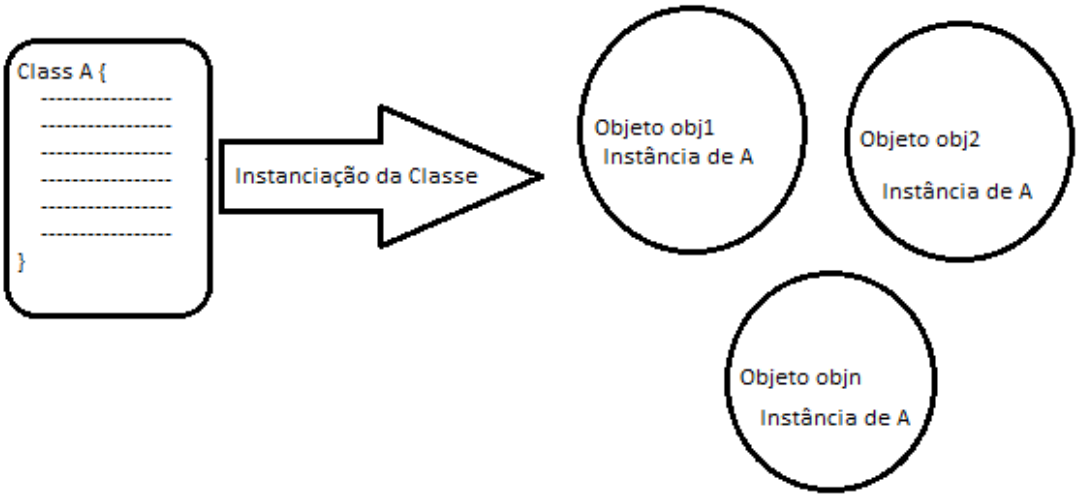
Programação Orientada a Objetos - UML

Na aula anterior, começamos a escrever nossos primeiros objetos Java, tais como `Usuario`, `ContaCorrente` e `Banco`.

Vimos também que objetos possuem atributos e comportamentos. Algo semelhante ao que ocorre na vida real, e que também podemos criar diversas instâncias de objetos a partir de uma classe:



Entretanto, não passamos por um passo importante da programação: a modelagem de software.

Modelagem de Software

Lembra quando estávamos no ensino fundamental, e tivemos que aprender expressões numéricas?
Considere a expressão abaixo:

2 + {5 x 3 - [42 / (2 + 5)] + (5 x 6)}

Segundo o que aprendemos, para resolvermos essa expressão, temos que seguir a seguinte ordem:

- primeiro resolva as contas que estiverem dentro dos parênteses
- depois resolva as contas que estiverem dentro dos colchetes
- por último, resolva as contas que estiverem dentro das chaves

Além disso, tínhamos que resolver as contas de acordo com as operações:

- operações de multiplicação e divisão deveriam ser resolvidas primeiro
- operações de adição e subtração deveriam ser resolvidas depois

E lá íamos nós resolvermos a expressão, com cada linha representando um passo resolvido.
Com o tempo, verificamos que com a prática, era possível "pular" alguns passos, fazendo com que a expressão fosse resolvida mais rapidamente (ou usando menos linhas).

Quando precisamos escrever um software, precisamos passar primeiramente pela modelagem do mesmo, desenhando todas as classes que imaginamos que o sistema deve ter. E só depois escrevemos o código de fato.

> *Com o tempo, pode ser que alguns problemas possam ser resolvidos de cabeça, sem a necessidade de modelagem desenhada. De qualquer forma, o passo de modelagem é importante, principalmente quando estamos aprendendo a programar.*

É difícil construirmos um carro sem que antes algum desenho tenha sido feito, com suas medidas e demais especificações. Mesmo os cozinheiros precisam de algum guia para prepararem seus pratos (as receitas). E com a construção de software não é diferente.

Mas como modelar? De que jeito?

A Unified Modeling Language (UML)

O Paradigma OO existe desde a década de 60. E desde essa época buscou-se entender quais seriam as melhores ferramentas para modelar um software segundo esse paradigma.

Ao longo dos anos surgiram diversas notações, como a OOSE (Object-Oriented Software Engineering, de Ivar Jacobson), a OMT (Object Modeling Technique, de James Rumbaugh) e a de Booch, de Grady Booch.

Grady Booch, por sinal, é um dos programadores mais ativos na internet, até hoje.

