

Orientação a objetos

Uma vez estudados os **fundamentos** e **sintaxe básicas** para a construção de **objetos**, e que devem ser muito bem assimilados, o objeto de estudo, em seguida, são os **principais conceitos de orientação a objetos**, detalhando suas **características** e **aplicabilidade**.

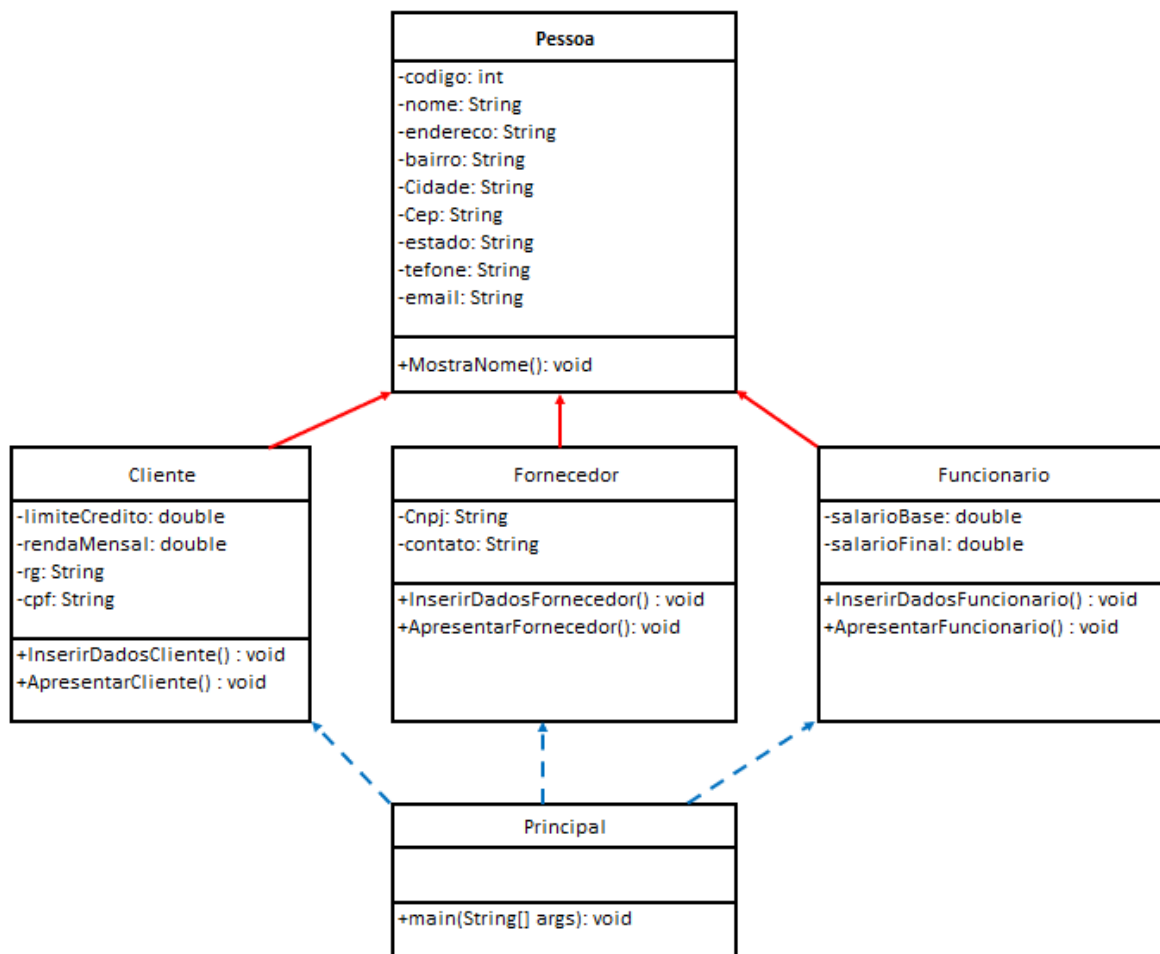
Herança

A herança é um conceito amplamente utilizado em linguagens orientadas a objetos. Além de vantagens facilmente identificadas, como a **reutilização** e **organização** de códigos, a herança também é a base para outros conceitos, como a **sobrescrita de métodos**, **classes e métodos abstratos** e **polimorfismo**. Tais conceitos são fundamentais para a modelagem de sistemas mais robustos.

