

Framework

3. Framework

- Como já falado, a reutilização de código tem sido um dos principais objetivos da engenharia de software e vimos também que a reutilização não é uma tarefa simples. Outra solução que surgiu para que possamos **atingir a reutilização é o framework**.
- O principal **propósito de um framework é ajudar no processo de desenvolvimento de aplicações**. Ele permite que as aplicações sejam desenvolvidas mais rapidamente e mais facilmente, e deve resultar em uma aplicação de qualidade superior.



Framework

- Framework é uma técnica que é **aplicada tanto no projeto quanto na implementação** de um software orientado a objetos. Ela implica, basicamente, em explorar o potencial de reutilização de partes de softwares já desenvolvidas ou em desenvolver novos componentes de software prevendo sua reutilização no futuro. Embora o conceito de framework possa ser **aplicado nos diferentes tipos de paradigmas de programação**, é na orientação a objetos que ele encontra seu cerne ideal de implementação, uma vez que existe um mapeamento bastante interessante entre os conceitos associados a estes dois assuntos.



Framework

3.1 O significado e o uso de framework

- Um framework é uma coleção de artefatos de software que é utilizável por várias aplicações diferentes. Esses artefatos são em geral classes, juntamente com o software exigido para utilizá-las. **Um framework é um tipo de denominador** comum para uma família de aplicações. Organizações que pensam em termos de desenvolvimento futuro **designam classes selecionadas** como pertencentes ao seu framework. Em geral, um framework começa a surgir no momento em que uma organização de desenvolvimento produz sua segunda, terceira ou quarta aplicação.



Framework

- As classes dentro de um framework podem ser relacionadas. Podem ser abstratas ou concretas. As aplicações podem utilizá-las por meio de herança, agregação ou dependência. Alternativamente, um framework pode **comportar-se como uma aplicação genérica** que personalizamos inserindo nossas próprias partes.
- Os artefatos do framework estão em geral na forma pronta para **usar e são códigos nativos**. Alternativamente, eles poderiam ser código de máquina virtual, como códigos de bytes.



Framework

- Como exemplo, suponha que uma organização desenvolva softwares de geração de imagens para reconhecimento facial automatizado. Suponha que a primeira aplicação tenha que reconhecer rostos em uma cena e suas classes incluam Face, Cena, Queixo, Cabelo e Chapéu. Talvez possamos considerar Face, Queixo e Chapéu como potenciais classes do framework, porque pensamos que elas aparecerão em várias aplicações que tenham que ser desenvolvidas no futuro. Se a **segunda aplicação precisar identificar as partes das faces** e utilizar Face, Sobrancelhas, Boca, Orelha, Queixo e Cabelo, talvez possamos decidir que o framework deverá conter Face, Queixo e Cabelo. No momento em que chegamos a nossa terceira ou quarta aplicação, provavelmente estaremos prontos para encerrar as classes do framework.



Framework

- Isso significa posicionar essas classes em um local reconhecido, **documentá-las particularmente bem** e mantê-las cuidadosamente. Tiramos proveito das classes do framework em aplicações subseqüentes e, em geral, adicionamos outras ao longo do tempo.
- Por exemplo, o Enterprise Java Bean (EJB), que é um framework especificado pela Sun, economiza uma tremenda **quantidade de tempo de projeto** e desenvolvimento ao implementar aplicações do lado do servidor envolvendo bancos de dados.

