Exercício 1) Apresente a implementação em linguagem orientada a objetos do seguinte enunciado:

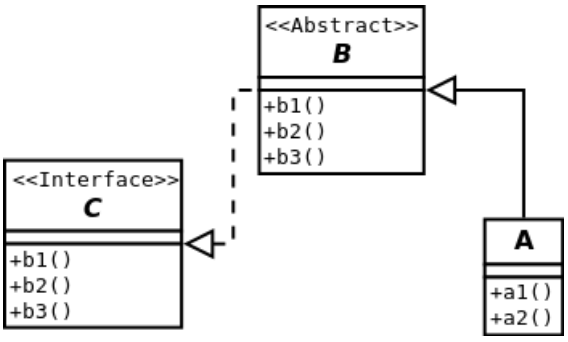


Figura 1

Nome	Tipo
A	Classe abstrata
B	Classe abstrata
C	Interface

Tabela 1

a) (1,0 ponto) Todos os métodos das classes da Tabela 1 são abstratos com retornos não-nulos. Apresente a implementação em linguagem orientada a objetos do diagrama da Figura 1 conforme a Tabela 1.

b) (1,0 ponto) Apresente 1 (um) exemplo de instanciação de 2 (dois) objetos de subclasses da classe A que exibam o retorno de todos os métodos abstratos do item a).

c) (1,0 ponto) Apresente 1 (um) exemplo de polimorfismo que mostre o nome da classe com a implementação do item b).

d) (1,0 ponto) Apresente 1 (um) exemplo de classe interna anônima na classe A para cada um dos métodos da interface C.

e) (1,0 ponto) Implemente uma classe com um método "main" que possua uma lista de alocação dinâmica da interface C, e que armazena objetos das subclasses da Classe A. Itere polimorficamente a lista com a impressão do nome da classe de cada objeto.

f) (1,5 ponto) Apresente 1 (um) exemplo de método que faça captura e imprima uma mensagem de exceção não-verificada com classe de exceção personalizada. A mensagem de exceção deve ser fornecida para o construtor sobrecarregado da classe de exceção. A exceção é capturada na leitura de um valor inválido inserido pelo usuário em um objeto da subclasse da classe A.

g) (1,5 ponto) Apresente 1 (um) exemplo de método que faça disparo e imprima uma mensagem de exceção verificada com classe de exceção personalizada. A mensagem de exceção deve ser fornecida para o construtor sobrecarregado da classe de exceção. A exceção é disparada na leitura de um valor inválido inserido pelo usuário em um objeto da subclasse da classe A.

h) (2,0 pontos) Apresente 2 (dois) exemplos de correção das entradas inválidas, após a captura da exceção, com classes de exceção personalizadas para o item f), e item g).