Durante a análise dos requisitos de um sistema (**solicitações que o sistema deverá atender**), podemos destacar os atributos ou os métodos comuns a um grupo de classes e concentrá-los em uma única classe (**processo conhecido como generalização**). Da mesma forma, é possível identificar o que é pertinente somente a determinada classe (**conhecido como especificação**). A primeira vantagem dessa organização é **evitar a duplicidade de código** (ter o mesmo trecho de código em lugares diferentes do sistema), o que traz maior **agilidade** e **confiabilidade** na manutenção e expansão do sistema.

Chamamos de **superclasses** essas classes que concentram atributos e métodos comuns que podem ser reutilizados (**herdados**) e de **subclasse** aquelas que reaproveitam (**herdam**) esses recursos.

Exemplo: Vejamos um exemplo, observe as definições das classes **Cliente**, **Fornecedor** e **Funcionário** utilizados no diagrama abaixo.




A classe **Pessoa** contém nove atributos e um método, os quais são comuns para clientes, fornecedores e funcionários e, portanto, deveriam constar nas classes **Cliente**, **Fornecedor** e **Funcionário**. Sem o recurso da herança, teríamos que replicar esses atributos e métodos nas três classes, procedimento desaconselhável em qualquer linguagem de programação, por trazer complexidades extras na manutenção e expansão dos sistemas. Por exemplo, vamos considerar um método para emissão de correspondência que foi atualizado para começar a gerar um histórico de remessas. Tal atualização deveria ser feita nas três classes envolvidas e, caso uma delas não fosse realizada, a atualização do controle de remessas (geração de histórico) ficaria inconsistente.

No modelo acima, a classe **Pessoa** foi definida como uma **superclasse** e as classes **Cliente**, **Fornecedor** e **Funcionário**, como suas **subclasses**. Do ponto de vista da funcionalidade, tudo o que foi definido na **superclasse** (atributos e métodos) será **herdado** pelas suas subclasses. Ou seja, um **objeto instanciado a partir da classe Cliente possui 13 atributos**. São eles: “**nome, endereço, bairro, cidade, estado, telefone e email**” declarados na superclasse **Pessoa**, além de “**limiteCredito, rendaMensal, rg e cpf**”, na subclasse **Cliente**. Há ainda três métodos,

nos quais “**MostrarNome**” foi definido na classe **Pessoa** e “**InserirDadosCliente** e **ApresentarCleinte**” foram definidos na classe **Cliente**. Na utilização desses atributos e métodos para um objeto do tipo **Cliente**, fica transparente o local onde cada um foi declarado ou definido.

Para estabelecer a herança em relação à codificação, as superclasses continuam com a mesma estrutura de uma classe comum. Já as subclasses recebem as seguintes definições:

```
public class Cliente extends Pessoa{...}
```



aqui está a “herança” em comando no Java.

O comando **extends** é o responsável por estabelecer a herança. É inserido na abertura da subclasse e indica o nome da superclasse, criando vínculo entre elas.