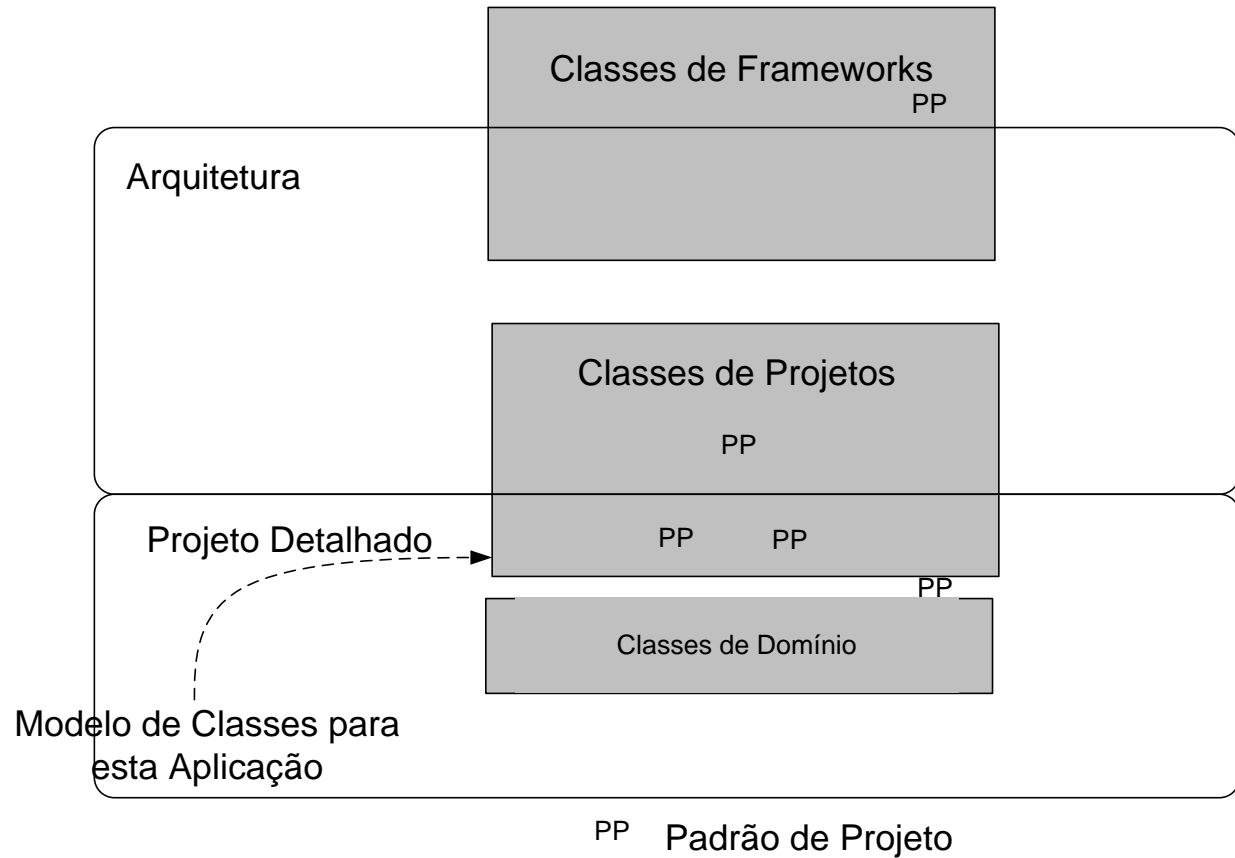


Framework

- A próxima figura mostra o relacionamento entre classes do framework, classes de domínio e classes de projeto restantes.
- O projeto completo para uma aplicação **consiste nas classes de domínio**, classes de projeto e algumas das classes do framework. Normalmente, não utilizamos todas as classes do framework em uma aplicação. As classes de projeto consistem em algumas que são exigidas pela arquitetura e outras que não são.
- As últimas são efetivamente as **classes de projeto com os detalhes exigidos** para completar o projeto.



Framework



Framework

- Como observado na figura anterior, os padrões de projeto às vezes são aplicados dentro do framework. Eles podem ser uma ajuda **significativa ao criar as classes de projeto**. Em geral, os padrões de projeto não ocorrem dentro das classes de domínio, pois estas são selecionadas separadamente. Entretanto, os padrões de projeto frequentemente incluem classes de domínio.
- As **classes do framework utilizadas no projeto** fazem parte da arquitetura de uma aplicação, embora elas também possam fazer parte do projeto detalhado. Normalmente, as classes de domínio fazem parte do projeto detalhado, uma vez que são específicas à aplicação e não são de natureza arquitetônica.



Framework

3.2 Utilizações do framework: conjuntos de ferramentas, arquitetura abstrata e controle

- Os frameworks têm três possíveis utilizações. Uma delas é fornecer uma "sacola de ferramentas". De fato, algumas coleções de classes simplesmente são chamadas **conjuntos de ferramentas** (toolkits). As classes em um conjunto de ferramentas podem estar relacionadas, mas esses relacionamentos não são de natureza arquitetônica. Por exemplo, o pacote Java `java.math` que contém `BigInteger` e `BigDecimal` é um conjunto de ferramentas. Ele não faz nenhuma suposição arquitetônica.




Framework

- O acesso a classes de um conjunto de ferramentas é irrestrito (em termos Java, public). Os conjuntos de ferramentas às vezes não são considerados frameworks, embora iremos considerá-los como tais.
- Outra possível utilização de frameworks ocorre quando eles contêm suposições arquitetônicas sobre as aplicações que os utilizam. Por exemplo, no pacote do framework (API) java.awt, a classe Container dá suporte ao método add(Component). Essa é uma propriedade arquitetônica, mas uma propriedade segura. (Com certeza, os contêineres precisam conter algo.)
- Novamente, o acesso a essas classes do framework é irrestrito.



Framework

- Nas duas utilizações de framework anteriores, o controle é responsabilidade do desenvolvedor. As classes do framework existem **simplesmente para ser utilizadas**. A terceira possível utilização de um framework se dá por ele conter o controle. Um exemplo simples é a API `java.applet`. Quando utilizamos Applet para criar nossa própria applet, não escrevemos um método `main()`. Em vez disso, devemos projetar e implementar de acordo com as regras ou protocolos fornecidos pelo framework. No caso de applets, as **regras informam o método** que é chamado quando uma applet é acessada (`init()`), o qual é chamado quando a applet é renderizada no monitor (`paint()`). 

