*0.33 + gB*0.67    ;
     }


}
```

2 a) Programe a superclasse ***Pessoa*** e uma subclasse ***Aluno*** que herde de *Pessoa.* A pessoa tem nome e ano de nascimento. O aluno tem nome, ano de nascimento e avaliação. Escreva construtores para ambas as classes e métodos para retornar o valor de todos os atributos. Sobreescreva o método ***toString( )*** *da classe Object,* em *Pessoa,* para que devolva um String contendo os valores dos atributos, um em cada linha, com título, e sobrescreva-o também em *Aluno.* Faça o método maisVelha na superclasse *Pessoa* que retorna a mais velha de duas pessoas. O método tem a seguinte declaração: **`public Pessoa maisVelha(Pessoa outra)`**

Obs. a classe Aluno deve ter dois construtores; um que recebe o nome, o ano de nascimento e o objeto avaliação e outro que recebe o nome, o ano de nascimento e as notas.

2 b) Programe uma classe de teste para o exercício anterior. Instancie pessoas e alunos, obtendo seus dados via teclado. Encerrar a leitura dos dados quando for lido ano de nascimento 0 (flag). Cada vez, solicitar a entrada dos dados comuns *(primeiro o ano de nascimento e se não for o flag, solicitar o nomee)* e, a seguir, solicitar a digitação o tipo ('A' ou 'P'). Se for aluno, devem ser lidos os outros dados, além dos dados já lidos. No final, após a leitura de todos os dados exibir os dados da pessoa *(que pode ser um aluno)* mais velha e a média geral dos alunos.

3 a) Implemente a hierarquia de classes:
ContaBancaria superclasse que contém o que é comum nas duas subclasses,
ContaCorrente (número, saldo , quantidade de transações realizadas (depósito, retirada e extrato) e duas constantes: número de transações livres (3) e valor cobrado pela transação (10 centavos))
ContaPoupanca (número, saldo e taxa de rendimento).
Faça o diagrama UML, antes de programar.

Deve ser possível sacar, depositar e consultar o extrato na conta corrente e na conta de poupança.
A conta corrente deve ter um método que será chamado a cada fim de mês para debitar o valor das transações que ultrapassaram a quantidade de transações livres.
A conta de poupança deve ter um método que será chamado a cada fim de mês para creditar o rendimento.

3 b) Implemente a classe **Menu** para operar sobre a hierarquia acima.

3 c) Implemente uma classe de teste, usando as classes acima.

4) a) Escreva a classe **FiguraBidimensional**
Atributo: cor
Construtor: um só que recebe como parâmetro a cor da figura
Métodos:
setCor – um parâmetro – muda a cor da figura
getCor - devolve o valor da cor

Classe **Círculo** – subclasse de FiguraBidimensional
Atributo: raio
Construtor: um só que recebe como parâmetros a cor e o raio Métodos:
calculaArea
calculaPerímetro
getRaio

Classe **Retângulo** – subclasse de FiguraBidimensional
Atributos: base e altura
Construtor: um só que recebe como parâmetros a cor, a base e a altura
Métodos:
calculaArea
calculaPerímetro
getBase
getAltura

4 b) Implemente a classe **Menu** para operar sobre a hierarquia acima.

4 c) Implemente uma classe de teste, usando as classes acima.