Resposta

```
public abstract class B implements C {
     public abstract int b1();
     public abstract int b2();
     public abstract int b3();
}

public abstract class A extends B {

  public abstract int a1();
  public abstract int a2();

  public A(){

        B  b  = new B () {   //classe interna anônima
            public int b1(){
                  return 1;
            }
           public int b2(){
              return 2;
           }
           public int b3(){
              return 3;
           }
        };

  }

}

//subclasse de A
public class A1 extends A {

      public int leitura(){
            Scanner leitor = new Scanner( System.in );
            int resultado = leitor.nextInt();
            return  resultado;
      }

public int a1(){
      return 1;
  }
```

```
  public int a2(){
      return 2;
  }

  public int b1(){
       return 1;
    }
    public int b2(){
       return 2;
    }
    public int b3(){
       return 3;
    }

    public String toString() {
        return a1() + " " + a2() + " " + b1() + " " + b2() + " " + b3();
    }
}//fim da subclasse A1

//Resposta da questao f)
public class ME extends RuntimeException {

          public ME(String m){
               super(m);
          }

          public int corrigir(int valor){
               if( valor == 0 )
                   valor = 1;
               return valor;
          }

}//fim classe ME

//Resposta da questao g)
public class MEV extends Exception {

          public MEV(String m){
               super(m);
          }

          //Resposta da questao h)
          public int corrigir(int valor){
               if( valor == 0 )
                   valor = 1;
               return valor;
          }
```

```
}//fim classe MEV


      //Resposta da questao c)
     public Principal(){

          //Resposta da questao b)
          A1  a = new A1();
          A1  b = new A1();

         //Resposta da questao c)
         C c = a;
         System.out.println( c );

         c = b;
         System.out.println( c );

         //Resposta e)
         List<C> lista = new ArrayList<>();
         lista.add( new A1() );
         for( C item : lista )
               System.out.println( item );

        //Resposta da questao f)
        try {
            if ( a.leitura() == 0 )
                  throw new ME("Minha mensagem!");
        } catch ( ME  me){
            System.out.println( me.getMessage() );
            a = me.corrigir( a.leitura() );
        }

        //Resposta da questao g)
       try {
            if ( a.leitura() == 0 )
                  throw new MEV("Minha mensagem!");
        } catch ( MEV  me){
            System.out.println( me.getMessage() );
            a = me.corrigir( a.leitura() );
        }




     }

      public static void main(String [  ] args ){
           new Principal();
```

```
    }

}//fim da classe Principal
```

Exercício 2) Uma fábrica possui 3 (três) tipos de máquinas de produção. Elabore um programa orientado a objetos que:

a) Calcule a média mensal de produção da empresa de acordo com a Tabela 1:

	JAN	FEV	MAR
Máquina1	10	20	30
Máquina2	40	50	60
Máquina3	70	80	90

b) Faça o tratamento de exceção verificada e personalizada para o caso do usuário informar um valor de produção mensal negativo.

```
public class Principal {

    private int [] maquina1 = { 10, 20, 30 };
    private int [] maquina2 = { 40, 50, 60 };
    private int [] maquina3 = { 70, 80, 90 };

    public class MEV extends Exception {
        public MEV(String m){
            super( m );
        }

    }

    public void leitura()  throws MEV {
        Scanner leitor = new Scanner( System.in );
        int resultado = leitor.nextInt();
        if ( resultado < 0  )
          throw new MEV(“PRODUCAO NEGATIVA” );
    }
```

```
    public void calcularMedia(){

        float media = 0;
        for(int i=0; i<3; i++){
            media = 0;
            media += maquina1[ i ] + maquina2[ i ] + maquina3[ i ];
            System.out.println( media / 3 );
        }
    }

    public Principal(){

        try {
           leitura();
        } catch( MEV  mev ){
             System.out.println(  mev.getMessage() );
        }

        calcularMedia();

    }

    public static void main(String [ ] a ){

       new Principal();
    }

}
```

Exercício 3) Implemente com Programação Orientada a Objetos:

a) (1,0 ponto) Crie 3 (três) classes: ContaCorrente, ContaSalário e ContaPoupança. Crie 2 (dois) construtores sobrecarregados para cada classe.

```
package src;

public class ex3 {

	public class ContaCorrente {
		private String nomeCliente;
		private Float saldo;

		// construtor
		public ContaCorrente (String nomecliente, Float saldo ) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
		}

		// construtor sobrecarregado
		public ContaCorrente (String nomeCiente) {
			this.nomeCliente = nomeCliente;
			this.saldo = 0f;
		}

	}

	public class ContaSalario {
		private String nomeCliente;
		private Float saldo;

		// construtor
		public ContaSalario (String nomecliente, Float saldo ) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
		}

		// construtor sobrecarregado
		public ContaSalario (String nomeCiente) {
			this.nomeCliente = nomeCliente;
			this.saldo = 0f;
		}
```

```
	}

	public class ContaPoupanca {
		private String nomeCliente;
		private Float saldo;

		// construtor
		public ContaPoupanca (String nomecliente, Float saldo ) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
		}

		// construtor sobrecarregado
		public ContaPoupanca (String nomeCiente) {
			this.nomeCliente = nomeCliente;
			this.saldo = 0f;
		}
	}


}
```