Framework

- Algumas vezes, ela é chamada de "fluxo de controle reverso".
- A seguir é apresentado um resumo das três possíveis utilizações de um framework discutidas.
 1. Conjunto de ferramentas
 - Uma coleção de classes úteis
 - Cada uma utilizável isoladamente
 - (Às vezes não chamado de framework)



Framework

2. Denominador comum de arquiteturas
 - Uma arquitetura abstrata
3. Um sistema que possui controle
 - Siga os protocolos para obter sua aplicação



Framework

3.3 Objetivos dos frameworks

- Algumas pessoas acreditam que os frameworks devem ser projetados somente se forem utilizados por um grande número de aplicações. A API Java é um desses frameworks. Entretanto, frequentemente é sensato desenvolver uma **camada de framework somente para uma única aplicação**, mesmo se não houver nenhuma garantia de que outras aplicações a utilizarão. Esse framework parcial geralmente forma uma camada genérica, da qual muitas das classes da aplicação serão herdadas.



Framework

- A divisão em camadas resultante promove melhores projetos. Em particular, a maioria dos padrões de projeto tem uma camada abstrata e uma não-abstrata: as classes na **camada abstrata frequentemente** contêm classes do framework, e aquelas na camada concreta frequentemente contêm classes de domínio.
- Um dos objetivos **comuns da introdução de frameworks** é gerenciar a persistência. O exemplo a seguir ilustra o problema.



Framework

- Suponha que queiramos permitir que visitantes do nosso website deixem-no a qualquer momento, mas retomem como se eles nunca tivessem saído. Por exemplo, eles podem encher seus carrinhos de compras com mercadorias, e então deixar o site temporariamente. Quando eles retornam ao site, poderíamos querer que eles tivessem as mesmas **mercadorias nos seus carrinhos de compras** ou, pelo menos, fornecer essa opção a eles. Isso pode ser feito com a criação de um objeto (de uma classe visita, por exemplo) no servidor, que é armazenado e ressuscitado quando o cliente retorna ao site.
- Essa propriedade de salvar um objeto entre execuções **chama-se persistência**. Como a persistência é um requisito comum, um framework pode apoiá-la em geral, aliviando o desenvolvedor de projetá-la repetidamente.



Framework

- No nosso exemplo, precisamos salvar e restaurar o objeto Visita: mas haverá muitos objetos Visita. Como reconhecemos aquele de que precisamos? Esse é o problema da identidade de objetos.
- Uma maneira de abordar o **problema da identidade de objetos** é gerar uma cadeia totalmente única para cada objeto (e para todos eles), salvando-a **entre as execuções**.



Framework

- Frameworks podem fornecer vários outros recursos úteis: o pooling é um exemplo disso. A questão é que precisamos **compartilhar objetos criados** em tempo de execução. Fazemos isso para evitar a proliferação de objetos (uma questão de eficiência de espaço) e para evitar a **criação e a destruição constante de objetos** (uma questão de eficiência de tempo). Essa questão surge quando acessamos bancos de dados repetidamente.
- Um requisito comum para servidores é a **criação de conexões com bancos de dados**: meios de entrar e recuperar a partir de bancos de dados. Estabelecer uma conexão é algo semelhante a conectar uma aplicação a um arquivo, como em

FileReader inputReader = new FileReader("c:Teste\123"); 🔊

Framework

- Operações como essa consomem tempo e requerem lembrar os nomes de caminho exatos. Entretanto, esse processo é **razoavelmente genérico** por natureza e não deveria ser necessário criar essas conexões repetidamente. Felizmente, contamos com os frameworks para gerenciar essas conexões: configurá-las quando necessário, lembrá-las, **identificá-las, colocá-las em um pooling**, mantê-las disponíveis e fechá-las quando apropriado, etc.



Framework

- Outro requisito comum dos poolings é necessário ao lidar com linhas de execução (threads) geradas para tratar da **interação de clientes**. Suponha, por exemplo, que estejamos implementando uma aplicação do lado do servidor que trata de compras on-line. É natural criar uma **linha de execução para cada sessão** com um determinado cliente, uma vez que as sessões demandam tempo. Entretanto, devemos remover sessões efetivamente quando elas não são mais necessárias; caso contrário, o servidor logo ficará sobrecarregado com uma população de objetos de sessão jamais montados. Contudo, esse ciclo ininterrupto de **criação/destruição desperdiça tempo**, e a solução mais uma vez é colocar as sessões em um pool.



Framework

- Segurança, a capacidade de verificar todos os acessos ao servidor, é outro recurso que não deveríamos ter de projetar todas as vezes que construímos uma aplicação. Alguns **frameworks também tratam da segurança** por nós.
- Abaixo segue um resumo dos possíveis objetivos do framework.
- **Serviço de Persistência**
 - Armazenar instâncias entre execuções.
- **Serviço de Identidade**
 - Identificar objetos o bastante para recuperá-los entre execuções.



