**1. Explique e discuta os três usos possíveis de UML:**

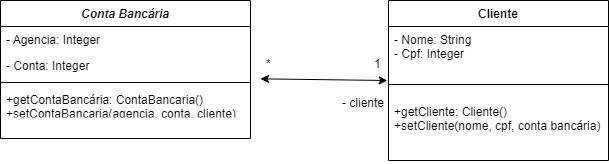
1. **Como blueprint (ou plantas técnicas detalhadas):** O uso de UML como blueprint é feito de maneira considerável em projetos do tipo waterfall ou UP, uma vez que suas fases são encadeadas e dependentes por possuírem um escopo muito bem definido. Assim, ao utilizar ferramentas CASE, o UML do tipo blueprint considera todas as etapas de um software, detalhando-as minunciosamente para o time de desenvolvimento logo após o levantamento de requisito para que então o código possa ser elaborado.
2. **Como sketches (esboços)**: UML como sketches surgiu basicamente como uma proposta ao blueprint, utilizado sobretudo em metodologias ágeis, já que consiste não em uma abstração do software como um todo, mas de suas frações. Esse tipo de UML possui um caráter não formal e é utilizado para comunicação entre desenvolvedores em dois possíveis momentos: a Engenharia Avante (Foward Engineering), que consiste numa discussão antes da elaboração do código das alternativas de design do requisito, e a Engenharia Reversa (Reverse Engineering), que consiste no uso de modelos UML para explicação de um código já existente. Como em ambos os casos o intuito não é gerar modelos completos e detalhados, ferramentas CASE não são utilizadas de forma massiva. Quadros, papéis e fluxos menos complexos são, por sua vez, explorados.
3. **Como linguagem de programação:**  UML como linguagem de programação, também conhecida como Model Driven Development (MDD) surgiu com a proposta de geração dos códigos do software a partir do próprio UML, utilizando a compilação de seus modelos. Para tanto, o UML precisou ganhar novos recursos e diagramas, mas por suas ferramentas serem muito “pesadas”, acaba sendo pouco utilizado.

**2. Descreva cenários de uso de diagramas de classes UML como instrumento dos seguintes tipos de engenharia:**

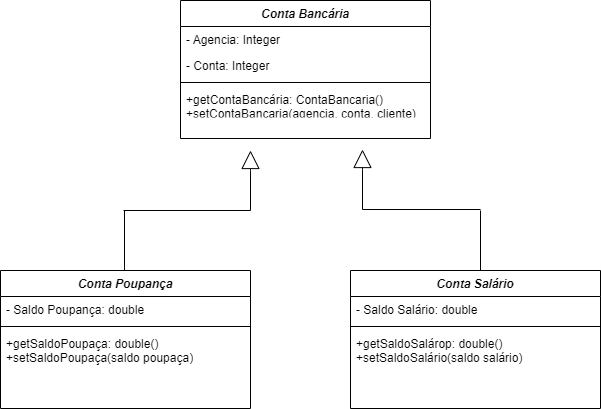
1. **Engenharia Reversa**: UML como instrumento de engenharia reversa é utilizado para explicar, seja para outros desenvolvedores, seja para pessoas sem um domínio técnico de desenvolvimento, como determinado código funciona.
2. **Engenharia Avante (*Forward Engineering*)**: UML como instrumento de engenharia avante consiste na discussão prévia ao desenvolvimento dos modelos de design possíveis para determinado requisito, levantando, por exemplo, as classes que serão necessárias e as relações estabelecidas entre e por elas.

**3. Modele os cenários descritos a seguir usando Diagramas de Classe UML. Veja que as classes são grafadas em uma fonte diferente.**

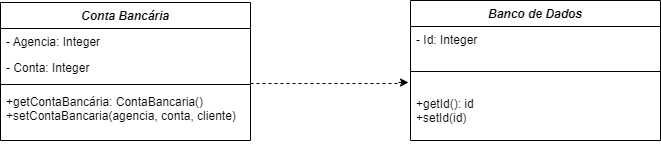
1. ContaBancaria possui exatamente um Cliente. Um Cliente, por sua vez, pode ter várias ContaBancaria. Existe navegabilidade em ambos os sentidos.



1. ContaPoupanca e ContaSalario são subclasses de ContaBancaria.



1. No código de ContaBancaria declara-se uma variável local do tipo BancoDados.

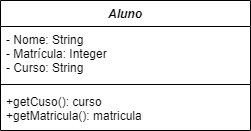


1. Um ItemPedido se refere a um único Produto (sem navegabilidade). Um Produto pode ter vários ItemPedido (com navegabilidade).

A diagram of a data flow

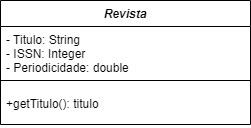
Description automatically generated

1. A classe Aluno possui atributos nome, matricula, curso (todos privados); e métodos getCurso() e cancelaMatricula(), ambos públicos.