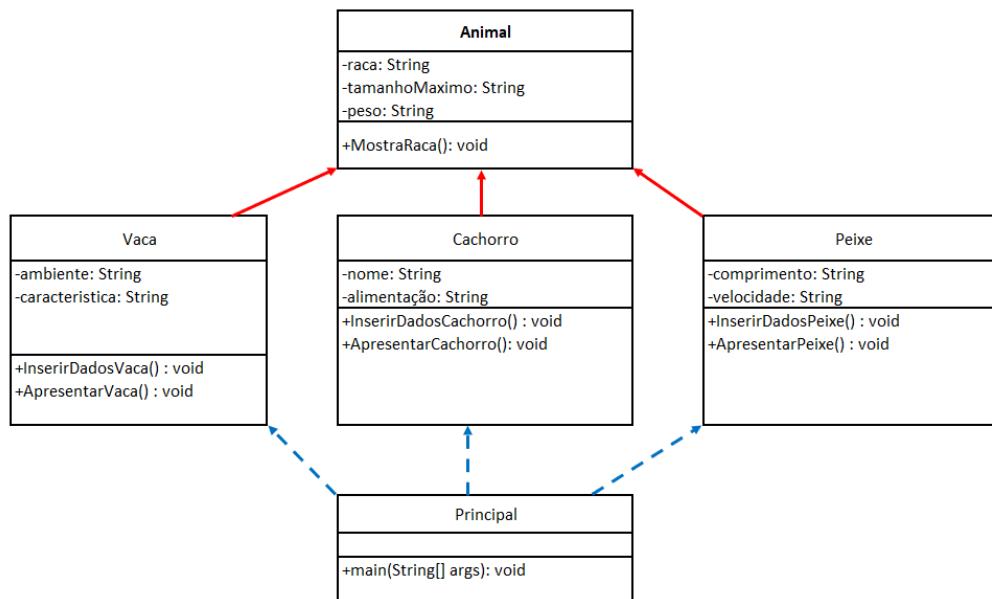
Construtores

Os construtores estão diretamente relacionados à inicialização dos atributos de uma classe. Partindo desse princípio e considerando o nosso exemplo, um objeto do tipo cliente possui todos os atributos declarados na sua **superclasse** (**Pessoa**) mais os declarados na classe **Cliente**. Portanto, o **construtor de uma subclasse deve estar preparado para inicializar os atributos** herdados e os declarados na própria classe.

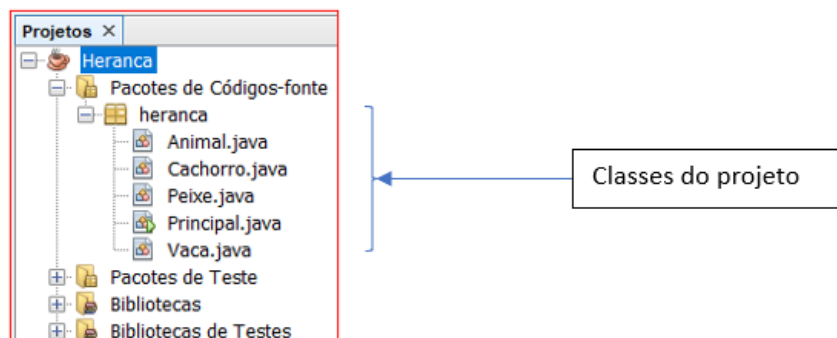
No construtor que recebe parâmetros (**aquele que inicializa os atributos com algum valor**), utilizamos o método **super** () para invocar o construtor da superclasse. Isso porque já foi definida nele a forma como esses atributos serão inicializados (**reutilizando o construtor já existente na superclasse**), restando apenas inicializar os atributos na subclasse.

Para acessar os atributos da **superclasse**, obrigatoriamente, devemos utilizar seus métodos de acesso (**getters e setters**), ao contrário do atributos instanciados na própria classe, que podem ser acessados diretamente. Porém, para garantir o conceito de **encapsulamento** e usufruir de seus benefícios (**segurança, manutenibilidade etc.**), sempre devemos criar e utilizar os métodos **getters** e **setters** para todos os atributos. Em relação ao nosso exemplo, ele se aplica às classes **Fornecedor** e **Funcionário**.

Exemplo-1: Vamos desenvolver o projeto de acordo com o diagrama abaixo.