Framework

- **Pooling**
 - de objetos: reutilizar objetos em tempo de execução para poupar tempo e espaço;
 - de threads;
 - de conexões de banco de dados.
- **Segurança**



Framework

3.4 Vantagens da Utilização dos Frameworks

- Os principais benefícios dos frameworks orientados a objetos decorrem da modularidade, reusabilidade, extensibilidade e inversão de controle que eles oferecem aos desenvolvedores.
- Frameworks aumentam modularidade através do encapsulamento de detalhes voláteis de implementação por trás de interfaces estáveis.
- A modularidade dos frameworks ajuda a aumentar a qualidade do software, concentrando o impacto das mudanças de projeto e implementação, o que reduz o esforço necessário para entender e manter softwares existentes.



Framework

- As interfaces estáveis fornecidas pelos frameworks aumentam a **reusabilidade**, definindo componentes genéricos que podem ser reutilizados para criar novas aplicações. A reusabilidade do framework alavanca o **conhecimento do domínio e o esforço** anterior dos desenvolvedores, a fim de evitar recriação e revalidação de soluções comuns, recorrendo aos requisitos da aplicação e aos desafios do projeto do software. Frameworks maduros permitem uma redução de mais de 90% do volume de código fonte que tem que ser escrito para **desenvolver uma aplicação**, quando comparado com software escrito com o suporte de uma biblioteca convencional de funções.



Framework

- Um framework aumenta a extensibilidade fornecendo métodos "ganchos" (hook) explícitos, que permitem que as aplicações estendam suas interfaces estáveis. Métodos hooks desacoplam, sistematicamente, as interfaces estáveis e os comportamentos de um domínio de aplicação, em um contexto particular.
- A **extensibilidade do framework é essencial** para assegurar customização adequada de serviços e características para a nova aplicação.



Framework

- A arquitetura de run-time de um framework é caracterizada por uma inversão de controle. Quando se usa um framework, usualmente apenas se implementam umas poucas **funções de chamada ou se especializam** umas poucas classes e, então, se invoca um único método ou procedimento. A partir desse ponto, o framework faz o resto do trabalho, invocando quaisquer chamadas a cliente ou métodos necessários, no tempo e lugar adequados.



Framework

3.5 Tipos de Frameworks

- A classificação dos frameworks se dá pela suas dimensões. Há dois tipos de dimensões e são avaliadas pela seguinte forma:
- Como o framework é usado?
- Onde o framework é usado?



Framework

3.5.1 – Classificação na dimensão como o framework é usado

- Em relação à estrutura de um framework, as suas partes variáveis são chamadas de hot-spots (pontos de flexibilização) [Pree 1996], já as fixas, são chamadas de frozen-spots. Os frameworks em que os hot-spots são implementados através da especialização de classes abstratas são chamados de whitebox frameworks. Já os frameworks em que os hot-spots são implementados através da composição de componentes são chamados de blackbox frameworks.



Framework

- Para utilizar os whitebox frameworks, o desenvolvedor precisa conhecer a estrutura do framework. Dependendo da qualidade da documentação e da complexidade do framework, este aprendizado pode ser bastante longo. A **tendência é que as repetidas especializações feitas em um whitebox framework possam indicar oportunidades de criação de parâmetros default para o uso em um framework do tipo blackbox.** Desta forma, a evolução e a estabilização podem transformar, ao longo do tempo, um whitebox framework num blackbox framework.



Framework

- Os frameworks whitebox estendem ou modificam suas funcionalidades pela definição de sub-classes com override de métodos, já os frameworks blackbox utilizam **as funcionalidades presentes no framework**, desta forma as funcionalidades internas não podem ser vistas e nem alteradas, ou seja, deve-se usar as funcionalidades fornecidas.



Framework

- **Responsabilidades comuns e tempo de ligação** são duas características importantes para a definição do tipo de implementação de um hot-spot. Responsabilidades comuns correspondem aos serviços que qualquer classe que implemente o hot-spot deverá oferecer para o restante do framework. Já o tempo de ligação, define o momento em **que as características do hot-spot** são selecionadas e associadas. Esta ligação pode ser feita pelo **desenvolvedor durante a customização** do framework ou pelo usuário durante a execução da aplicação, neste caso, a ligação pode ser feita múltiplas vezes.



Framework

3.5.2 – Classificação na dimensão onde o framework é usado

- De acordo com Sauv  (2007), os frameworks s o classificados segundo sua utiliza  o em: frameworks de suporte, de aplica  o e de dom nio.

3.5.2.1 – Framework de suporte

- O framework de suporte prov  servi os no n vel de sistema operacional, como acesso a arquivos e computa  o distribu da. S o um tipo raro de framework.



