UEMG - Universidade Estadual de Minas Gerais
PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA
LISTA DE EXERCÍCIOS (Compilação) – Versão 2.0

1) Escreva uma classe **Conta** que contenha o nome do cliente, o numero da conta, o saldo e o limite. Estes valores deverão ser informados no construtor, sendo que o limite não poderá ser maior que o valor do salário mensal do cliente. Faça um método deposito e um método retira. O método retira ira devolver *true* ou *false*, dependendo se o cliente pode retirar. Faça um método **saldo** que retorne o saldo do cliente.

2) Escreva uma classe **ContaEspecial** que funciona como a classe do exercício 1, mas que aceite um limite de até 3x o valor do salário do cliente.

3) Escreva uma classe **Cartao** que receba um objeto do tipo conta e uma senha. Devera conter um método **saque** e um método **saldo**, semelhante à classe do exercício 1, mas que receba uma senha que deverá ser a mesma armazenada no cartão. Faça também um método que altere a senha, desde que receba a senha antiga como parâmetro.

4) Escreva uma classe **Produto** que contenha o número serial, o volume (inteiro) e também uma *string* que inicialmente possui o valor "não testado". O número de série será passado no construtor. Possuirá um método *boolean* **testaUnidade** que somente poderá ser executado uma única vez. O produto terá 90% de chance de estar OK. Caso esteja OK, a string passara de "não testado" para "aprovado". Caso não esteja OK, passara para "reprovado". Retorna *true* se foi aprovado e *false* se não foi. Deverá também conter um método **alteraVolume** e um método *toString* que retornara em uma *string* do número de série, o volume e o resultado do teste. (dica: java.lang.Math.random() gera um numero de 0.0 a 1.0)

5) Escreva uma classe **Radio** que herdará de **Produto**. Deverá ter um método **escutar** que retornará uma *String* contendo a estação e a banda (ex.: 94.9 FM) da rádio. Deverá conter um método **trocaEstacao** e um método **trocaBanda**. Deverá alterar o método *toString* de forma a acrescentar a estação e a banda.

6) Escreva uma classe **TV** que herdará **Produto**. Deverá ter um método **assistir** que retornará uma *String* contendo o canal que esta assistindo. Conterá um método **trocaCanal**. Alterará o *toString* de forma a acrescentar o canal.

7) Escreva uma classe **Controle** que ira receber um produto, testá-lo e imprimir seu status (método *toString*). Faca um programa principal que a utilize em conjunto com rádios e TVs.

8) Escreva o jogo **Papel**, **Pedra** e **Tesoura**. Deverá conter uma classe **Coisa** que será a superclasse de **Papel**, **Pedra** e **Tesoura**. Deverá conter uma classe **Jogo** que jogará o tempo todo. (Aqui vem o famoso: Coisa c1 = new Pedra();) Faca o computador escolher aleatoriamente que objetos ele escolherá e imprimirá os resultados. Note que durante os confrontos o método que receber os dois objetos deverá receber duas coisas sem saber exatamente qual o tipo. Neste caso, usando os métodos específicos de cada um deverá realizar a comparação. **Papel** ganha de **Pedra** (envolve), **Pedra** ganha de tesoura (quebra) e **Tesoura** ganha de **Papel** (**Corta**).

9) Crie uma classe que representa um ponto no plano cartesiano. Em seguida, crie uma classe que representa um triângulo, reusando a classe anterior por composição. Finalmente, escreva um programa que receba do usuário as coordenadas dos vértices do triângulo e imprima seu perímetro.

10) Crie uma classe que representa um funcionário, registrando seu nome, salário e data de admissão. Em seguida, crie uma classe que represente um departamento de uma empresa, registrando o nome e os funcionários que nele trabalham (para uso de vetores, considere um máximo de 100 funcionários). Por fim, crie uma classe que representa uma empresa, registrando seu nome, CNPJ e departamentos (considere um máximo de 10 departamentos). Faça um programa que:
    - Criar uma empresa;
    - Adicionar a esta empresa alguns departamentos;
    - Adicionar aos departamentos alguns funcionários;
    - Dar aumento de 10% a todos os funcionários de um determinado departamento;
    - Transferir um funcionário de um departamento para outro.

    É esperado que seu código seja bem encapsulado. Por exemplo, para adicionar um departamento em uma empresa (ou um funcionário a um departamento), não se deve acessar o vetor (ou lista) de departamentos diretamente, mas sim ter um método na classe que representa a empresa para adicionar um departamento.

11) Projete e implemente um sistema que modele um banco. Seu projeto deve permitir a criação de vários bancos e várias contas para cada banco. Para um dado banco deve ser possível:
    - Obter seu nome;
    - Obter seu código;
    - Criar uma nova conta.

    Não armazene as contas criadas no banco: quando um banco cria uma conta ele deve apenas retornar o objeto que modela essa conta, assuma que quem pediu a criação da conta vai se encarregar de guardar essa conta em algum lugar (ou seja, não se preocupe com a armazenagem das contas). Para cada conta criada deve ser possível:
        - Obter o nome do correntista;
        - Obter seu código;
        - Obter o banco ao qual a conta pertence;
        - Obter seu saldo;
        - Fazer uma aplicação;
        - Efetuar um débito.

    Faça com que cada banco tenha um código próprio, o mesmo para as contas. Permita que contas de bancos diferentes tenham o mesmo número. Escreva um programa de teste que crie dois bancos e duas contas e faça depósitos, retiradas e consultas. Imprima os resultados para conferir se eles estão corretos.

12) Crie a seguinte hierarquia de classes:
    - Uma interface para representar qualquer forma geométrica, definindo métodos para cálculo do perímetro e cálculo da área da forma;
    - Uma classe abstrata para representar quadriláteros. Seu construtor deve receber os tamanhos dos 4 lados e o método de cálculo do perímetro já pode ser implementado;
    - Classes para representar retângulos e quadrados. A primeira deve receber o tamanho da base e da altura no construtor, enquanto a segunda deve receber apenas o tamanho do lado;
    - Uma classe para representar um círculo. Seu construtor deve receber o tamanho do raio.

