Orientação a Objetos (Parte 1)
Enviado por Quarta, agosto 14 @ 14:38:17 EST por **Erro! A referência de hyperlink não é válida.**

http://www.portaljava.com/home/modules.php?name=Search&query=&topic=2Técnicas para Desenvolvimento de Aplicações Orientadas a Objetos Utilizando a Linguagem Java

Tutorial desenvolvido e enviado por
Marco Aurélio Souza Mangan
Patrícia Kayser Vargas
Denny Azzolin

Alunos da Universidade Federal do Rio Grando do Sul (UFRGS)

---

1 INTRODUÇÃO

Atualmente, o desenvolvimento de aplicações voltadas para a Internet tem se apresentado como uma das áreas de maior crescimento na indústria de informática. Rapidamente, a linguagem Java se estabeleceu como uma das melhores alternativas para implementação desse tipo de aplicação. Java é uma linguagem orientada a objetos moderna que adota uma série de mecanismos já consagrados em outras linguagens como C++, Eiffel e Smalltalk; ao mesmo tempo que é capaz de manipular vários mecanismos relacionados com a programação em rede.

Entretanto, surgem as questões: Esses mecanismos estão sendo explorados pelos desenvolvedores de aplicações? Um desenvolvedor consegue tirar proveito de todos os recursos que a linguagem oferece? Existe uma vantagem real em utilizar tais recursos?

Este tutorial tenta abordar essas questões apresentando algumas alternativas de implementação de uma aplicação. Através de um exemplo real, são apresentados os principais conceitos envolvidos no desenvolvimento de software orientado a objetos. Também são apresentados os principais problemas envolvidos no desenvolvimento de aplicações comerciais enfrentados pelo desenvolvedor.

A proposta deste tutorial é chamar a atenção para tópicos importantes da programação orientada a objetos de forma que possam ser encontrados maiores detalhes através da consulta à bibliografia. As referências apresentadas ao final deste texto e na página de hipertexto do tutorial (http://www.inf.ufrgs.br/~kayser/sblp99) são um bom ponto de partida. Na medida do possível serão apresentados exemplos práticos da aplicação desses tópicos. O enfoque será dado na organização e estruturação da aplicação. A explicação da programação de cada método não é um aspecto essencial do

exemplo.

Este trabalho foi possível devido a interação entre pesquisadores universitários e profissionais da área de informática de Porto Alegre – RS.

A aplicação utilizada como exemplo foi um applet Java encomendado pela companhia Telefonica Celular. A função principal do applet é o envio de mensagens em formato texto para telefones celulares. O envio de mensagens é feito em duas etapas. Primeiro, a aplicação deve entrar em contato com o servidor instalado na central telefônica. Através de um protocolo proprietário desse servidor a mensagem é programada para envio aos aparelhos de telefonia celular. Esta etapa ocorre através da Internet. Na segunda etapa, o servidor da telefônica se encarrega de enviar a mensagem através da rede celular.

A primeira versão dessa aplicação foi desenvolvida através de uma ferramenta de prototipação. O resultado obtido foi uma aplicação que atendia as necessidades expostas pela Telefonica. Entretanto, a implementação era pouco legível e de difícil manutenção. O código fonte foi definido em uma única classe com algo em torno de 500 linhas, onde eram tratadas a interface com o usuário, consistência de dados e a comunicação com o servidor.

Justamente a necessidade de manutenção e o reuso é que obrigaram a reengenharia da aplicação. A segunda versão foi escrita separando cada aspecto da implementação em várias classes. O resultado foi um código mais legível, mais fácil de ser alterado e a criação de um conjunto de componentes reusáveis.

Acredita-se que o principal desafio na aprendizagem de uma linguagem orientada a objetos seja compreender um novo paradigma, mais do que aprender uma nova sintaxe, de forma a poder tirar um maior proveito dos recursos oferecidos pela linguagem.

Este tutorial encontra-se organizado da seguinte forma. No Capítulo 2 são apresentados alguns conceitos do desenvolvimento orientado a objetos. No Capítulo 3 é apresentado o problema utilizado como estudo de caso no decorrer deste texto. No Capítulo 4 é descrita a etapa de engenharia do desenvolvimento da aplicação No Capítulo 5 é apresentado o particionamento da aplicação em três níveis: apresentação, modelo e integração de sistemas. No Capítulo 6 é apresentado como ficou a organização de classes após a reengenharia, demonstrando o potencial de reuso. Finalmente, no Capítulo 7 são apresentadas as conclusões.

---

2 DESENVOLVIMENTO ORIENTADO A OBJETOS

Antes de iniciar nosso estudo de caso é importante que algumas considerações gerais sejam feitas. A primeira delas visa esclarecer qual a necessidade da utilização da orientação a objetos no desenvolvimento de software. Em

segundo lugar, é preciso oferecer uma visão geral dos mecanismos e facilidades permitidos por esse paradigma. Por fim, vamos comentar um pouco sobre as características da linguagem Java e seu papel no cenário atual do desenvolvimento de software.

2.1 ASPECTOS ECONÔMICOS DAS GERAÇÕES DE LINGUAGENS DE PROGRAMAÇÃO

Para compreender qual a necessidade da orientação a objetos é interessante entender a evolução recente das linguagens de programação de um modo geral. Entre as diversas classificações propostas, podemos classificar as linguagens de programação em cinco famílias de acordo com os aspectos econômicos que lhes originaram:

- Primeira Geração: Linguagem de máquina
- Segunda Geração: Linguagem de montagem (assembly)
- Terceira Geração: Linguagens de alto nível
- Quarta Geração: Linguagens para a geração de aplicações
- Geração Orientada a Objetos: Linguagens voltadas para o reuso e manutenção

O maior determinante da primeira geração de linguagens de programação era o custo do equipamento. Não existiam facilidades para a operação e programação dos dispositivos. A demanda por aplicações era muito pequena.

Na segunda geração, o custo do equipamento ainda era alto, mas já permitia transferir um pouco do trabalho do programador para a máquina. A principal característica dessa geração é o aparecimento de programas capazes de interpretar instruções simbólicas e de transformá-las em linguagem de máquina (montadores).

Na terceira geração o maior determinante passou a ser o custo do desenvolvimento de programas. Com o relativo barateamento do equipamento, mais tarefas do desenvolvedor foram transferidas para programas como compiladores e interpretadores.

A quarta geração enfrentou o desafio de atender uma demanda crescente de aplicações. As ferramentas de desenvolvimento e as bibliotecas de aplicação se popularizaram nesse período. O usuário passou então a desfrutar de capacidades que antes eram reservadas aos desenvolvedores.

