UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

Disciplina: Projeto e Engenharia de Software
Professor: Eduardo de Lucena Falcão
Aluno: Thiago Theiry de Oliveira
Tópico: Modelagem de Sistemas e Padrões de Projetos

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

   a. **Como blueprint (ou plantas técnicas detalhadas)**

R= Nessa forma de uso, defende-se que, após o levantamento de requisitos, seja produzido um conjunto de modelos ou plantas técnicas (blueprints), documentando diversos aspectos de um sistema e sempre usando diagramas UML.

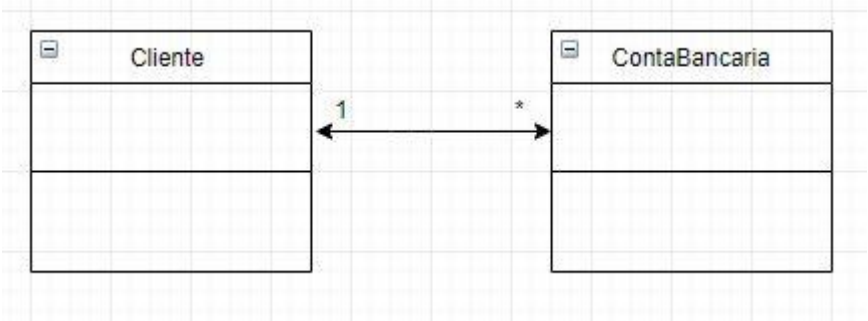
   b. **Como sketches (esboços**)

R= Nela, usamos UML para construir diagramas leves e informais de partes de um sistema, vindo daí o nome esboço (sketch). Esses diagramas são usados para comunicação entre os desenvolvedores, em duas situações principais: Engenharia Avante e Engenharia Reversa.

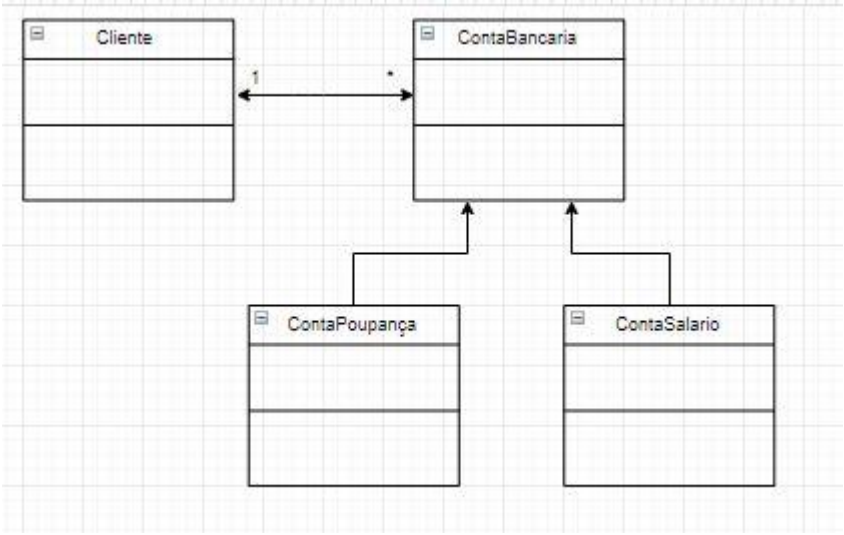
   c. **Como linguagem de programação.**

R= corresponde ao uso de UML vislumbrado pela OMG, após a padronização da linguagem de modelagem. De forma ambiciosa e pelo menos durante um período, vislumbrou-se a geração de código automaticamente a partir de modelos UML. Em outras palavras, não haveria mais uma fase de codificação, pois o código seria gerado diretamente a partir da compilação de modelos UML

