A Plataforma Java

Você já deve ter ouvido falar que Java é tanto uma linguagem quanto uma plataforma. Isso quer dizer que, além da linguagem, o programador Java também conta com um conjunto de APIs que facilitam o desenvolvimento ao oferecerem soluções para situações comuns de desenvolvimento.

Confira como a plataforma Java foi organizada para atender as necessidades de diferentes tipos de aplicações:

- Java SE: Representa a base do Java, sendo composta pelas APIs e bibliotecas básicas para possibilitar o desenvolvimento de aplicativos de linha de comando e desktop. Já ouviu falar em Swing? Sim, ele faz parte do Java SE (Java Standard Edition);

- Java ME: A Java Micro Edition é a plataforma voltada para dispositivos móveis e embarcados, com capacidade de processamento reduzida, como os utilizados na criação de produtos para a Internet das Coisas;

- Java EE: Voltada para o desenvolvimento de soluções web e corporativas, a Java Enterprise Edition é uma especificação que agrupa outras, sendo, por isso, conhecida como especificação “guarda-chuva”.

Conhecendo As Plataformas Java

Com o objetivo de apresentar algumas das plataformas Java, este artigo aborda os conceitos de cada uma delas, onde podem ser utilizadas e quais os tipos de aplicação que podem ser desenvolvidas. Também serão apresentadas suas características e algumas especificações aplicadas e disponíveis para cada plataforma, descrevendo suas utilidades.

Este tema é útil para quem deseja conhecer e utilizar as plataformas Java EE, Java SE e JavaFX no desenvolvimento de aplicações web e desktop. É válido também para profissionais que desejam conhecer algumas das especificações que podem ser aplicadas nessas plataformas.

Diversas plataformas de desenvolvimento estão presentes fortemente no mercado de TI, como é o caso do Java. Ela foi lançada pela empresa Sun Microsystems em 1995 e atualmente está sob a responsabilidade da Oracle. Seu objetivo é fornecer ferramentas para o desenvolvimento de aplicativos que contemplam áreas como mobilidade, sistemas web, sistemas desktop, dentre outros.

A plataforma Java é composta atualmente por quatro edições para segmentos específicos de aplicações, a saber: Java EE, Java SE, Java ME e JavaFX. Uma característica forte e presente em todas elas é a contribuição significativa de seus usuários, que ajudam cada vez mais em seu processo de evolução e divulgação.

Além da colaboração da comunidade, uma das grandes vantagens do Java é ser multiplataforma, o que possibilita a execução de aplicativos em qualquer sistema operacional ou hardware, desde que o interpretador Java esteja instalado. Os programas desenvolvidos são emulados por meio de uma máquina Virtual, conhecida como JVM (Java Virtual Machine), que converte os bytecodes (codificação do programa) para uma linguagem que a máquina entenda.

Outra vantagem do Java é a oferta de um vasto conjunto de tecnologias (ou especificações) que estão disponíveis e ajudam a enriquecer cada vez mais as plataformas. Alguns conceitos dessas tecnologias serão abordados ao longo do artigo.

Java Enterprise Edition

A Java EE, ou Java Platform, Enterprise Edition, é a plataforma que disponibiliza recursos para o desenvolvimento de aplicações corporativas voltadas para web e servidores de aplicação.

Java EE foi projetada para suportar sistemas de uso em larga escala, ou seja, para uma quantidade significativa de usuários, possibilitando o desenvolvimento de aplicações escaláveis, robustas e multicamadas. Toda essa estrutura incorpora características como segurança e confiabilidade, muitas vezes consideradas difíceis de serem implementadas. Com o objetivo de amenizar as dificuldades de

implementação dessas características, o Java EE fornece um conjunto de tecnologias que reduz significativamente o custo e a complexidade de desenvolvimento.

Em uma aplicação multicamadas, por exemplo, tem-se a estrutura do aplicativo separada em camadas, onde cada uma possui uma responsabilidade específica. Atualmente é possível classificar essa estrutura em duas partes: arquitetura lógica e arquitetura física.

A arquitetura física refere-se à infraestrutura sobre a qual o sistema ou aplicação é executado, que pode ser dividida em três camadas, a saber:

· Cliente (Client Machine): uma máquina cliente pode ser um dispositivo móvel ou um equipamento com algum dispositivo inteligente.

Entretanto, um Cliente não necessariamente é considerado uma máquina, podendo ser também softwares que acessam outras camadas ou que recebem notificações delas e, no contexto da Java EE, um exemplo pode ser o acesso ao servidor. Diversos tipos de softwares podem ser considerados clientes Java EE, desde um navegador web até uma aplicação que não é desenvolvida em Java;

· Servidor de aplicação (Application Server): é onde as aplicações web e corporativas são instaladas. Quando alguma regra de negócio é alterada na aplicação, para que os clientes tenham acesso a essa alteração, basta atualizar o sistema no servidor de aplicações. Feito isso, todos os usuários passam a ter acesso à nova versão.

Os servidores de aplicação são acessados pelos clientes através de uma conexão de rede. É o servidor de aplicação que controla a conexão com o banco de dados em função das requisições do cliente;

· Servidor de banco de dados (Database Server): é onde residem os registros da aplicação desenvolvida, por exemplo, informações de cadastros de usuários. Os dados são acessados somente pelo servidor de aplicação e não diretamente pela aplicação cliente.