Finalmente, a quinta geração enfrenta hoje o desafio de efetuar a manutenção e o reuso das milhares de aplicações desenvolvidas pelas gerações anteriores. A orientação a objetos é um paradigma de programação que oferece vários mecanismos voltados para o reuso e a manutenção de sistemas de computador. O que torna a programação orientada a objetos tão atraente é, principalmente, a possibilidade de gerenciar de forma mais simples problemas bastante complexos e de reusar código de forma mais eficiente.

Como pudemos constatar, as vantagens da orientação a objetos estão

relacionadas com a produção, reuso e manutenção de aplicações, pois este é o contexto econômico em que se encontram inseridas. Características como o alto desempenho da execução, eficiência e simplicidade devem ser buscadas em outros paradigmas e/ou gerações de linguagens.

A seção seguinte apresenta os mecanismos que permitem a orientação atender ao reuso e a manutenção de aplicações de forma mais completa que os paradigmas anteriores.

2.2 CONCEITOS BÁSICOS

Os dois conceitos chaves da programação orientada a objetos são classe e objeto.

Classe é “uma descrição de um ou mais objetos através de um conjunto uniforme de atributos e serviços. Além disso, pode conter uma descrição de como criar novos objetos na classe” [COA 93]. “Uma classe é um protótipo que define os **atributos** (variáveis que definem estados internos de um objeto) e **métodos** (tipos de serviços ou procedimentos que determinam o comportamento possível dos objetos) comuns a todos os objetos de um certo tipo” [SUN 99].

Objeto é “uma abstração de alguma coisa no domínio de um problema ou em sua implementação, refletindo a capacidade de um sistema para manter informações sobre ela, interagir com ela ou ambos” [COA 93]. Podemos entender um objeto como um pacote de software contendo variáveis (atributos) e métodos relacionados [SUN 99] que é a instância de uma classe.

Desse modo uma classe define comportamento e forma padrão para a construção de objetos. Uma vez definida uma classe A, é possível definir uma nova classe B como uma extensão de A. Nesse caso, utiliza-se o conceito de **herança**, onde a **subclasse** B herda atributos e métodos da **superclasse** A. Uma subclasse pode reescrever métodos definidos na superclasse e criar novos atributos e métodos.

Na Figura 2.1 tem-se um exemplo de herança. As subclasses Quadrado e Retângulo herdam o atributo cor e o método desenha() da superclasse Polígono. Outra característica importante na programação orientada a objetos é o polimorfismo. O **polimorfismo** caracteriza-se pela possibilidade de classes distintas possuírem métodos com nomes idênticos mas com implementações distintas. Note que no exemplo da Figura 2.1 as subclasses Quadrado e Retângulo irão reescrever o método desenha() para exibir, respectivamente, um quadrado com lado lado e um retângulo com as dimensões base x altura.

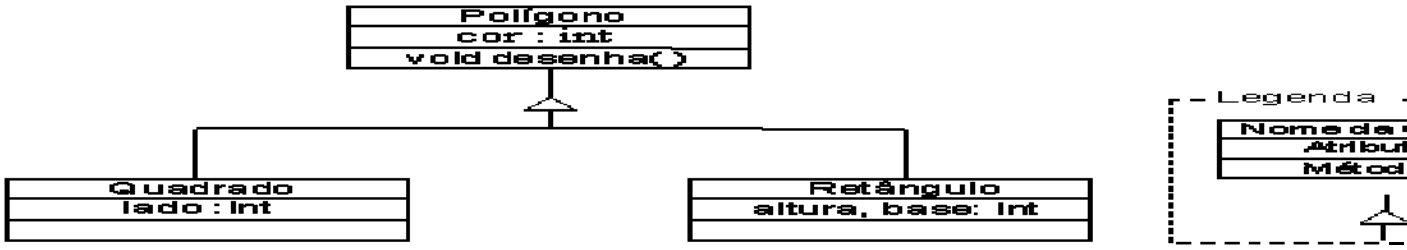

Figura 2.1 - Exemplo de herança e de polimorfismo.

2.3 PADRÕES

Quando um desenvolvedor experiente implementa um novo programa, dificilmente criará uma nova solução totalmente distinta de outras já implementadas. Normalmente, o programador se recorda de outros problemas similares que já foram resolvidos anteriormente e reusará a essência dessas soluções para resolver o novo programa. Esse cenário é válido para o desenvolvimento de aplicações e para a maioria das atividades humanas.

Neste contexto, a essência dessas soluções já executadas pode ser apresentada como um padrão (pattern), que se repete em diversos problemas, os quais nem sempre se encontram relacionados [BUS96].

Por exemplo, de uma maneira grosseira, podemos comparar o diagnóstico e tratamento de doenças na medicina com a aplicação de padrões. Quando um paciente procura um médico, ele informa a sua situação atual e o médico tenta identificar sintomas de uma anomalia (problema). Ao identificar uma série de sintomas, o médico tenta encontrar os registros de outros casos semelhantes. Esses casos anteriores fazem parte da experiência do médico ou foram apresentados durante o seu curso de medicina. Ao detectar qual a moléstia do paciente o médico tenta iniciar um tratamento (solução) que vai apresentar muitas vezes efeitos benéficos (como a cura do paciente) e alguns efeitos colaterais não desejados (como desconforto durante o tratamento, imobilização etc.). Esses efeitos em conjunto são a conseqüência da aplicação da solução. Para facilitar a comunicação entre si, os médicos convencionaram nomes para as doenças.

Em geral, um padrão possui quatro elementos essenciais [GAM 95]:

1. **nome**: é um identificador utilizado para descrever o cenário do padrão como um todo;
2. **problema**: descreve quando aplicar o padrão;
3. **solução**: descreve os elementos que fazem parte do projeto, seus relacionamentos, responsabilidades e colaborações;
4. **conseqüência**: são os resultados alcançados com a aplicação do padrão e a relação entre as vantagens e desvantagens dessa aplicação.

No desenvolvimento de aplicações, padrões ajudam a promover bons projetos e implementações de programas por capturarem experiências no desenvolvimento de software, descrevendo um problema recorrente e uma solução básica para o problema de tal forma que essa solução possa ser usada da mesma forma em novos problemas.

Existem diferentes tipos de padrões. Podemos destacar: padrões para projeto (design patterns [GAM 95]), padrões para análise (analysis patterns [FOW 96]) e padrões para organização de bancos de dados (data patterns [HAY]). Existem também padrões propostos para outras áreas do conhecimento como a arquitetura e urbanismo [ALE 77] e administração [DIK 97].

As bibliotecas da linguagem Java fazem amplo uso de padrões. A compreensão dos padrões auxilia, portanto, a compreensão das bibliotecas e, de forma indireta, da programação orientada a objetos.

