

Orientação a Objetos

Operações e Associações

Operações - Exemplo 1

Métodos do Objeto

Operações

- ▷ Trabalhamos anteriormente com métodos *get* e *set*. Estes métodos são utilizados em Java para controlar o acesso a atributos.
 - Em outras linguagens, como em C#, existe o conceito de propriedades, que são utilizadas para este fim.
- ▷ Da mesma forma que utilizamos métodos para encapsular atributos, também podemos utilizar outros tipos de métodos, que podem realizar cálculos que alteram ou não alteram o estado do objeto.
 - Chamamos estes métodos de operações.

Operações

- ▷ Seguindo com o nosso exemplo, chegou até nós que o salário líquido do nosso funcionário é calculado da seguinte forma:
 - $\text{Salário líquido} = \text{salário bruto} - \text{valorDescontos};$

Operações

- ▷ Novos atributos encapsulados na classe Funcionario:

```
public double getSalarioBruto() {  
    return salarioBruto;  
}  
public void setSalarioBruto(double salario) {  
    this.salarioBruto = salario;  
}  
public double getValorDescontos() {  
    return valorDescontos;  
}  
public void setValorDescontos(double valor) {  
    this.valorDescontos = valor;  
}  
private double salarioBruto;  
private double valorDescontos;
```

Operações

- ▷ Como podemos calcular o salário líquido?

Operações

- ▷ Como podemos calcular o salário líquido?

```
public double getSalarioLiquido() {  
    return salarioBruto - valorDescontos;  
}
```

Operações

- ▷ Novo código do main:

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```


Operações

- ▷ Novo código do main:

codigo:
nome:
salarioBruto:
valorDescontos:

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▷ Novo código do main:

codigo: 1
nome:
salarioBruto:
valorDescontos:

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▷ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto:
valorDescontos:

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▷ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto: 5000
valorDescontos:

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▷ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto: 5000
valorDescontos: 1200

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▷ Novo código do main:

codigo: 1
nome: Bruno
salarioBruto: 5000
valorDescontos: 1200

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▶ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto: 5000
valorDescontos: 1200

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
  
        public double getSalarioLiquido() {  
            return salarioBruto - valorDescontos;  
        }  
  
        System.out.printf("Salário líquido: %.2f\n",  
            func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▶ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto: 5000
valorDescontos: 1200

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
  
        public double getSalarioLiquido() {  
            return salarioBruto - valorDescontos;  
        }  
  
        System.out.printf("Salário líquido: %.2f\n",  
            func1.getSalarioLiquido());  
    }  
}
```


Operações

- ▷ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto: 5000
valorDescontos: 1200

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
  
        public double getSalarioLiquido() {  
            return 5000 - 1200;  
        }  
  
        System.out.printf("Salário líquido: %.2f\n",  
            func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▷ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto: 5000
valorDescontos: 1200

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
  
        public double getSalarioLiquido() {  
            return 3800;  
        }  
  
        System.out.printf("Salário líquido: %.2f\n",  
            func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▷ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto: 5000
valorDescontos: 1200

```
public class Main {  
    public static void main(String[] args) {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```

Operações

- ▷ Novo código do main:

codigo: 1
nome: **Bruno**
salarioBruto: 5000
valorDescontos: 1200

```
public class Main {  
    public static void main()  
    {  
        Funcionario func1 = new Funcionario();  
        func1.setCodigo(1);  
        func1.setNome("Bruno");  
        func1.setSalarioBruto(5000);  
        func1.setValorDescontos(1200);  
        System.out.printf("Salário líquido: %.2f\n",  
                           func1.getSalarioLiquido());  
    }  
}
```

Salário líquido: 3800,00

Operações - Exemplo 2

Controlando as ações de um objeto

Operações

- ▷ Trabalharemos com um exemplo à partir de agora que é a modelagem inicial de um pequeno sistema bancário.
- ▷ Vamos iniciar pela classe Conta, que possui os atributos número e saldo.

Operações

- ▷ Existem métodos que controlam o comportamento de um objeto.
 - Estes são chamados de operações.
- ▷ Em nosso exemplo, o usuário do sistema pode depositar e sacar quantias em uma conta.