Estrutura do projeto:



```
package heranca;

public class Animal {
    // declarando os atributos da classe
    private String raca;
    private String tamanhoMaximo;
    private String peso;

    // metodo da classe Animal
    public void MostrarRaca() {
        System.out.println("Raça do animal: " + this.raca);
    }

    // metodos get's e set's da classe
    public String getRaca() {
        return raca;
    }
    public void setRaca(String raca) {
        this.raca = raca;
    }
    public String getTamanhoMaximo() {
        return tamanhoMaximo;
    }
    public void setTamanhoMaximo(String tamanhoMaximo) {
        this.tamanhoMaximo = tamanhoMaximo;
    }
    public String getPeso() {
        return peso;
    }
    public void setPeso(String peso) {
        this.peso = peso;
    }
}
```

```
package heranca;

public class Vaca extends Animal {
    // declarando os atributos da classe
    private String ambiente;
    private String caracteristica;

    // declarando os metodos da classe
    public void InserirDadosVaca() {
        super.setRaca("Nelore");
        super.setPeso("600kg");
        super.setTamanhoMaximo("média 165cm de comprimento e 155cm de altura");
        this.ambiente = "Terra";
        this.caracteristica = "Fêmeas apresentam musculatura menos desenvolvida";
    }
    public void ApresentarVaca() {
        System.out.println("Raça da vaca....: " + super.getRaca());
        System.out.println("Tamanho Maximo..: " + super.getTamanhoMaximo());
        System.out.println("Peso da vaca....: " + super.getPeso());
        System.out.println("Ambiente da vaca: " + this.ambiente);
        System.out.println("Caracteristica..: " + this.caracteristica);
    }
}
```

```
package heranca;

public class Cachorro extends Animal {
    // declarando os atributos da classe
    private String nome;
    private String alimentacao;

    // declarando os metodos da classe
    public void InserirDadosVaca() {
        super.setRaca("Golden");
        super.setPeso("50kg");
        super.setTamanhoMaximo("média 55cm de comprimento e 45cm de altura");
        this.nome = "Peter";
        this.alimentacao = "Ração Premier";
    }
    public void ApresentarVaca() {
        System.out.println("Raça.....: " + super.getRaca());
        System.out.println("Tamanho Maximo..: " + super.getTamanhoMaximo());
        System.out.println("Peso.....: " + super.getPeso());
        System.out.println("Nome.....: " + this.nome);
        System.out.println("Alimentação.....: " + this.alimentacao);
    }
}
```

```
package heranca;

public class Peixe extends Animal {
    // declarando os atributos da classe
    private String comprimento;
    private String velocidade;

    // declarando os metodos da classe
    public void InserirDadosVaca() {
        super.setRaca("Tubarao-branco");
        super.setPeso("2,5 toneladas");
        super.setTamanhoMaximo("8 metros");
        this.comprimento = "em média de 7 metros de comprimento";
        this.velocidade = "50 m/s";
    }
    public void ApresentarVaca() {
        System.out.println("Raça.....: " + super.getRaca());
        System.out.println("Tamanho Maximo...: " + super.getTamanhoMaximo());
        System.out.println("Peso.....: " + super.getPeso());
        System.out.println("Comprimento.....: " + this.comprimento);
        System.out.println("Velocidade.....: " + this.velocidade);
    }
}
```

```
package heranca;

public class Principal {

    public static void main(String[] args) {
        // fazendo as instancias das classes (subclasses)
        Vaca vaca = new Vaca();
        Peixe peixe = new Peixe();
        Cachorro dog = new Cachorro();

        vaca.InserirDadosVaca();
        vaca.ApresentarVaca();
        System.out.println(""); // pula linha em branco

        peixe.InserirDadosVaca();
        peixe.ApresentarVaca();
        System.out.println(""); // pula linha em branco

        dog.InserirDadosVaca();
        dog.ApresentarVaca();
        System.out.println(""); // pula linha em branco
    }
}
```

Resultado:

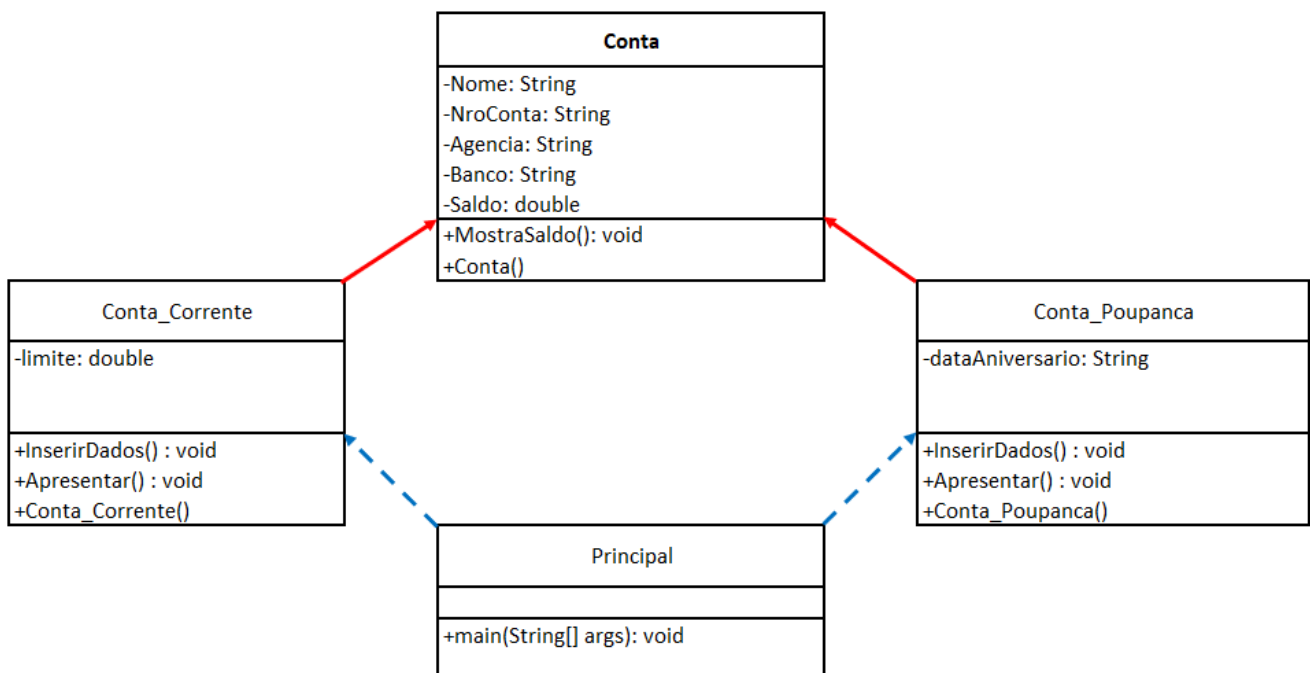
```

Raça da vaca.....: Nelore
Tamanho Maximo...: média 165cm de comprimento e 155cm de altura
Peso da vaca.....: 600kg
Ambiente da vaca: Terra
Caracteristica...: Fêmeas apresentam musculatura menos desenvolvida

Raça.....: Tubarao-branco
Tamanho Maximo...: 8 metros
Peso.....: 2,5 toneladas
Comprimento.....: em média de 7 metros de comprimento
Velocidade.....: 50 m/s

Raça.....: Golden
Tamanho Maximo...: média 55cm de comprimento e 45cm de altura
Peso.....: 50kg
Nome.....: Peter
Alimentação.....: Ração Premier
    
```

Exemplo-2: Vamos desenvolver o projeto de acordo com o diagrama abaixo, lembrando que agora temos o construtor da classe.




```
package herancacomconstrutor;

public class Conta {
    // declarando os atributos da classe
    private String Nome;
    private String NroConta;
    private String Agencia;
    private String Banco;
    private final double Saldo;

    // construtor da classe
    public Conta() {
        this.Nome = "nome do cliente";
        this.NroConta = "nro da conta";
        this.Agencia = "Agência";
        this.Banco = "Banco";
        this.Saldo = 0;
    }

    // metodo da classe
    public void MostrarSaldo() {
        System.out.println("Saldo: R$ " + String.format("%.2f", this.Saldo));
    }

    // metodos get's e set's
    public String getNome() {
        return Nome;
    }
    public void setNome(String Nome) {
        this.Nome = Nome;
    }
    public String getNroConta() {
        return NroConta;
    }
    public void setNroConta(String NroConta) {
        this.NroConta = NroConta;
    }
    public String getAgencia() {
        return Agencia;
    }
    public void setAgencia(String Agencia) {
        this.Agencia = Agencia;
    }
    public String getBanco() {
        return Banco;
    }
    public void setBanco(String Banco) {
        this.Banco = Banco;
    }
}
```

```
package herancacomconstrutor;

import javax.swing.JOptionPane;

public class Conta_Poupanca extends Conta {
    // declarando os atributos da classe
    private String dataAniversario;

    // construtor da classe
    public Conta_Poupanca() {
        super(); // chamando o construtor da superclass
        this.dataAniversario = "15 do mês";
    }

    // metodo da classe
    public void InserirDados() {
        super.setNome("Marcos Antonio");
        super.setBanco("23 - Bradesco");
        super.setAgencia("1363-3");
        super.setNroConta("100.987-0");
    }

    public void Apresentar() {
        String aux = "\nDados da conta Poupança\n\n";

        aux += "Nome: " + super.getNome() + "\n";
        aux += "Banco: " + super.getBanco() + "\n";
        aux += "Agência: " + super.getAgencia() + "\n";
        aux += "Nro da Conta: " + super.getNroConta() + "\n";
        aux += "Limite da Conta: " + this.dataAniversario + "\n";

        JOptionPane.showMessageDialog(null, aux);
    }
}
```