2.4 TÉCNICAS DE DESENVOLVIMENTO

O desenvolvimento de software pode ser compreendido como composto de quatro fases: especificação, projeto, implementação e integração e teste. Essas fases são interativas e dinâmicas e, na prática, possuem características diferentes em cada grupo de desenvolvimento. No entanto existem alguns pontos que se seguidos podem determinar o sucesso de um projeto.

Na fase de projeto de um sistema orientado a objetos, normalmente são definidas as características do sistema, construídos protótipos de interface com o usuário e projetadas as classes do modelo.

Um objetivo importante da prototipação é servir de base para um desenvolvimento do sistema utilizando refinamentos sucessivos, isto é, o sistema não deve ser planejado para ser implementado em uma única versão. Os autores acreditam que o desenvolvimento é muito mais eficaz com a adição sucessiva de funcionalidades. É muito mais produtivo criar uma primeira versão em que exista um conjunto de funcionalidades mínimas e/ou com baixo desempenho e que esse protótipo seja gradativamente aprimorado a cada nova versão. Esse tipo de desenvolvimento além de facilitar o cumprimento de prazos, esconde a complexidade do problema.

A questão da prototipação é fundamental no desenvolvimento comercial pela possibilidade de interação com o cliente, permitindo a determinação das suas reais necessidades e preferências. Além disso, em muitos casos, contratos são firmados apenas depois da demonstração de protótipos satisfatórios por parte dos desenvolvedores. Note que o esforço de desenvolvimento de protótipos em linguagens como Java é minimizado pela existência de ferramentas visuais que permitem o rápido desenvolvimento de interfaces gráficas.

O projeto de sistemas orientados a objetos possui como principal ponto a necessidade de definir as classes envolvidas no sistema e de caracterizar a sua hierarquia, bem como a interação durante a execução entre as mesmas. Sem

dúvida uma boa modelagem facilita a fase de implementação, mas o fundamental é que durante a implementação seja possível rever a organização das classes. Um ponto importante a ser lembrado, é a quase inexistência de projetos no desenvolvimento de sistemas comerciais de pequeno porte, o que pode levar ao desenvolvimento de sistemas de difícil depuração e manutenção.

O desenvolvimento de software é uma atividade humana complexa, semelhante a tantas outras atividades que envolvem domínio da técnica, criatividade e raciocínio e, principalmente, perseverança. Desta maneira, as técnicas usadas na programação são também aplicadas e identificadas em outros ramos do empreendimento humano. Por exemplo, as seguintes técnicas utilizadas na “solução criativa de problemas” são amplamente adaptáveis no desenvolvimento de software orientado a objetos:

1. **busca de outras experiências e soluções para problemas semelhantes**
   soluções já existentes devem ser consultadas antes de se tentar propor uma nova solução. Através do refinamento, composição e instanciação de objetos de classes já existentes, e através da aplicação de padrões, a orientação a objetos facilita o reuso e a procura de outras soluções e experiências semelhantes;
2. **divisão e conquista**
   ao se deparar com um problema complexo, tenta-se reduzi-lo a problemas menores e mais fáceis de serem tratados;
3. **refinamentos sucessivos**
   deve-se resolver versões mais simples do problema em um seqüência contínua de dificuldade.
4. **diversidade**
   não é aconselhável comprometer-se imediatamente com uma única solução. Durante a solução do problema podem ser obtidas novas informações que permitirão a existência de novas alternativas.

As técnicas apresentadas foram adotadas por programadores de todo mundo, assim como por físicos, matemáticos e outros cientistas. Cada ramo do conhecimento possui sua própria versão dessas técnicas.

No desenvolvimento de software, um boa linguagem de programação oferece mecanismos que incentivam a aplicação dessas técnicas de desenvolvimento. Nesse contexto, as linguagens de programação são, em última análise, instrumentos para auxiliar na solução de problemas. Nosso instrumento nesse tutorial é a linguagem Java.

2.5 A LINGUAGEM DE PROGRAMAÇÃO JAVA

Em 1990 a Sun Microsystems desenvolveu uma linguagem orientada a objetos denominada Oak cujo propósito era o desenvolvimento de pequenos aplicativos e programas para controle de eletrodomésticos. Com essa

linguagem poderíamos programar novas funções para os aparelhos domésticos, tornando-os mais flexíveis. Por exemplo, um forno de microondas poderia ser programado para preparar diversos tipos de alimentos. Os fabricantes de um aparelho poderiam oferecer vários aplicativos para tarefas e usuários diferentes.

Além disso, a proposta da Sun previa o desenvolvimento de um sistema operacional que controlasse uma rede de eletrodomésticos. Com isso seria possível utilizar um microcomputador para automatizar uma residência e seus eletrodomésticos.

Com a popularização da rede mundial de computadores (Internet), através da World Wide Web, a Sun decidiu postergar a idéia dessa rede doméstica e direcionou a linguagem Oak para o desenvolvimento de aplicações na Web, dando origem à linguagem Java.

2.5.1 JDK 1.0

Em 1995 a linguagem Java foi disponibilizada pela primeira vez para o grande público através da primeira versão do Java Development Kit (JDK 1.0). Essa versão da linguagem utilizava vários conceitos da linguagem C++ [MEY 97], que era a linguagem comercial orientada a objetos mais popular da época. A linguagem C++ não possuía bibliotecas padronizadas e a implementação era muito dependente de uma dada plataforma. Java corrigiu uma série de problemas de C++ com a simplificação da linguagem e com a utilização de uma máquina virtual independente de plataforma.

A principal razão para a adoção de Java na Web foi a questão da segurança. Todo o programa Java tem um acesso limitado aos recursos disponíveis. Existem dois tipos de programas Java: aplicação e applet. Uma aplicação é um programa independente enquanto um applet executa em um ambiente controlado a partir de um navegador (browser), estando sob um controle mais severo de acesso aos recursos computacionais.

Também, a existência do interpretador Java e a sua biblioteca padronizada foram outros pontos a favor da linguagem. Nessa primeira versão do JDK, foram introduzidas várias bibliotecas (pacotes). Java.lang e java.applet disponibilizam as classes que formam a base da linguagem.

O pacote java.awt é o que define as ferramentas para desenvolvimento de interfaces gráficas, com o tratamento de eventos através de um sistema de callbacks e implementação nativa de componentes de interface. Note que o uso de callbacks já era amplamente utilizado no sistema operacional UNIX e em ambientes como o ET++ [WEI 94].O pacote Java.net fornece facilidades para acesso aos recursos da rede bem como para programação concorrente e distribuída. Foram disponibilizados ainda os pacotes java.io e java.util. Java também permite a definição e controle de processos leves (threads) [OAK 99] através de elementos da própria linguagem desde essa versão inicial.