    No programa principal, crie quadrados, retângulos e círculos com tamanhos diferentes e armazene num vetor. Em seguida, imprima os dados (lados ou raio), os perímetros e as áreas de todas as formas.

13) Defina uma classe **IntervaloDeTempo** cujos objetos representam um intervalo de tempo em número de horas, minutos e segundos. O construtor de objetos dessa classe deve receber como argumento um número inteiro positivo, representando o número de segundos decorridos desde o instante inicial 00:00:00 horas e retornar um objeto da classe **IntervaloDeTempo** correspondente.

Por exemplo, a expressão **new IntervaloDeTempo(3500)** deve retornar um objeto que represente 0 horas, 58 minutos e 20 segundos. Crie também uma classe para testar a classe definida.

14) Defina uma classe **Pessoa** cujos objetos representam uma pessoa contendo as seguintes informações: nome, sexo, número do documento de identidade e data de nascimento da pessoa (dia, mês e ano). Defina um construtor de objetos dessa classe e também um método idade() para retornar a idade da pessoa (um número inteiro de anos) dado como argumento a data atual. Caso a data passada como argumento seja anterior _a data de nascimento da pessoa o método deve retornar -1.

15) Defina uma classe **Aluno**, subclasse da classe **Pessoa**, que contenha as seguintes informações adicionais: número de matrícula, curso no qual está matriculado, data de ingresso na universidade. Redefina o construtor de objetos da classe de maneira apropriada. Crie também uma classe para testar as classes definidas.

16) Defina uma classe **Conta**, cujos objetos representam contas bancárias, contendo cada qual as informações: número da conta, nome e CPF do correntista, data de abertura da conta, senha e saldo corrente. Defina um construtor de objetos dessa classe que tenha como parâmetros essas informações (exceto o saldo da conta) e retorne um objeto da classe Conta com o saldo corrente igual a zero. Defina também dois métodos para implementação das operações de saque e de depósito na conta. Esses métodos devem retornar o saldo corrente da conta depois de efetuada a operação correspondente (suponha que o valor passado como argumento para o método é sempre um valor positivo). O método de saque não deverá alterar o saldo corrente da conta corrente da conta caso o montante do saque seja superior a esse saldo. Crie também uma classe para testar a classe definida.

17) Defina uma classe **ChequeEspecial**, subclasse da classe **Conta**, que contenha como informação adicional o valor limite (negativo) para o saldo da conta. Redefina o construtor de objetos da classe de maneira apropriada e os demais métodos levando em consideração o limite do cheque especial. Crie também uma classe para testar a classe definida.

18) Desenvolva uma classe de nome **CodigoPostal**, cujas instâncias sejam capazes de guardar o Código Postal de uma dada rua. Note que cada Código Postal é constituído por dois números inteiros, que designaremos respectivamente por "indicativo" e "extensão", e o nome da rua (Ex: 38408-046 Armando Lombardi). Deverão poder ser criados códigos postais dados:
    - "indicativo", "extensão" e rua
    - "indicativo" e rua (ficando nesse caso a extensão igual a zero)
    - nenhum parâmetro (ficando nesse caso os atributos numéricos a zero e a rua com a mensagem "Indisponível".

    Para além dos usuais métodos seletores (*getters*) e modificadores (*setters*) implemente também o método mostra, cuja evocação, permita visualizar a informação relativa a um determinado código postal no formato:
    CEP: 38408 - 046 Armando Lombardi (ou CEP: Indisponível se for o caso).
    Escreva um programa de teste para a classe CodigoPostal. Crie várias instâncias da classe e teste os vários métodos implementados.

19) A empresa **BadSoft** desenvolveu uma classe **Xpto** capaz de armazenar 3 valores do tipo *int* como se indica a seguir:

```
// Arquivo: Xpto.java
public class Xpto {
        //VARIAVEIS DE INSTANCIA
        public int a;
        public int b;
```

```
        public int c;

        //CONSTRUTORES
        public Xpto() {
                a=0;
                b=0;
                c=0;
        };

        public Xpto (int a1, int b1, int c1) {
                a=a1;
                b=b1;
                c=c1;
        };

        //METODOS
        public int produto() {
                return (a*b*c);
        };

        public int soma() {
                return (a+b+c);
        };
}; // class Xpto
```

Note que os autores da classe **Xpto** não incluíram, na mesma, métodos seletores (*getters*) e métodos modificadores (*setters*) capazes de acessar ou alterar os valores das respectivas variáveis de instância. Estas foram declaradas como *public* pelo que o respectivo acesso pode fazer-se, do exterior da classe, de forma direta sem recurso a tais métodos. Vários programadores adquiriram a classe **Xpto** e utilizaram-na nos seus programas.

Um desses programas, designado por Exemplo1, é indicado a seguir:
Passado algum tempo os engenheiros da **BadSoft** acharam que seria muito melhor guardar os 3 valores do tipo *int* num vetor. Assim lançaram um *update* da classe **Xpto** com a seção correspondente às variáveis de instância substituída por:

```
        //VARIAVEIS DE INSTANCIA
        public int v[];
```

a) Tendo em conta esta alteração indique que outras modificações tiverem os engenheiros da **BadSoft** de efetuar a nível de construtores e métodos de forma a manter a classe **Xpto** funcional.
b) Indique também quais as alterações que o programador, cliente da BadSoft, teria de efetuar no seu programa Exemplo1.