Erro: correto Data Aniversário

```
package herancacomconstrutor;

import javax.swing.JOptionPane;

public class Conta_Corrente extends Conta {
    // declarando os atributos da classe
    private double limite;

    // construtor da classe
    public Conta_Corrente() {
        super(); // chamando o construtor da superclass
        this.limite = 0;
    }

    // metodo da classe
    public void InserirDados() {
        super.setNome("Carlos Eduardo Mattos");
        super.setBanco("001 - Banco do Brasil");
        super.setAgencia("0978-8");
        super.setNroConta("22.545-9");
    }

    public void Apresentar() {
        String aux = "\nDados da Conta Corrente\n\n";

        aux += "Nome: " + super.getNome() + "\n";
        aux += "Banco: " + super.getBanco() + "\n";
        aux += "Agência: " + super.getAgencia() + "\n";
        aux += "Nro da Conta: " + super.getNroConta() + "\n";
        aux += "Limite da Conta: " + this.limite + "\n";

        JOptionPane.showMessageDialog(null, aux);
    }
}
```

```

package herancacomconstrutor;

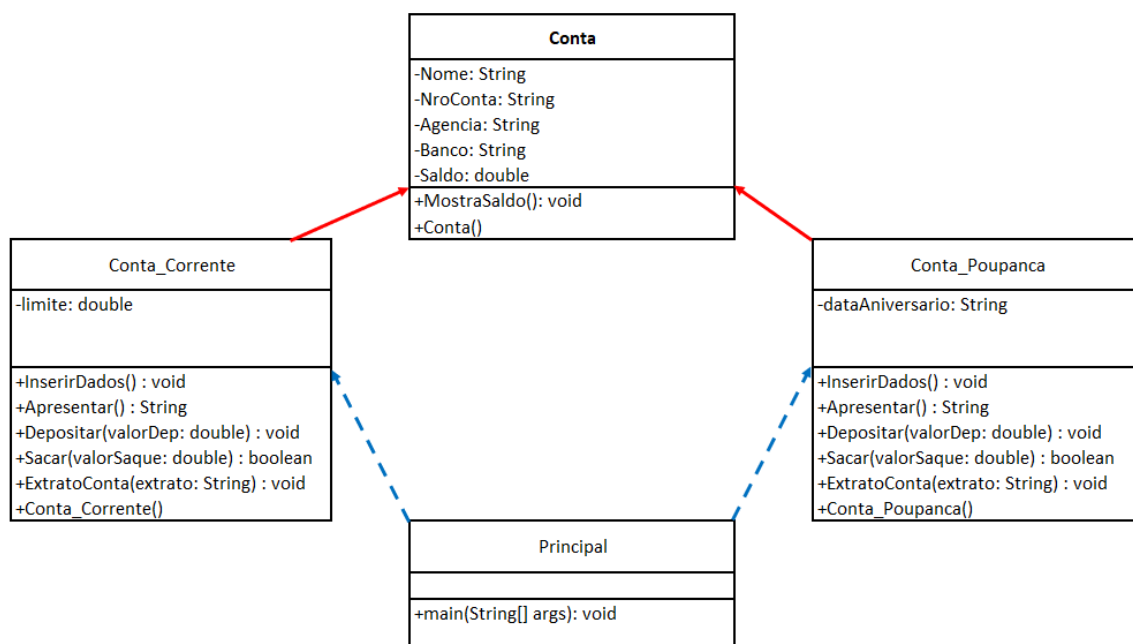
public class Principal {

    public static void main(String[] args) {
        Conta_Corrente conta = new Conta_Corrente();
        Conta_Poupanca poupanca = new Conta_Poupanca();

        conta.InserirDados();
        conta.Apresentar();

        poupanca.InserirDados();
        poupanca.Apresentar();
    }
}
    
```