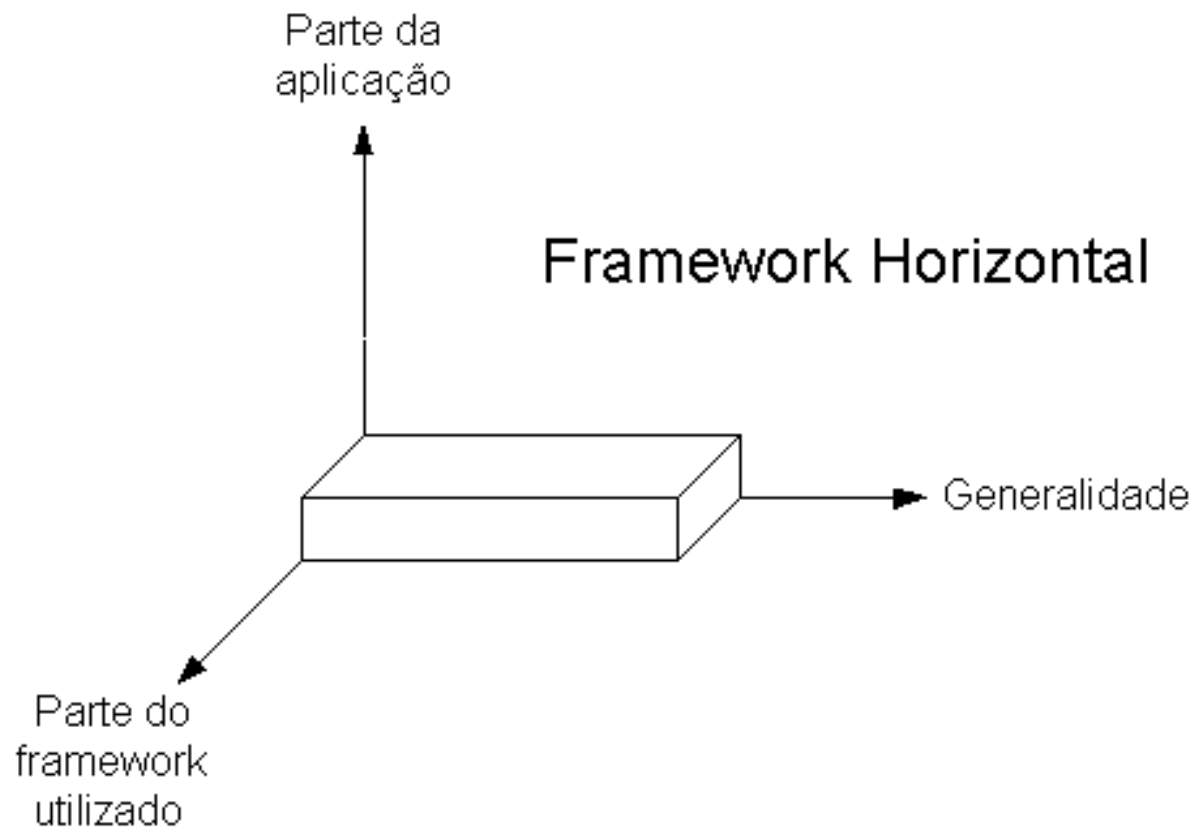
Framework

3.5.2.2 – Framework de serviço

- O framework de serviço, também chamado de framework horizontal, encapsula o conhecimento aplicável a uma vasta gama de aplicações, resolvendo apenas uma fatia do problema da aplicação, conforme ilustrado na figura a seguir.



