Sistema Empresa XYZ

Considere-se as seguintes classes implementadas em Java, a partir de uma análise bastante simplificada do corpo funcional de uma grande empresa.

```
public class boy{
        private int ID;
        private String nome;
        private int departamento;
        private double salario, previdencia, desconto;

        public boy (int ID, String nome, int departamento, double salario){
                this.ID = ID;
                this.nome=nome;
                this.departamento= departamento;
                this.salario= salario;
                this.previdencia=0.05; // Desconto fixo de um sistema de previdência da empresa
                this.desconto =0.02;
        }
        void setDesconto(double desconto){
                this.desconto =desconto;
        }
        double pagamento(){
                // Calcula o salário líquido
                return(this.salario*(1- this.previdencia-this.desconto));
        }
}

public class funcionario{
        private int ID;
        private String nome;
        private int departamento;
        private double salario, previdencia, desconto, adicional;

        public funcionario(int ID, String nome, int departamento, double salario, double adicional){
                this.ID = ID;
                this.nome=nome;
                this.departamento= departamento;
                this.salario= salario;
                this.previdencia=0.05; // Desconto fixo de um sistema de previdência da empresa
                this.desconto =0.02;
                this. adicional = adicional;  // adicional caso o funcionário seja por exemplo chefe.
        }
        void setDesconto(double desconto){
                this.desconto =desconto;
        }
        void setAdicional(double novo_percentual){
                this.adicional = adicional*(1+novo_percentual);
        }
        double pagamento(){
                // Calcula o salário líquido
                return(salario*(1- this.previdencia- this.desconto+ this.adicional));
        }
}
```

```
public class funcionarioSindicalizado{
	private int ID;
	private String nome;
	private int departamento;
	private double salario, previdencia, desconto, adicional, desconto_sindicato;

	public funcionarioSindicalizado (int ID, String nome, int departamento, double salario,
					double adicional)
		this.ID = ID;
		this.nome=nome;
		this.departamento= departamento;
		this.salario= salario;
		this.previdencia=0.05;  // Desconto fixo de um sistema de previdência da empresa
		this.desconto =0.02;
		this. adicional = adicional;  // adicional caso o funcionário seja por exemplo chefe.
		this.desconto_sindicato = 0.1;
	}
	void setDesconto(double desconto){
		this.desconto =desconto;
	}
	void setAdicional(double novo_percentual){
		this.adicional = adicional*(1+novo_percentual);
	}

	double pagamento(){
		// Calcula o salário líquido
		return(salario*(1-this.previdencia-this.desconto+
			this.adicional-this.desconto_sindicato));
	}
}

public class presidente{
	private int ID;
	private String nome;
	private int departamento;
	private double salario, previdencia, adicional_whisky, adicional_helicoptero,
		adicional_adicional;

	public presidente( int ID, String nome, int departamento, double salario){
		this.nome=nome;
		this.data_nascimento= data_nascimento;
		this.data_admissao= data_admissao;
		this.departamento= departamento;
		this.salario= salario;
		this.previdencia=0.05;  // Desconto fixo de um sistema de previdência da empresa
		this.adicional_whisky=0.9;
		this.adicional_helicoptero=0.7;
		this.adicional_adicional=3.8;

	}
	void setAdicional(double novo_percentual){
		adicional_whisky=0.9*(1+novo_percentual);
		adicional_helicoptero=0.7*(1+novo_percentual);
		adicional_adicional=3.8*(1+novo_percentual);
	}
```

```
        double pagamento(){
            // Calcula o salário líquido
            return(salario*(1-this.previdência+this.adicional_whisky+this.adicional_helicoptero
                    + this.adicional_adicional));
        }
    }
```

1ª Parte do trabalho

Observando-se cada classe isoladamente, nota-se que cada uma representa bem cada funcionário. Porém, quando observadas em conjunto, várias modificações podem ser introduzidas para se aproveitar as funcionalidades da orientação a objetos, tais como herança e polimorfismo. Assim reescreva as classes acima de forma aproveitando as funcionalidades da orientação a objetos vistas na disciplina.

2ª Parte do trabalho

Após você reescrever as classes escreva a **classe Empresa** que armazena os funcionários da empresa. A **classe Empresa** utiliza um ***container*** (**coleção objetos**), ou seja, um **vetor objetos** com **no máximo 100 posições** para armazenar os objetos que representam os funcionários da empresa. Além disso a **classe Empresa** disponibiliza **métodos** que implementam as seguintes operações:

1) Adicionar um funcionário em uma dada posição do *container*;
2) Remover um funcionário em uma dada posição do *container*;
3) Calcular pagamento do funcionário.
4) Alterar o desconto do funcionário.
5) Listar os funcionários da empresa.