Em relação à arquitetura lógica, a mesma é caracterizada pela organização dos componentes do sistema ou aplicação, podendo ser dividida em quatro camadas, a saber:

· Apresentação (Web Pages): representa a interface gráfica (botões, janelas, formulários) que provém opções que permitem ao usuário interagir com o aplicativo. Uma camada de apresentação mal projetada pode resultar em uma alta complexidade, ocasionando uma experiência frustrante ao usuário;

· Controladora (Web Tier): objetiva controlar as requisições realizadas pelos usuários. Suas principais tarefas ou funções são: coletar as informações dos usuários na camada de apresentação, gerar conteúdo dinâmico para cada cliente e manter o estado dos dados para a sessão de cada usuário.

Basicamente esta camada controla o fluxo de interação que ocorre na camada de apresentação, servindo como intermediária entre a camada de apresentação e a camada de negócios;

· Negócios (Business Tier): objetiva implementar toda a lógica de negócio da aplicação, desde funções simples como validação de dados de entrada (por exemplo, CPF, CNPJ), até funções mais complexas como o controle de estoque.

Essa camada encapsula as funcionalidades da aplicação em EJBs (classes de negócio) sem se preocupar com a interface com o usuário ou a camada de persistência;

· Persistência (EIS Tier): é onde são implementados os métodos responsáveis por buscar, remover, editar e inserir informações na base de dados. Não é o próprio mecanismo de persistência (banco de dados), mas sim um front-end que manipula o acesso a ele.

Para compreender melhor como funciona essa arquitetura observe a Figura 1 que exemplifica graficamente a estrutura de uma aplicação Java EE.

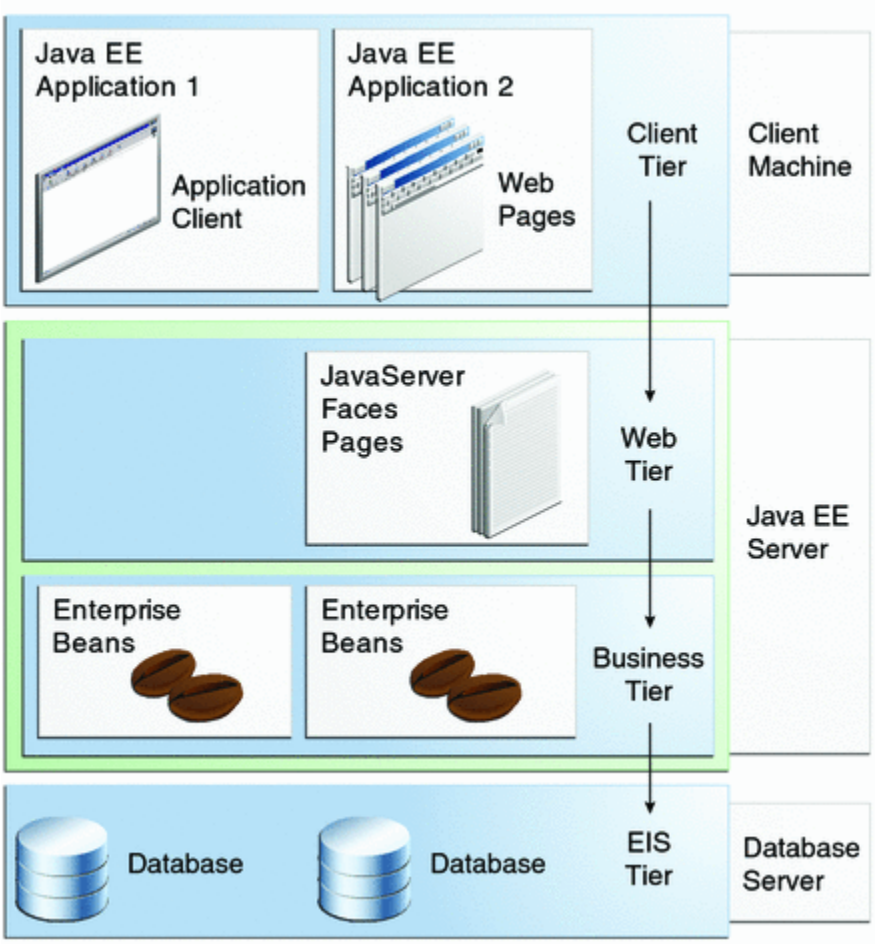

Figura 1. Estrutura de uma aplicação Java EE.

As quatro camadas lógicas interagem entre si através do seguinte fluxo: a camada de apresentação tem a função de interagir com o usuário e apresentar resultados. Ela dispara eventos gerados a partir das ações dos usuários (cliques ou seleções de menus) para a camada de controle. Esta gerencia todo o fluxo de informação recebida e ativa processos da camada de negócio. A camada de negócios, por sua vez, mantém toda a regra de negócios da aplicação, e de acordo com a ação recebida da camada de controle, executa a lógica necessária e se comunica com a camada de persistência, responsável pelo acesso às informações no banco de dados. Esta última camada, enfim, provê a criação, atualização, busca ou remoção de dados solicitados na camada de negócios, que são processados e devolvidos por meio da camada de controle à camada de apresentação.