Operações

- ▷ A classe ficaria assim:

Conta
-numero: int -saldo: double
+sacar(quantia: double) +depositar(quantia: double)

- ▷ Vamos verificar como fica a implementação da mesma:

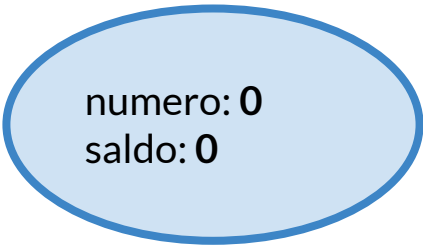
Operações

▷ Classe Conta:

```
public class Conta {  
    public int getNumero() {  
        return numero;  
    }  
  
    public void setNumero(int numero) {  
        this.numero = numero;  
    }  
  
    public double getSaldo() {  
        return saldo;  
    }  
}
```

```
    public void depositar(double quantia) {  
        this.saldo += quantia;  
    }  
  
    public void sacar(double quantia) {  
        this.saldo -= quantia;  
    }  
  
    private int numero;  
    private double saldo = 0;  
}
```

Operações



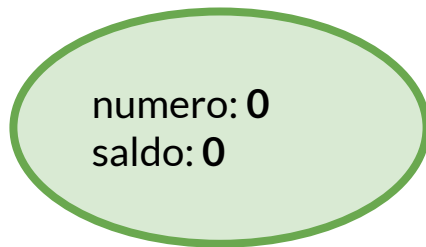
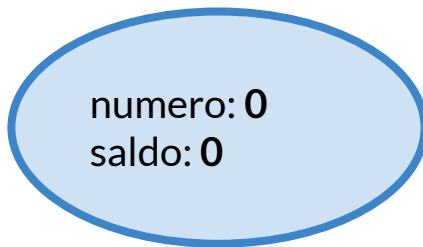
numero: 0
saldo: 0

▷ Main:

```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
c1.setNumero(1);  
c2.setNumero(2);  
c1.depositar(1000);  
c2.depositar(2000);  
c1.sacar(200);  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c1.getNumero(), c1.getSaldo());  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c2.getNumero(), c2.getSaldo());
```

Operações

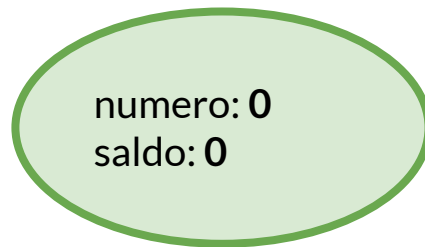
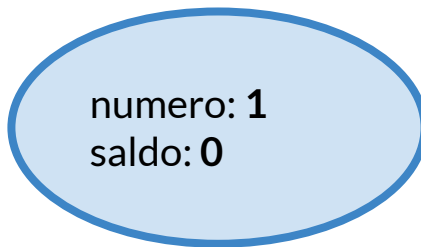
▷ Main:



```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
c1.setNumero(1);  
c2.setNumero(2);  
c1.depositar(1000);  
c2.depositar(2000);  
c1.sacar(200);  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c1.getNumero(), c1.getSaldo());  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c2.getNumero(), c2.getSaldo());
```

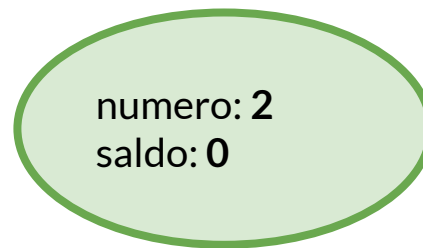
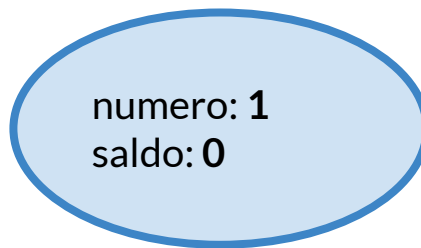
Operações

▷ Main:



```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
c1.setNumero(1);  
c2.setNumero(2);  
c1.depositar(1000);  
c2.depositar(2000);  
c1.sacar(200);  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c1.getNumero(), c1.getSaldo());  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c2.getNumero(), c2.getSaldo());
```