Cada uma das notações possuía a sua maneira de entender o software orientado a objetos. Tinha, portanto, seus documentos e diagramas. E também pontos fortes e pontos fracos. Abaixo, um resumo sobre elas:

Notação	Pontos fortes	Pontos fracos
OMT	Análise	Projeto
Booch	Projeto	Análise
OOSE	Comportamento	Todas as demais áreas

As empresas podiam escolher entre essas notações já consagradas, ou criar uma própria. Entretanto, isso causava diversos problemas:

- como comunicar aos desenvolvedores novos a notação utilizada?
- caso a notação fosse mudada, como transmitir o conhecimento para a nova notação?
- teríamos que escolher qual notação usar de acordo com o projeto a ser modelado?

Percebendo esses e outros problemas, os autores acima citados se reuniram e criaram a Linguagem de Modelagem Unificada (ou Unified Modeling Language - UML) trazendo as boas ideias de cada uma das notações.

Estrutura da UML

A UML é composta de diversos diagramas e documentos. Essas ferramentas não precisam, no entanto, serem utilizadas na sua totalidade, mas apenas aquelas que você, como profissional, entende que são úteis para o momento.

Imagine um marceneiro que precise construir uma casa de cachorros. Se ele possui todas as tábuas nas medidas certas, ele provavelmente não precisará do serrote, mas com certeza precisará de martelo e alguns pregos.

De forma resumida, iremos estudar 2 dos principais diagramas da UML, o **diagrama de classes** e o **diagrama de sequência**.

A relação completa de diagramas e documentos da UML pode ser vista com detalhes no livro ***"UML - Guia do Usuário, de Grady Booch, James Rumbaugh e Ivar Jacobson"***

O Diagrama de Classes

Vamos rever a classe `Usuario`, vista na aula anterior:

```
package primeiros.objetos;

class Usuario {

	int idade;
	double salario;
	char sexo;
	boolean casado;
	String nome;

	Usuario(int idade, double salario, char sexo, boolean casado,
String nome){
		this.idade = idade;
		this.salario = salario;
		this.sexo = sexo;
		this.casado = casado;
		this.nome = nome;
	}

	void exibir() {
		System.out.println("As informações da pessoa são:\n " +
				"Nome: " + nome
				+ ", Idade: " + idade
				+ ", Salário: " + salario
				+ ", Sexo: " + sexo
				+ ", Casado: " + casado);
	}
}
```

A estrutura básica dessa classe (e da maioria das classes) é composta de 4 partes:

- a classe em si
- atributos
- construtor
- comportamentos (representados pelos métodos)

A classe Usuario, portanto, pode ser representada usando o diagrama de classes da seguinte forma:

Curiosamente, o construtor não é representado no diagrama de classes.

É possível encontrar o nome da classe logo no topo do retângulo. Em seguida, encontramos os atributos e, por último, os comportamentos (métodos) da classe.

Ao lado dos atributos e dos métodos, encontramos o símbolo # . Esse será melhor explicado ao longo do curso.

Agora, vamos rever as classes `ContaCorrente` e `Banco`.

*As classes acima serão copiadas para o novo projeto **Aula5-UML-Associacoes**. Também será criado o pacote `banco.fiap` nesse novo projeto.*

```
package banco.fiap;

public class ContaCorrente {

	double saldo;

	ContaCorrente(double saldo){
		this.saldo = saldo;
	}

	void saque(double valor){
		saldo = saldo - valor;
	}

	void deposito(double valor) {
		saldo = saldo + valor;
	}
```

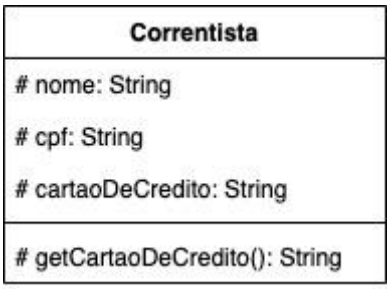
```
	double saldo() {
		return saldo;
	}
}
```

```
package banco.fiap;

public class Banco {

	public static void main(String[] args) {
		ContaCorrente umaConta = new ContaCorrente(1000);
		ContaCorrente outraConta = new ContaCorrente(2000);

		umaConta.saque(100);
		outraConta.deposito(200);

		System.out.println("Saldo de umaConta: " +
umaConta.saldo());
		System.out.println("Saldo de outraConta: " +
outraConta.saldo());
	}

}
```