É importante lembrar que nessa versão foram introduzidos vários mecanismos importantes como RMI (Remote Method Invocation) e JNI (Java Native Interface). RMI é usado em programas cliente-servidor e distribuídos para acessar métodos de objetos remotos. JNI permite o acesso a funções

implementadas em outras linguagens, como C e C++, como se fossem métodos Java, facilitando a integração de sistemas legados.

2.5.2 JDK 1.1

A primeira versão serviu para consolidar a posição da linguagem no desenvolvimento da Web, mas haviam muitos pontos que podiam ser melhorados e vários bugs a serem corrigidos. A versão JDK 1.1 lançada em 1997 trouxe uma série de inovações dentre as quais destaca-se uma nova forma de tratar os eventos através do uso de listeners substituindo o sistema de callbacks da versão 1.0. Com este novo mecanismo, os objetos se cadastram como interessados em obter informações de determinados eventos (princípio de Hollywood: “não telefone para mim, deixe que eu telefono para você”).

Nessa versão a integração com bancos de dados foi facilitada com a introdução do Java Data Base Connection (JDBC). Com o JDK 1.1 também foi introduzido o conceito de modelo de componentes (beans) e de uma apresentação e comportamento da interface independente de plataforma (swing). Esses conceitos já haviam sido explorados em outras linguagens. O modelo de componentes apareceu anteriormente em toolkits de desenvolvimento como o OpenWindows Motif e o Microsoft Fundation Classes. O swing foi oferecido como um pacote extra que podia ser utilizado com o JDK1.1.

Essas alterações possibilitaram o desenvolvimento de aplicações comerciais e impulsionaram o aparecimento de ambientes de programação e novas bibliotecas desenvolvidas por empresas como Microsoft (Visual J++), IBM (VisualAge for Java), Inprise (JBuilder) e Symantec (Visual Café). Nesse período, a linguagem deixou de ser apenas uma opção para desenvolvimento Web e passou a ser uma opção para o desenvolvimento de aplicações.

Bibliotecas de programação importantes foram traduzidas para Java como, por exemplo, a Java Generic Library (JGL), uma biblioteca de programação genérica que faz uso intensivo da orientação a objetos. Outros sistemas importantes foram migrados para Java como: bancos de dados (ObjectSpace da ODI), implementações CORBA (Visibroker da Visigenic) e bibliotecas para programação de agentes distribuídos (JavaAglet da IBM, Odissey, Voyager).

Na medida que a linguagem crescia em importância econômica, passou a enfrentar as suas primeiras dificuldades. Primeiro a disputa entre a Sun e a Microsoft pela propriedade e permissão para alteração da máquina virtual. A Sun era contrária a inclusão das implementações do modelo de componentes e das bibliotecas nativas da Microsoft junto a máquina Java. Essa inclusão tornaria as bibliotecas de Java dependentes da plataforma MS Windows, segundo a Sun.

Ainda nesse mesmo período, os programadores de applets reagiram a proposta da Sun de criar uma linguagem tão abrangente. O tempo de compilação e de execução de Java aumentou em muito com a nova versão do JDK. A Sun não arriscou a criação de duas linguagens separadas: uma para Web e outra para o desenvolvimento de aplicações. Com isso parte do público de Java migrou para alternativas mais simples como JavaScript e VBScript.

Para contornar a falta de velocidade da linguagem, foram lançados os primeiros compiladores de código nativo (Symantec Just-In-Time compiler e SuperCede). Nesse tipo de compilador, o código Java é transmitido em byte codes até a máquina cliente e então é convertido para instruções nativas antes de iniciar a execução. Depois de compilado, a velocidade de execução é comparável com a de um programa nativo.

Ainda nesse período crítico da história de Java, surgiram compiladores para linguagens como Basic e C++ que geravam byte codes para a máquina virtual Java [TOL 99].

2.5.3 JDK 2

A versão mais recente do JDK, inicialmente lançada como JDK 1.2 e agora chamada JDK 2, trouxe inovações na área de segurança, mas as principais mudanças ocorreram na programação de interface com o usuário. A versão 1.1 já havia introduzido modificações nas classes awt (abstract window toolkit). A versão 1.2, por sua vez, incorporou o pacote swing [ECR 98], que é uma alternativa mais completa que a awt. O swing possui como principal característica um look and feel independente de plataforma. Ou seja, uma aplicação em Java pode ser desenvolvida para manter a mesma aparência e comportamento na interface, independente da plataforma onde está executando.

2.6 CONCLUSÃO

Java é um sistema composto de três elementos: uma linguagem orientada a objetos, um conjunto de bibliotecas e uma máquina virtual no qual os programas são executados. Conceitualmente, Smalltalk apresenta uma abordagem semelhante, embora o ambiente de programação Smalltalk seja muito superior ao de Java do ponto de vista de coerência e elegância. Entretanto, Smalltalk é limitada em relação ao tratamento de novos recursos computacionais como programação em rede, utilização de banco de dados, integração com a Web, internacionalização e segurança. Claramente, Java herda muitas das características de C++ e Smalltalk e as simplifica, oferecendo uma linguagem eficiente e adaptada ao ambiente da Internet.

A importância de Java no cenário de desenvolvimento de software atual é justificada por uma série de características importantes da linguagem, entre elas:

(a) simplicidade da linguagem aliada a uma série de recursos disponíveis através de bibliotecas;

(b) independência de plataforma, uma vez que o código compilado pode ser executado em qualquer plataforma que possua uma máquina virtual Java instalada;

(c) mecanismos de segurança principalmente para os programas do tipo applet.

Muitos outros mecanismos foram integrados em Java tais como, threads, comunicação entre objetos e tratamento de eventos. Essas características não são conceitos novos, já existindo em outras linguagens e ambientes, sendo amplamente utilizados em linguagens conceituais como Oberon. No entanto, muitas vezes essas características, e até a programação orientação a objetos, são novidades para os profissionais do nosso mercado. Deste modo, um programador Java novato é obrigado a dedicar um esforço muito maior para dominar esses novos conceitos e passar a tirar efetivo proveito dessa nova ferramenta.

O objetivo desse tutorial é apresentar técnicas que auxiliem o desenvolvedor a explorar todo o potencial da linguagem Java. Entretanto, as técnicas apresentadas são aplicáveis a outras linguagens orientadas a objeto.

A abordagem deste tutorial é a apresentação dessas técnicas juntamente com a sua aplicação sobre um problema real. A seção seguinte detalha o estudo de caso proposto.