Para um usuário conseguir administrar os funcionários armazenados na **classe Empresa**, implemente uma **classe principal** com a função **main()**, que apresenta um menu para o usuário escolher qual das operações acima irá executar, em seguida é executado o método correspondente a operação, e disponibilizado novamente o menu para que usuário escolha uma próxima operação, e é claro, você deve disponibilizar no menu uma opção para finalizar o programa.

Na implementação dos **métodos** da **classe Empresa** as informações necessárias para a sua execução devem ser passadas por parâmetro, ou seja, os métodos realizam a entrada de ados (**Scanner**)e nem saída de dados (**System.out.println**) .

Por exemplo, se for solicitada a 1ª operação (Adicionar um funcionário em uma dada posição) o usuário deverá informar os dados do funcionário e a posição no *container* onde será inserido o funcionário, em seguida é criado um novo funcionário (objeto) e a **classe principal** faz a chamada do **método da classe Empresa** informando o objeto e a posição a ser inserida, que realiza inserção e atualização no *container*.

Detalhamento das operações

1) Para **adicionar** um funcionário em uma dada posição, no **método da classe Empresa** responsável por essa operação, deve primeiro verificar se a posição faz sentido ou não, ou seja, só podemos adicionar um funcionário em alguma posição que já estava ocupada ou na primeira posição disponível no final do *container*. Caso a posição seja válida, devemos tomar cuidado para não colocar um elemento sobre outro. É preciso deslocar todos os elementos a "direita" da posição onde vamos inserir uma vez para a "frente". Isso abrirá um espaço para guardar o novo elemento.

2) Para **remover** um funcionário em uma dada posição, no **método da classe Empresa** responsável por essa operação recebe por parâmetro a posição que será posição e verifica se a posição está ocupada ou não, se a posição estiver ocupada então podemos remover o funcionário. Para isso basta deslocar os elementos que estavam à direita daquele que removemos uma vez para esquerda e fechamos o "buraco" aberto pela remoção, ao final o método retorna **true** caso tenha sucesso ou **false** caso contrário.

3) Para **calcular o pagamento** do funcionário o método da classe Empresa responsável por essa operação recebe por parâmetro o ID do funcionário, em seguida busca no *container* o funcionário calcula e devolve o valor do salário do funcionário para a classe principal.

4) Para **alterar o desconto** do funcionário o método da classe Empresa responsável por essa operação recebe por parâmetro o ID do funcionário e o novo valor de desconto, em seguida busca no *container* o funcionário e atualiza o valor de desconto do funcionário, caso essa alteração seja permitida para o funcionário em questão, por exemplo o presidente não tem desconto, e não deve ser alterado. Caso o método consiga alterar o valor do desconta do funcionário é retornado **true** e **false** caso contrário.

5) Para **listar os funcionários da empresa** o método da classe Empresa responsável por essa operação deve devolver uma String com a lista com todos os funcionários da empresa contendo o ID e o nome do funcionário de cada um dos funcionários armazenados no *container*.

Restrições do projeto

1) Para armazenar os funcionários do seu sistema você deverá usar **vetor de objetos** como **container**, não é permitido usar classes prontas do Java (**Java Collections**) para isso, como **ArrayList**.

2) O programa deve estar bem documentado e implementado na **linguagem Java**, aplicando os conhecimentos sobre programação orientado a objetos e tipo abstrato de dados vistos nesse semestre.

3) Este trabalho pode ser desenvolvido em grupos de até **3 alunos**. Como este trabalho pode ser feito em grupo, evidentemente você pode "discutir"' o problema dado com outros grupos, inclusive as "dicas"' para chegar às soluções, mas você deve ser responsável pela solução final e pelo desenvolvimento do seu programa. Ou seja, qualquer tentativa de fraude será punida com a **nota zero**. Para maiores esclarecimentos leiam o documento "**Orientações para Desenvolvimento de Trabalhos Práticos**".

4) A entrega consistirá em uma apresentação do projeto ao professor responsável pela disciplina e será a avaliado de acordo com os seguintes critérios:
   a) Funcionamento do programa;
   b) O quão fiel é o programa quanto à descrição do enunciado;
   c) Clareza na nomenclatura de variáveis, métodos e classes;
   d) E principalmente a aplicação dos conceitos de orientação a objetos vistos durante as aulas da disciplina;