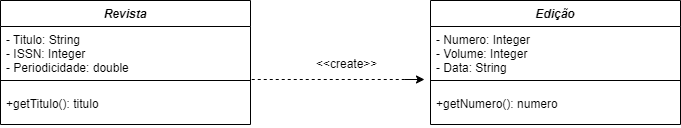


**4. (ENADE 2014, Tec. e Análise de Sistemas) Construa um diagrama de classes para representar as seguintes classes e associações:**

* Uma revista científica possui título, ISSN e periodicidade;



* Essa revista publica diversas edições com os seguintes atributos: número da edição, volume da edição e data da edição. Importante destacar que cada instância da classe edição relaciona-se única e exclusivamente a uma instância da classe revista científica, não podendo relacionar-se com nenhuma outra;



* Um artigo possui título e nome do autor. Um artigo é um conteúdo exclusivo de uma edição. E uma edição obrigatoriamente tem que possuir no mínimo 10 e no máximo 15 artigos.

A close-up of a pen

Description automatically generated

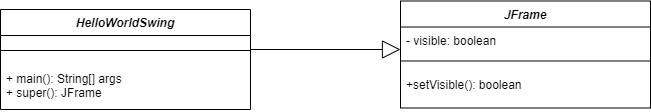
Obs: Uma edição deve ter no mínimo 10 e no máximo 15 artigos

**5. Crie diagramas de classes para os seguintes trechos de código:**

public class HelloWorldSwing {    
   
public static void main(String[] args) {   
JFrame frame = new JFrame("Hello world!");   
frame.setVisible(true);   
}   
}

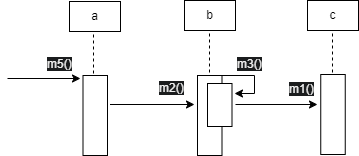


class HelloWorldSwing extends JFrame {   
   
public HelloWorldSwing() {   
super("Hello world!");   
}   
   
public static void main(String[] args) {   
HelloWorldSwing frame = new HelloWorldSwing();   
frame.setVisible(true);   
}   
}



**6.** **Mostre o diagrama de sequência relativo ao seguinte código. O diagrama deve começar com a seguinte chamada *a*.*m5()*.**

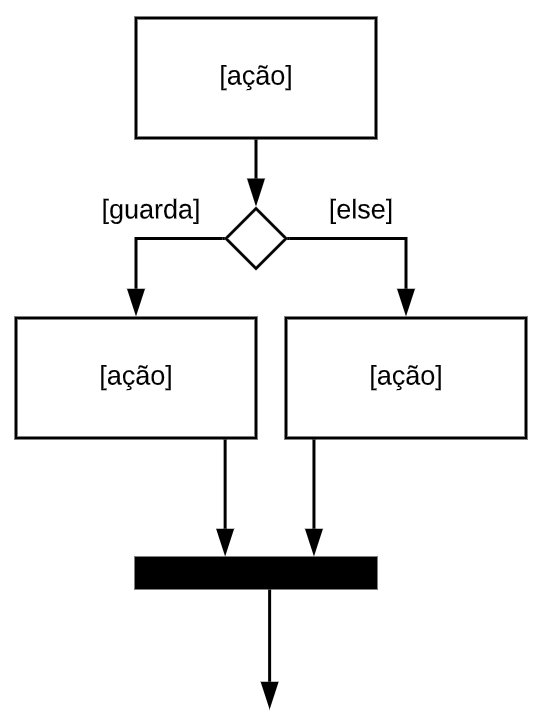
A a = new A(); // variáveis globais   
B b = new B();   
C c = new C();   
   
class C {    
void m1() { ... }    
}   
   
class B {    
void m2() { ... c.m1(); ... this.m3(); ... }   
void m3() { ... c.m1(); ... }   
void m4() { ... }   
}   
   
class A {    
void m5() { ... b.m2(); ... b.m3(); ... b.m4(); ... }   
}



**7.** **Em diagramas de atividades, explique a diferença entre um nodo de *merge* e um nodo de *join*.**

A principal diferença entre o nó de merge e o nó de join está na maneira como eles lidam com os fluxos de execução, pois o primeiro simplesmente combina os fluxos de execução à medida que eles chegam, enquanto o segundo espera até que todos os fluxos de entrada estejam disponíveis antes de prosseguir com a execução, transformando vários fluxos de execução em apenas um.

**8. Qual é o erro do seguinte diagrama de atividades? Refaça o diagrama de forma a refletir corretamente a intenção do projetista.**



Proposto pelo projetista

O erro está acontecendo pois após o step de decisão não existe um nó de merge para capturar as múltiplas respostas e direcionar para apenas uma saída. Nesse caso de decisão, nós do tipo *joins* não são adequados. O correto seria um merge e um fork, podendo ser seguido da finalização da aplicação:

