



**GOVERNO DO
ESTADO DO CEARÁ**

Secretaria da Educação

**ESCOLA ESTADUAL DE
EDUCAÇÃO PROFISSIONAL - EEEP**
ENSINO MÉDIO INTEGRADO À EDUCAÇÃO PROFISSIONAL

CURSO TÉCNICO EM INFORMÁTICA

POO/JAVA



GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação

Governador
Cid Ferreira Gomes

Vice Governador
Domingos Gomes de Aguiar Filho

Secretária da Educação
Maria Izolda Cela de Arruda Coelho

Secretário Adjunto
Maurício Holanda Maia

Secretário Executivo
Antônio Idilvan de Lima Alencar

Assessora Institucional do Gabinete da Seduc
Cristiane Carvalho Holanda

Coordenadora da Educação Profissional – SEDUC
Andréa Araújo Rocha



GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação

Coordenação Técnica Pedagógica

Renanh Gonçalves de Araújo

Equipe de Elaboração

Adriano Gomes da Silva
Cíntia Reis de Oliveira
Fernanda Vieira Ribeiro
Francisco Aislan da Silva Freitas
João Paulo de Oliveira Lima
Liane Coe Girão Cartaxo
Mirna Geyla Lopes Brandão
Moribe Gomes de Alcântara
Niltemberg Oliveira Carvalho
Paulo Ricardo do Nascimento Lima
Renanh Gonçalves de Araújo
Renato William Rodrigues de Souza

Colaboradores

Maria Analice de Araújo Albuquerque
Maria Danielle Araújo Mota
Sara Maria Rodrigues Ferreira Feitosa

Programação Orientada Objetos / Java

MANUAL DO (A) ALUNO (A)

**Janeiro DE 2013
FORTALEZA/CEARÁ**

Apresentação

No intuito de deixar claro à (o) professor (a) o que é esperado do aluno ao final da disciplina, este manual propõe os objetivos de aprendizagem referentes ao tema, acompanhado do conteúdo da disciplina. Disponibiliza uma bibliografia para o (a) professor (a), subsidiando-o (a) para aprofundar os debates, bem como, uma bibliografia de referência.

Elaborado no intuito de qualificar o processo de formação, este Manual é um instrumento pedagógico que se constitui como um mediador para facilitar o processo de ensino-aprendizagem em sala de aula.

A disciplina de POO/Java faz parte do eixo de programação Java para Desktop e segue um fio condutor iniciado pela Lógica de Programação e será seguida pela disciplina de Banco de Dados, e posteriormente por Laboratório de Software.

O principal objetivo deste manual é apresentar para o aluno uma alternativa para o processo de desenvolvimento de software orientado a objetos, embora existam outros processos, outros padrões, em fim, outras alternativas que, certamente, eles irão se deparar no mundo do trabalho. Esta experiência o fará entender a importância do processo de desenvolvimento de software para termos produtos eficiente e eficaz.

Sendo assim, temos a apresentação da UML, na Fase I do manual, ferramenta de documentação de projetos de software mais usada pelo mercado atualmente. Na Fase II, descrevemos alguns itens complementares da linguagem Java que não foram abordados no manual de Lógica de Programação, mas julgamos importantes. Em seguida temos a Fase III, onde teremos os conceitos de Orientação a Objetos. Por fim, na Fase IV, iremos implementar um projeto usando padrão MVC e interface usando Java.swing, como a essa altura os conceitos de banco de dados ainda não foram aprendidos, usaremos uma simulação do banco baseada em vetores de objetos.

É importante que o (a) professor (a) compreenda o propósito do método do curso, e assim, se aproprie do conteúdo e da metodologia proposta por meio das atividades pedagógicas, fazendo um estudo cuidadoso deste Manual e buscando aperfeiçoar sua didática para conduzir com sucesso as atividades propostas. Assim, Esperamos contribuir com a consolidação do compromisso e envolvimento de todos (professores e alunos) na formação desses profissionais.

Apostila

Sumário da Apostila

FASE I – A UML COMO FERRAMENTA DE PROGRAMAÇÃO ORIENTADA A OBJETOS	7
1 INTRODUÇÃO A SISTEMAS	7
1.1 MODELAGEM DE SISTEMAS	7
1.2 DADO, CONHECIMENTO E INFORMAÇÃO.....	8
1.3 O QUE É UM SISTEMA.....	9
1.4 COMO CONSTRUIR UM SISTEMA DE INFORMAÇÃO BASEADO EM COMPUTADOR.....	10
1.5 O PAPEL DO ANALISTA DE SISTEMAS	11
1.6 FASES DO DESENVOLVIMENTO DE SISTEMAS	12
1.6.1 CONCEPÇÃO DO SISTEMA.....	12
1.6.2 ESTUDO DA VIABILIDADE	12
1.6.3 PROCESSO DE ANÁLISE.....	12
1.6.4 PROJETO DO SISTEMA.....	13
1.6.5 PROJETO DETALHADO.....	13
1.6.6 IMPLEMENTAÇÃO.....	13
1.6.7 IMPLANTAÇÃO E MANUTENÇÃO	14
2 REQUISITOS.....	15
2.1 INTRODUÇÃO	15
2.2 LEVANTAMENTO DE REQUISITOS	15
2.2.1 ENTREVISTA	15
2.2.2 PESQUISA	16
2.2.3 QUESTIONÁRIO.....	16
2.2.4 REUNIÕES.....	16
2.2.5 OBSERVAÇÕES.....	16
2.2.6 ANÁLISE DE REQUISITOS.....	16
3 HISTÓRIA E EVOLUÇÃO DA UML	18
4 DIAGRAMAS ESTRUTURAIS DA UML.....	19
4.1 DIAGRAMA DE CLASSE.....	19
4.2 DIAGRAMA DE OBJETO.....	21
4.3 DE COMPONENTES	21
4.4 DE IMPLANTAÇÃO	22
4.5 DE PACOTES	22

4.6 DE ESTRUTURA.....	23
5 <u>DIAGRAMAS COMPORTAMENTAIS DA UML.....</u>	23
5.1 DE CASO DE USO	23
5.2 DIAGRAMA DE SEQUÊNCIA	24
5.3 DIAGRAMA DE MÁQUINA DE ESTADO	25
5.4 DE ATIVIDADES.....	26
6 <u>RELACIONAMENTOS EM UML</u>	27
6.1 RELACIONAMENTO DE DEPENDÊNCIA	27
6.2 RELACIONAMENTO DE ASSOCIAÇÃO.....	28
6.3 RELACIONAMENTO DE GENERALIZAÇÃO.....	29
<u>FASE II – PROGRAMAÇÃO NA LINGUAGEM JAVA.....</u>	31
7 <u>VISÃO GERAL DAS TECNOLOGIAS JAVA.....</u>	31
7.1 INTRODUÇÃO	31
7.2 TECNOLOGIAS JAVA	33
7.2.1 J2SE (STANDARD EDITION)	33
7.2.2 J2EE (ENTERPRISE EDITION)	33
7.2.3 J2ME (MICRO EDITION)	33
8 <u>CLASSES JAVA.....</u>	34
8.1 CLASSE STRING	34
8.2 CLASSE MATH	37
<u>FASE III – PROGRAMAÇÃO ORIENTADA A OBJETOS</u>	40
9 <u>ORIENTAÇÃO A OBJETOS.....</u>	40
9.1 INTRODUÇÃO	40
9.2 HISTÓRICO	41
9.3 FUNDAMENTOS DE ORIENTAÇÃO A OBJETOS.....	42
9.3.1 UMA ANALOGIA.....	42
9.3.2 OBJETO X CLASSE X INSTÂNCIA	43
9.3.3 MENSAGEM	45
9.3.4 ATRIBUTO.....	46
9.3.5 MÉTODO	47
9.3.6 CONSTRUTORES.....	48

<u>10 ENCAPSULAMENTO.....</u>	51
10.1 O QUE É ENCAPSULAMENTO?.....	51
10.1.1 MODIFICADORES DE ACESSO.....	51
10.1.2 ACESSO PÚBLICO (PUBLIC)	51
10.1.3 ACESSO PRIVADO (PRIVATE)	51
10.1.4 ACESSO PROTEGIDO (PROTECTED).....	51
10.1.5 MÉTODOS GET E SET	51
<u>11 HERANÇA.....</u>	55
11.1 O QUE É HERANÇA?	55
11.2 SUPERCLASSE E SUBCLASSE	55
11.3 HERANÇA MÚLTIPLA.....	55
11.4 CLASSE OBJECT	55
<u>12 POLIMORFISMO.....</u>	59
12.1 O QUE É POLIMORFISMO?	59
12.2 SOBRECARGA DE MÉTODOS.....	60
12.3 CLASSES E MÉTODOS ABSTRATOS.....	62
<u>FASE IV – PROJETO ORIENTADO A OBJETOS</u>	64
<u>13 CONHECENDO E ORGANIZANDO UM PROJETO NO PADRÃO MVC</u>	64
13.1 ENTENDO O PADRÃO MODEL-VIEW-CONTROLLER – MVC	64
<u>14 IMPLEMENTAÇÃO DO PROJETO.....</u>	66
14.1 IDENTIFICAÇÃO DO PROJETO	66
14.2 CRIANDO O PROJETO.....	66
14.3 CRIANDO OS PACOTES MVC	68
14.4 PACOTE MODEL.....	69
14.5 PACOTE VIEW.....	72
14.5.1 TELA DE LOGIN	72
14.5.2 TELA DE CADASTRO DE PRODUTOS.....	77
14.5.3 CODIFICANDO O BOTÃO OK DA TELA DE LOGIN	78
14.5.4 CODIFICANDO O BOTÃO CADASTRAR DA TELA DE PRODUTO.....	80
14.5.5 CODIFICANDO O BOTÃO BUSCAR DA TELA DE PRODUTO	83
14.5.6 CODIFICANDO O BOTÃO ALTERAR DA TELA DE PRODUTO	85
14.5.7 CODIFICANDO O BOTÃO EXCLUIR NA TELA DE PRODUTO	86
14.6 GERANDO UM ARQUIVO EXECUTÁVEL DO SISTEMA	87

<u>15 ASSUNTOS COMPLEMENTARES</u>	<u>89</u>
15.1 TRATAMENTO DE EXCEÇÕES EM JAVA.....	89
15.2 CONVENÇÕES DE CÓDIGO JAVA.....	94
<u>16 REFERÊNCIAS BIBLIOGRÁFICAS</u>	<u>98</u>
<u>17 EXERCÍCIOS PROPOSTOS.....</u>	<u>100</u>

FASE I – A UML COMO FERRAMENTA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

1 Introdução a Sistemas

1.1 Modelagem de Sistemas

Modelar é:

- ✓ Representar de forma gráfica ou textual partes reais ou imaginárias do sistema.
- ✓ Por no papel a concepção que se tem de como funcionará o sistema concebido.
- ✓ Documentar de forma gráfica ou em texto um sistema existente (engenharia reversa).

Por que é importante modelar?

Se observarmos tudo que será construído primeiro passa por uma fase de modelagem, isso ocorre em todas as áreas. Exemplo: um engenheiro primeiro constrói a planta de uma casa e só então começa a construção do imóvel. A planta o guiará durante toda a construção.

Em desenvolvimento de sistema não é diferente primeiro o analista deve estruturar toda a modelagem do sistema, conversando com o usuário, colhendo informações e requisitos do sistema, depois ele deve desenhar o sistema para realizar esse fase ele precisará de uma linguagem padrão para desenhar o sistema, essa linguagem deve ser conhecida por todos da área, pois assim qualquer pessoa que trabalhe com desenvolvimento de sistemas pode dar andamento ao projeto. A última fase da modelagem é a aprovação do projeto, após isso passamos para etapa de construção do software. Note que assim como a planta da casa para o engenheiro o guiará durante toda a construção do imóvel, para a equipe que desenvolve o software a modelagem do sistema será importante em todas as fases de desenvolvimento.

Inicialmente, vamos definir alguns conceitos básicos para posteriormente assimilarmos o conteúdo de Análise de sistemas.

O que é dado?

“Dados são itens referentes a uma descrição primária de objetos, eventos, atividades e transações que são gravados, classificados e armazenados, mas não chegam a ser organizados de forma a transmitir algum significado específico” (Turban, McLean e Wetherbe, 004, pg. 63).

O que é uma informação?

“Informação é todo conjunto de dados organizados de forma a terem sentido e valor para seu destinatário. Este interpreta o significado, tira conclusões e faz deduções a partir deles. Os dados processados por um programa aplicativo têm uso mais específico e maior valor agregado do que aqueles simplesmente recuperados de um

banco de dados. Esse aplicativo pode ser um sistema de gerenciamento de estoques, um sistema de matrículas online de uma universidade, ou um sistema de Internet para compra e venda de ações". (Turban, McLean e Wetherbe, 2004, pg. 63).

O que é conhecimento?

"Conhecimento consiste de dados e informações organizados e processados para transmitir compreensão, experiência, aprendizado acumulado e técnica, quando se aplicam a determinado problema ou atividade. Os dados processados para extrair deduções críticas e para refletir experiência e perícia anteriores fornecem a quem os recebe conhecimento organizacional, de alto valor potencial". (Turban, McLean e Wetherbe, 2004, pg. 63).

1.2 Dado, Conhecimento e Informação.

Observe nas imagens abaixo a distinção entre Dados, Informações e Conhecimento:

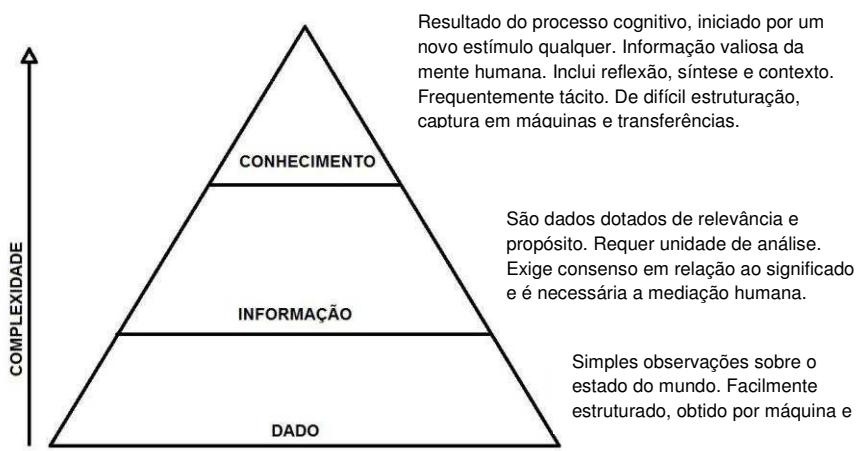


FIGURA: Dado, Informação e Conhecimento (Fanderuff e Gall, 2005)

Figura 1 - Dado, Informação e Conhecimento

Tabela 1 - Dado, Informação e Conhecimento (1)

DADO	INFORMAÇÃO	CONHECIMENTO
Simples observações sobre o estado do mundo	Dados dotados de relevância e propósito	Informação valiosa da mente humana. Inclui reflexão, síntese, contexto.
Facilmente estruturado	Requer unidade de análise	De difícil estruturação
Facilmente obtido por máquinas	Exige consenso em relação ao significado	De difícil captura em máquinas
Frequentemente quantificado	Exige necessariamente a mediação humana	Frequentemente tácito
Facilmente transferível		De difícil transferência



Exemplo

Através dos conceitos acima, já podemos ter uma ideia do significado de dados, informações e conhecimentos para um Sistema de Informação. Vamos analisar um exemplo:

Milena é aluna de uma EEEP. Assim como todos os demais, possui um boletim com notas de diversas disciplinas. Essas notas nada mais são do que números que de alguma forma foram friamente calculados, gerados e impressos em um papel através de uma programação. As notas ali contidas são nada mais do que dados.

Se analisarmos seu boletim, podemos observar que todas as suas notas são boas, pois estão bem acima da média. Contextualizando esses dados, obtemos uma informação: Milena é uma ótima aluna.

Indo mais além, analisando o seu comportamento, temos que ela é uma aluna proativa em sala de aula, administra bem o seu tempo, utiliza as aulas de estudo para de fato estudar e pratica a TESE como sua filosofia de vida. Contextualizando essas informações chegamos a um conhecimento sobre a pessoa de Milena.

1.3 O que é um sistema

Para termos uma real compreensão do que é um Sistema de Informação, faz-se necessário anteriormente entender o que é um Sistema. Nesse sentido, é válida a definição apontada no site Wikipédia, em junho de 2012: “Sistema (do grego sietemiu) é um conjunto de elementos interconectados, de modo a formar um todo organizado. É uma definição que acontece em várias disciplinas, como biologia, medicina, informática, etc. Vindo do grego o termo “sistema” significa “combinar”, “ajustar”, “formar um conjunto””.

Segundo Chiavenato (1999) e Oliveira (2002), o sistema apresenta os seguintes componentes:

- ✓ Objetivos: é a finalidade para o qual o sistema foi criado;
- ✓ Entradas do Sistema: é o que inicia o sistema, traz a informação para a operação do sistema;
- ✓ Processamento: fenômeno que realiza as mudanças, é o mecanismo que converte as entradas em saídas;
- ✓ Saídas do Sistema: são os resultados do processamento.
- ✓ Retroalimentação ou feed back do sistema: é a informação gerada pelo sistema que informa sobre o comportamento do mesmo;
- ✓ Ambiente: é o meio que envolve externamente o sistema.

De posse da definição podemos definir o Sistema de Informação ou simplesmente SI, como sendo um conjunto organizado de dados, cujo elemento principal é a informação. Sua função principal é o armazenamento, tratamento e fornecimento de informação que de forma organizada servem de apoio a funções ou processos de uma

empresa, por exemplo. Um SI não necessita, necessariamente, ser computadorizado. As fichas organizadas dos alunos em uma academia de musculação formam, por exemplo, um sistema de informações.

Os sistemas de Informação podem ser divididos em subsistemas e, em muitos modelos aparecem dois deles, sendo um com processos, pessoas, documentos e informações, enquanto o outro apresenta os meios automatizados como, por exemplo, as máquinas, redes de comunicação e computadores. Ao primeiro chamamos de subsistema social e o segundo denomina-se subsistema automatizado. Do apresentado até aqui fica fácil perceber que o Sistema de Informações é conteúdo bastante amplo, já que inclui pessoas, dados, processos, programas (softwares), maquinários (hardware), etc.

Todos os aspectos que envolvem o Sistema possuem importância significativa, já que funcionam como uma verdadeira engrenagem, ou seja, uma fase depende da outra, uma peça ajuda a outra, enfim, todos se complementam. Assim, hardware, software, fator humano e qualidade das informações, são igualmente válidos para o funcionamento com excelência.

Tomemos por exemplo uma lanchonete que buscou implantar um Sistema de Informação com o intuito de garantir maior agilidade no atendimento dos clientes. Para tanto pensou em um cardápio fixo, comprou computadores eficazes e alocou código de barras nas embalagens. Aparentemente estas medidas poderiam sim agilizar o atendimento, porém, se os colaboradores não forem capacitados para utilizar as ferramentas, certamente ao invés de garantir maior agilidade no atendimento, pode até mesmo causar maior embaraço.

Os sistemas de informação são poderosas ferramentas e, nesta concepção apresentam diversas aplicações e benefícios, dentre eles podemos citar: organizar/incrementar a produtividade; fortalecer estratégias de marketing; formatar a qualificação dos colaboradores; estabelecer rotinas de controle de produtos (entrada e saída – preços – lucros, etc.).