b) (1,0 ponto) Forneça para cada Classe 1 (um) atributo e 1 (um) comportamento únicos que não estão presentes nas outras classes. Inicialize os atributos em um dos construtores da Classe.

```
package src;

public class ex3 {

//—---------------------------------- conta corrente —----------------------------------------------------
	public class ContaCorrente {
		private String nomeCliente;
		private Float saldo;
		private Float debitoAutomatico; // atributo unico

		// construtor
		public ContaCorrente (String nomecliente, Float saldo, Float chequeEspecial
) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
			this.debitoAutomatico = debitoAutomatico;
		}
```

```
		// construtor sobrecarregado
		public ContaCorrente (String nomeCiente) {
			this.nomeCliente = nomeCliente;
			this.saldo = 100f;
			this.debitoAutomatico = 0f;
		}

		// comportamento unico
		public void cadastrarDebito ( Float novoDebito) {
			if (saldo > novoDebito) {
				saldo = saldo - novoDebito;
				debitoAutomatico += novoDebito;
			}else {
				System.out.println("Voce nao tem saldo suficiante para
cadastrar um novo debito automatico");
			}
		}
	}
//—---------------------- conta salario —-------------------------------------------------------------------
	public class ContaSalario {
		private String nomeCliente;
		private Float saldo;
		private Integer tempoTrabalho; // atributo unico

		// construtor
		public ContaSalario (String nomecliente, Float saldo, Integer
tempoTrabalhado ) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
			this.tempoTrabalho = tempoTrabalhado;
		}

		// construtor sobrecarregado
		public ContaSalario (String nomeCiente) {
			this.nomeCliente = nomeCliente;
			this.saldo = 0f;
			this.tempoTrabalho = 0;
		}

		//comportamento unico
		public void Bonifacao () {
			tempoTrabalho = tempoTrabalho/100;
			saldo += tempoTrabalho;
			System.out.println("Bonificacao bem sucedida");

		}
	}
// —---------------------- conta poupanca —-----------------------------------------------------
```

```
	public class ContaPoupanca {
		private String nomeCliente;
		private Float saldo;
		private Float taxaJuros; //atributo unico

		// construtor
		public ContaPoupanca (String nomecliente, Float saldo ) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
		}

		// construtor sobrecarregado
		public ContaPoupanca (String nomeCiente) {
			this.nomeCliente = nomeCliente;
			this.saldo = 0;
		}

		// comportamento unico
		public void Rendimento () {
			double rendimento = saldo * taxaJuros;
		   saldo += rendimento;
		   System.out.println("Rendimento calculado e saldo atualizado.");
		}
	}


}
```