Framework



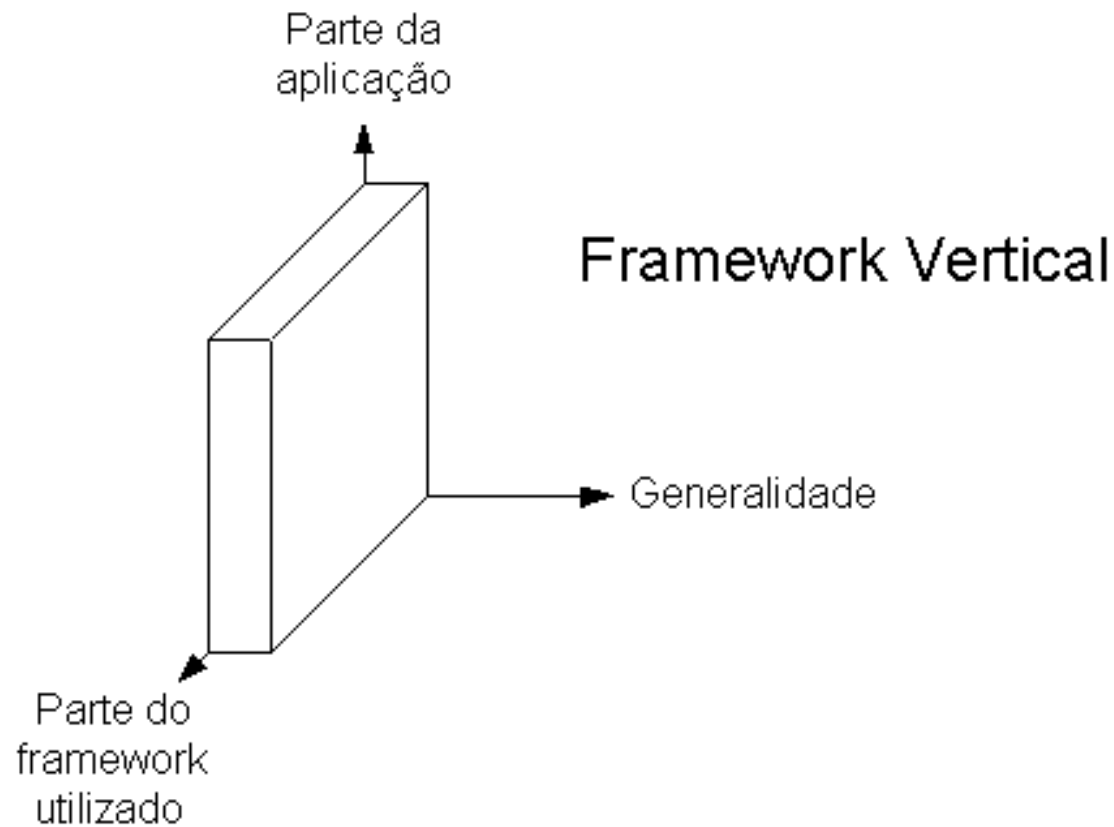
Framework

3.5.2.3 – Framework de domínio

- O framework de domínio, também chamado de framework vertical, **encapsula o conhecimento** voltado às aplicações, pois pertence a um domínio particular de problema, resolvendo boa parte da aplicação, conforme ilustrado na próxima figura.



Framework



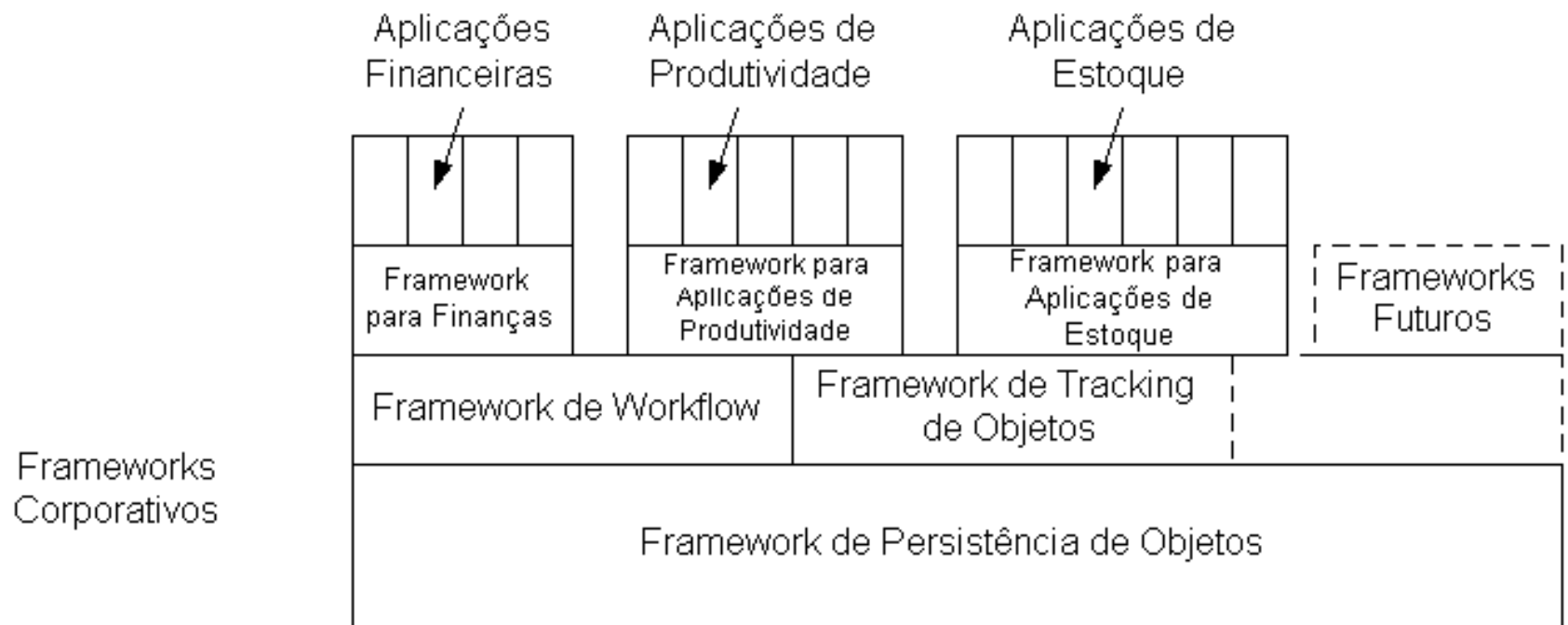
Framework

3.5.2.4 – Estrutura de Framework

- Uma empresa pode utilizar de vários frameworks para construir suas aplicações. Ela pode **mesclar os tipos de frameworks**, como por exemplo, uns podem ser horizontais e outros verticais.
- A figura abaixo utiliza de vários frameworks para a construção de um aplicativo corporativo. Acima da figura estão os **frameworks verticais** e **abaixo** na figura os frameworks horizontais.