Operações



▷ Main:

```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
c1.setNumero(1);  
c2.setNumero(2);  
c1.depositar(1000);  
c2.depositar(2000);  
c1.sacar(200);  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c1.getNumero(), c1.getSaldo());  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c2.getNumero(), c2.getSaldo());
```

Operações

▷ Main:

numero: 1
saldo: 1000

numero: 2
saldo: 0

```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
c1.setNumero(1);  
c2.setNumero(2);  
c1.depositar(1000);  
c2.depositar(2000);  
c1.sacar(200);  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c1.getNumero(), c1.getSaldo());  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c2.getNumero(), c2.getSaldo());
```

Operações

▷ Main:

numero: 1
saldo: 1000

numero: 2
saldo: 2000

```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
c1.setNumero(1);  
c2.setNumero(2);  
c1.depositar(1000);  
c2.depositar(2000);  
c1.sacar(200);  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c1.getNumero(), c1.getSaldo());  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c2.getNumero(), c2.getSaldo());
```

Operações

▷ Main:

numero: 1
saldo: 800

numero: 2
saldo: 2000

```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
c1.setNumero(1);  
c2.setNumero(2);  
c1.depositar(1000);  
c2.depositar(2000);  
c1.sacar(200);  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c1.getNumero(), c1.getSaldo());  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c2.getNumero(), c2.getSaldo());
```


Operações

▷ Main:

numero: 1
saldo: 800

numero: 2
saldo: 2000

```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
c1.setNumero(1);  
c2.setNumero(2);  
c1.depositar(1000);  
c2.depositar(2000);  
c1.sacar(200);  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c1.getNumero(), c1.getSaldo());  
System.out.printf("Conta %d: saldo: %.2f\n",  
                  c2.getNumero(), c2.getSaldo());
```

Exercícios

1) Na classe Conta, crie a operação Transferir, que deve receber uma quantia (double) e uma conta de destino (Conta). Esta operação deve sacar a quantia da conta que receber a chamada a esta operação e depositar na conta de destino recebida por parâmetro.

Associações

Objetos associados a outros objetos

Associações

- ▷ Em nosso exemplo, temos a definição do que é uma conta, porém ainda faltam controlar alguns dados:
 - Uma conta está relacionada a um correntista.
 - Um correntista é uma pessoa
 - Uma pessoa possui um nome e um e-mail

Associações

- ▷ Vamos nos concentrar primeiramente no relacionamento entre uma conta e a pessoa que é o correntista.
 - A nossa classe conta possui um número, que é um número inteiro.
 - A nossa classe conta possui um saldo, que é um número real.
 - A nossa classe conta possui um correntista, que é uma pessoa.
 - Pessoa deve ser uma classe!

Associações

▷ Classe Pessoa:

```
public class Pessoa {  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

Associações

▷ Continuação da Classe Pessoa:

```
public String getEmail() {  
    return email;  
}  
  
public void setEmail(String email) {  
    this.email = email;  
}  
  
private String nome;  
private String email;  
}
```

Associações

- ▷ Como vamos representar este relacionamento?
- ▷ Lembre-se que a classe Conta contém:
 - número (int)
 - saldo (double)
 - correntista (Pessoa)
- ▷ As três características listadas acima são propriedades, porém não costumam ser representadas da mesma forma.

Associações

- ▷ Uma propriedade pode ser um atributo ou uma associação.
 - Geralmente atributos são utilizados para representar coisas pequenas, muitas vezes são propriedades cujo tipo é um dos tipos básicos da linguagem em questão ou uma classe pertencente ao framework da própria empresa ou pessoa desenvolvedora.
 - Geralmente associações são utilizadas para representar propriedades cujo tipo é uma classe, possivelmente do próprio modelo de domínio.

Associações

- ▷ A escolha entre atributo ou associação está muito mais relacionada à ênfase do que a qualquer significado subjacente.
 - É muito mais importante destacar a relação entre a classe Conta e a classe Pessoa do que destacar a relação entre a classe Conta e o tipo primitivo double, pois o tipo primitivo é um tipo estável (raramente é alterado) e pertencente ao próprio framework.
 - Ao levar a classe Conta para outro sistema, você terá que levar a classe Pessoa também. Esta preocupação é desnecessária com o tipo primitivo double.

Associações

- ▷ Alteração na classe Conta para adicionar a associação:

```
public class Conta {  
    ...  
    public Pessoa getCorrentista() {  
        return correntista;  
    }  
  
    public void setCorrentista(Pessoa correntista) {  
        this.correntista = correntista;  
    }  
    ...  
    private Pessoa correntista;  
}
```

Associações

- ▷ Método principal:

```
public class Main {  
    public static void main(String[] args) {  
        var p = new Pessoa();  
        p.setNome("Ana");  
        p.setEmail("ana@email.com");  
  
        var conta = new Conta();  
        conta.setNumero(123);  
        conta.setCorrentista(p);  
    }  
}
```

Associações

- ▷ Método principal:

```
public class Main {  
    public static void main(String[] args) {  
        var p = new Pessoa();  
        p.setNome("Ana");  
        p.setEmail("ana@email.com");  
  
        var conta = new Conta();  
        conta.setNumero(123);  
        conta.setCorrentista(p);  
    }  
}
```

nome:
email:

Associações

▷ Método principal:

```
public class Main {  
    public static void main(String[] args) {  
        var p = new Pessoa();  
        p.setNome("Ana");  
        p.setEmail("ana@email.com");  
  
        var conta = new Conta();  
        conta.setNumero(123);  
        conta.setCorrentista(p);  
    }  
}
```

nome: Ana
email:

Associações

- ▷ Método principal:

```
public class Main {  
    public static void main(String[] args) {  
        var p = new Pessoa();  
        p.setNome("Ana");  
        p.setEmail("ana@email.com");  
  
        var conta = new Conta();  
        conta.setNumero(123);  
        conta.setCorrentista(p);  
    }  
}
```

nome: Ana
email: ana@email.com

Associações

- ▷ Método principal:

```
public class Main {  
    public static void main(String[] args) {  
        var p = new Pessoa();  
        p.setNome("Ana");  
        p.setEmail("ana@email.com");  
  
        var conta = new Conta();  
        conta.setNumero(123);  
        conta.setCorrentista(p);  
    }  
}
```

numero: 0
saldo: 0
correntista:

nome: Ana
email: ana@email.com

Associações

- ▷ Método principal:

```
public class Main {  
    public static void main(String[] args) {  
        var p = new Pessoa();  
        p.setNome("Ana");  
        p.setEmail("ana@email.com");  
  
        var conta = new Conta();  
        conta.setNumero(123);  
        conta.setCorrentista(p);  
    }  
}
```

numero: 123
saldo: 0
correntista:

nome: Ana
email: ana@email.com

Associações

- ▷ Método principal:

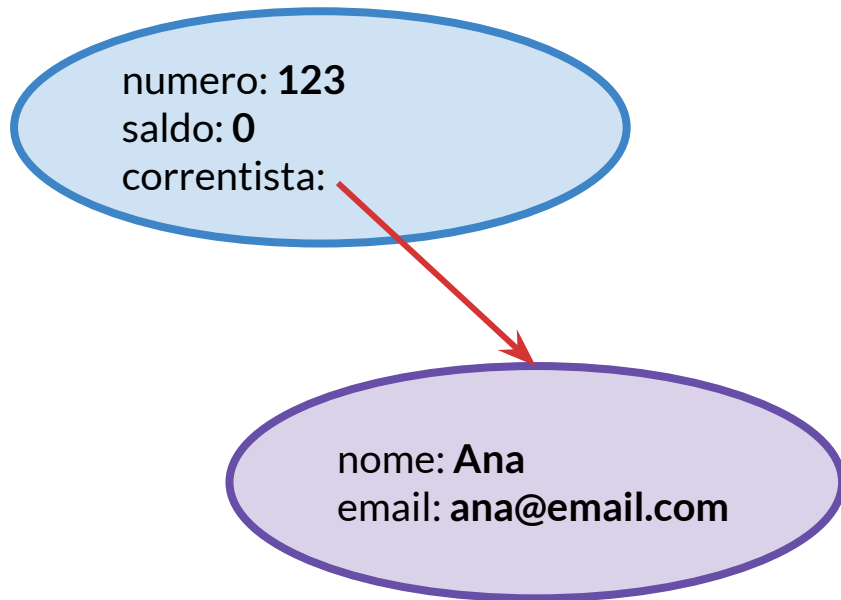
```
public class Main {  
    public static void main(String[] args) {  
        var p = new Pessoa();  
        p.setNome("Ana");  
        p.setEmail("ana@email.com");  
  