c) (1,0 ponto) Crie a Interface 'ICliente' com um método 'getCliente()' que retorna o nome do cliente. Cada Classe deve implementar a Interface 'ICliente'.

```
package src;

public class ex3 {

	public interface ICliente  {
		String getCliente();
	}

// ------------------------------ conta corrente ------------------------------
	public class ContaCorrente implements ICliente {
		private String nomeCliente;
```

```
		private Float saldo;
		private Float debitoAutomatico; // atributo unico

		// construtor
		public ContaCorrente (String nomecliente, Float saldo, Float chequeEspecial
) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
			this.debitoAutomatico = debitoAutomatico;
		}

		// construtor sobrecarregado
		public ContaCorrente (String nomeCiente) {
			this.nomeCliente = nomeCliente;
			this.saldo = 100f;
			this.debitoAutomatico = 0f;
		}

		// comportamento unico
		public void cadastrarDebito ( Float novoDebito) {
			if (saldo > novoDebito) {
				saldo = saldo - novoDebito;
				debitoAutomatico += novoDebito;
			}else {
				System.out.println("Voce nao tem saldo suficiante para
cadastrar um novo debito automatico");
			}
		}

		public String getCliente () {
			return this.nomeCliente;
		}
	}

//---------------------------- Conta Salario -------------------------------------

	public class ContaSalario implements ICliente {
		private String nomeCliente;
		private Float saldo;
		private Integer tempoTrabalho; // atributo unico

		// construtor
		public ContaSalario (String nomecliente, Float saldo, Integer
tempoTrabalhado ) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
			this.tempoTrabalho = tempoTrabalhado;
		}
```

```
                // construtor sobrecarregado
                public ContaSalario (String nomeCiente) {
                        this.nomeCliente = nomeCliente;
                        this.saldo = 0f;
                        this.tempoTrabalho = 0;
                }

                //comportamento unico
                public void Bonifacao () {
                        tempoTrabalho = tempoTrabalho/100;
                        saldo += tempoTrabalho;
                        System.out.println("Bonificacao bem sucedida");

                }

                public string getCliente () {
                        return this.nomeCliente;
                }
        }

// ---------------------- conta poupanca ----------------------------------
        public class ContaPoupanca implements ICliente {
                private String nomeCliente;
                private Float saldo;
                private Float taxaJuros; //atributo unico

                // construtor
                public ContaPoupanca (String nomecliente, Float saldo ) {
                        this.nomeCliente = nomecliente;
                        this.saldo = saldo;
                }

                // construtor sobrecarregado
                public ContaPoupanca (String nomeCiente) {
                        this.nomeCliente = nomeCliente;
                        this.saldo = 0;
                }

                // comportamento unico
                public void Rendimento () {
                        double rendimento = saldo * taxaJuros;
                   saldo += rendimento;
                   System.out.println("Rendimento calculado e saldo atualizado.");
                }

                public String getCliente() {
                        return this.nomeCliente;
```

```
		}

	}


} // fim classe ex3
```

d) (1,0 ponto) Crie uma classe abstrata 'Conta' com um método abstrato 'getSaldo()' que retorna o saldo do cliente. A Classe Conta tem um relacionamento de Generalização com as Classes ContaCorrente, ContaSalário e ContaPoupança.

```
package src;

public class ex3 {

// ----------- classe abstrata Conta ----------------------------------------------------

	public abstract  class Conta {
		 public abstract Float getSaldo();
	}

// ------------ interface ICliente ----------------------------------------------------

	public interface ICliente  {
		String getCliente();
	}

// ------------------------------- conta corrente -------------------------------
	public class ContaCorrente extends Conta  implements ICliente {
		private String nomeCliente;
		private Float saldo;
		private Float debitoAutomatico; // atributo unico

		// construtor
		public ContaCorrente (String nomecliente, Float saldo, Float chequeEspecial
) {
			this.nomeCliente = nomecliente;
			this.saldo = saldo;
			this.debitoAutomatico = debitoAutomatico;
		}

		// construtor sobrecarregado
		public ContaCorrente (String nomeCiente) {
```

```
                        this.nomeCliente = nomeCliente;
                        this.saldo = 100f;
                        this.debitoAutomatico = 0f;
                }

                // comportamento unico
                public void cadastrarDebito ( Float novoDebito) {
                        if (saldo > novoDebito) {
                                saldo = saldo - novoDebito;
                                debitoAutomatico += novoDebito;
                        }else {
                                System.out.println("Voce nao tem saldo suficiante para
cadastrar um novo debito automatico");
                        }
                }

                public String getCliente () {  // retorna nome do cliente
                        return this.nomeCliente;
                }

                public Float getSaldo () {  // retorna saldo do cliente
                        return this.saldo;
                }
        }

//---------------------------- Conta Salario -------------------------------------

        public class ContaSalario extends Conta implements ICliente {
                private String nomeCliente;
                private Float saldo;
                private Integer tempoTrabalho; // atributo unico

                // construtor
                public ContaSalario (String nomecliente, Float saldo, Integer
tempoTrabalhado ) {
                        this.nomeCliente = nomecliente;
                        this.saldo = saldo;
                        this.tempoTrabalho = tempoTrabalhado;
                }

                // construtor sobrecarregado
                public ContaSalario (String nomeCiente) {
                        this.nomeCliente = nomeCliente;
                        this.saldo = 0f;
                        this.tempoTrabalho = 0;
                }

                //comportamento unico
```

```
                public void Bonifacao () {
                        tempoTrabalho = tempoTrabalho/100;
                        saldo += tempoTrabalho;
                        System.out.println("Bonificacao bem sucedida");

                }

                public string getCliente () {
                        return this.nomeCliente;
                }

                public Float getSaldo () {
                        return this.saldo;
                }
        }

// ---------------------- conta poupanca ------------------------------------
        public class ContaPoupanca extends Conta implements ICliente {
                private String nomeCliente;
                private Float saldo;
                private Float taxaJuros; //atributo unico

                // construtor
                public ContaPoupanca (String nomecliente, Float saldo ) {
                        this.nomeCliente = nomecliente;
                        this.saldo = saldo;
                }

                // construtor sobrecarregado
                public ContaPoupanca (String nomeCiente) {
                        this.nomeCliente = nomeCliente;
                        this.saldo = 0;
                }

                // comportamento unico
                public void Rendimento () {
                        double rendimento = saldo * taxaJuros;
                   saldo += rendimento;
                   System.out.println("Rendimento calculado e saldo atualizado.");
                }

                public String getCliente() {
                        return this.nomeCliente;
                }

                public Float getSaldo () {
                        return this.saldo;
                }
```

```
        }


} // fim classe ex3
```