Tecnologias Aplicadas Na Camada De Apresentação

Na camada de apresentação estão contidos todos os componentes visuais (formulários, menus, botões, por exemplo) presentes na aplicação. Para a construção da interface gráfica, tecnologias como HTML, JavaScript, CSS e Ajax são utilizadas.

Essas tecnologias trabalham em conjunto com algumas especificações Java, a saber:

· JavaServer Faces (JSF): tecnologia para desenvolvimento web que utiliza um conjunto de componentes de interface gráfica baseados em eventos. O JSF busca aprimorar a construção de interfaces com o usuário ricas e amigáveis, possibilitando aos desenvolvedores trabalhar na camada de apresentação sem precisar conhecer a fundo, por exemplo, JavaScript, pois a tecnologia JSF oferece componentes prontos;

· JavaServer Pages (JSP): é uma linguagem de script que tem como objetivo a geração de conteúdo dinâmico para páginas da internet, pois permite adicionar código Java a páginas estáticas, como as páginas HTML.

Embora presentes na camada de apresentação, as tecnologias JSP e JSF também possuem recursos que permitem o seu uso na camada de controle.

Com o objetivo de enriquecer ainda mais o client-side de Java, foi criada a plataforma JavaFX, que vem conquistando o mercado, provendo uma nova e moderna coleção de controles de interface com o usuário. Esta coleção permite aos desenvolvedores construir aplicações RIA (Rich Internet Applications) que podem ser executadas em vários navegadores, desktops e dispositivos móveis. Mais informações sobre ela serão abordadas em seu respectivo tópico.

Tecnologias Aplicadas Na Camada De Controle

Na camada de controle, que intermedia a interação entre a camada de apresentação e de negócios, o Java EE oferece tecnologias para atender as necessidades das tarefas descritas anteriormente. Dentre estas tecnologias podem-se citar alguns dos componentes fundamentais do Java EE, que são:

· Servlets: é um componente com o objetivo de receber requisições HTTP, processar e devolver respostas dinamicamente para a camada de apresentação, em formato de páginas HTML e XML, por exemplo;

· Expression Language: são tags padrões utilizadas dentro de páginas JSP e Facelets para referenciar componentes Java EE. As tags EL possuem algumas funções, a saber: possibilidade de invocar métodos estáticos e públicos, acesso dinâmico a dados armazenados em componentes JavaBeans e basicamente todas as operações de lógica e aritmética da linguagem Java (+, -, *, /, < =, ||);

· JavaServer Pages Standard Tag Lib: São bibliotecas de tags que encapsulam funcionalidades comuns dentro de páginas, como controle de laços (for), controle de fluxos (do tipo if else), dentre outros. O JSTL permite escrever páginas JSPs sem código Java, usando somente tags.

· Componentes JavaBeans: são componentes de software que são projetados para serem reutilizáveis. Para uma classe ser considerada um JavaBeans, deve possuir as seguintes características: implementar a interface Serializable, possuir um construtor sem argumentos e conter métodos get() e set() para acesso aos campos. Apesar da semelhança nos nomes, os JavaBeans não devem ser confundidos com os Enterprise JavaBeans, ou EJBs, que são componentes utilizados em servidores.

Tecnologias Aplicadas Na Camada De Negócios

A camada de negócio é responsável por implementar toda a lógica de negócios para o aplicativo. Nesta camada se concentra todo o código com as regras desenvolvidas para um caso específico, como um setor financeiro, um site de comércio eletrônico, entre outros. Em um aplicativo empresarial devidamente projetado, o núcleo da aplicação é implementado nos componentes da camada de negócio. As tecnologias que são oferecidas nesta camada são:

· Enterprise JavaBeans (EJB): é um modelo de componentes com foco em arquiteturas multicamadas, responsável por encapsular a lógica de negócios da aplicação. Fazendo uso de EJBs é possível construir aplicações escaláveis, transacionais, seguras e portáteis;

· JAX-RS RESTful web services: é uma API para criação de web services que usa a arquitetura REST (Representational State Transfer), sendo adotada no projeto de aplicações web que contam com recursos nomeados (URL, URI, URN). O objetivo da API é fornecer um conjunto de anotações para expor uma classe POJO como um serviço RESTful, a fim de simplificar o desenvolvimento de web services;

· JAX-WS web service endpoints: é uma API para criar e consumir web services SOAP. SOAP (Simple Object Access Protocol) é um protocolo para troca de informações estruturadas que se baseia em XML para seu formato de mensagem. Uma mensagem SOAP encapsula o conteúdo e pode ser trafegada via HTTP, JMS ou outro protocolo;

· Java Persistence API (JPA): é uma API que possibilita aos desenvolvedores gerenciar os dados dos usuários utilizando o mapeamento relacional de objetos (ORM). Tipicamente, uma entidade (classe)

representa uma tabela em um banco de dados relacional, e cada instância dessa entidade corresponde a uma linha (registro ou tupla) nessa tabela. A Java Persistence API possibilita criar essas entidades, modificá-las e removê-las do banco de dados. Os dados encapsulados pelos campos persistidos por essa API podem ser consultados usando, por exemplo, JPQL (Java Persistence Query Language), uma linguagem similar à SQL;

· Java EE managed beans: são componentes que podem prover a lógica de negócio da aplicação, mas não exigem os recursos transacionais ou de segurança dos EJBs.