1.4 Como construir um Sistema de informação baseado em computador

Dizemos que um Sistema de informação é baseado em computador quando ele realiza parte - ou mesmo todas – as tarefas desejadas por meio da computação. Para que um sistema de informação obtenha sucesso na realização de suas rotinas, é preciso que se entenda todos os processos e procedimentos relacionados à tarefa a ser executada. Ou seja, é indispensável que se tenha um convívio direto com os problemas e soluções diárias naquele contexto em que o sistema será implantado, pois esse conhecimento profundo é o que vai dar embasamento para o desenvolvimento de um Sistema de Informação que seja realmente eficiente.

Em posse de um objetivo, qual seja planejar e desenvolver a construção de um Sistema de Informação eficiente para determinada situação, devemos analisar outros fatores decisivos nesse processo. O ambiente no qual o sistema será implantado é um deles, e pode influenciar de forma direta ou indireta no funcionamento de um sistema.

Por exemplo, poderemos obter resultados completamente diferentes ao implantar um sistema on-line em uma cidade capital de estado, onde o acesso à internet é incorporado ao cotidiano dos supostos usuários, e em uma pequena cidade do interior onde o acesso à rede ainda é um privilégio restrito a poucos. Nesse caso, o fator internet é determinante para o sucesso ou insucesso do Sistema de Informação.

Outro fator que influencia de forma direta o desenvolvimento do sistema são os chamados recursos de sistema, que são os recursos indispensáveis à construção de um Sistema de Informação. São exemplos deles: dinheiro, máquinas, pessoas capacitadas, ambiente físico, papéis, etc.

Além dos fatores citados, outro a ser considerado é a análise dos dados relevantes para o Sistema de Informação. Tais dados devem ser cuidadosamente examinados, ponderados e utilizados de forma consistente, a fim de gerarem informações que de fato sejam úteis aos usuários. Um sistema que considera a entrada de dados inconsistentes em seu funcionamento, como data em branco ou preços com valores nulos, certamente não chegará a uma saída interessante.

Mas, afinal, a quem compete este trabalho?

1.5 O papel do Analista de Sistemas

O profissional da Tecnologia da Informação responsável por todo este trabalho é o Analista de Sistemas. Ele é, na prática, um solucionador de problemas e exerce uma função bastante complexa, que é concretizar em um software todo um sistema de informação.

Um bom analista de sistemas tem como principais características:

- ✓ Conhecimento teórico e prático de computação;
- ✓ Ampla visão organizacional e empresarial;
- ✓ Bom senso na tomada de decisões;
- ✓ Bom relacionamento interpessoal para lidar com todos envolvidos no projeto;



Figura 2 - Qualidades do analista de sistemas

“É muito difícil criar produtos usando grupos foco. Muitas vezes, as pessoas não sabem o que elas querem até que você mostre a elas.”

Steve Jobs

1.6 Fases do desenvolvimento de sistemas

Já sabemos o que é um sistema de informação, quais seus fatores principais e quem é o responsável por desenvolvê-lo. Vamos agora entender como se dá todo esse processo, desde o início da ideia até a conclusão de seus trabalhos.

Como vimos, a construção de um software é um processo complexo e para que se obtenha sucesso, seu desenvolvimento deve seguir uma exigente metodologia de trabalho. Vejamos cada uma das fases desse processo:

1.6.1 Concepção do Sistema

É uma fase de descobertas. Requer que todos os envolvidos – desenvolvedores e usuários - mantenham postura colaborativa para que os requisitos levantados sejam o mais próximo possível da realidade. Baseado nessas informações, o analista de sistemas consegue fazer um diagnóstico da situação e uma previsão de ações.

1.6.2 Estudo da Viabilidade

Após a apropriação do contexto na fase anterior, é feito um estudo para avaliar se o projeto é ou não viável de ser implementado, do ponto de vista organizacional, tecnológico e financeiro. A decisão da continuidade ou não do projeto cabe aos analistas e aos gestores da organização que será beneficiada.

Nessa fase, são analisados os seguintes questionamentos:

- ✓ O sistema poderá contribuir de fato para realização dos objetivos previstos?
- ✓ O sistema poderá ser implementado, apesar de restrições de cunho tecnológico, organizacional e temporais?
- ✓ Existe outro modo eficiente de realização dessas tarefas sem a necessidade de criação desse sistema?
- ✓ O sistema atual realmente pode resolver os problemas não solucionados pelos anteriores?
- ✓ É possível sua integração a sistemas já em funcionamento?

Caso os responsáveis optem pela continuidade do projeto, o mesmo será submetido à fase seguinte. Caso contrário, o ciclo termina aqui.

1.6.3 Processo de Análise

Essa etapa consiste em realizar um levantamento de dados e de fatos, a fim de entender o que realmente precisa ser feito para solucionar o problema. O contato entre analista de sistemas e usuários deve se estreitar nesse momento, pois é preciso um entendimento detalhado e técnico por parte do analista. Caso já exista algum sistema em funcionamento, este é o momento em que o analista se apropria de maiores detalhes do mesmo.

Deverá ser criado um modelo lógico do sistema, constituído, entre outros, do diagrama de fluxo de dados, dicionário de dados e principais algoritmos.

Até o término dessa fase, já devem ser de conhecimento do analista os objetivos principais do sistema em questão, quais setores da empresa serão impactados pelo sistema, como este deverá funcionar, qual será o fluxo de dados, quais serão os arquivos utilizados e a forma como serão atualizados, qual setor será responsável pela população do sistema, qual o prazo para que esses dados sejam processados e apresentado algum resultado aos usuários.

1.6.4 Projeto do Sistema

Nesse momento, o analista propõe soluções para o problema baseado em conclusões das fases anteriores. Para cada alternativa sugerida, apresenta-se aos gestores da organização um diagrama de fluxo de dados, para que seja feita uma escolha baseada em custos e benefícios. O modelo lógico foi sendo transformado em modelo físico.

Ao final da fase, já deverá estar definido pelo analista um relatório contendo qual tipo de banco de dados será adotado pelo sistema, os arquivos que serão utilizados, a definição dos arquivos de dados e seus layouts, relatórios a serem emitidos e os tipos de dispositivos de armazenamento de dados.

1.6.5 Projeto Detalhado

Tomada a decisão, é hora de dar início a implementação do sistema. O analista de sistemas já sabe o que deve ser feito, tecnicamente, para dar inicio a esse processo.

Nesse momento, os relatos dos usuários ainda podem ser de grande valia para a condução do sistema. O analista de sistemas deve definir quais programas irão compor o sistema, e produzir as especificações referentes a cada um deles, especificações essas que serão posteriormente entregues e codificadas pelos programadores.

Exemplos de especificações definidas nesta fase:

- ✓ Aprimoramento do fluxo de dados da alternativa escolhida;
- ✓ Identificação do banco de dados ou arquivos a serem utilizados;
- ✓ Detalhamento de arquivos de dados e seus layouts;
- ✓ Criação de layouts de relatórios a serem emitidos pelo sistema;
- ✓ Especificação de todos os programas que irão compor o sistema;
- ✓ Revisão na estimativa de custos;
- ✓ Preparação da documentação inicial do sistema;
- ✓ Definição dos tipos de dispositivo de armazenamento;
- ✓ Elaboração de um plano de testes;

1.6.6 Implementação

A fase de implementação é quando de fato o programa começa ser construído na plataforma escolhida, tendo como base as especificações produzidas. É, de todas, a fase mais cara.

Os programas que se referem ao gerenciamento do sistema em geral deverão ser os primeiros a serem construídos, e só após, os demais.

Inicia-se, após a finalização dos programas, os testes. Todo o sistema deve ser testado exaustivamente antes de ser posto em funcionamento. Ainda nesta fase, os erros detectados na homologação (testes) são corrigidos e testados novamente.

Uma vez sem erros, a equipe deve dedicar-se a escrever os manuais do usuário, um tutorial contendo todas as informações necessárias para seu funcionamento.

1.6.7 Implantação e Manutenção

Finalmente, chegamos à implantação e manutenção, última fase da criação de um sistema. Implantar significa colocar o sistema em fase de operação, ou seja, realizar as instalações necessárias para o uso do sistema no local predefinido. Uma vez implantado, os usuários deverão ser treinados a operar corretamente o sistema.

Ao ser submetido ao uso real, o sistema ainda poderá apresentar alguma falha, portanto a equipe de analista e programadores ainda pode realizar ajustes.

Agora, é apenas manter em pleno funcionamento, desenvolvendo pequenas melhorias e adequações quando necessário.

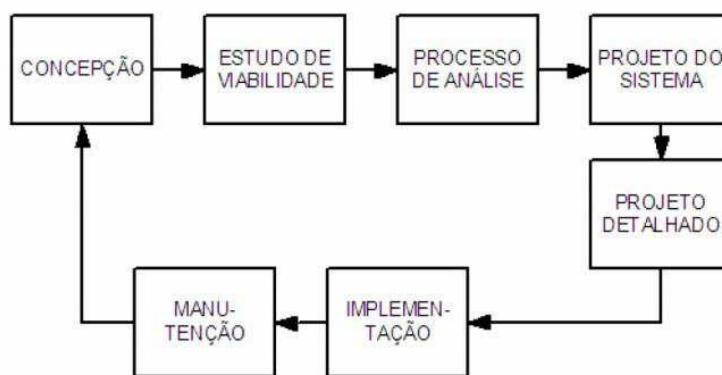


Figura 3 - Fases do desenvolvimento de sistemas

2 Requisitos

2.1 Introdução

Os requisitos de um sistema são os elementos que a equipe de desenvolvimento precisa conhecer para que o software seja desenvolvido. São as necessidades que deverão ser atendidas pelo sistema de informação que será proposto. Em posse dessas informações, o analista tem propriedade para propor soluções, especificar e desenvolver um sistema de informação. Segundo Sewbok, “Um requisito de software é uma propriedade que deve ser implementada para resolver um problema do mundo real”.

Acontece que tomar conhecimento desses requisitos é uma tarefa que, na prática, não é tão simples como parece. Muitas vezes os próprios gestores e usuários não conseguem expressar claramente seus objetivos. Por desconhecerem os procedimentos técnicos que regem um Sistema de Informação, muitas vezes eles ocultam informações de extrema importância, fundamentais para o planejamento da solução.

Nesse ponto, a atuação do analista de sistemas é decisiva para o sucesso ou não do trabalho. Dificilmente todas essas informações irão estar reunidas em uma pessoa só, portanto o analista tem que conversar com pessoas de todos os setores que irão utilizar o sistema, pois precisa ter uma visão de todas as funcionalidades que irá implementar. É preciso conhecer a dinâmica de trabalho de cada envolvido. Por exemplo, dificilmente o gerente de vendas de uma empresa saberá com precisão como o almoxarifado controla a saída de produtos, ou como exatamente a cozinha mantém o controle sobre seu estoque, portanto, é atribuição do analista capturar esse processo pessoalmente.

2.2 Levantamento de requisitos

Como vimos, o analista de sistemas é o responsável intermediar e identificar os requisitos. Para isso, ele dispõe de algumas técnicas que o auxiliam a estreitar as relações com os usuários, visando obter dos clientes o máximo de informações úteis. Entre as técnicas utilizadas para este processo, podemos destacar, entre outras, as seguintes:

2.2.1 Entrevista

É uma das formas mais eficientes de se obter as informações desejadas. Não existe uma fórmula ou receita certa para a sua condução, mas, para obter um melhor resultado, o analista deve atuar de forma racional e metódica, com certa flexibilidade, porém, sem improvisações desnecessárias.

O analista deve ter manter na entrevista um clima amistoso, deixando o entrevistado à vontade. Ele deve entender que a entrevista não é um julgamento, pois se por algum motivo se sentir acanhado, poderá não prezar pela veracidade das informações.

Outros fatores importantes em uma entrevista são: bom relacionamento interpessoal do entrevistador, habilidade ao se comunicar e humildade.

2.2.2 Pesquisa

Embora a entrevista seja uma técnica altamente eficaz, não deve ser a única fonte de levantamento de requisitos. Outra forma de se obter dados interessantes é a pesquisa. O analista poderá analisar documentos e relatórios gerados em diversos setores, além de arquivos, e baseado neles tirar outras conclusões, que podem vir a somar com o resultado já obtido na entrevista.

2.2.3 Questionário

Quando o sistema envolve um número muito grande de usuários, a aplicação de questionários é uma boa estratégia de pesquisa. Com perguntas sucintas, objetivas e sem dubiedade, é possível observar o padrão de respostas e dele retirar pertinentes conclusões.

2.2.4 Reuniões

São encontros com a participação de alguns usuários envolvidos, para discussões em grupo. Deve ser conduzida de forma hábil pelo mediador, mantendo sempre o foco, qual seja a discussão sobre as rotinas da organização para uma maior percepção dos requisitos por parte do analista.

2.2.5 Observações

Mesmo utilizando técnicas como entrevista ou reunião, algumas informações podem passar despercebidas. Observar o comportamento e o ambiente de trabalho dos usuários pode ser uma técnica eficaz para perceber alguma dessas informações que porventura não foi levada em consideração em outro momento.