e) (1,0 ponto) Crie uma classe Principal com um vetor 'ICliente' que armazena objetos das Classes ContaCorrente, ContaSalario e ContaPoupanca. Itere polimorficamente o vetor imprimindo o conteúdo de cada objeto da seguinte forma: um objeto da Classe ContaCorrente imprime o nome da Classe e o valor do saque; um objeto da Classe ContaSalário imprime o nome da Classe, o valor e o CPF do cliente; um objeto da Classe ContaPoupança imprime o nome da Classe, o saldo e a data da consulta.

*** colocar tudo em arquivos .java separados

```
package src;

public class Principal {

// ----------- classe abstrata Conta ----------------------------------------------------

        public abstract class Conta {
                public abstract double getSaldo();
        }

// ------------ interface ICliente ------------------------------------------------------

        public interface ICliente {
                String getCliente();
        }

// ------------------------------- conta corrente ------------------------------
        public class ContaCorrente extends Conta implements ICliente {
                private String nomeCliente;
                private double saldo;

                // construtor
                public ContaCorrente(String nomeCliente, double saldo) {
                        this.nomeCliente = nomeCliente;
                        this.saldo = saldo;
                }

                // construtor sobrecarregado
```

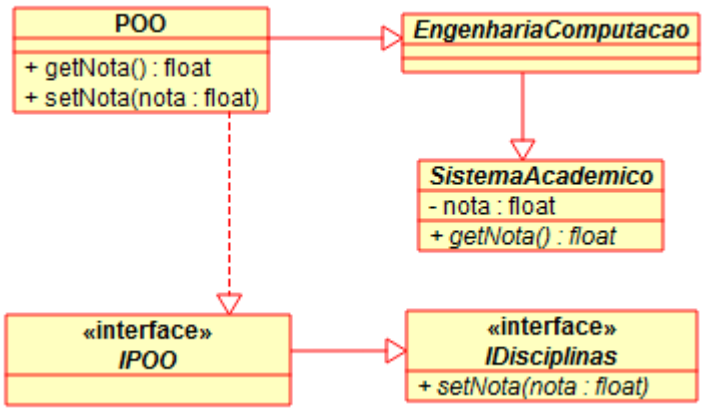
```
		public ContaCorrente(String nomeCiente) {
			this.nomeCliente = nomeCliente;
			this.saldo = 0;
		}

		public String getNomeCliente() {
			return nomeCliente;
		}

		public void setNomeCliente(String nomeCliente) {
			this.nomeCliente = nomeCliente;
		}

		public void setSaldo(double saldo) {
			this.saldo = saldo;
		}

		// Comportamento único
		public void realizarTransferencia(double valor, ContaCorrente contaDestino)
{
			if (this.saldo >= valor) {
				this.saldo -= valor;
				contaDestino.saldo += valor;
				System.out.println("Transferência realizada com sucesso.");
			} else {
				System.out.println("Saldo insuficiente para transferência.");
			}
		}

		public String getCliente() { // retorna nome do cliente
			return this.nomeCliente;
		}

		public double getSaldo() { // retorna saldo do cliente
			return this.saldo;
		}
	}

//--------------------------- Conta Salario -------------------------------------

	public class ContaSalario extends Conta implements ICliente {
		private String nomeCliente;
		private String cpf;
		private double valor;

		// construtor
		public ContaSalario(String nomeCliente, String cpf, double valor) {
			this.nomeCliente = nomeCliente;
```

```
        this.cpf = cpf;
        this.valor = valor;
    }

    // construtor sobrecarregado
    public ContaSalario(String nomeCiente) {
        this.nomeCliente = nomeCliente;
        this.cpf = "";
        this.valor = 0;
    }

    public String getNomeCliente() {
        return nomeCliente;
    }

    public void setNomeCliente(String nomeCliente) {
        this.nomeCliente = nomeCliente;
    }

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    public double getValor() {
        return valor;
    }

    public void setValor(double valor) {
        this.valor = valor;
    }

    // Comportamento único
    public void receberSalario(double valor) {
        this.valor += valor;
