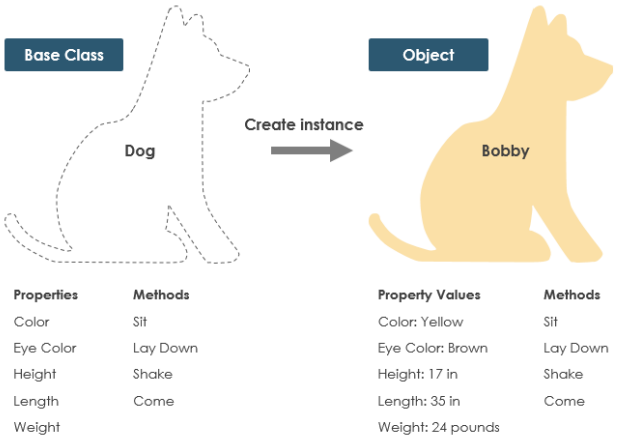
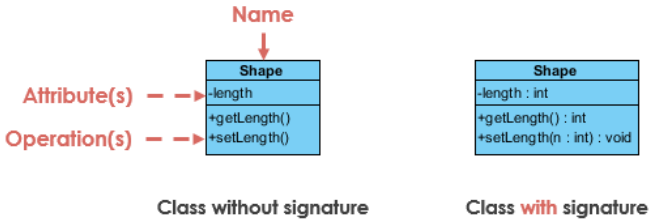
O diagrama de classes UML é uma notação gráfica usada para construir e visualizar sistemas orientados a objetos. Um diagrama de classes na Unified Modeling Language (UML) é um tipo de diagrama de estrutura estática que descreve a estrutura de um sistema mostrando:

- **Classes;**
- **Atributos;**
- **Operações (ou métodos);**
- **Relações entre os objetos;**

OBS: As classes descrevem o tipo de objeto, enquanto os objetos são instâncias utilizáveis de classes, e possuem estados e comportamentos.



Uma classe representa um conceito que encapsula estado (atributos) e comportamento (operações). Cada atributo tem um tipo. Cada operação tem um valor. O nome da classe é a única informação obrigatória.



Operações de Classe (Métodos):

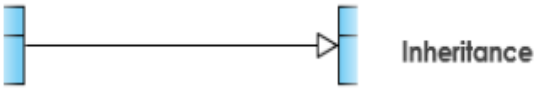
- São serviços que a classe oferece.

Relacionamentos (Associações)

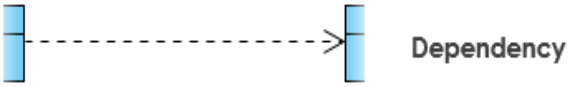
Uma classe pode estar envolvida em um ou mais relacionamentos com outras classes. Um relacionamento pode ser um dos seguintes tipos:

- **Associação:** é representada por uma linha sólida entre as classes. As associações são normalmente nomeadas usando um verbo ou frase verbal.

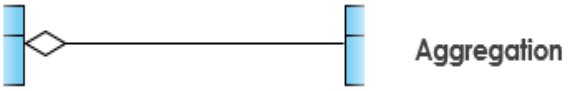
- **Herança (ou generalização):** é uma relação entre uma classe geral e uma mais específica. A classe específica herda as características do classificador mais geral. O relacionamento é exibido como uma linha sólida com uma ponta de seta oca que aponta do elemento filho para o elemento pai.



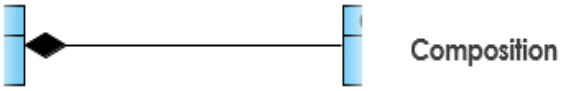
- **Dependência:** uma classe pode precisar do objeto de outra classe. Exemplo: carros e oficina, o carro alguma vez precisará da oficina para manutenções.



- **Agregação:** uma classe é parte de outra; uma pode existir sem outra. Exemplo: casa possui cadeira, mas casa pode existir sem cadeira.



- **Composição:** uma classe compõe a outra, elas têm tempo de vida iguais, isso significa que uma não pode existir sem outra. Exemplo: corpo e coração, carro e motor.



Visibilidade

Define se atributos e operações de classes específicas podem ser vistos e utilizados por outras classes. Exemplo: os atributos e operações em uma classe com visibilidade pública podem ser vistos e utilizados por outras classes, enquanto os atributos e operações com visibilidade privada podem ser vistos e utilizados somente pela classe que os contém.

- **+** público (qualquer classe, de qualquer container pode ver)
- **-** privado (classes no mesmo container)
- **~** package (classes no mesmo pacote do container)
- **#** protegido (classes do mesmo container ou descentes do container)

Multiplicidades

Informa a quantidade de instâncias de objetos que uma classe pode ter em relação a outra classe.

0..1 = no máximo um;
1..1 = um e somente um;
0..* = muitos;
1..* = um ou muitos;

Tópicos Importantes: projeto pode ser **conduzido por dados ou por responsabilidade**: identifica todos os dados/responsabilidades do sistema e verifica se estão cobertos por uma coleção de objetos das classes do sistema. **Passos para criação do diagrama**: identificar frases nominais, elimine candidatos inapropriados, identifique os verbos (indicam métodos e operações), valide o modelo. **Erros comuns**: Análise excessiva da frase nominal, nomes de classe e associação incompreensíveis, atribuição de multiplicidades às associações prematuramente, abordagem das questões de implementação antes do tempo, comprometimento com as construções de implementação.