20) A **GoodSoft**, concorrente da **BadSoft**, também lançou no mercado uma classe semelhante com o nome **Ypto**. Os engenheiros da **GoodSoft** optaram por declarar as variáveis de instância com a visibilidade *private* (e não *public* como os seus colegas da **BadSoft**) sendo o respectivo acesso feito à custa de métodos seletores (*getters*) e métodos modificadores (*setters*)como se indica a seguir:

```
// Arquivo: Exemplo1.java
public class Exemplo1 {
        public static void main(String[] args) {
                Xpto x1=new Xpto();
                Xpto x2= new Xpto(5,6,7);
                x1.a=1;
                x1.b=2;
                System.out.println(x1.soma());
```

```
                System.out.println(x2.soma());
                x1.c=x2.a;
                x2.a=x2.a+1;
                System.out.println(x1.c);
                System.out.println(x2.a);
                //.....
        };
}; // class Exemplo1

// Arquivo: Ypto.java
public class Ypto {
        //VARIÁVEIS DE INSTÂNCIA
        private int a;
        private int b;
        private int c;

        //CONSTRUTORES
        public Ypto() {
                a=0;
                b=0;
                c=0;
        };
        public Ypto(int a1, int b1, int c1) {
                a=a1;
                b=b1;
                c=c1;
        };

        //MÉTODOS SELETORES
        public int getA() {
                return a;
        };
        public int getB(){
                return b;
        };
        public int getC(){
                return c;
        };

        //MÉTODOS MODIFICADORES
        public void setA(int a1){
                a=a1;
        };
        public void setB(int a1){
                b=a1;
        };
        public void setC(int a1){
                c=a1;
        };
        //OUTROS MÉTODOS
        public int produto(){
                return (a*b*c);
        };
        public int soma(){
                return (a+b+c);
        };
```

```
}; // class Ypto
```

Vários programadores adquiriram a classe **Ypto** e utilizaram-na nos seus programas. Um desses programas, designado por Exemplo2, é indicado a seguir:
Os engenheiros da **GoodSoft** também concluíram que seria muito melhor se os 3 valores fossem guardados num vetor e por isso também lançaram um *update* da classe **Ypto**.

a) Tendo em conta esta alteração indique que outras modificações tiverem os engenheiros da **GoodSoft** de efetuar a nível de construtores e métodos por forma a manter a classe **Ypto** funcional.

b) Indique também quais as alterações que o programador, cliente da **GoodSoft**, teria de efectuar no seu programa Exemplo2.

a) Que conclusão podemos tirar destes dois exercícios ?

21) Desenvolva uma classe Java que modela um objeto **Livro**, que contém ainda um método construtor e um método (public static void) *main*. Um livro possui um título e quantidade de páginas. Use **String** para representar o título. Use inteiros para representar a quantidade de páginas. O construtor deve receber através de argumentos, os dois dados suficientes para criar um livro. O método main deve criar um array com quatro posições, e quatro livros com dados quaisquer, cada um deles associado a uma posição do array.

22) Desenvolva uma classe pública denominada **Grupo**, que permita representar um grupo de trabalhos constituído por dois alunos, onde se incluem os seguintes membros públicos:
a) um construtor, onde são passados os dois alunos do grupo e uma nota do grupo (que é atribuída pela avaliação de um trabalho de grupo).
b) um método que devolve a média das notas individuais dos alunos do grupo (atenção que esta não é a nota do grupo);
c) um método que devolve o índice de desvio das notas individuais dos alunos em relação à nota do grupo; este índice é dado pela seguinte fórmula:

$$\sqrt{(Ng-a)^2+(Ng-b)^2}$$

onde Ng é a nota do grupo e a,b são as notas individuais de cada um dos membros do grupo.

```
// Arquivo: Exemplo2.java
public class Exemplo2 {
        public static void main(String[] args) {
                Ypto x1=new Ypto();
                Ypto x2= new Ypto(5,6,7);
                x1.setA(1);
                x1.setB(2);
                System.out.println(x1.soma());
                System.out.println(x2.soma());
                x1.setC(x2.getA());
                x2.setA(x2.getA()+1);
                System.out.println(x1.getC());
                System.out.println(x2.getA());
                //.....
        }; // main
}; // class Exemplo2
```

UEMG - Universidade Estadual de Minas Gerais
PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA
LISTA DE EXERCÍCIOS (Compilação) – Versão 2.0

23) Crie uma classe **Televisao**:

- Atributos :
  - status (on/off);
  - canal.
- Defina um construtor que inicialize o campo status como off e o campo canal como 11.
- Defina um método **ligaDesliga**
- Se estiver ligado ele desliga e se estiver desligado ele liga.
- Defina um método **trocaCanal**
- Recebe troca o canal da tv, pedindo o canal a ser trocado.
- Essa televisão só vai até o canal 15, se for um canal inválido permaneça no mesmo canal.
- Defina um método **verStatus** que irá exibir uma mensagem dizendo se a TV está ligada ou desligada e se estiver ligada mostrar também o canal em que ela está.
- Defina um método *main* para testar a classe.

24) Crie uma classe **Termometro**:

– Atributos: temperatura

– Defina um construtor que não receba nenhum parâmetro e inicialize o campo temperatura com o valor de 15.

– Defina um método aquecer.

- A temperatura é alterada de 5 em 5.
  - Defina um método esfriar.
- A temperatura e alterada de 5 em 5.
  - Defina um método para retornar o valor de temperatura.
  - Defina o método *main*

a. Crie um objeto da classe **Termometro**.

b. Teste o aumento e a baixa da temperatura.

25) Crie uma classe **Carro**:

- Atributos:
  - cor;
  - marca;
  - modelo;
  - combustível.
- Defina um construtor que receba: cor, marca, modelo, combustível.
- Defina um método **mostrarCarro**, para mostrar os atributos do carro.
- Defina o método *main* para testar a classe.

a) Criar um objeto a classe **Carro**.

b) Chamar o método **mostrarCarro**.