---

3 APRESENTAÇÃO DO EXEMPLO

Este capítulo apresenta um exemplo prático de uma aplicação orientada a objetos em Java: um applet para envio de mensagens para telefones celulares utilizado pela companhia Telefonica Celular. Será apresentado, além da descrição do exemplo, um breve relato de negociações com o cliente, apresentando uma perspectiva prática do assunto. Também são apresentados os aspectos culturais envolvidos na adoção de uma nova linguagem.

3.1 APRESENTAÇÃO DO PROBLEMA

A companhia Telefonica Celular desejava oferecer um serviço de mensagens curtas em formato texto para os assinantes de seu serviço de telefonia móvel. Esse serviço se encontra disponível através da central digital de telefones móveis. A Telefonica necessitava de uma aplicação que permitisse a um usuário da Web enviar a mensagem até a central telefônica, de onde ela seria transmitida ao telefone móvel correspondente.

Para escolher a empresa que desenvolveria essa aplicação foi aberta uma concorrência, que teve como selecionada a empresa IntexNet, de Porto Alegre.

3.2 NEGOCIAÇÃO COM O CLIENTE

Diversas empresas participaram da concorrência aberta pela Telefonica Celular. No entanto, a IntexNet Network foi a única empresa a apresentar uma

solução desenvolvida em Java. As outras empresas optaram pela utilização de linguagens de script tais como PERL [SRI 97], JavaScript [GOO 98] e VBScript [MAN 96].

A reação inicial do cliente, e em especial do gerente de marketing da Telefonica Celular, foi negativa, demonstrando que o mercado tem reservas com relação a linguagem Java. A principal reserva é em relação ao desempenho de Java em dois momentos distintos:

(a) inicialização da máquina virtual;

(b) desempenho de execução.

A primeira execução de um applet Java é precedida pela inicialização da máquina virtual. Esse processo leva alguns segundos dependendo da versão da linguagem e da plataforma do navegador. Durante a inicialização da máquina virtual o navegador Web deixa de responder a comandos do usuário, o que causa um certo desconforto em determinados tipos de usuários. Versões mais antigas de navegadores não permitem a utilização de applets e em alguns casos o sistema pode se tornar instável por alguns momentos. Logo, a primeira impressão causada por Java no mercado é baseada neste tipo de experiência negativa.

As dúvidas em relação ao desempenho foram geradas, em parte, em uma das diversas campanhas negativas desencadeada por concorrentes da empresa Sun, criadora e fornecedora da linguagem Java. A preocupação com desempenho é causada pela falta de compreensão do modelo de execução de um applet. Na Web existem applets de todo tipo e complexidade. Mas a maioria deles foi criada para executar aplicações de tamanho reduzido e com um modesto uso das capacidades de processamento dos computadores atuais. Existem applets de tamanho reduzido que executam milhares de instruções de ponto flutuante, comprometendo o desempenho da aplicação. Mas esse tipo de aplicação não é o mais comum em sistemas de informação, onde os applets são encarregados, basicamente, da interação com o usuário.

Naturalmente, essa impressão negativa causada pela linguagem preocupou o departamento de marketing, muito mais do que a equipe técnica. A solução encontrada pela IntexNet foi a apresentação de um protótipo para desfazer qualquer dúvida sobre o desempenho efetivo da aplicação. O resultado final do applet pode ser visitado a partir do endereço eletrônico http://www.telefonicacelular.com.br.

3.3 O PORQUÊ DA OPÇÃO JAVA

A solução apresentada possuía uma série de vantagens, tal como permitir um controle muito maior que uma solução com o uso de script. No entanto, para que a negociação continuasse, a IntexNet teve que provar que a solução era possível de ser realizada e que as outras não eram tão boas. Em uma semana foi apresentado um primeiro protótipo tal como será apresentado no capítulo seguinte. Assim, para garantir o fechamento do contrato, foi preciso tanto uma prova de que a empresa dominava a tecnologia Java quanto uma

prova das vantagens sobre outras tecnologias.

O prazo total para o desenvolvimento da aplicação foi de um mês. Note que no mercado os prazos são curtos e precisam ser obedecidos sob o risco de perda do cliente. Por exemplo, se o protótipo inicial tivesse levado mais de uma semana para ser produzido, dificilmente a concorrência teria sido ganha. Pode-se citar como vantagens do uso da linguagem Java que garantiram que a concorrência fosse ganha:

- Java dificulta ao usuário perceber o funcionamento do programa, aumentando um pouco a segurança;
- é possível controlar quem, quando e de que forma enviou a mensagem, bem como o número de mensagens enviadas;
- o applet foi utilizado como uma forma de transmitir um imagem positiva da empresa, pois auxilia a Telefonica a mostrar o domínio de um tecnologia nova;
- o uso de Java permite enviar um retorno (feedback) ao usuário dos passos que estão sendo executados, ou de avanço, no processo de envio. Com script normalmente apenas na finalização o usuário é informado do sucesso ou fracasso na operação.

Note que para o cliente não é importante a forma como um sistema é desenvolvido e sim que as funcionalidades desejadas sejam atendidas, e nesse caso Java permitia que essas funcionalidades fossem atendidas de forma mais eficiente.

3.4 DESCRIÇÃO DA APLICAÇÃO

Através de uma das páginas Web da Telefonica, o usuário deve fornecer informações para o envio da mensagem, tais como o número do telefone celular e a mensagem. A interface com o usuário deve testar a consistência dos dados submetidos, como por exemplo, evitar a submissão de um número inválido de telefone. Uma vez que essa aplicação é acessada pela Internet, também é preciso incluir mecanismos de segurança para o acesso. Outra característica importante é que o applet esteja totalmente integrado à página da Telefonica.

Quanto ao envio de mensagens, ele é feito em duas etapas. Primeiro, a aplicação deve entrar em contato com o servidor instalado na central telefônica. Através de um protocolo proprietário desse servidor a mensagem é programada para envio aos aparelhos de telefonia celular. Esta etapa ocorre através da Internet. Na segunda etapa, o servidor da telefônica se encarrega de enviar a mensagem através da rede celular.

---

4 ENGENHARIA

Nesta etapa do desenvolvimento da aplicação o desenvolvedor está preocupado em atender exclusivamente as necessidades do cliente. Preocupações com a engenharia de software e a elegância da solução são marginais. A funcionalidade aparente, revelada na interface com o usuário, é o aspecto mais importante da aplicação.