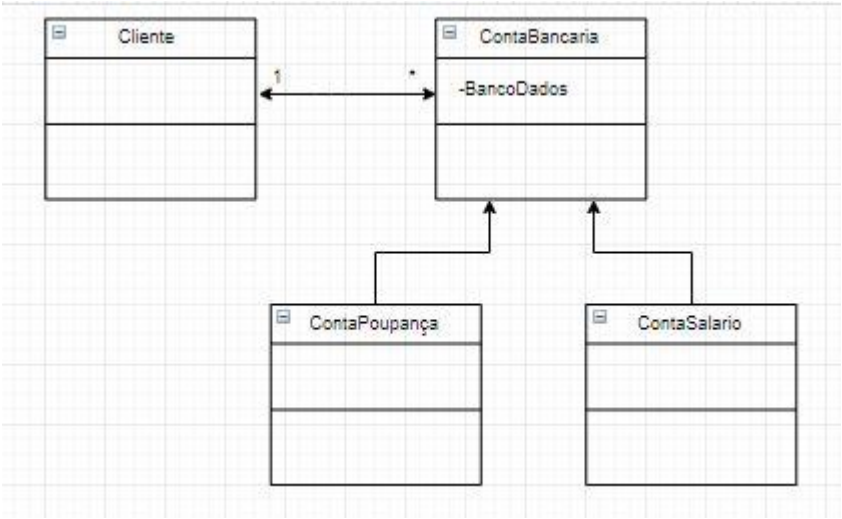
2. **Modele os cenários descritos a seguir usando Diagramas de Classe UML. Veja que as classes são grafadas em uma fonte diferente**.
   a. **ContaBancaria possui exatamente um Cliente. Um Cliente, por sua vez, pode ter várias ContaBancaria. Existe navegabilidade em ambos os sentidos.**



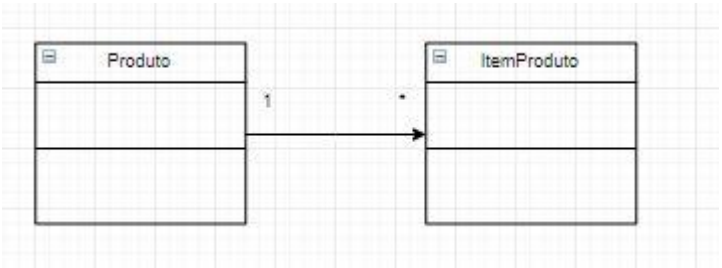
   b. **ContaPoupanca e ContaSalario são subclasses de ContaBancaria**.



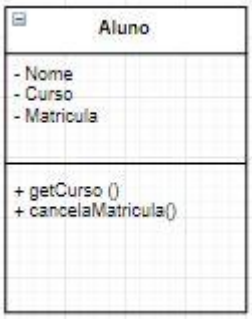
c. No código de ContaBancaria declara-se uma variável local do tipo BancoDados.



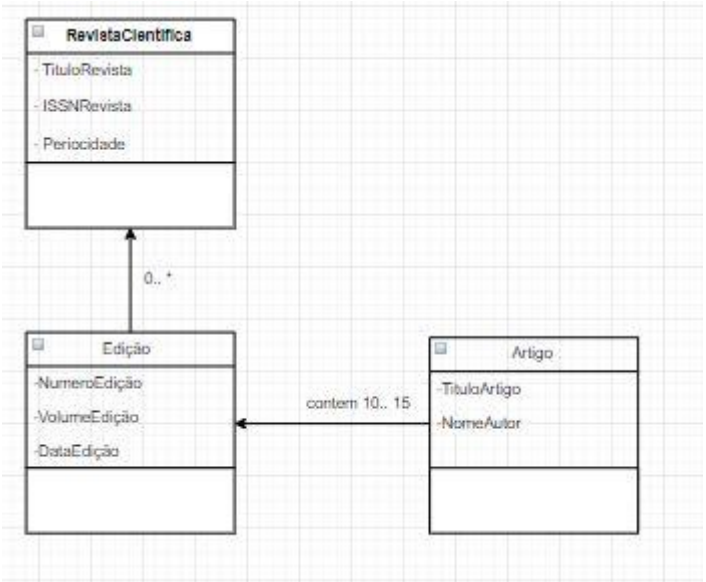
d. Um ItemPedido se refere a um único Produto (sem navegabilidade). Um Produto pode ter vários ItemPedido (com navegabilidade).