Unidade 4 – Projetar a solução

A arquitetura de software refere-se às estruturas de alto nível de um sistema de software, à disciplina de criação de tais estruturas e à documentação dessas estruturas.

Características da arquitetura de software:

- **Atender stakeholders**: os sistemas de software devem atender a uma variedade de artes interessadas.
- **Separação de necessidades:** para diminuir a complexidade; descrições separadas são chamadas de vistas arquitetônicas.

- **Orientado pela qualidade:** atributos de qualidade, como tolerância a falhas, compatibilidade com versões anteriores, confiabilidade, manutenção, disponibilidade, segurança, usabilidade.
- **Estilos recorrentes**: maneiras-padrão de abordar as necessidades recorrentes.
- **Integridade**: uma visão geral do sistema do que deve fazer e como deve fazê-lo.

Atividades de definição de arquitetura:

Auxiliam um arquiteto de software a realizar análise, síntese, avaliação e evolução.

- **Gestão do conhecimento e comunicação:** é a atividade de explorar e gerenciar o conhecimento essencial para projetar uma arquitetura de software.
- **Raciocínio de projetos e a tomada de decisão:** reúne e associa contextos de decisão, formula problemas de decisão de projeto, encontrar opções de solução e avalia trocas antes de tomar decisões.
- **Documentação:** descrição do projeto gerado durante o processo de arquitetura de software.

Padrões da arquitetura de software:

Blackboard, Cliente-servidor, Peer-to-peer, Plug-ins, REST...

Projeto de software: do problema à arquitetura

Atividade que ocorre entre a etapa de especificação de requisitos e a programação. **Projeto estrutural**: versão abstrata em alto nível do sistema; representa o software como um sistema formado por múltiplos componentes que interagem entre si. Nesse nível, os projetistas obtêm a ideia do domínio da solução proposta. **Projeto de alto nível**: visão menos abstrata de subsistemas e módulos. **Projeto detalhado**: trata da parte de implementação do que é visto como um sistema e de seus subsistemas nos dois projetos anteriores; mais detalhado para módulos e suas implementações; define a estrutura lógica de cada módulo e suas interfaces para se comunicar com outros módulos.

Modularização

É uma técnica para dividir um sistema de software em múltiplos módulos independentes e discretos, em que se espera que sejam capazes de executar tarefas independentemente. Vantagens: pequenos componentes mais fáceis de manter, maior segurança, podem ser reutilizados, possibilidade de execução simultânea.

Concorrência

2 tipos de execução de software: sequencial e paralela (ou concorrente). A concorrência é implementada dividindo o software em múltiplas unidades de execução independentes, como módulos e executando-as em paralelo.

Acoplamento e coesão

Medidas pelas quais a qualidade de um projeto de módulos e sua interação entre eles é medida.

Coesão: medida que define o grau de acoplamento dentro dos elementos de um módulo.

- **Coesão coincidental**: coesão não planejada e aleatória que pode ser o resultado de quebrar o programa em módulos menores por motivos de modularização; não é aceita.
- **Coesão lógica**: quando elementos logicamente categorizados são agrupados em um módulo.
- **Coesão temporal**: elementos do módulo são organizados de forma que são processados em um ponto similar.
- **Coesão processual**: elementos do módulo são agrupados e executados sequencialmente.
- **Coesão comunicativa**: elementos do módulo são agrupados, executados sequencialmente e funcionam nos mesmos dados (informação).
- **Coesão sequencial**: elementos do módulo são agrupados porque a saída de um elemento serve como entrada para outro.
- **Coesão funcional**: maior grau de coesão; elementos do módulo em coesão funcional são agrupados, porque todos contribuem para uma única função bem definida, também pode ser reutilizado.

Acoplamento: medida que define o nível de interconfiabilidade entre os módulos de um programa; quanto menor for o acoplamento, melhor será o programa

- **Acoplamento de conteúdo**: módulo pode acessar ou modificar diretamente ou se referir ao conteúdo de outro módulo.
- **Acoplamento comum**: vários módulos têm acesso à leitura e à gravação de alguns dados globais (acoplamento global).
- **Controle de acoplamento**: é quando dois módulos são chamados de controle acoplado, se um deles decidir a função do outro módulo ou mudar o fluxo de execução.
- **Acoplamento de selo**: vários módulos compartilham a estrutura de dados comum e trabalham em diferentes partes.
- **Acoplamento de dados**: dois módulos interagem por meio de dados passantes (como parâmetro). Se um módulo passa a estrutura de dados como parâmetro, o módulo de recepção deve usar todos os seus componentes.

Documentação do projeto:

Índice → Introdução → Visão geral do sistema → Arquitetura do sistema → Projeto de dados → Projeto de Componente → Projeto de Interface → Matriz de Requisitos → Apêndice.

Diagrama de Estado

Estado Inicial: ponto de entrada da utilização do objeto

Estado Final: ponto de saída da utilização do objeto

Estado: estados em que o objeto se encontra em cada momento

Evento: transição