26) Crie uma classe **Livro**:

- Atributos :
  - titulo;
  - autor
  - editora
  - quantidade de páginas.
- Defina métodos que retornem os valores de cada atributo da classe
- Defina métodos que alterem os valores q de cada atributo da classe
- Defina um método main para testar a classe.

a) Criar um objeto da classe livro.

b) Popular o objeto pelos métodos get e set.

c) Mostrar o título do livro e o nome do autor.

d) Mostrar o título do livro a editora e a quantidade de páginas.

e) Alterar a quantidade de páginas.
f) Mostrar título do livro,nome do autor e quantidade de páginas.

27) Crie uma classe **Retangulo**:
- Atributos:
  - altura;
  - largura.
- Defina um construtor que chame o método "ObterAlturaLargura".
- Defina um método calcular área (área=altura*largura)
- Defina um método para ler altura e largura.
- Defina um método main para testar a classe.

a) Criar um objeto da classe Retângulo.
b) Chamar o método que calcula a área.

28) Crie uma classe Tempo:
- Atributos:
  - hora;
  - minutos;
  - segundos.
- Defina um construtor que receba hora, minuto e segundo.
- Defina um método para mostrar a hora no formato hora, minuto e segundo.
- Defina um método para retornar a hora em segundos.
- Defina um método main para testar a classe.

a) Criar um objeto da classe Tempo.
b) Chamar o método para mostar a hora
c) Chamar o método parar mostrar a hora em segundos.

29) Crie uma classe **Funcionario**:
- Atributos:
  - nome;
  - RG;
  - salário
  - quantidade de anos que o funcionário está na firma.
- Defina um construtor que receba nome, RG, salário e quantidade de anos.
- Defina um método que calcule e altere o salário.
  - A cada ano que o funcionário está na firma tem um aumento de 10%
- Defina um método que mostre todos os dados do funcionário.
- Defina o método main para testar a classe.

a) Crie três objetos da classe Funcionário
b) Mostre os dados dos funcionários.
c) Calcule seus salários.
d) Mostre os dados dos funcionários.

30) Crie um projeto chamado Trabalho2, que possui 2 pacotes: **br.uemg.principal** e **br.uemg.empresa**.

a) Crie uma classe em Java com as seguintes características: Classe **Funcionário** com **mat**, **nome**, **função** e **salário**, todos os atributos com escopo público. Todas as propriedades, com exceção de mat, devem ser de instância. Esta classe deve estar no pacote **br.uemg.empresa**.

b) Escreva uma classe aplicação **UsaTrabalho2** para criar 3 objetos do tipo **Funcionário** e solicite ao usuário que entre com dados para os três funcionários criados. Ao final imprima os valores fornecidos para estes funcionários. A classe **UsaTrabalho2** deve estar no pacote **br.uemg.principal**.

c) Altere a classe **Funcionário** para que esteja de acordo com as definições da Orientação a Objetos com relação ao encapsulamento, ou seja, torne as propriedades private e recompile a classe. Tente executar o programa do exercício b.

d) Altere a classe **Funcionário** adicionando métodos de acesso à classe de modo que somente será possível ter acesso a estas variáveis mediante os métodos criados.

e) Altere o exercício b para que utilize os métodos ao invés das propriedades diretamente.

f) Altere as classes **Funcionário** e a criada no exercício b para que a propriedade **mat** seja auto-incrementada, ou seja, a matricula do funcionário seja automaticamente fornecida a partir do valor 1 a cada funcionário criado. Sendo assim, a matricula não será mais solicitada na criação de um funcionário e sim gerada na construção do objeto da classe.

g) Crie um método construtor para a classe **Funcionário** de modo que os valores das propriedades sejam fornecidos na criação de um funcionário.

h) Altere o exercício b para que utilize este novo método construtor.

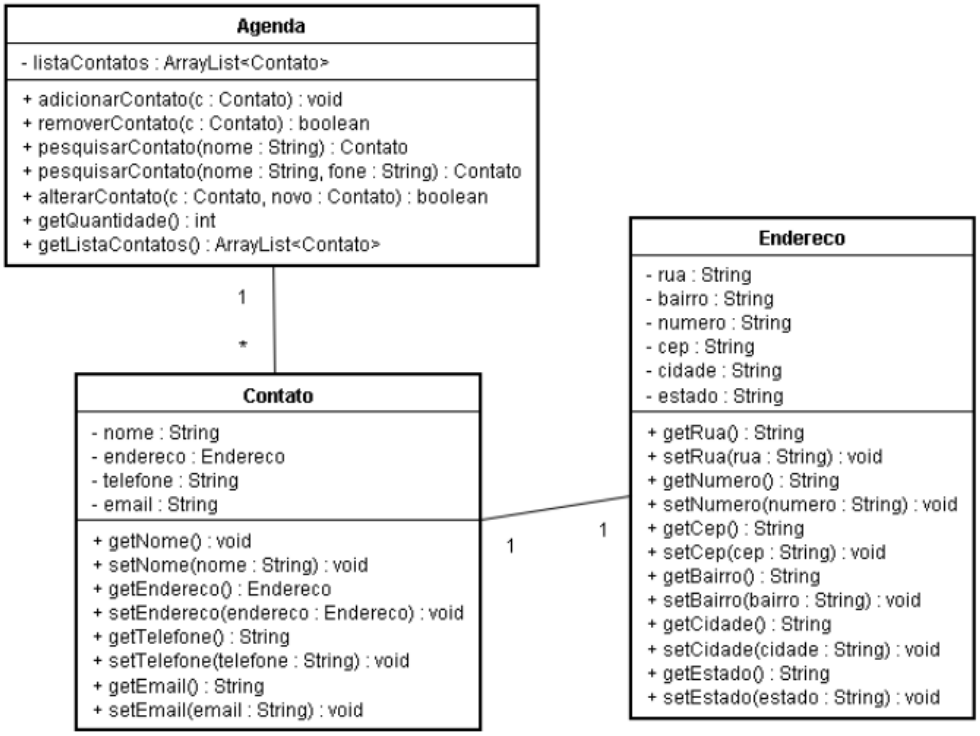
i) Crie um novo método construtor para a classe **Funcionário** que permita a construção de um **Funcionário** apenas com o nome e função. Este construtor deverá chamar o outro construtor já existente passando o salário como zero.

j) Crie uma classe chamada **CargosSalarios** que possua um método de classe chamado **buscaSalario** que receba como parâmetro a função do funcionário e retorne o salário correspondente. Este método deverá ser chamado pelo construtor do exercício h ao invés de se passar o salário igual a zero. A classe **CargosSalarios** deve estar no pacote **br.uemg.empresa**.

Utilize a tabela de funções a seguir como exemplo.
Função: diretor-salário: 5000,00
Função: vendedor-salário: 3000,00
Função: secretaria-salário: 1000,00

UEMG - Universidade Estadual de Minas Gerais
PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA
LISTA DE EXERCÍCIOS (Compilação) – Versão 2.0

31) Seja o diagrama de classes abaixo:



a) Implementar todas as classes definidas no diagrama de classes;
b) Criar uma classe chamada **GerenciadorAgenda** que tenha o seguinte menu:

1 – Incluir Contato
2 – Alterar Contato
3 – Excluir Contato
4 – Pesquisar Contato por Nome
5 – Pesquisar Contato por Telefone
6 – Listar contatos
7 – Ver quantidade de contatos

32) Qual a diferença entre classe e objeto?

33) Qual é a finalidade do método construtor?

34) Identifique na instrução abaixo: a classe, o objeto, o construtor e a operação de instanciação.

```
Computador computador = new Computador();
```

35) O que diferencia um construtor de um método qualquer?

36) Observe que a classe abaixo não possui um construtor, porém para que seja criado um objeto sempre é necessário utilizar o operador new seguido do nome do construtor. Desta forma, não será possível criar um objeto desta classe? Explique.

```
public class Garrafa{
	private String tipo;

	public void setTipo (String tipo){
		this.tipo = tipo;
	}

	public String getTipo(){
		return tipo;
	}
}
```

37) A classe abaixo é parecida com a classe do exercício anterior, porém agora ela possui um construtor alternativo. O que irá acontecer se em outra classe você desejar criar um objeto da mesma com a seguinte instrução:

```
Garrafa gar = new Garrafa();
```

```
public class Garrafa{
	private String tipo;

	public Garrafa (String tipo){
		this.tipo = tipo;
	}

	public void setTipo (String tipo){
		this.tipo = tipo;
	}

	public String getTipo(){
		return tipo;
	}
}
```

38) As duas classes abaixo não estão no mesmo pacote. Faça uma análise das mesmas e corrija possíveis erros de compilação.

```
public class Carro{
	protected int litrosNoTanque;
	protected boolean carroLigado;

	private void encherTanque(int litros){
		litrosNoTanque = litros;
	}
}
```

```
public class TesteCarro{
	public static void main (String args[]){
		Carro carro = new Carro();
		carro.encherTanque(10);
		carro.carroLigado = true;
	}
}
```

39) Observe as classes **Circulo** e **TesteCirculo** abaixo e realize as seguintes tarefas:

- Declare o atributo raio da classe **Circulo** como privado;
- Crie os métodos necessários na classe **Circulo** de modo que seja possível obter ou alterar o valor do atributo raio pela classe **TesteCirculo**;
- Caso o objetivo seja alterar o valor do atributo raio, faça a seguinte verificação: Se o valor do novo raio for positivo, faça a atribuição, caso contrário não faça;
- Siga as orientações presentes na classe **TesteCirculo** e execute cada tarefa;

```
public class Circulo{
      double raio;
}

public class TesteCirculo{
      public static void main (String args[]){
            / Crie um objeto da classe Circulo
            / Coloque aqui o codigo para alterar o raio para 10
            / Coloque aqui o codigo para obter o valor do raio
      }
}
```

40) Observe as classes **AcessaBanco** e **TesteBanco** abaixo e realize as seguintes tarefas:

- Declare os atributos *login* e conectado da classe **AcessaBanco** como privado;
- Crie os métodos necessários na classe **AcessaBanco** de modo que seja possível obter ou alterar o valor dos atributos *login* e conectado pela classe **TesteBanco**;
- Siga as orientações presentes na classe **TesteBanco** e execute cada tarefa;

```
public class AcessaBanco{
      String login;
      boolean conectado;
}

public class TesteBanco{
      public static void main (String args[]){
      /* Crie um objeto da classe AcessaBanco */
      /* Coloque aqui o codigo para alterar o login para o seu nome */
      /* Coloque aqui o codigo para ajustar conectado para true */
      }
}
```

41) Analise o código abaixo e descreva em poucas linhas onde está o erro de compilação e qual seria a solução para corrigir o mesmo.

```
public class TesteStatic{
      private int a1;
      private int a2;

      public static void main (String args[]){
            iniciar(10,40);
      }

      public void iniciar(int v1, int v2){
```

```
            int b1 = v1;
            int b2 = v2;
      }
}
```

42) Analise as classes abaixo e descreva em poucas linhas se existe erro de compilação ou não. E caso exista algum, descreva qual deveria ser a solução.

```
public final class Veiculo{
      private String chassi;

      public String getChassi(){
            return chassi;
      }
}

public class Carro extends Veiculo{
      private String cad;

      public String getCad(){
            return cad;
      }
}
```

43) A classe **Jogo** abaixo foi descrita sem atributos e métodos.

```
public class Jogo{ }
```

Logo após o processo de compilação foi utilizado o utilitário **javap** (que transforma código binário em código Java) e o mesmo retornou o código abaixo.

```
public class Jogo extends java.lang.Object{
      public Jogo();
}
```

Note que existem mais informações no código gerado pelo utilitário **javap** do que o código original. O que aconteceu?