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

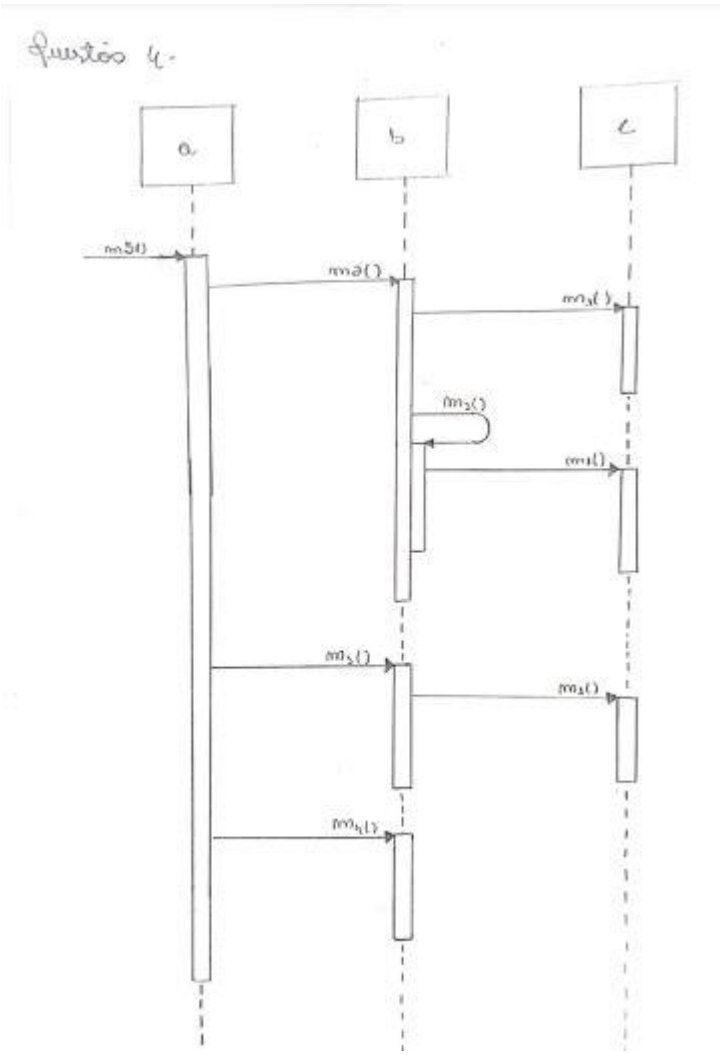


3. **(ENADE 2014, Tec. e Análise de Sistemas) Construa um diagrama de classes para representar as seguintes classes e associações:**
   a. **Uma revista científica possui título, ISSN e periodicidade;**
   b. **Essa revista publica diversas edições com os seguintes atributos: número da edição, volume 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;**
   c. **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.**

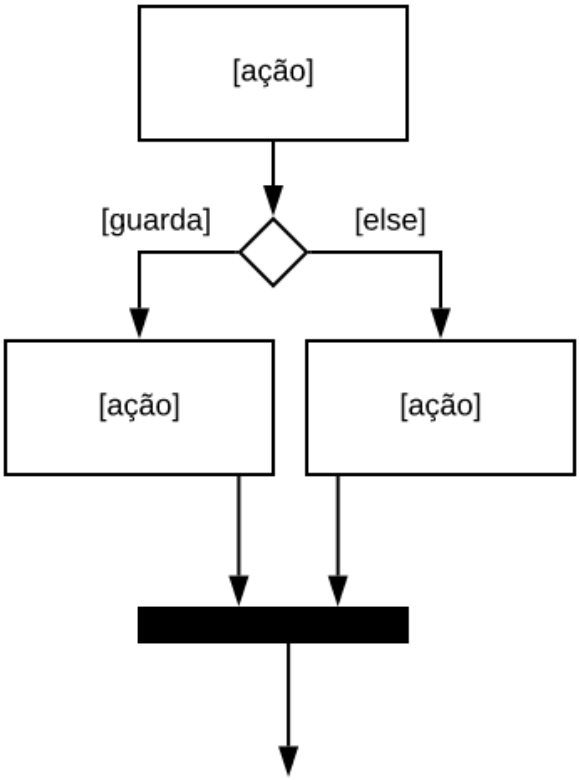


4. **Mostre o diagrama de sequência relativo ao seguinte código. O diagrama deve começar com a seguinte chamada a.m5().**

```
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(); ...  }
}
```



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



R = no nodo merge há fluxos convergentes para um único ponto e existe apenas uma saída, o que difere do nodo join, onde vários fluxos chegam concorrentemente . ou seja, o Merge é um nó de controle que reúne vários fluxos alternativos de entrada para aceitar o fluxo de saída único. Não há junção de tokens. O merge não deve ser usado para sincronizar fluxos simultâneos, um merge reúne vários fluxos sem sincronização. Enquanto o Join é um nó de controle que possui várias arestas de entrada e uma aresta de saída e é usado para sincronizar fluxos simultâneos de entrada. Os Joins são introduzidos para suportar o paralelismo nas atividades

6. **(ENADE 2011, adaptado) Sobre padrões de projeto, assinale V ou F.**

   (F) **Prototype é um tipo de padrão estrutural**.