2.2.6 Análise de Requisitos

Requisitos Funcionais

Requisitos funcionais são aqueles que descrevem funcionalidades que o sistema deve ter para atender às expectativas dos usuários. Por exemplo:

- ✓ O software deverá calcular a média de notas dos alunos;
- ✓ O software deverá enviar, a cada semestre, um email para os pais com o boletim de seus filhos em anexo;
- ✓ O software deverá emitir relatório de compras quinzenalmente;
- ✓ O software deverá alertar ao chefe do restaurante quando o estoque estiver abaixo do estabelecido.

Requisitos Não Funcionais

Já os requisitos não funcionais são aqueles que descrevem as características do sistema, como usabilidade, desempenho, confiabilidade, custo, etc. São exemplos de requisitos não funcionais:

- ✓ O acesso às funcionalidades de gestão deve ser restrito aos diretores da empresa;
- ✓ Se o tempo de resposta ultrapassar 30 segundos, redirecionar para página de erro;
- ✓ O software deverá ser desenvolvido para sistema operacional Linux;
- ✓ O prazo de entrega do sistema ao cliente deve ser de 2 anos.

Requisitos do usuário

Requisitos de usuário são declarações escritas, em linguagem natural, das funcionalidades que o sistema oferece. No documento, deve constar basicamente a descrição dos requisitos funcionais, requisitos não funcionais, devendo conter inclusive as restrições operacionais do sistema. Este material destina-se ao cliente, portanto deve ser escrito em linguagem clara e compreensível, além de conter tabelas e diagramas, visto que os usuários não possuem conhecimentos técnicos em desenvolvimento de sistemas.

A quem interessa:

- | | |
|--|--|
| <ul style="list-style-type: none">✓ Gerentes de clientes✓ Usuários finais do sistema✓ Engenheiros do cliente | <ul style="list-style-type: none">✓ Gerentes do fornecedor✓ Arquitetos do sistema |
|--|--|

Requisitos do sistema

Consiste em um documento estruturado, que estabelece detalhes das funcionalidades e restrições do sistema. Pode ser elaborado como forma de contrato entre o cliente e contratante.

A quem interessa:

- | | |
|---|---|
| <ul style="list-style-type: none">✓ Usuários finais do sistema✓ Engenheiros da organização cliente | <ul style="list-style-type: none">✓ Arquitetos do sistema✓ Desenvolvedores de software |
|---|---|

Especificação de Software

É um documento que consiste em uma minuciosa descrição do sistema, que vem a servir como base para a implementação do mesmo. As especificações de software são destinadas aos próprios desenvolvedores.

A quem interessa:

- | | |
|--|---|
| <ul style="list-style-type: none">✓ Engenheiros da organização cliente | <ul style="list-style-type: none">✓ Arquitetos do Sistema✓ Engenheiros de Software |
|--|---|

3 História e evolução da UML

A UML é uma linguagem para modelagem de softwares, ela é utilizada no processo de definição do software, na análise e estruturação de como o programa será implementado. Ela permite que os desenvolvedores de software possam modelar seus programas de forma unificada, isso quer dizer que qualquer pessoa que entenda UML poderá entender a especificação de um software. O Objetivo da UML é descrever “*o que fazer*”, “*como fazer*”, “*quando fazer*” e “*porque deve ser feito*”.

Para modelar os sistemas a UML possui um conjunto de diagramas que devem ser utilizados em combinação para obter todas as visões do sistema.

A UML é utilizada para modelar sistemas escritos no padrão orientado a objetos, portanto para todas as definições existentes na orientação a objetos haverá um diagrama correspondente. Exemplo: Uma classe, um objetos, a comunicação dos objetos e etc.

Na criação da UML estão envolvidos três personagens: Jim Rumbaugh, Grady Booch e, Ivar Jacobson.

Jim Rumbaugh e Grady Booch combinaram dois métodos populares de modelagem orientada a objeto: Booch e OMT (Object Modeling Language), mas tarde, Ivar Jacobson, o criador do método Objectory uniu-se aos dois para a concepção da primeira versão da linguagem UML (Unified Modeling Language).

Em 1997 a UML foi adotada como padrão pela OMG (Object Management Group). Que é uma organização internacional que aprova padrões abertos para aplicações orientadas a objetos.

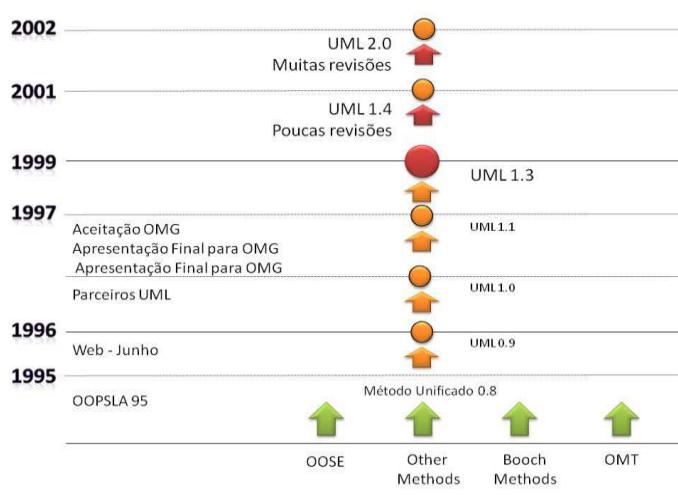


Figura 4 - Evolução da UML

A Figura 4 mostra a evolução da UML. Em 1995 ocorre a união de vários métodos de modelagem como OOSE, BOOCH, OMT e outros.

- I. Em 1996 surge a UML 0.9
- II. Em 1997 surge a UML 1.0 que no mesmo ano evoluiu para a UML 1.1. Nesse ano a OMG que é uma organização mundial de padrões orientados a objetos reconhece a UML.

III. Em 2001 a UML passa por revisões e é lançado a UML 1.4

IV. Em 2002 a UML é reestruturada e da origem a UML 2.0.

A UML divide seus diagramas em Estruturais e Comportamentais, como demonstrado pela Figura 5. No sentido horário, a partir do Diagrama de Caso de Uso, até o Diagrama de Interação, temos os diagramas Comportamentais. Os demais são os Diagramas Estruturais.