Tecnologias Aplicadas Na Camada De Persistência

A terceira camada consiste em receber requisições da camada de negócio e executar essas requisições na base de dados. A seguir, são demonstradas algumas tecnologias que podem ser aplicadas para comunicação com o banco de dados a partir de aplicações Java EE:

· Java Database Connectivity API (JDBC): é um conjunto de APIs de baixo nível que tem como função estabelecer uma conexão, realizar consultas, inserção, remoção e atualização de dados, por meio de instruções SQL, em uma base de dados;

· Java EE Connector Architecture (JCA): é uma API para conexão de um servidor de aplicação com sistemas legados, geralmente sistemas que não evoluíram com o passar do tempo;

· Java Transaction API (JTA): API para definição e gerenciamento de transações. Por meio de interfaces é possível padronizar o uso de transações distribuídas, feitas por aplicativos Java. Basicamente é usada para verificar se uma transação é confirmada (commit), ou não (rollback). Um exemplo do uso de JTA é quando se necessita acessar dois SGBDs;

· Java Persistence API (JPA): A JPA, além de fazer parte da camada de negócios, como descrito anteriormente, também faz parte da camada de persistência.

Algumas das tecnologias supracitadas continuam sendo evoluídas nas novas versões das plataformas. Atualmente, a Java EE, por exemplo, está na versão 7, lançada oficialmente em 2013. Alguns dos principais marcos da evolução da Java EE estão ilustrados na Figura 2.

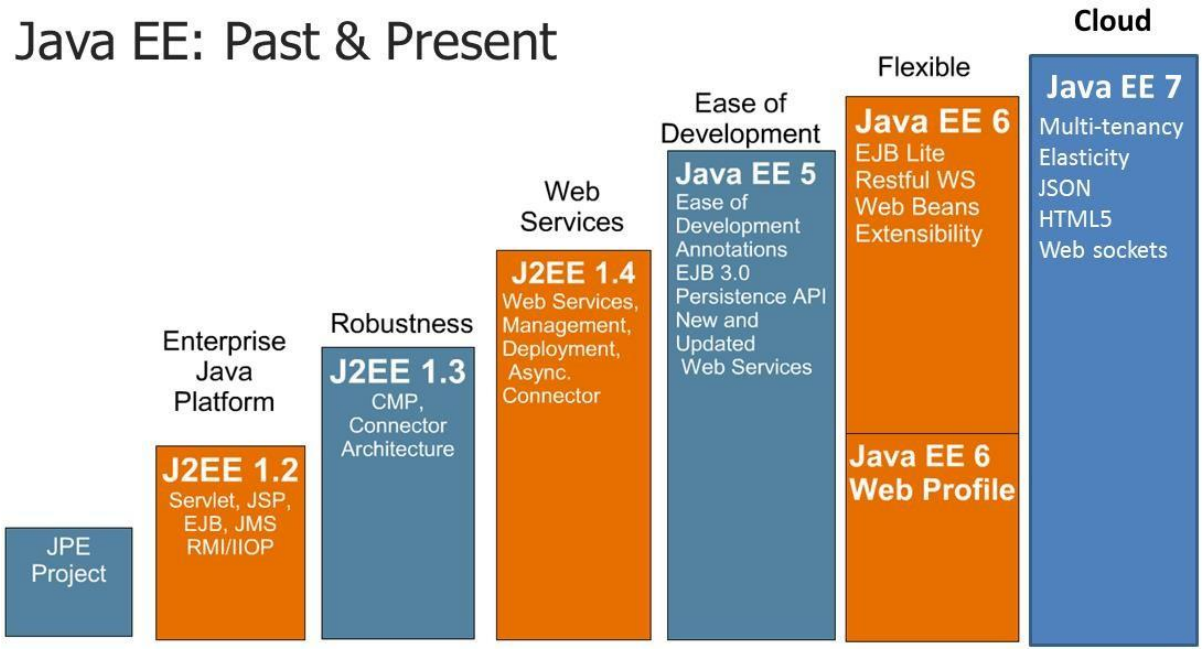

Figura 2. Evolução da plataforma Java EE.

A Java EE 7 introduziu um conjunto de novas APIs, bem como melhorou algumas das APIs já existentes. Seu maior objetivo foi facilitar a implementação e execução de aplicações em ambientes na nuvem, sejam elas públicas e/ou privadas. Um dos aspectos mais importantes do crescimento do Java é seu foco, que busca sempre acompanhar tendências com as novidades presentes em cada

nova versão de plataforma. As tecnologias e suas respectivas versões, que compõem essa plataforma, estão ilustradas na Figura 3.

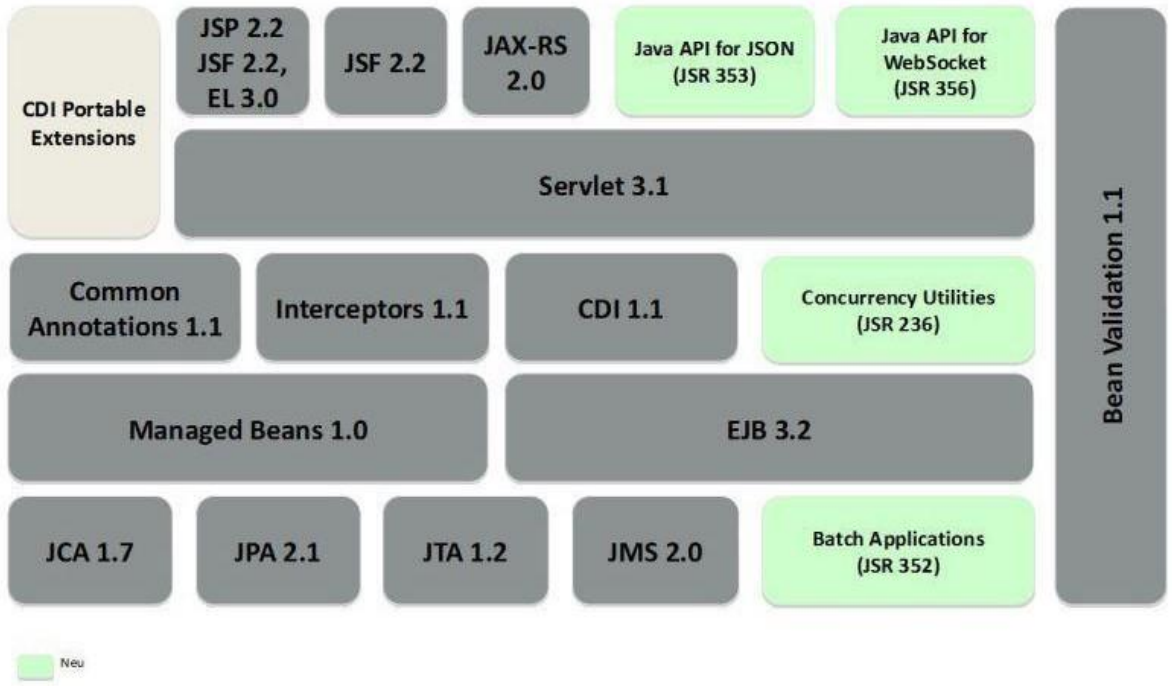

Figura 3. Especificações da Java EE 7.

Dentre as diversas melhorias projetadas para a Java EE 7, pode-se citar a tecnologia Bean Validation. Com ela é possível realizar validações no código fazendo uso de anotações. Bean Validation permite que a mesma validação de um campo obrigatório aplicada na camada de apresentação, por exemplo, possa ser utilizada nas demais camadas. Esta regra em todas as camadas fornece diversos benefícios ao desenvolvedor, gerando produtividade e evitando possíveis falhas.

Java SE

A Java SE (Java Platform, Standard Edition) é a plataforma de programação voltada para criação de applets e desenvolvimento de softwares para desktop, destinados a computadores pessoais, notebooks ou outras arquiteturas com maior capacidade de processamento e memória.

Os aplicativos podem ser executados em Windows, Mac OS, Linux, Solaris ou outros sistemas operacionais, contanto que estes tenham instalado o ambiente de execução JRE(Java Runtime Environment).

Tecnologias Aplicadas À Java SE

A plataforma Java SE fornece um conjunto de tecnologias para a criação de aplicações, sejam elas desktop ou não. Dentre as que ela oferece, são citadas algumas a seguir:

· Java 3D: API que oferece um conjunto de interfaces que possibilitam a construção e controle de gráficos 3D. Com esta API é possível incorporar gráficos de alta qualidade a aplicativos e applets;

· >Java Sound: fornece uma API para operações de áudio, como reprodução, captura (gravação) e mixagem;

· Java Advanced Imaging (JAI): provê um conjunto de interfaces que suportam um modelo de programação simples e de alto nível para manipulação de imagens;

· AWT (Abstract Window Toolkit): é uma API cuja função é fornecer suporte à criação de janelas, botões e outros elementos visuais usando métodos nativos. Ou seja, ela faz uso do ambiente gráfico do sistema operacional onde o software desenvolvido está sendo executado. Por exemplo, se você usa o Windows, toda a interface do seu aplicativo ficará com a aparência do Windows;

· JDesktop Integration Components (JDIC): é um projeto que fornece acesso a funcionalidades e facilidades oferecidas pelo ambiente nativo onde a aplicação é executada. Por exemplo, uma aplicação sendo executada em um ambiente desktop poderá abrir uma página web usando o navegador padrão definido pelo usuário em seu perfil na máquina. O JDIC suporta uma grande variedade de recursos, como criar ícones na área de trabalho do usuário, acesso ao editor de e-mails, dentre outros.

JavaFX

JavaFX é uma plataforma projetada para oferecer um rico conjunto de APIs que simplificam o desenvolvimento de softwares que utilizam conteúdo de multimídia. Para que isso seja possível, são utilizados avançados engines de mídia e gráficos de aceleração por hardware.

Essa tecnologia provê uma nova e moderna coleção de controles de interface com o usuário, tornando-se uma ótima opção para o desenvolvimento RIA com Java. As aplicações RIA (Rich Internet Applications) podem ser acessadas de qualquer computador que esteja conectado a internet. O termo RIA é usado para descrever Aplicações Ricas para Internet, que são executadas em ambiente web, mas que possuem características similares a softwares desenvolvidos para execução em ambiente desktop.

Investir nesta plataforma pode significar muitas vantagens para os desenvolvedores e empresas que já trabalham, principalmente, com Java. Por ser baseada nesta linguagem, é possível utilizar todos os seus poderosos recursos, como multithreading e suporte a anotações.

Outra vantagem é que a tecnologia utiliza FXML, uma linguagem de marcação como opção à codificação em Java. FXML permite modelar a UI separadamente da lógica da aplicação, não sendo necessário recompilar o código toda vez que uma mudança no layout ocorrer. Com essa tecnologia é possível criar tabelas, botões, aplicar efeitos em campos, criar animações, fazer uso de CSS, exibir conteúdo multimídia, dentre muitas outras funcionalidades que contribuem para fornecer ao usuário uma melhor usabilidade. Na Figura 4 é exibida a tela de uma aplicação disponibilizada pela Oracle com exemplos de aplicações desenvolvidas em JavaFX.

Figura 4. Aplicação com exemplos do JavaFX.

Para incrementar ainda mais a plataforma JavaFX, algumas tecnologias e projetos vêm contribuindo para o desenvolvimento de aplicações desse tipo. São eles:

· e(fx) eclipse: plugin do Eclipse para a construção de aplicações JavaFX;

· FxForm2: API para criação de formulários utilizando JavaBeans. FXForm2 usa a API deBean Validation para validação dos dados;

· DataFX: Atualmente, muitos dos web services têm seus dados disponibilizados em formatos como XML e/ou JSON. O objetivo do projeto DataFX é facilitar a leitura desses dados usando ListViews, TableViews e TreeViews de uma forma mais simples, produtiva e sem muitas linhas de código;

· JFXtras: projeto que tem como objetivo oferecer diversos componentes extras aos desenvolvedores, enriquecendo ainda mais a interface gráfica das aplicações e, consequentemente, sua usabilidade.

Conclusão

Neste artigo foi possível conhecer as principais plataformas Java (Java SE, Java EE e JavaFX) e abordar diversos conceitos de tecnologias que podem ser aplicadas especificamente em cada uma delas.

A edição Java SE é considerada a base da plataforma Java, sendo voltada, principalmente, para a criação de software para ambiente desktop. Como é a base da plataforma, Java EE estende Java SE, e é usada para a construção de aplicações corporativas e web. JavaFX, por sua vez, tem por objetivo focar no desenvolvimento de aplicações multimídia, proporcionando uma interface visual muito mais agradável que enriquece o desenvolvimento client-side da plataforma Java.

Para o futuro, espera-se que sejam implementadas ainda mais tecnologias que venham a fortalecer as plataformas analisadas, buscando uma melhora contínua para tornar o processo de desenvolvimento de aplicações Java ainda mais produtivo.

Máquina Virtual Java

Na máquina virtual Java, ou JVM, é onde a sua aplicação será executada. É ela, também, a responsável pela característica multiplataforma do Java. Um programa escrito nessa linguagem será executado em qualquer plataforma, seja ela um notebook, smartphone ou torradeira, que possua uma máquina virtual Java implementada.

Um dos recursos mais conhecidos da JVM é o Garbage Collection. É ele que é acionado com certa frequência para limpar da memória objetos que não estão sendo utilizados, evitando desperdício de espaço e que sua aplicação deixe de funcionar por falta dela.

Algumas vezes, a depender da proposta de sua aplicação, pode ser necessário fazer customizações na JVM visando melhoria na performance.

Primeiros Passos

Assim como em qualquer outra linguagem, no Java você encontrará tópicos de conhecimento fundamental. Dominá-los permite programar um código mais limpo e fácil de ser compreendido.

Diferente de outras linguagens, no Java o seu primeiro código será escrito dentro de um método de uma classe.

Após aprender sobre classes você deve estar curioso para saber como declarar variáveis e realizar operações com elas.

No Java as enums são um poderoso recurso para a criação de objetos imutáveis.

Você já ouviu falar que tudo no Java são objetos? Se ainda não, não tem problema.

Agora que você já conheceu os componentes básicos de um programa Java, que tal olhar um pouco mais de perto a criação de uma classe?

Strings

A manipulação de texto é algo bastante comum no desenvolvimento de aplicações. Em Java, a estrutura principal para isso é a classe String, que nos fornece vários métodos para sua manipulação. Nos posts abaixo você encontrará conteúdos que abordam desde os métodos básicos, até outras opções para que possa manipular texto da maneira mais adequada para sua solução.

Além da classe String, temos também as classes StringBuilder e StringBuffer.

Algumas vezes precisamos de uma ferramenta mais específica para processar Strings. É nesse momento que as expressões regulares podem ajudar.

Estruturas De Condição E Repetição No Java

Duas ferramentas importantes e presentes na lógica de programação de qualquer código são as estruturas de condição e repetição. A estrutura de condição nos permite criar um bloco de código a ser executado apenas se determinada condição for atendida. Já a estrutura de repetição nos permite criar um bloco de código que pode ser executado várias vezes, enquanto a condição especificada for verdadeira.

Entrada E Saída De Dados

Saber apresentar dados na tela e ler dados do teclado é um passo fundamental no aprendizado da programação. Lembre-se que o principal objetivo de um programa é processar os dados do usuário e exibir os resultados. Para isso temos os recursos de Entrada/Saída, ou In/Out da linguagem.

Módulos Do Código Java

Saber como organizar o código é fundamental para sua manutenção e reutilização. Em Java, o código é organizado em pacotes, classes e métodos. Dominados esses conceitos, e utilizando modificadores de acesso, podemos controlar como cada classe ou método poderá ser reaproveitado pelos demais.

Estrutura De Dados

Ao começar a desenvolver sistemas um pouco mais avançados, você se deparará com a necessidade de organizar os dados que são manipulados no código. Para isso, é comum o uso de estruturas de dados, que, como o nome indica, fornecem uma forma padrão para agrupar e simplificar a gerência de dados com características em comum.

Tratamento De Exceções

É comum a ocorrência de exceções durante a execução de um programa. O que não é comum é não nos prepararmos para elas. Uma exceção é um evento diferente daquilo que normalmente o software espera que aconteça. Quando não tratamos exceções, o software fica sem saber o que fazer com ela e como continuar sua execução, o que normalmente leva à paralisação do sistema.

Todo programa pode e vai falhar em algum momento. Aplicações consistentes se preparam adequadamente para esse momento.

Serialização E Fluxo De Dados

Logo você poderá se perguntar: Como salvar os dados que utilizo em minha aplicação para que consiga utilizá-los posteriormente? A resposta é simples: Serialização. Quando serializamos um objeto, estamos o transformando em um array de bytes, formato utilizado para salvar arquivos.

Sockets

Caso seu interesse seja aprender como enviar e receber dados e arquivos pela rede, saber programar com Sockets é fundamental. Compreender sockets é a primeira etapa para iniciar na programação distribuída.

Threads

Com a multiplicação do número de núcleos presentes em um processador, saber como programar com threads é algo de grande relevância para fornecer sistemas com alto desempenho, que explorem ao máximo o poder computacional de servidores e desktops.

Collections

Neste momento é bem provável que você já tenha tido o primeiro contato com as coleções, afinal, uma coleção é uma estrutura de dados. Mas, por que só agora falar delas? É importante compreender os fundamentos por trás de um conceito antes de começar a utilizar algo mais avançado, com um nível mais alto de abstração.

De forma simples, uma coleção é uma lista de objetos. Como quando lidamos com dados é comum nos depararmos com diferentes necessidades de organização dos mesmos, o Java facilita esse trabalho nos fornecendo diferentes opções para mantê-los - de listas que se preocupam apenas em armazenar os dados em memória, a listas que evitam dados repetidos e os mantém ordenados.

Generics

Generics é uma importante funcionalidade da linguagem Java que foi criada para facilitar o reuso do código e tornar a implementação mais segura e menos propensa a erros. Generics são bastante utilizados quando lidamos com coleções, por exemplo, e sua declaração é feita com os caracteres '<>'. Assim, quando declaramos uma lista, podemos especificar que essa lista somente poderá receber objetos de um tipo.

Com isso, caso o programador tente inserir um objeto de tipo diferente, o erro será identificado em tempo de compilação, e não mais em tempo de execução.

Expressões Lambda

Um dos recursos mais jovens da linguagem Java agrega um toque de programação funcional a ela. Uma definição simples e um pouco abstrata é entender que uma expressão lambda é uma opção para programar uma função sem que para isso seja necessário declarar um nome e um tipo de retorno. Em geral, a expressão lambda é declarada no mesmo lugar em que será utilizada e possibilita um código menor e mais simples de manter.

A sintaxe de uma expressão lambda é bastante simples. Começamos declarando os parâmetros. Em seguida utilizamos o operador arrow.

Por fim, declaramos o bloco de código a ser executado.

(Zero ou mais parâmetros) -> { /* bloco de código */ }

Streams

Com o intuito de aprimorar a forma como manipulamos coleções de dados, o Java 8 trouxe a Streams API, conjunto de classes e interfaces que faz uso de conceitos da programação funcional e das expressões lambda para que escrevamos código mais claro e em menor quantidade.

Para isso, obtemos uma stream, a partir de uma coleção, por exemplo, e, sem nos preocuparmos com a forma como os dados serão percorridos, programamos algum tipo de processamento sobre os dados, como filtros, mapeamentos, entre outros, já fornecidos pela API.

Como exemplo, considere que temos uma lista de produtos e desejamos selecionar apenas aqueles cujo valor seja maior do que 5. O código para solucionar esse problema poderia ser:

```
List<produto> produtosFiltrados = produtos

 .stream()

 .filter((p) -> p.getValor() > 5)
```

```
.collect(Collectors.toList());
```

Reflection

O Java também dispõe de um recurso conhecido como Reflection. Com ele, em tempo de execução conseguimos acessar informações de uma classe, como os atributos e métodos, assim como instanciar essa classe e invocar um método. Isso nos permite estender funcionalidades de uma aplicação, por exemplo.

Mas antes de prosseguir, sem querer dar spoiler, você vai precisar de um pouco de familiaridade com as anotações em Java para entender esse poderoso conceito na prática.