44) Analisando a classe Java abaixo podemos observar que a mesma possui apenas um atributo, um construtor e dois métodos. Perceba que dentro do método *main* estão sendo invocados métodos e atributos que não pertencem à classe. Isto é um erro de compilação? Justifique sua resposta.

```
public class PessoaFisica extends Pessoa{
      private String RG;
      public PessoaFisica(){
            super();
      }
      public String getRG(){
            return RG;
      }

      public static void main (String args[]){
            PessoaFisica pf = new PessoaFisica();
            pf.setEndereco("Rua XV n. 10");
            pf.setFone("2546-3274");
            System.out.println(pf.endereco);
            System.out.println(pf.fone);
```

```
        }
}
```

45) Faça um programa em Java para implementar uma calculadora simples com as quatro operações básicas da matemática. Crie três métodos para cada uma das operações e cada método deverá ser sobrecarregado, pois um deles deve receber apenas dois parâmetros do tipo *int*, o outro apenas dois parâmetros do tipo *float* e o último apenas dois parâmetros do tipo *String*. Quando os parâmetros forem do tipo *String*, os mesmos deverão ser convertidos para o tipo *int*.

46) Crie uma classe Java chamada **ClienteBanco** com os seguintes atributos (privados):
- nome;
- Data de Nascimento;
- CPF;
- Endereço.

Além do construtor padrão, crie um construtor alternativo para iniciar cada um dos atributos. Crie os métodos necessários para acessar estes atributos.Faça a sobrescrita do método ***toString()*** da classe *Object* para o mesmo retornar a seguinte mensagem:

"O Sr." <nome> "portador do CPF n." <CPF> "nascido em" <data de nascimento> "residente e domiciliado a " <endereço> "vem por meio desta solicitar o encerramento de sua conta corrente".

Crie um método *main* e dentro do mesmo faça com que a mensagem gerada pelo método ***toString()*** seja impressa na tela.

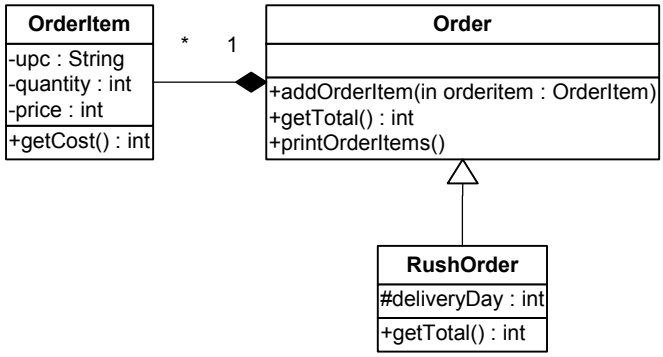
47) Crie uma classe Java chamada **Pessoa** com os seguintes atributos privados:
- nome;
- Data de Nascimento;

Crie os métodos necessários para acessar estes atributos. Crie também um método chamado "informarIdade", este método deve imprimir na tela a idade desta pessoa em anos, meses e dias. Crie um método *main* para criar um objeto da classe **Pessoa**, inicie os atributos "nome" e "Data de nascimento" e em seguida chame o método "informarIdade" para que o mesmo informe a idade desta Pessoa.

48) Crie uma classe Java para um **Celular**, com os seguintes atributos: modelo (*String*) e número (*int*).
- Faça o encapsulamento destes atributos;
- Faça a sobrecarga do método que irá alterar o valor do atributo número, permitindo que o programador informe uma string ou um inteiro;
- Faça a sobrescrita do método correto da classe **Object** para retornar uma string que representa a concatenação dos atributos da classe.

UEMG - Universidade Estadual de Minas Gerais
PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA
LISTA DE EXERCÍCIOS (Compilação) – Versão 2.0

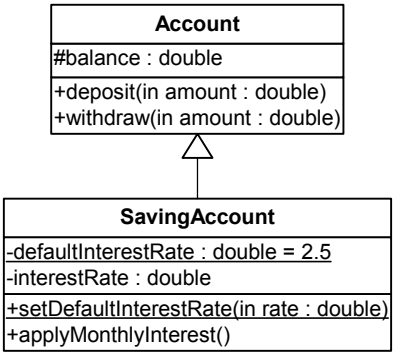
32) Considerar o modelo conceitual em UML mostrado na figura abaixo. Escrever cada uma as classes do diagramas de classes, e também uma classe aplicação de teste, conforme as instruções a seguir:



a) Escrever as classes **Item**, **Pedido**, **Entrega** em classes separadas, todas em um mesmo pacote. Observar que todas as classes são públicas.
b) A classe **Item** possui os atributos: uma String upc, um inteiro de quantidade e um inteiro preço, todos privados. O método **getCusto()** retorna a multiplicação de sua quantidade pelo preço.
c) A classe **Pedido** possui uma lista de objetos do tipo Item. O método **adicionaItem()** recebe uma referência de um objeto **Item** como parâmetro e armazena ma lista. O método **getTotal()** retorna o custo total de todos os itens do pedido. O método **listaItens()** lista informações sobre cada item encomendado via o método da classe **Item**.
d) A classe **PedidoExpresso** estende a classe **Pedido**. Ela possui um atributo de instância **DataEntrega**, com o especificador de escopo *protected*, que representa em quantos dias o pedido deveria ser entregue. O método **getTotal()** sobrescreve o método definido na superclasse da seguinte forma: primeo invoca o método **getTotal()** da superclasse para determinar o total para todos os itens do pedido. Em seguida, soma o frete. O frete para um prazo de um dia de entrega é de R$ 50,00, para dois dias é de R$35,00 e para três dias é de R$ 20,00. E para quatro ou mais dias é grátis. Note que o custo do frete deveria ser adicionado somente se há itens no pedido. Isto significa que se o método **getTotal()** da superclasse retorna 0, não custo de frete, mas apenas retorna 0.
e) Implemente uma classe aplicação como a seguir:
    i. Criar quatro pedidos, e adicioná-los em um *ArrayList*, identificado como ***pedidos***. O primeiro objeto é uma instância da classe **Pedido**, o Segundo é uma instância de **PedidoExpresso** com um dia para entrega, e o terceiro objeto é uma instanciada de **PedidoExpresso** com três dias de entrega;
        - Creiar um objeto **Item** com UPC, quantidade e preço;
        - Adicione o item do pedido aos pedidos, correspondendo the order item to the element of the orders array corresponding to the delivery day via the addOrderItem(). If delivery day is four or bigger, add the order item into the first element, which is the instance of Order class. If the delivery day is one, add it to the second element of the orders array, which is the instance of the RushOrder class with one day delivery, and so on.
        - Continue the loop until either user enters “done”.
    ii. For each order in the orders array:
        - Print out the type of order via the toString() methods.
        - Print out all the order items.
        - Print out the subtotal for this order.
    iii. Print out the total cost in all of the orders.

UEMG - Universidade Estadual de Minas Gerais
PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA
LISTA DE EXERCÍCIOS (Compilação) – Versão 2.0

1. Implement the inheritance hierarchy shown in **Erro! Fonte de referência não encontrada.** in Java. Have Account and SavingAccount classes in a single file named Homework3_2. The defaultInterestRate field in the SavingAccount class is static with an initial value of 2.5. The setDefaultInterestRate() is a static function. Have the Homework3_2 class to create objects of Account and SavingAcount classes, invoke deposit and withdraw methods on them, and invoke the setDefaultInterestRate() and the applyMonthlyInterest() on the SavingAccount object.



1. Consider the inheritance hierarchy in Figure 1.

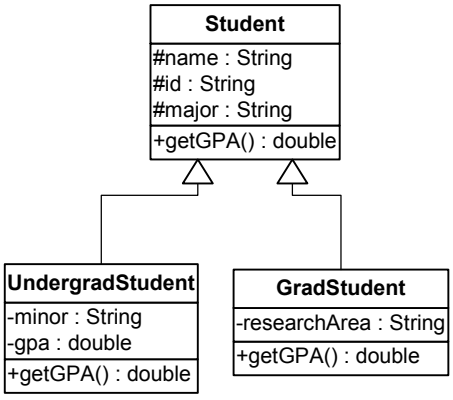


Figure 1 Inheritance Hierarchy for Exercise 3

Introduce Student, UndergradStudent and GradStudent classes in a package called edu.heinz.cmu.oop95712c.Student. Also define a StudentFactory class in this package as explained below.

Student is an abstract class with abstract getGPA() method. UndergradStudent and GradStudent are final classes. Information about students is kept in a data file as in the following format:

G|Adam Smith|123-4567|CISC|Network Management

U|Chris Smith|987-6543|MATH|ENGL|3.98

The fields are separated by the | character. The first field is either a G representing a graduate student or a U representing an undergraduate student. Graduate student entries have name, id, major and research area

fields, whereas the undergraduate entries have name, id, major, minor and gpa fields. The getGPA() method for the graduate students should always return 4.0.

Introduce the StudentFactory class with just a static function of the form:

public static ArrayList createStudents(String filename) throws IOException {…}

This function should take the name of the data file and read it line by line until the end of file. For each line read, it should examine the first field, create either a graduate or an undergraduate student object, store the student object reference in an array list. Once, all the entries are read, it should return the array list object.

Define your main function in a class named Homework4_3 in the default package. Your main function should invoke the createStudents() function of the StudentFactory class and get the list of students into a local array list variable. It should then loop over the array list and print out the information about each student on a separate line onto the console.

There is only one exercise in this homework, which is 100 points.

Follow the commenting and coding convention, and the minimal class description guidelines as discussed in the lectures for all the classes you introduce, except for the test driver classes that contain the main function.

Name your files as specified below. Generate the Java documentation by running the javadoc utility over all the classes and packages. Put all your java files, compiled class files and the documentation files into a zip file named Homework5.zip and submit it via the Drop Box on the blackboard before the beginning of the class on October 27, 2004. Also, print all your .java files with your name and bring them to me at the beginning of the class on October 27, 2004. Don’t print nor bring the documentation files.