   (F) **Singleton tem por objetivo garantir que uma classe tenha ao menos uma instância e fornecer um ponto global de acesso para ela.**

   (V) **Template Method tem por objetivo definir o esqueleto de um algoritmo em uma operação, postergando a definição de alguns passos para subclasses.**

(V) **Iterator fornece uma maneira de acessar sequencialmente os elementos de um objeto agregado sem expor sua representação subjacente.**

7. **Dê o nome dos seguintes padrões de projeto:**
   a. **Oferece uma interface unificada e de alto nível que torna mais fácil o uso de um sistema.**
      R= Façade
   b. **Garante que uma classe possui, no máximo, uma instância e oferece um ponto único de acesso a ela.**
      R= Singleton
   c. **Facilita a construção de objetos complexos com vários atributos, sendo alguns deles opcionais.**
      R= Builder
   d. **Converte a interface de uma classe para outra interface esperada pelos clientes. Permite que classes trabalhem juntas, o que não seria possível devido à incompatibilidade de suas interfaces.**
      R= Adapter
   e. **Oferece uma interface ou classe abstrata para criação de uma família de objetos relacionados.**
      R= Abstract Factory
   f. **Oferece um método para centralizar a criação de um tipo de objeto.**
      R= Abstract Factory
   g. **Funciona como um intermediário que controla o acesso a um objeto base**.
      R= Mediator
   h. **Permite adicionar dinamicamente novas funcionalidades a uma classe.**
      R= Decorator
   i. **Oferece uma interface padronizada para caminhar em estruturas de dados.**
      R= Iterator
   j. **Permite parametrizar os algoritmos usados por uma classe.**
      R= Strategy
   k. **Torna uma estrutura de dados aberta a extensões, isto é, permite adicionar uma função em cada elemento de uma estrutura de dados, mas sem alterar o código de tais elementos**.
      R = Visitor
   l. **Permite que um objeto avise outros objetos de que seu estado mudou.**
      R= Observer
   m. **Define o esqueleto de um algoritmo em uma classe base e delega a implementação de alguns passos para subclasses.**
      R= Template Method

8. **Dado o código abaixo de uma classe Subject (do padrão Observador):**

```
interface Observer {
  public void update(Subject s);
}

class Subject {

  private List<Observer> observers=new ArrayList<Observer>();

  public void addObserver(Observer observer) {
    observers.add(observer);
  }

  public void notifyObservers() {
    (A)
  }

}
```

Implemente o código de notifyObservers, comentado com um (A) acima.

R= não feita.

9. **Em uma entrevista dada ao site InformIT, em 2009, por ocasião dos 15 anos do lançamento da primeira edição do GoF, três dos autores do livro mencionaram que — se tivessem que lançar uma segunda edição do trabalho — provavelmente manteriam os padrões originais e incluiriam alguns novos, que se tornaram comuns desde o lançamento da primeira edição, em 1994. Um dos novos padrões que eles mencionaram na entrevista é chamado de Null Object. Estude e explique o funcionamento e os benefícios desse padrão de projeto. Para isso, você vai encontrar diversos artigos na Web. Mas se preferir consultar um livro, uma boa referência é o Capítulo 25 do livro Agile Principles, Patterns, and Practices in C#, de Robert C. Martin e Micah Martin. Ou então o refactoring chamado Introduce Null Object do livro de Refactoring de Martin Fowler.**

R= O padrão Object Null é um objeto que encapsula a ausência de um objeto. Ele fornece o comportamento para não fazer nada e retornar um valor padrão. Esse padrão de projeto é usado sempre que a referência a um objeto pode ser nula. Um dos seus benefícios é que o uso do padrão Object Null simplifica o código e o torna menos propenso a erros. (Este padrão permite a criação de um objeto que é utilizado para substituir a lógica de verificação de nulos), o código do cliente é simplificado e consequentemente, menores condicionais requerem menos casos de teste.