Exemplo-3: Aproveitando o diagrama acima e modificando alguns itens, faça um projeto em Java que simule 20 transações de depósito e saques com as classes `Conta_Corrente` e `Conta_Poupanca`, ou seja, 10 transações para cada classe.



```
package herancacomconstrutor;

public class Conta {
    // declarando os atributos da classe
    private String Nome;
    private String NroConta;
    private String Agencia;
    private String Banco;
    private double Saldo;

    // construtor da classe
    public Conta() {
        this.Nome = "nome do cliente";
        this.NroConta = "nro da conta";
        this.Agencia = "Agência";
        this.Banco = "Banco";
        this.Saldo = 0;
    }

    // metodo da classe
    public void MostrarSaldo() {
        System.out.println("Saldo: R$ " + String.format("%.2f", this.Saldo));
    }

    // metodos get's e set's
    public String getNome() {
        return Nome;
    }
    public void setNome(String Nome) {
        this.Nome = Nome;
    }
    public String getNroConta() {
        return NroConta;
    }
    public void setNroConta(String NroConta) {
        this.NroConta = NroConta;
    }
    public String getAgencia() {
        return Agencia;
    }
    public void setAgencia(String Agencia) {
        this.Agencia = Agencia;
    }
    public String getBanco() {
        return Banco;
    }
    public void setBanco(String Banco) {
        this.Banco = Banco;
    }
    public double getSaldo() {
        return Saldo;
    }
    public void setSaldo(double Saldo) {
        this.Saldo += Saldo;
    }
}
```



```
package herancacomconstrutor;  
  
import javax.swing.JOptionPane;  
  
public class Conta_Corrente extends Conta {  
    // declarando os atributos da classe  
    // "final" pq não sofre alteracao  
    private final double limite;  
  
    // construtor da classe  
    public Conta_Corrente() {  
        super(); // chamando o construtor da superclass  
        this.limite = 100;  
    }  
  
    // metodo da classe  
    public void InserirDados() {  
        super.setNome("Carlos Eduardo Mattos");  
        super.setBanco("001 - Banco do Brasil");  
        super.setAgencia("0978-8");  
        super.setNroConta("22.545-9");  
    }  
  
    public String Apresentar() {  
        String aux = "\nDados da Conta Corrente\n";  
  
        aux += "Nome: " + super.getNome() + "\n";  
        aux += "Banco: " + super.getBanco() + "\n";  
        aux += "Agência: " + super.getAgencia() + "\n";  
        aux += "Nro da Conta: " + super.getNroConta() + "\n";  
        aux += "Limite da Conta: " + this.limite + "\n";  
        return aux;  
    }  
  
    public void Depositar(double valorDep) {  
        // atribuindo o valorDep ao atributo da classe saldo  
        this.setSaldo(valorDep);  
    }  
  
    public boolean Sacar(double valorSaque) {  
        // buscando o saldo da conta corrente  
        double saldo = super.getSaldo();  
        // total de saldo + limite da conta  
        double total = saldo + this.limite;  
  
        if ( total >= valorSaque ) {  
            return true; // sim, o cliente tem saldo para saque  
        }  
        return false; // não, o cliente não tem saldo para saque  
    }  
  
    public void ExtratoConta(String extrato) {  
        String aux = "";  
  
        aux += Apresentar() + "\n";  
        aux += extrato + "\n";  
        aux += "\nSaldo Final: R$ " + String.format("%.2f", super.getSaldo());  
        JOptionPane.showMessageDialog(null, aux);  
    }  
}
```

```
package heranca.com.construtor;

import javax.swing.JOptionPane;

public class Conta_Poupanca extends Conta {
    // declarando os atributos da classe
    private String dataAniversario;

    // construtor da classe
    public Conta_Poupanca() {
        super(); // chamando o construtor da superclass
        this.dataAniversario = "15 do mês";
    }

    // metodo da classe
    public void InserirDados() {