Neste contexto, o desenvolvedor tem seu primeiro contato com a orientação a objetos, e com a linguagem Java. Através de uma das diversas ferramentas de desenvolvimento rápido disponíveis no mercado, é gerado um protótipo que irá convencer o cliente da viabilidade da aplicação. Neste tutorial, chamaremos esta fase do desenvolvimento de prototipação. Nesta fase, o desenvolvedor deve dominar uma determinada ferramenta de programação visual.

Em um segundo momento, após a estabilização da interface e dos requisitos da aplicação, o desenvolvedor deverá consolidar a funcionalidade aparente com a inclusão de código no protótipo. Nesta fase de complementação o desenvolvedor deve apresentar domínio sobre a biblioteca de programação Java.

As duas próximas seções apresentam um panorama de como ocorre o desenvolvimento de software nessas duas fases.

4.1 PROTOTIPAÇÃO

Nesta fase, os principais recursos do programador são a biblioteca de classes e o ambiente de programação visual. No caso do applet exemplo, o navegador (browser) que o usuário utilizará limita uma série de possibilidades do desenvolvimento.

O primeiro contato de um desenvolvedor de software típico com a orientação a objetos ocorre durante o uso de uma ferramenta de desenvolvimento rápido ou uma biblioteca de programação. Esse primeiro contato é imposto pelas alterações incluídas na nova versão de sua ferramenta de desenvolvimento preferida.

Um último problema ainda nesta fase é a integração gráfica do applet com o “design” do restante da página de hipertexto.

As próximas seções, comentam sobre o papel de cada um desses recursos no ambiente de desenvolvimento.

4.1.1 Bibliotecas de Programação Orientadas a Objetos

A linguagem Java possui um conjunto padronizado de classes. Entretanto, essas classes oferecem somente recursos básicos para a programação. A maioria das ferramentas de desenvolvimento é acompanhada de um conjunto extra de classes que facilitam a programação de aplicações específicas.

Por exemplo, um botão do pacote java.awt não permite a colocação de uma ícone (glyph) juntamente com o texto do botão. Os desenvolvedores da plataforma Windows estão acostumados com a existência de ícones nos botões, o que faz com que a maioria das ferramentas de desenvolvimento do

ambiente Windows ofereçam uma extensão da classe java.awt.button com essa funcionalidade. Essa característica é comum nos programas da plataforma Microsoft Windows, mas não é encontrada nas demais plataformas, como o Commom Desktop Environment, um dos ambientes gráficos do UNIX, ou o System 7, do Machintosh.

Outras extensões são mais complexas e úteis, como as classes de programação para interface com bancos de dados relacionais da Inprise.

Com a ferramenta certa e um pouco de sorte, o desenvolvedor deve encontrar a maior parte das necessidades da sua aplicações já implementada e disponível na biblioteca de classes que acompanha sua ferramenta.

Dois problemas são comuns nesta fase:

a) o desenvolvedor não conhece a biblioteca;

b) o desenvolvedor não sabe como combinar as classes disponíveis para criar sua aplicação.

A solução para o primeiro problema é escrever e ler muitos programas até que seja adquirida uma certa experiência no uso da biblioteca. O segundo problema pode ser contornado, em parte, pelo uso de um ambiente de programação visual.

4.1.2 Programação Visual

A programação visual utiliza a manipulação direta de elementos gráficos como forma de combinar componentes. Este tipo de programação é relativamente recente embora as linguagens visuais possuam amplo fundamento teórico de outras áreas, como a comunicação e a lingüística.

A principal vantagem da programação visual é o seu aspecto intuitivo e direto. Por ser intuitiva, possibilita ao desenvolvedor iniciante um primeiro contato com uma variedade de classes que será aos poucos dominada.

A programação visual é muito utilizada para a instanciação de objetos. Em algumas ferramentas mais sofisticadas pode ser feita a associação entre eventos e a ativação de métodos através de menus ou elementos visuais. A composição de elementos também é amplamente utilizada pela programação gráfica, e é essa característica que permite a criação de elementos mais complexos.

A linguagem Java define uma estrutura padronizada de classes que permite que diversos ambientes visuais possam manipulá-los. Este padrão é conhecido por Java Bean. Um bean é simplesmente uma classe Java que adota uma série de padrões nos nomes de métodos e propriedades e que possui uma ícone pré-definida para sua visualização dentro do ambiente de programação.

4.1.3 Navegadores

O código gerado pelo ambiente visual é gravado em um arquivo que contém os byte codes do applet. Este arquivo é transferido para a máquina cliente no momento da execução. Além de ser responsável pela requisição, recepção e execução do applet, o navegador provê um ambiente de execução controlado e toda a implementação da máquina virtual Java para a plataforma onde o navegador se encontra instalado.

O fato da máquina virtual encontrar-se presente no navegador diminui em muito o tamanho do applet. Entretanto essa abordagem possui um desvantagem séria: o applet deverá executar em uma versão de JDK compatível com aquela usada no ambiente de execução.

Os primeiros navegadores não incluíam a máquina virtual Java. Quando os navegadores se popularizaram, já incluíam a máquina virtual Java para o JDK 1.0. Mais tarde se verificou que a maior parte da plataforma instalada nas empresas não era adequada para a execução de navegadores mais complexos.

A Sun desenvolveu um adendo (plug-in) para as versões recentes dos navegadores que, uma vez instalado, permite ao navegador obter e instalar automaticamente novas versões da máquina virtual através da Internet.

Ainda assim, grande parte dos usuários não adotou as atualizações lançadas posteriormente, que incluíam as máquinas virtuais para as versões mais recentes do JDK. Com isso, quando uma aplicação é desenvolvida para a Internet, como o caso do applet da Telefonica, o desenvolvedor fica restrito as classes do JDK 1.0.

4.1.4 Utilização de ferramentas de prototipação

Na resolução do exemplo apresentado nesse tutorial, utilizou-se o Symantec Café para o desenho da interface. Com a ajuda desse ambiente visual e da biblioteca de classes da ferramenta, foi possível criar uma aplicação com uma interface muito semelhante a da versão final do applet.

Com a ajuda de ferramentas visuais, o desenvolvedor pode criar uma classe que herda as funcionalidades da classe applet e em um segundo momento, pode utilizar a nova classe para compor instâncias de outras classes.

O desenvolvimento com uso de ferramentas visuais facilita a colocação de componentes gráficos. Porém, para a codificação do comportamento da interface é necessário que o programador conheça os recursos da linguagem. Isso causa problemas ao programador devido a dificuldade em dominar as biblioteca de classes e a necessidade de conhecimento das bibliotecas que não possuem forma gráfica. Por esse motivo, ferramentas como VisualAge da IBM utilizam beans para representar na interface de programação elementos não visuais necessários para o desenvolvimento de sistemas.