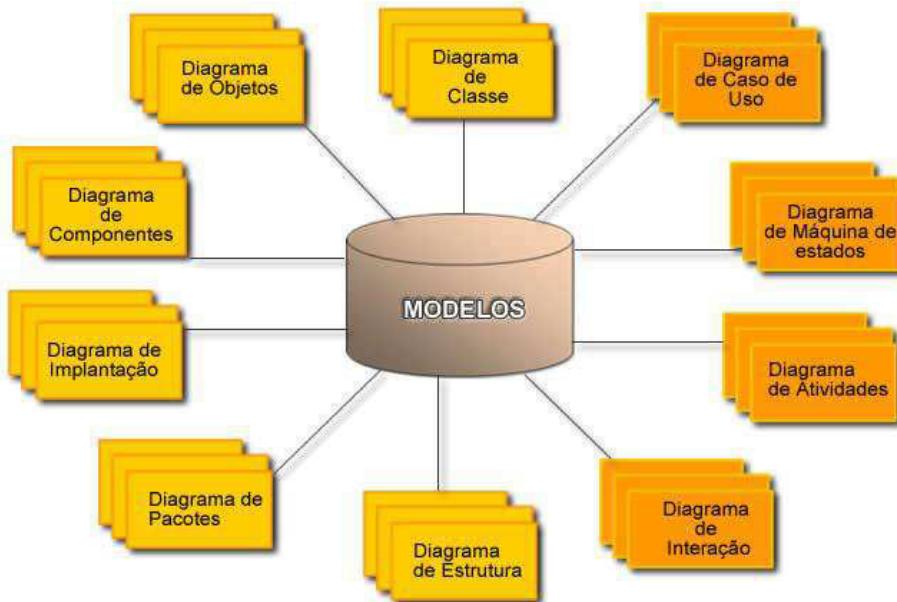


Figura 5 - Estrutura de diagramas UML

Aqui vamos estudar apenas alguns diagramas, em toda a apostila você irá encontrar exemplos com codificação em Java e seu respectivo diagrama. Então é muito importante que entendam os diagramas estudados e saiba reconhecê-los.



Exercícios Propostos

4 Diagramas Estruturais da UML

Usados para visualizar, especificar, construir e documentar aspectos estáticos de um sistema.

4.1 Diagrama de Classe

Este diagrama é fundamental e é o mais utilizado na UML, serve de apoio aos outros diagramas. O Diagrama de Classe mostra o conjunto de classes com seus atributos e métodos e os relacionamentos entre classes. Em todo sistema baseado na orientação a objetos as um dos princípios fundamentais são as classes que definem o sistema.

Uma classe deve ser desenhada com um retângulo dividido em três partes:

- ✓ Parte Superior: fica o nome da classe. Habitualmente escreve-se o nome da classe no singular, com a 1^a letra em maiúscula.

- ✓ Parte Central: ficam as propriedades ou atributos da classe (variáveis), já sendo identificado o tipo do atributo, do seu lado esquerdo. Para descobrir os atributos de uma classe verifique quais as características da classe deseja guardar daquela classe no sistema. Note que às vezes uma mesma classe pode ter atributos diferentes dependendo do sistema. Note que atributos são definidos ao nível da classe, enquanto que os valores dos atributos são definidos ao nível do objeto.
- ✓ Parte Inferior: ficam os métodos que a classe pode realizar (ações), já sendo definidos seu retorno e parâmetros. Os métodos são definidos ao nível da classe, enquanto que a invocação desses métodos é definida ao nível do objeto.

Veja na Figura 6 as partes da definição de uma classe.

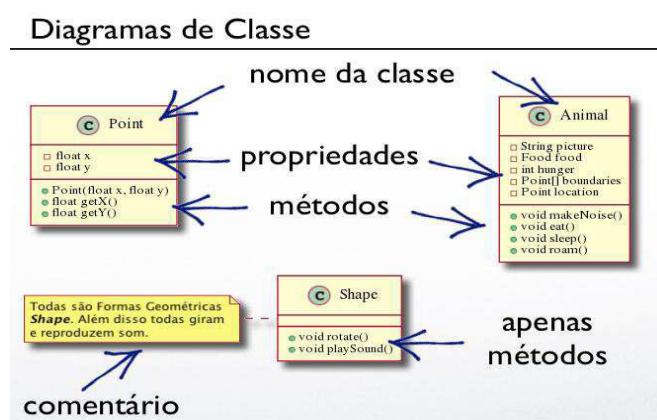


Figura 6 - Partes do diagrama de classe

Para identificar a visibilidade de atributos e métodos é usada uma sinalização, chamada de modificadores de acesso, do lado esquerdo de cada elemento. Veja na tabela abaixo (Tabela 2) a representação para os modificadores usados na linguagem Java.

Tabela 2 - Modificadores de Acesso em Java e UML

Símbolo	Modificador de Acesso	Representação
~	Default ou Package-Private	Conhecido como package em UML, não há palavra chave associada na linguagem Java e é usado quando não se identifica um modificador no diagrama.
-	Private	Visível somente pelos objetos criados pela classe
#	Protected	Visível somente pelos objetos criados pela classe ou suas subclasses
+	Public	Visível por todos

Também é possível inserir um comentário no diagrama, para isso temos um símbolo próprio da UML, um cartão com a ponta superior direita dobrada, direcionado para o item que requer comentário.

Observe no diagrama abaixo a comunicação entre três classes do sistema



Exemplo 1:

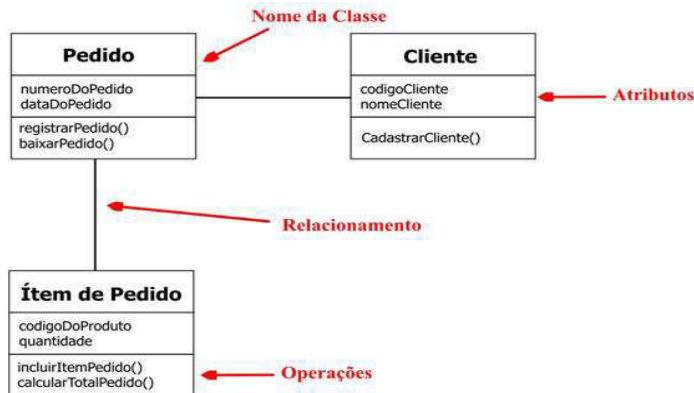


Figura 7 - Diagrama de Classe



Exercícios Propostos

4.2 Diagrama de Objeto

O diagrama de objeto está relacionado com o diagrama de classes e, é praticamente um complemento dele. Fornece uma visão dos valores armazenados pelos objetos de um diagrama de classe em um determinado momento da execução do processo do software. São muito úteis para exemplificar relacionamentos complexos entre objetos em determinado momento

Observe na imagem abaixo que o objeto da classe cliente guarda os valores armazenados pelo objeto no momento atual da execução.



Figura 8 - Diagrama de Objetos

4.3 De Componentes

Está associado à linguagem de programação e tem por finalidade indicar os componentes do software e seus relacionamentos.

Um componente é elemento externo de software que compõe programa que estamos implementando, é como um código externo que é reaproveitado. Esse componente pode ser substituído por outro componente, como, por exemplo: executáveis, bibliotecas, tabelas, pastas e documentos.