        System.out.println("Salário recebido com sucesso.");
    }

    public String getCliente() {
        return this.nomeCliente;
    }

    public double getSaldo() {
        return this.valor;
    }
```

```
        }

// ---------------------- conta poupanca -----------------------------------
        public class ContaPoupanca extends Conta implements ICliente {
                private String nomeCliente;
                private double saldo;
                private String dataConsulta;

                public ContaPoupanca(String nomeCliente, double saldo, String
dataConsulta) {
                        this.nomeCliente = nomeCliente;
                        this.saldo = saldo;
                        this.dataConsulta = dataConsulta;
                }

                // construtor sobrecarregado
                public ContaPoupanca(String nomeCiente) {
                        this.nomeCliente = nomeCliente;
                        this.saldo = 0;
                        this.dataConsulta = "";
                }

                public String getNomeCliente() {
                        return nomeCliente;
                }

                public void setNomeCliente(String nomeCliente) {
                        this.nomeCliente = nomeCliente;
                }

                public String getDataConsulta() {
                        return dataConsulta;
                }

                public void setDataConsulta(String dataConsulta) {
                        this.dataConsulta = dataConsulta;
                }

                public void setSaldo(double saldo) {
                        this.saldo = saldo;
                }

                // Comportamento único
                public void realizarInvestimento(double valor) {
                        this.saldo += valor;
                        System.out.println("Investimento realizado com sucesso.");
                }
```

```
		public String getCliente() {
			return this.nomeCliente;
		}

		public double getSaldo() {
			return this.saldo;
		}

	}

// ---------- main ----------
	public static void main(String[] args) {

		// Adicionando objetos das classes ContaCorrente, ContaSalario e
ContaPoupanca
		ContaCorrente corrente = new ContaCorrente("João", 500);
		ContaSalario salario = new ContaSalario("Maria", "123456789", 1000);
		ContaPoupanca poupanca = new ContaPoupanca("Carlos", 2000,
"20/10/2023");

		ICliente[] clientes = { corrente, salario, poupanca };


		// Iterando polimorficamente o vetor e imprimindo os detalhes
		for (ICliente cliente : clientes) {
			if (cliente instanceof ContaCorrente) {
				ContaCorrente contaCorrente = (ContaCorrente) cliente;
				System.out.println("Classe: ContaCorrente");
				System.out.println("Saldo: " + contaCorrente.getSaldo());
			} else if (cliente instanceof ContaSalario) {
				ContaSalario contaSalario = (ContaSalario) cliente;
				System.out.println("Classe: ContaSalario");
				System.out.println("Valor: " + contaSalario.getValor());
				System.out.println("CPF: " + contaSalario.getCpf());
			} else if (cliente instanceof ContaPoupanca) {
				ContaPoupanca contaPoupanca = (ContaPoupanca) cliente;
				System.out.println("Classe: ContaPoupanca");
				System.out.println("Saldo: " + contaPoupanca.getSaldo());
				System.out.println("Data da Consulta: " +
contaPoupanca.getDataConsulta());
			}
		}
	}

} // fim classe ex3
```