Além de uma ContaCorrente, é importante que o Banco conheça seus correntistas. E, para o nosso banco, é importante que um correntista tenha:

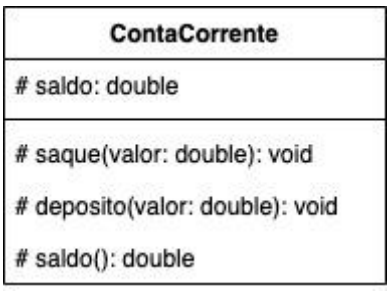
- um nome
- um cpf
- um cartão de crédito
- sua conta corrente
- e que informe seu cartão de crédito quando for necessário.

Se precisarmos modelar o `Correntista`, precisaremos desenhar algo como:

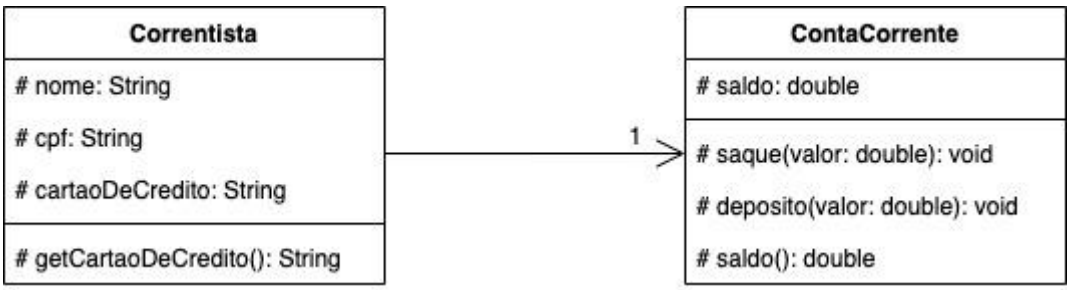


Note que o diagrama não contém a informação de `ContaCorrente`.

Por se tratar de uma classe do nosso modelo, podemos criar uma **associação** entre as classes.
Inicialmente, vamos modelar a classe `ContaCorrente`:



Agora, fazendo a associação entre as classes:



Notem a seta saindo da classe `Correntista` para a classe `ContaCorrente`.
Há também um número "1" do lado direito da associação, indicando que um `Correntista` pode ter 1 `ContaCorrente`. Esse número é chamado de **cardinalidade da associação**.

Caso entendêssemos que um `Correntista` pudesse ter mais de uma `ContaCorrente`, teríamos que colocar a cardinalidade como n.

> *A associação entre `Banco` e `ContaCorrente` é uma relação de 1 pra n. Ou seja, temos 1 banco e n contas corrente.*

A classe `Correntista` deverá então ser escrita da seguinte forma:

```
package banco.fiap;

public class Correntista {

	String nome;
	String cpf;
	String cartaoDeCredito;
```

```
	ContaCorrente contaCorrente;

	Correntista(String nome, String cpf, String cartaoDeCredito,
ContaCorrente contaCorrente){
		this.nome = nome;
		this.cpf = cpf;
		this.cartaoDeCredito = cartaoDeCredito;
		this.contaCorrente = contaCorrente;
	}

	String getCartaoDeCredito() {
		return cartaoDeCredito;
	}
}
```

—----------

Exercícios

5.1 - Desenhe a associação entre `Banco` e `ContaCorrente` usando UML.

5.2 - Desenhe o diagrama de classes de uma livraria online. Essa livraria deve lidar com conceitos como `Pedido` e `ItemPedido`. Um pedido pode ter um ou mais itens de pedido. Um `ItemPedido` contém o nome do produto e o seu valor. Já um `Pedido` contém o valor total de todos os `ItemPedido`. E tanto o `Pedido` quanto o `ItemPedido` devem ter um método `valor()` que não recebe nenhum parâmetro e deverá retornar o respectivo valor.

5.3 - Com base na modelagem realizada no exercício anterior, escreva as classes modeladas acima, usando a linguagem Java. Escreva também uma classe chamada `Main`, que deve conter o método `main`. Dentro desse método, crie alguns objetos da classe `ItemPedido` e instancie um objeto da classe `Pedido` passando os objetos da classe `ItemPedido` criados anteriormente, via construtor da classe `Pedido`. Ao final, o programa deve imprimir o valor total do pedido, com base nos valores de cada `ItemPedido`.