2. Consider the inheritance hierarchy in Figure 1.

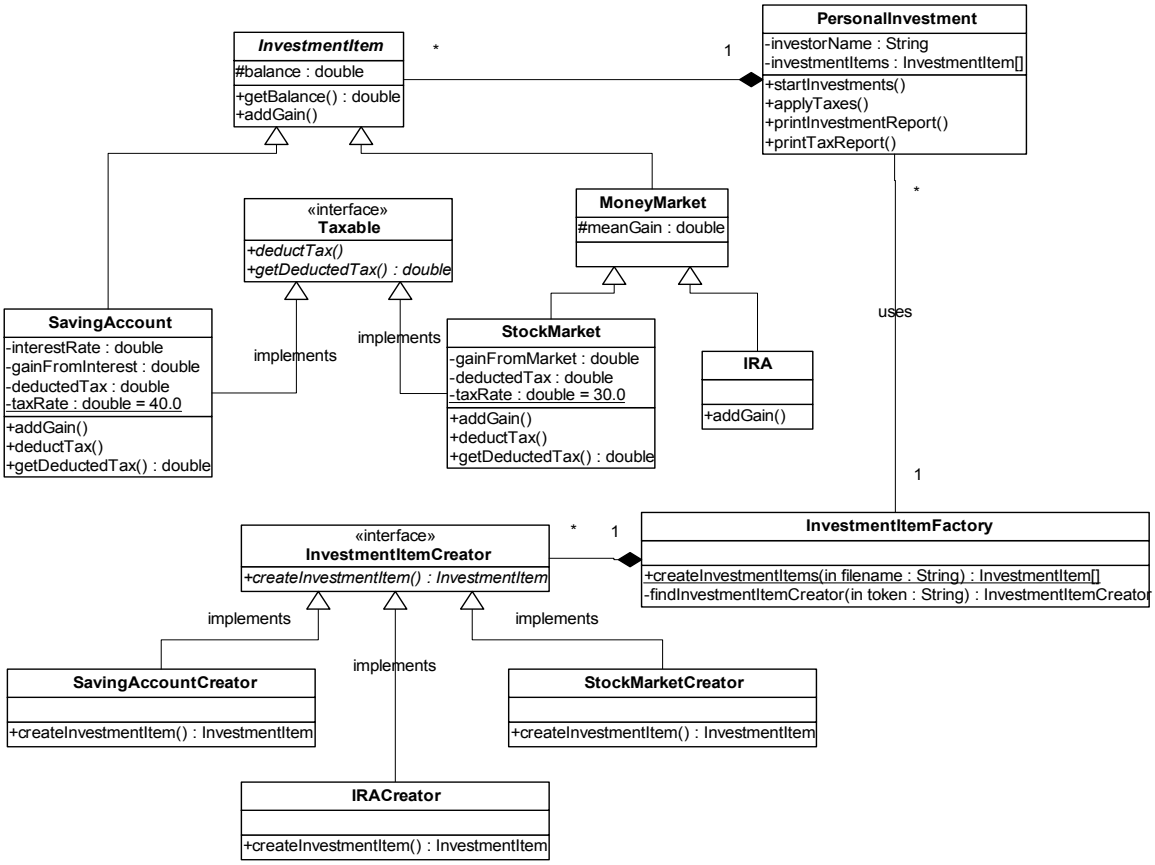


Figure 2 Inheritance Hierarchy for Exercise 3

Create a package named edu.heinz.cmu.oop95712c.Investment and put all the classes shown in Figure 1 into this package as explained below:

- InvestmentItem is an abstract class with an abstract addGain() method. getBalance() is the getter method for protected member balance.
- Taxable is an interface with deductTax() and getDeductedTax() methods.
- SavingAccount is a subclass of InvestmentItem and it implements the Taxable interface. taxRate is static and final whereas all others are instance members. addGain() method calculates the (yearly) interest and adds it into the balance and to the gainFromInterest. deductTax() method calculates the tax based on the gainFromInterest, reduces the balance by the tax, adds the tax to the deductedTax member and sets the gainFromInterest to 0. getDeductedTax() returns the deductedTax member variable.
- MoneyMarket extends InvestmentItem and has an instance variable named meanGain.
- The IRA class is a subclass of MoneyMarket. Its addGain() will generate an exponentially distributed random number with a mean of meanGain and add it to the balance. The method to generate exponentially distributed random number will be provided to you.
- StockMarket is a subclass of InvestmentItem and implements Taxable interface. The taxRate is static and final whereas all others are instance variables. addGain() generates exponentially distributed random number with a mean of meanGain and add it to the balance. It also increments the gainFromMarket by that generated number. deductTax() and getDeductedTax() work similarly to those of the SavingAccount class.
- PersonalInvestment has the investorName and investmentItems, both instance members. The startInvestments() calls the following static function of the InvestmentItemFactory class in a try –catch block

```
investmentItems =

    InvestmentItemFactory.createInvestmentItems(investorName + "_investment.data");
```

This function reads a file and creates all the InvestmentItem objects into an array, as explained later on. It then creates a local ActionListener object named adder and passes that object to a Timer instance as the ActionListener as follows:

```
Timer t = new Timer(1000, adder);

t.start();
```

The actionPerformed() method of the adder object should iterate over all the InvestmentItem objects inside the investmentItems and invoke their addGain() method.

The applyTaxes() method should iterate over all the InvestmentItem objects inside the investmentItems and invoke the deductTax() method only on the investment objects that implement the Taxable interface.

The printInvestmentReport() method should iterate over all the InvestmentItem objects inside the investmentItems, print them out and invoke the getBalance() method in order to add the balance of this item into a running total balance. It should print out the total balance at the end.

The printTaxReport() method should find the total taxes paid by this personal investment account by iterating over all the InvestmentItem objects inside the investmentItems, invoking the getDeductedTax() method only on the investment objects that implement the Taxable interface and adding the return values of getDeductedTax() into a running total tax variable. It should print out the total tax at the end.

- InvestmentItemFactory implements the factory design pattern discussed in the class. The database for investment items are stored in a file named name_investment.data where name is the investorName instance variable of the PersonalInvestment object. The format of the file is shown below:

```
SA|1000|2.5

IRA|10000|200

SM|5000|100
```

The first field in each of the lines represents the type of the investment: SA for SavingAccount, IRA for IRA and SM for StockMarket. For SA, the remaining fields are the initial balance and the interest rate; for IRA and SM, the first field is the initial balance and the second is the mean gain.

Define InvestmentItemCreator interface and SavingAccountCreator, IRACreator, StockMarketCreator classes as private inner classes inside the InvestmentItemFactory class.

The createInvestmentItems() static function opens the buffered input stream and read it line by line. For each line read from the input stream, it will get the first token using a StringTokenizer. This token identifies the type of the investment item. This function will pass the identifier token to the private findInvestmentItemCreator() which finds the creator object corresponding to the identifier of the investment item. The createInvestmentItems() will invoke the createInvestmentItem() method of the creator object.

The createInvestmentItem() methods of all the creator classes take a StringTokenizer parameter and uses it to read the values that are necessary to create an investment item object. The createInvestmentItem() method of the SavingAccountCreator class calls the nextToken() on the StringTokenizer parameter to get the values for the initial balance and the interest rate. It uses these two values to create and return a SavingAccount instance. Other creator objects work similarly.

Once the createInvestmentItem() method of the creator object returns an investment item, the createInvestmentItems() will put that object into an array list. Once all the lines are complete reading, createInvestmentItems() will call the toArray() method on that array list and return the array of investment items.

All the creator objects are stored in a static array inside the InvestmentItemFactory class. A skeleton will be provided for you to use these creator objects.

The main function inside the Homework5_1.java file in default package will be provided for you to experiment with your program.

The skeleton code is available in Homework5 directory of http://www.andrew.cmu.edu/user/syucel/Java/homework_solutions/