        super.setNome("Marcos Antonio");
        super.setBanco("23 - Bradesco");
        super.setAgencia("1363-3");
        super.setNroConta("100.987-0");
    }

    public String Apresentar() {
        String aux = "\nDados da conta Poupança\n";

        aux += "Nome: " + super.getNome() + "\n";
        aux += "Banco: " + super.getBanco() + "\n";
        aux += "Agência: " + super.getAgencia() + "\n";
        aux += "Nro da Conta: " + super.getNroConta() + "\n";
        aux += "Data aniversario: " + this.dataAniversario + "\n";
        return aux;
    }

    public void Depositar(double valorDep) {
        // atribuindo o valorDep ao atributo da classe saldo
        this.setSaldo(valorDep);
    }

    public boolean Sacar(double valorSaque) {
        // buscando o saldo da conta corrente
        double saldo = super.getSaldo();
        // total de saldo + limite da conta
        double total = saldo;

        if ( total >= valorSaque ) {
            return true; // sim, o cliente tem saldo para saque
        }
        return false; // não, o cliente não tem saldo para saque
    }

    public void ExtratoConta(String extrato) {
        String aux = "";

        aux += Apresentar() + "\n";
        aux += extrato + "\n";
        aux += "\nSaldo Final: R$ " + String.format("%.2f", super.getSaldo());
        JOptionPane.showMessageDialog(null, aux);
    }
}
```



```
public class Principal {  
  
    public static void main(String[] args) {  
        Conta_Corrente conta = new Conta_Corrente();  
        Conta_Poupanca poupanca = new Conta_Poupanca();  
        Random rand = new Random(); // Classe que gera nros aleatorios  
  
        // inserindo dados da conta poupança e conta corrente  
        poupanca.InserirDados();  
        conta.InserirDados();  
  
        // variaveis primitivas  
        int i;  
        double valor;  
        String transacaoPoup = "\nTransacao Poupanca\n"; // cabeçalho  
        String transacaoConta = "\nTransacao Conta Corrente\n"; // cabeçalho  
        String dados;  
  
        // simular 50 transações utilizando o for  
        for (i = 0; i < 20; i++) {  
            // transacao pares direciona para Conta corrente  
            if (i % 2 == 0) {  
                valor = rand.nextDouble() * 1000; // valor aleatorio  
                poupanca.Depositar(valor); // soma com o saldo  
                transacaoPoup += i + "-Deposito: R$ " + String.format("%.2f", valor) + "\n";  
  
                valor = rand.nextDouble() * 1000; // valor aleatorio  
                valor = valor * -1; // para deixar negativo  
                if (poupanca.Sacar(valor)) { // verifica se pode sacar  
                    poupanca.setSaldo(valor);  
                    transacaoPoup += i + "-Saque: R$ " + String.format("%.2f", valor) + "\n";  
                } else { // não tem saldo  
                    transacaoPoup += i + "-Saldo insuficiente" + "\n";  
                }  
            } else {  
                valor = rand.nextDouble() * 1000; // valor aleatorio  
                conta.Depositar(valor); // soma com o saldo  
                transacaoConta += i + "-Deposito: R$ " + String.format("%.2f", valor) + "\n";  
  
                valor = rand.nextDouble() * 1000; // valor aleatorio  
                valor = valor * -1; // para deixar negativo  
                if (conta.Sacar(valor)) { // verifica se pode sacar  
                    conta.setSaldo(valor);  
                    transacaoConta += i + "-Saque: R$ " + String.format("%.2f", valor) + "\n";  
                } else { // não tem saldo  
                    transacaoConta += i + "-Saldo insuficiente" + "\n";  
                }  
            }  
        }  
  
        dados = poupanca.Apresentar();  
        dados = transacaoPoup;  
        poupanca.ExtratoConta(dados);  
  
        dados = conta.Apresentar();  
        dados = transacaoConta;  
        conta.ExtratoConta(dados);  
    }  
}
```