Reflection também é muito utilizado por IDEs, bibliotecas e frameworks, os quais precisam lidar com classes criadas por terceiros. Você já imaginou como aquela biblioteca que faz o parsing de objetos para JSON e vice-versa funciona? Como ela consegue converter qualquer objeto para JSON e o JSON para seu respectivo objeto?

Orientação A Objetos

A Orientação a Objetos é o paradigma de programação mais utilizado para o desenvolvimento de sistemas e tem como principal característica o planejamento e implementação do software a partir da representação de ‘coisas’ da vida real por meio de objetos.

Antes desse paradigma era comum criarmos sistemas utilizando o paradigma de programação estruturada. Porém, devido a limitações, aos poucos essa opção começou a ser substituída.

Pilares Da Orientação A Objetos

Ao planejar a programação baseada em objetos, esse paradigma trouxe consigo uma série de conceitos, os quais levam a um código mais fácil de entender, manter e reutilizar.

O encapsulamento é o pilar da Orientação a Objetos que nos permite ocultar informações e, ao mesmo tempo, atribuir segurança ao código. Quando encapsulamos o acesso a uma variável, por exemplo, não importa para o objeto que consome essa informação (o cliente), se o valor é obtido lendo o dado em memória ou se foi necessário acessar um banco de dados.

Imagine que você esteja em um restaurante e faça um pedido. É importante para você saber qual panela está sendo utilizada, a ordem de preparo dos nutrientes, entre outras coisas? Então, encapsular é a "arte" de deixar acessível ao cliente apenas aquilo que ele precisa saber.

E a herança? Como assim programar uma herança? Você herdou características de seus pais, avós e de outras gerações passadas, não é mesmo? Na Orientação a Objetos, quando precisamos fazer com que a classe B possua características de outra, definimos via código que a classe B herda dessa outra. Interessante, não? :) Ainda está achando um pouco confuso?

O polimorfismo é um pilar da Orientação a Objetos que tem forte ligação com a herança. Algumas vezes, quando herdamos uma característica, precisamos adicionar a ela um comportamento diferente. Por exemplo: todo animal sabe correr, porém, cada um corre de uma maneira diferente.

Coesão E Acoplamento

Você já deve ter ouvido falar que um bom código precisa ter alta coesão e baixo acoplamento. No entanto, o que é a coesão e o acoplamento? A coesão é uma medida que indica quão bem delimitado está cada módulo do nosso código. Caso ele faça mais coisas do que deveria, dizemos que temos um módulo com baixa coesão. Já o acoplamento mede o grau de dependência de um módulo em relação aos demais. Caso ele dependa de muitos módulos, dizemos que ele está com acoplamento alto. Essas situações devem ser evitadas, pois tornam o código muito mais difícil de manter e evoluir.

Princípios SOLID

Após compreender os pilares da Orientação a Objetos na teoria e na prática, um importante passo na busca pelo código de qualidade é aprender sobre os princípios SOLID.

Esses princípios nos ajudam a pensar no problema a ser modelado de forma orientada a objetos, propondo definições que nos levam a um design de código coeso e com baixo acoplamento, o que o torna fácil de ler, manter e reutilizar.

E por que esse nome, por que SOLID? Esse termo é um acrônimo para as cinco "regras" que o compõem:

- S - Single Responsibility Principle - O Princípio da Responsabilidade Única define que cada classe deve ter apenas uma responsabilidade;

- O - Open Closed Principle - O Princípio do Aberto Fechado define que uma classe deve ser extensível (aberta para extensões) por outras classes, sem que para isso precise ser modificada (fechada para modificações);

- L - Liskov Substitution Principle - O Princípio da Substituição de Liskov está relacionado ao uso da herança e dita que devemos ser capazes substituir a classe filha pela classe pai sem que o funcionamento do software seja prejudicado;

- I - Interface Segregation Principle - O Princípio da Segregação de Interfaces preza pela correta definição de interfaces, de forma que os clientes que a consumam tenham acesso apenas àquilo que de fato é necessário;

- D - Dependency Inversion Principle - O Princípio da Inversão de Dependências sinaliza que devemos depender apenas de classes abstratas, e não de classes concretas;

Orientação A Objetos Na Prática

Agora que você já conhece os pilares da Orientação a Objetos, que tal pôr as mãos na massa? Pode dizer! Já estava na hora, não é mesmo? :)

Para isso reunimos uma seleção de artigos que abordam esse tema na prática.

Como apoio a esse conteúdo, não deixe de verificar os posts abaixo e aprenda como solucionar e/ou evitar os erros mais cometidos quanto adotamos o paradigma OO.

Boas Práticas

Até o momento você aprendeu as principais tecnologias que compõem o Java, bem como utilizá-las de forma consistente. Que tal agora colocar em prática esses conhecimentos enquanto aprende técnicas avançadas no desenvolvimento com Java?

Lembre-se que o aprendizado da linguagem Java precisa ser constante. Somente programando você conseguirá implementar códigos melhores.

Ao prosseguir com a busca pelo código de qualidade, em algum momento você se deparará com o termo Clean Code. Mas, o que é isso? O Clean Code, ou Código Limpo, reúne uma série de práticas a serem aplicadas para obter um código fácil de compreender e evoluir.

A qualidade do código também influencia na performance.