No exemplo do applet da Telefonica, o código inicial obtido através do Symantec Café foi alterado de forma a garantir em primeiro lugar desempenho, que era a principal preocupação do cliente. Note que o código gerado por esse tipo de ferramentas não é muito otimizado. Além disso, embora a ferramenta gerasse toda o código relativo a interface gráfica, várias funcionalidades não podiam ser modeladas através do ambiente desenvolvimento oferecido pela ferramenta. A Figura 4.1 apresenta o relacionamento da classe Phone com as demais classes da biblioteca da ferramenta.

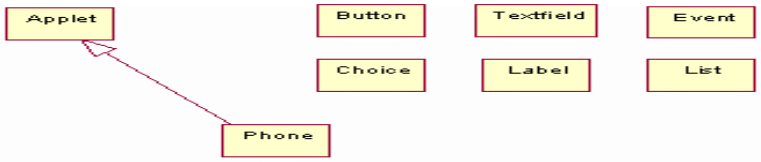

Figura 4.1 – Classes da aplicação (engenharia, prototipação).

A Figura 4.2 mostra a declaração da classe Phone criada pela ferramenta. Note que o desenvolvedor teve o cuidado de escolher apenas as classes provenientes da biblioteca padrão de Java. o método button1_ActionPerformed é acionado a cada vez que o botão button1 for pressionado.

```
import java.awt.*;
import java.applet.*;

public class Phone extends Applet
{
    public void init(){ }

    java.awt.TextField t1;
    java.awt.Choice c1;
    java.awt.TextField t2;
    java.awt.TextField c2;
    java.awt.TextField tx;
    java.awt.TextArea t4;
    java.awt.Button button1;
    java.awt.Label label1;
    java.awt.Label label2;
    java.awt.Label label3;
    java.awt.Label label4;
    java.awt.Label label6;
    java.awt.Button button2, button3, button4, button5;
    java.awt.List list1;

    void button1_ActionPerformed(java.awt.Event event) { }
}
```

Figura 4.2 – Métodos e campos da aplicação (engenharia, prototipação).

As classes oferecidas pela ferramenta geralmente são mais atrativas e completas, entretanto:

a) as classes podem ser familiares aos usuário de uma plataforma, mas completamente desconhecidos por usuários de outra plataforma;
b) o código dessas classes deve ser inserido no applet, uma vez que não fazem parte da máquina virtual oferecida pelos navegadores, aumentando o tamanho do applet e conseqüentemente aumentando o tempo de carga;
c) o uso dessas classes impede a utilização do applet gerado em uma outra ferramenta de prototipação;
d) a utilização de classes jdk 1.0 resolve o problema de compatibilidade entre navegadores, já que muitos usuários usam navegadores sem suporte as novas versões do jdk.

Neste ponto do desenvolvimento, o applet já possui a apresentação final de sua interface (Figura 4.4) conforme pode ser visualizado na página da Telefonica Celular (Figura 4.3). Observe como o applet é apenas um dos elementos da página.

Uma questão importante no desenvolvimento de applets, é que esses programas necessariamente serão embutidos em uma página HTML. É importante que o applet respeite o “design” gráfico pré-existente da página do seu cliente, principalmente em relação a tamanho e cores dos elementos gráficos. Nos casos em que não existem ainda as páginas onde o programa será executado, é aconselhável que a tarefa de projeto da página seja deixada para um profissional especializado em design gráfico.

Como foi dito, o applet é inserido através em uma página HTML e visualizado através de navegadores. Para tal utiliza-se tags de hipertexto <APPLET> e </APPLET>. Mais detalhes podem ser obtidos na referência sobre Java [ECK 98] [FLA 97] [SUN 99]. Nesta etapa da engenharia, a apresentação visual do applet já se encontra completa, é possível então passar para a fase de complementação.

TELEFÔNICA CELULAR - Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search N

Bookmarks Location: http://200.238.10.17/pin

Instant Message Members WebMail Co

Telefonica

CELULAR

- > Novidades
- > Produtos e Serviços
- > Preços e Tarifas
- > Mensagens Curtas
- > MoviStar Amigo
- > Atendimento ao Cliente
- > A Empresa
- > Lojas
- > Agentes Credenciados
- > Mapa de Cobertura
- > Mapa do Site

Através dele você passa a c
com uma nova maneira de s
comunicar com seus amigos
colegas de trabalho, fornece
dentre outros, de forma rápic
objetiva, utilizando a mais av
tecnologia em termos de
comunicação pessoal.

Clique AQUI para maiores

Figura 4.3 – Interface gráfica do applet já inserida na página Web.

4.2 COMPLEMENTAÇÃO

Nesta fase, o desenvolvedor deve fazer uso das classes disponibilizadas pela biblioteca de programação Java a fim de criar o comportamento desejado ao applet. A Figura 4.4 apresenta quais são as funcionalidades requeridas da aplicação.