Framework



Framework

3.6 – Exemplo de Frameworks

- Agora será apresentado um framework comercial bastante utilizado no mercado.
- O Hibernate é um framework para persistência escrito na linguagem Java. Sua principal característica é a transformação das classes em Java para tabelas de dados.
- O Hibernate gera as chamadas SQL evitando o trabalho manual da conversão dos dados, tornando a aplicação portátil para **quaisquer bancos de dados SQL**, porém causando um pequeno aumento no tempo de execução.



Framework

- O Hibernate **conta com um dialeto SQL chamado HQL** (Hibernate Query Language) que se parece muito com SQL exceto pelo fato de ser orientado a objeto. É possível executar os pedidos SQL sobre as classes de persistência do Java ao invés de tabelas do banco de dados.
- O Hibernate foi criado por desenvolvedores Java, espalhados ao redor do mundo, e liderado por Gavin King. Posteriormente, a JBoss Inc. **contratou os principais desenvolvedores do programa** para fazer o seu suporte.
- Segundo Larman (2000), as seguintes idéias-chave devem ser tratadas por um framework para persistência:



Framework

- **Mapeamento.** Deve haver algum mapeamento entre uma classe e seu armazenamento persistente (tabela) e entre os atributos dos objetos e os campos (colunas) em um registro.
- **Identidade de objeto.** A fim de poder relacionar facilmente os registros em objetos e garantir que não exista duplicidade, os registros e os objetos devem ter um único identificador de objeto.
- **Materialização e desmaterialização.** Materialização é o ato de transformar uma representação de dados não orientada a objeto, em objeto. Desmaterialização é a atividade contrária.



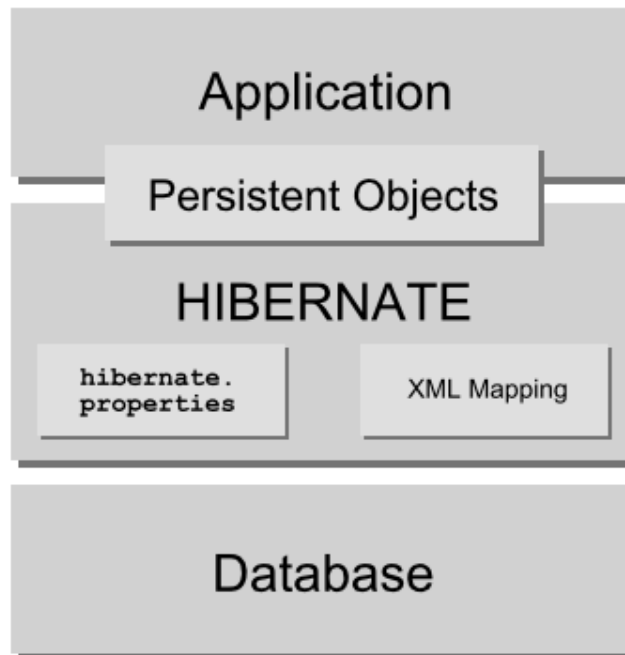
Framework

- **Caches.** São armazenados em uma área da memória os objetos materializados.
- **Materialização sob demanda (Lazy).** Nem todos os objetos são materializados de uma só vez. Uma instância somente é materializada sob demanda, quando necessária.
- **Referências Inteligentes.** A materialização sob demanda é implementada usando uma forma de referência inteligente, conhecida como proxy virtual.



Framework

- De acordo com a referência que acompanha a instalação do Hibernate (HIBERNATE, 2008), em uma visão de alto nível, **a arquitetura do framework** se apresenta da seguinte forma:

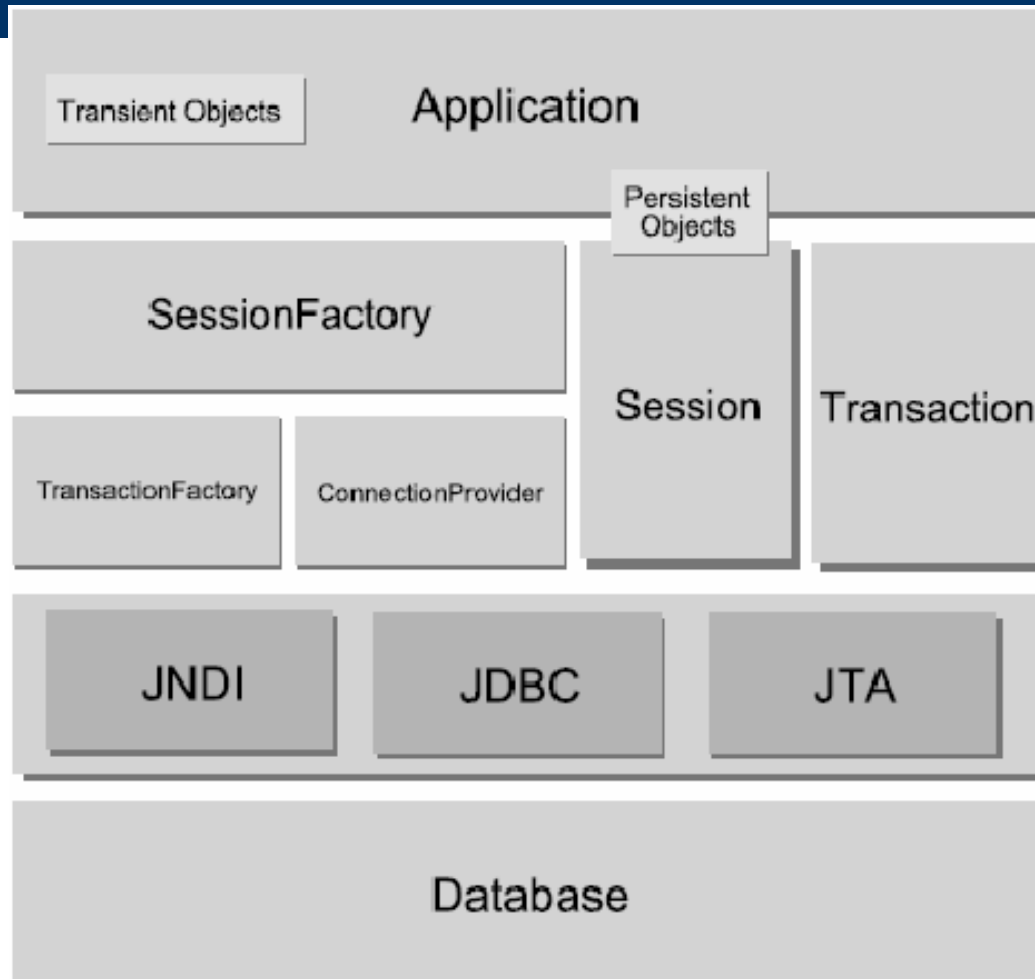


Framework

- O número de componentes utilizados aumenta conforme a **complexidade do projeto**. Abaixo uma visão detalhada sobre a arquitetura do Hibernate:



Framework



Framework

- De acordo com a **especificação do Hibernate**, os componentes da figura anterior são definidos da seguinte forma (HIBERNATE, 2008):
 - **SessionFactory**: Armazena os mapeamentos e configurações compiladas para um banco de dados.
 - **Session**: Objeto que representa o diálogo entre a aplicação e a persistência, encapsulando uma conexão JDBC. Este objeto controla um cache dos objetos persistentes.



Framework

- **Persistent Objects:** Objetos que contém as informações persistentes e regras de negócio. Devem ser Java Beans, possuir um construtor sem parâmetros e métodos get ou set para os atributos persistidos. Eles só podem estar associados à exatamente uma Session.
- **Transient Objects:** Instâncias de classes persistentes que não estão atualmente associadas a uma Session.
- **Transaction:** Usado pela aplicação para especificar unidades atômicas de acesso ao banco de dados.



Framework

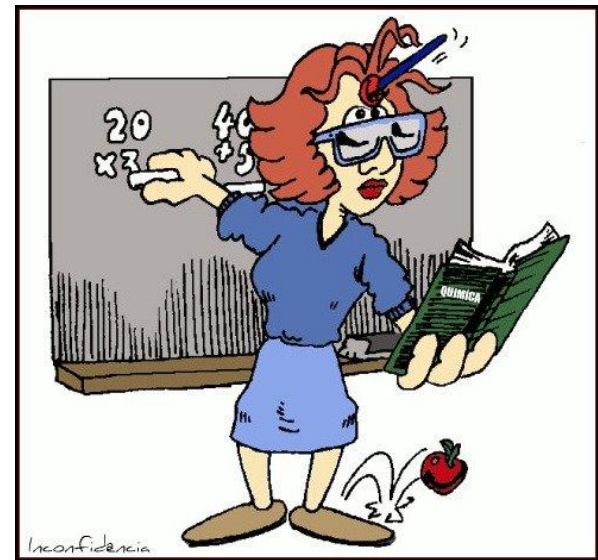
- **ConnectionProvider:** Abstrai a aplicação dos detalhes do Datasource ou DriverManager. Não exposto à aplicação.
- **TransactionFactory:** Instâncias de Transaction. Não exposto à aplicação.



Componentes



**Não durma
no ponto.**



**Exercício de
fixação.**