        var conta = new Conta();  
        conta.setNumero(123);  
        conta.setCorrentista(p);  
    }  
}
```

numero: 123
saldo: 0
correntista:

nome: Ana
email: ana@email.com

Associações

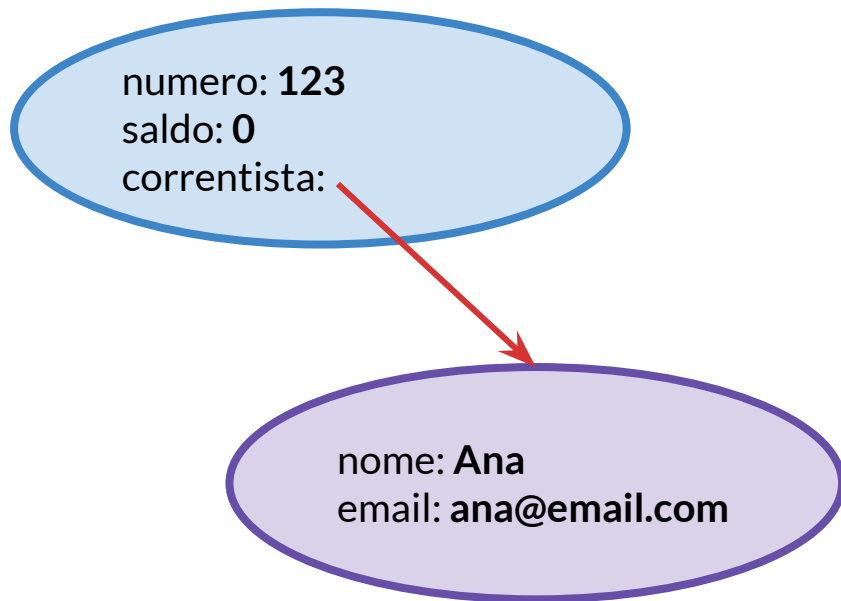
- ▶ Acessando uma pessoa à partir de um objeto do tipo Conta:



```
String nome, email;  
Pessoa p = conta.getCorrentista();  
nome = p.getNome();  
email = p.getEmail();
```

Associações

- ▶ Acessando uma pessoa à partir de um objeto do tipo Conta:



```
String nome, email;  
nome = conta.getCorrentista().getNome();  
email = conta.getCorrentista().getEmail();
```

Exercícios

2) Além de nome e e-mail, uma pessoa possui um endereço. Crie uma classe Endereco para representar um endereço e faça a associação entre o Pessoa e Endereco.

Exercícios

3) Trabalhando com data e hora:

```
import java.time.LocalDate;  
import java.time.LocalTime;
```

```
public class Main {  
    public static void main(String[] args) {  
        LocalDate hoje = LocalDate.now();  
        LocalTime agora = LocalTime.now();  
  
        System.out.printf("%s %s", hoje, agora);  
    }  
}
```

2024-04-10 23:01:02.000070000

Exercícios

3) Crie a associação “transacoes”, que deve ser um `ArrayList` do tipo `Transacao`, sendo que `Transacao` é uma classe contendo três atributos: valor, data e hora. Para débitos, o valor deve ser negativo. Cada transação (saque, depósito ou transferência) deve adicionar uma `Transacao` na lista de transações da conta. A data e a hora da transação precisa ser a data e a hora do computador no momento da transação. Os atributos data e hora devem ser do tipo `LocalDate` e `LocalTime`, respectivamente.

Fim de material

Dúvidas?