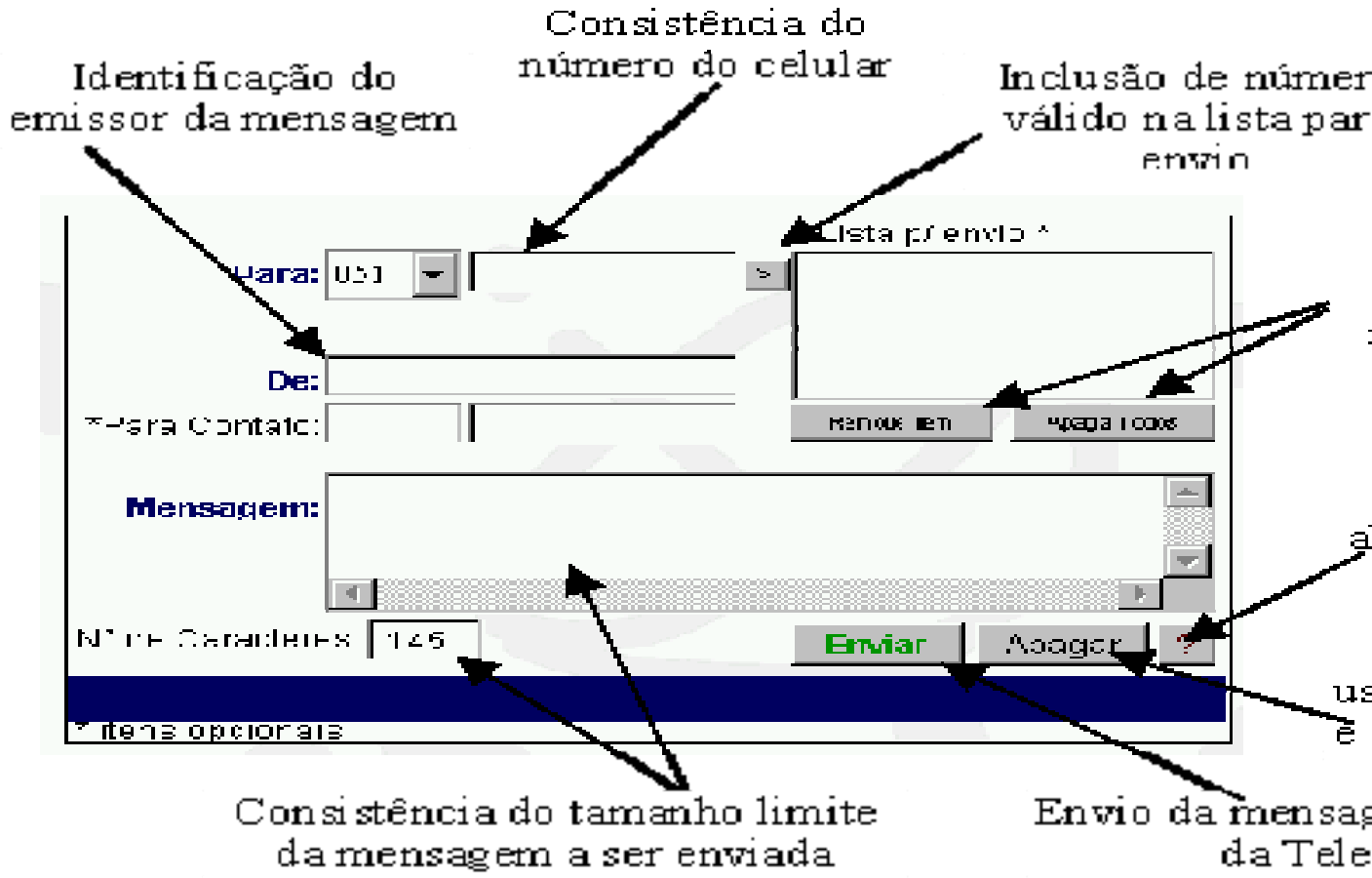

Figura 4.4 – Funcionalidades do applet.

Dois grupos de funcionalidades devem ser implementadas:

a) teste da consistência dos dados digitados na interface;
b) envio da mensagem ao servidor instalado na Telefonica.

O primeiro grupo foi implementado através da interceptação de eventos. O segundo exigiu o uso de três classes da biblioteca padrão de Java: DataOutputStream, DataInputStream e Socket. As classes envolvidas na aplicação são apresentadas na Figura 4.5.

Figura 4.5 – Classes da aplicação (engenharia, complementação).

4.2.1 Consistência dos dados

As funcionalidades se refletem na criação de novos atributos e métodos nessa fase da engenharia. A Figura 4.6 apresenta o esqueleto do código onde as classes criadas pelo desenvolvedor estão destacadas em negrito.
Dos atributos quase todos foram criados para controlar a conexão com o servidor. A exceção é a variável max utilizada para controlar o tamanho da mensagem a ser enviada.

Com relação aos métodos, *IsInsideList()* impede a inserção de um telefone repetido na lista de telefones. O método *ClearData()* remove os valores de todos os componentes da interface. *ListenServer()* e *CloseCom()* foram criados para realização da comunicação via sockets. *LastIoError()* retorna o valor verdadeiro se houve um erro na última comunicação com o servidor.

Por fim, *Validate()*, é uma função que determina se a mensagem não excedeu o tamanho máximo permitido pelo sistema de mensagens curtas.

Note-se que a maioria dos navegadores não permitem o uso de listeners. Por esse motivo o método *handleEvent()* da classe applet teve de ser sobreescrito, usando o modelo de callbacks do jdk 1.0. Nesse modelo de programação de evento, o *handleEvent()* é acionado toda vez que um evento ocorrer na interface. O parâmetro event do método informa que tipo de evento ocorreu. Na Tabela 4.1 são apresentados os atributos de Event utilizados e a Figura 4.7 apresenta trechos do método sobreescrito.

Tabela 4.1 – Alguns atributos da classe Event.

Atributo	Significado
target	objeto que sofreu o evento
id	tipo de evento (ACTION_EVENT ou KEY_PRESS, no nosso exemplo)
key	tecla associada ao evento KEY_PRESS

```
import java.awt.*;
import java.applet.*;
import java.net.*;
import java.io.*;

public class Phone extends Applet
{
  static public Socket soquete;
  static public DataInputStream in;
  static public DataOutputStream out;
  int portnumber;
  String host;
```

```
  String linex;
  int max=0;

  public boolean isInsideList() {…}
  public boolean LastIoError() {…}
  public void ClearData(){…}
  public int Validate(){…}
  public boolean ListenServer() {…}
  public void CloseCom(){…}
  public void init(){…}

  java.awt.TextField t1;
  java.awt.Choice c1;
  java.awt.TextField t2;
  java.awt.TextField c2;
  java.awt.TextField tx;
  java.awt.TextArea t4;
  java.awt.Button button1;
  java.awt.Label label1;
  java.awt.Label label2;
  java.awt.Label label3;
  java.awt.Label label4;
  java.awt.Label label6;
  java.awt.Button button2, button3, button4, button5;
  java.awt.List list1;

  public boolean handleEvent(Event event) {…}
  void button1_ActionPerformed(java.awt.Event event) {…}
  void button2_ActionPerformed(java.awt.Event event) {…}
  void button3_ActionPerformed(java.awt.Event event) {…}
  void button4_ActionPerformed(java.awt.Event event) {…}
  void button5_ActionPerformed(java.awt.Event event) {…}
}
```

Figura 4.6 – Métodos e campos da aplicação (engenharia, complementação).

O método handleEvent() é acionado a cada vez que um evento ocorre. O parâmetro event fornece dados sobre esses eventos. Este método deve retornar o valor lógico verdadeiro se o evento foi consumido (processado) ou o valor lógico falso se o evento não foi consumido. A classe java.applet.Applet apresenta uma implementação básica deste método que processa as ações normais de um interface, como o pressionar de botões e a digitação de caracteres.

```
public boolean handleEvent(Event event) {
  // se o botão button4 foi pressionado...
  if (event.target == button4 && event.id == Event.ACTION_EVENT) {
    button4_ActionPerformed(event);
    return true;
  }
...
  // limita o comprimento máximo do textfield t1 em 14 caracteres
```

```
  if (event.target == t1 && event.id == Event.KEY_PRESS) {
     if ( (t1.getText().length() > 14) || (Validate() >= 150) )
        return (event.key != '_');
     else
        if (event.key=='
')
           t2.requestFocus();
```