Exercício 4) No Diagrama UML da figura a seguir:



Nome	Tipo
SistemaAcademico	Classe Abstrata
EngenhariaComputacao	Classe Abstrata
POO	Classe Concreta
IPOO	Interface
IDisciplinas	Interface

a) (3,0 pontos) Implemente o Diagrama UML apresentado em linguagem de Programação Orientada a Objetos.

**** fazer tudo em .java separado

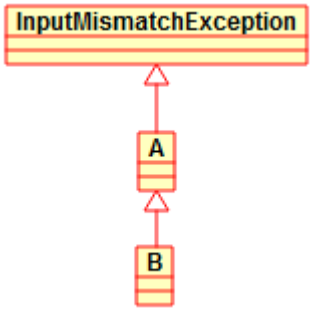
```
package src;

public class Principal {


        public abstract class SistemaAcademico {
```

```
    private float nota;

    public float getNota() {
      return nota;
    }
  }


  public abstract class EngenhariaComputacao extends SistemaAcademico {

  }


  public class POO extends EngenhariaComputacao implements IPoo {
    private float nota;


    public float getNota() {
      return nota;
    }


    public void setNota(float nota) {
      this.nota = nota;
    }
  }


  public interface IPoo extends IDisciplinas {

  }


  public interface IDisciplinas {
    void setNota(float nota);
  }

  public static void main(String[] args) {

        // Instanciando um objeto da classe POO
POO poo = new POO();

// Atribuindo e recuperando a nota com tratamento de exceções
try {
  // Atribuindo a nota
  poo.setNota(8.5f);

  // Recuperando a nota
```

```
        float notaRecuperada = poo.getNota();
        System.out.println("Nota recuperada: " + notaRecuperada);
      } catch (Exception e) {
        System.out.println("Ocorreu um erro ao atribuir ou recuperar a nota: " +
e.getMessage());
      }

   }

}// fim da principal
```

b) (2,0 pontos) Implemente a Classe Principal que instancia um objeto da Classe POO. A partir desse objeto da Classe POO, ilustre a invocação dos métodos acessores e mutadores para atribuir e recuperar a nota do Sistema Acadêmico, com tratamento de exceções.

Exercício 5) Observe o diagrama UML a seguir:



a) Implemente um método de leitura de dados do usuário que capture a exceção do tipo A.
b) Implemente um método de leitura de dados do usuário que capture a exceção do tipo B.
c) Implemente um método de leitura de dados do usuário que capture a exceção do tipo A. A seguir, dispare a exceção do tipo B e faça a captura.

d) Modifique o programa anterior para que a superclasse seja uma classe de Exceção verificada.