Na figura ao lado temos um exemplo de diagrama de componente, perceba comunicação entre componentes executáveis, bibliotecas (DLL), arquivos XML, imagens, tabelas e arquivos de texto. Tudo isso é considerado um componente do software.

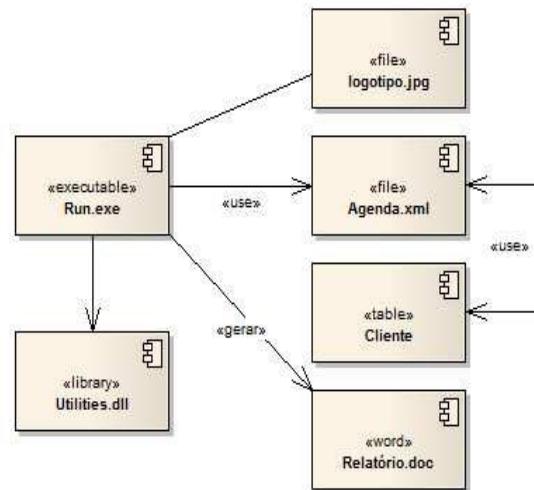


Figura 9 - Diagrama de Componentes

4.4 De Implantação

Diagrama de Implantação é usado para mostrar a organização do hardware e a ligação do software aos dispositivos físicos, Como processador, impressora, memória, disco; é necessário para que o engenheiro de software especifique a plataforma em que o sistema é executado.

O diagrama de implantação ao lado demonstra a plataforma física que o software será implantado.

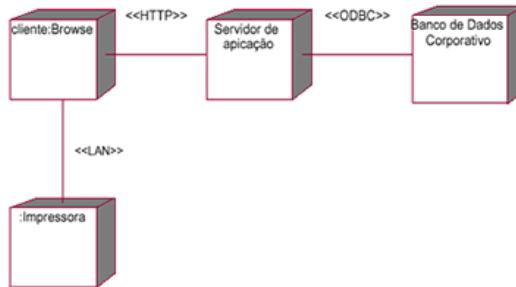


Figura 10 - Diagrama de implantação

4.5 De Pacotes

Representa os pacotes que o sistema está dividido, demonstrando a interação lógica entre as partes.

Observe a comunicação entre os pacotes de parte do sistema.

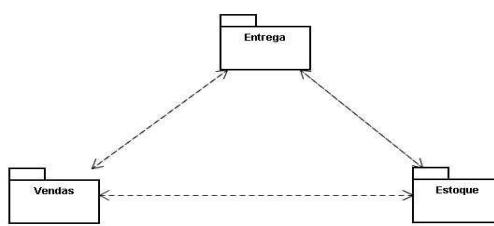


Figura 11 - Diagrama de pacotes

4.6 De Estrutura

Descreve a estrutura interna da comunicação entre classes em um dado momento do sistema. Ou seja, é muito similar ao diagrama de classes e, portanto não é muito utilizado.

O exemplo ao lado mostra como as instâncias das classes autor, submissão, tema e tipo colaboram entre si.

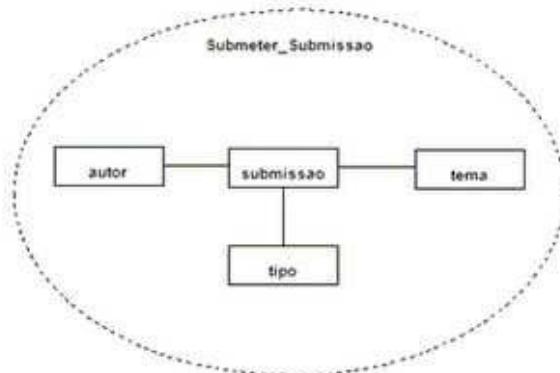


Figura 12 - Diagrama de estrutura



Exercícios Propostos

5 Diagramas Comportamentais da UML

Usados para visualizar, especificar, construir e documentar aspectos dinâmicos de um sistema.

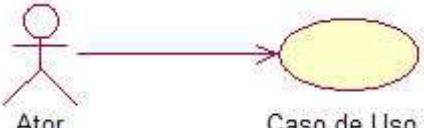
5.1 De Caso de Uso

É o diagrama comportamental mais utilizado na modelagem de sistemas, ele representa uma funcionalidade do sistema. A

Tabela 3, abaixo, mostra os elementos usados em um diagrama de caso de uso.

Tabela 3 – Elementos do Diagrama de Casos de Uso

Elementos	Como usar	Função
 Ator	Boneco com o nome do ator embaixo	Um ator é normalmente um usuário que dispara uma ação no sistema.

 Caso de Uso	Elipse com o nome do caso de uso embaixo	Um caso de uso define uma grande função do sistema.
 Autor	 Caso de Uso	Linha direcionada

O diagrama de caso de uso ajuda a definir a sequência de ações executadas por um ou mais atores e pelo próprio sistema para produzir resultados para um ou mais atores. Seu foco é identificar os objetivos do usuário, em vez das funções do sistema através da análise de como o ator, de forma externa, interage com as funcionalidades do sistema. Cada caso de uso no diagrama deve ter um documento correspondente determinando o seu comportamento.

Para identificar os atores do diagrama o analista deve primeiro identificar as áreas da empresa que serão afetadas pelo software. Existem algumas perguntas básicas que podem ajudar a descobrir os atores.

- 1) Que órgãos, empresas ou pessoas utilizarão o sistema?
- 2) Existem outros sistemas que irão se comunicar com o sistema que será construído?
- 3) Alguém deve ser informado de alguma ocorrência do sistema.

5.2 Diagrama de Sequência

Descreve a ordem temporal em que as **mensagens** são trocadas entre os **objetos**.

A função desse diagrama é estabelecer como os objetos interagem e seus relacionamentos dentro de um contexto.

A leitura desse diagrama é feito em duas dimensões: horizontal, onde estão representados os objetos, e vertical que representa a linha do tempo. A interação entre os objetos é feita através de mensagens, representadas através de linhas sólidas direcionadas, partindo o objeto solicitando para o solicitado.

Cada objeto, representado no diagrama por um retângulo contendo seu nome, corresponde à uma coluna. Cada objeto possui uma linha tracejada vertical, que indica o seu tempo de vida.

A linha de vida de um objeto tem a função de:

- ✓ Apresentam o tempo de vida dos objetos
- ✓ Pode apresentar a ativação ou a desativação dos objetos
- ✓ Indicam que os objetos estão executando algo
- ✓ Indica chamada de método do próprio objeto
- ✓ Podem representar a criação e a destruição de objetos



Exemplo Diagrama de sequência

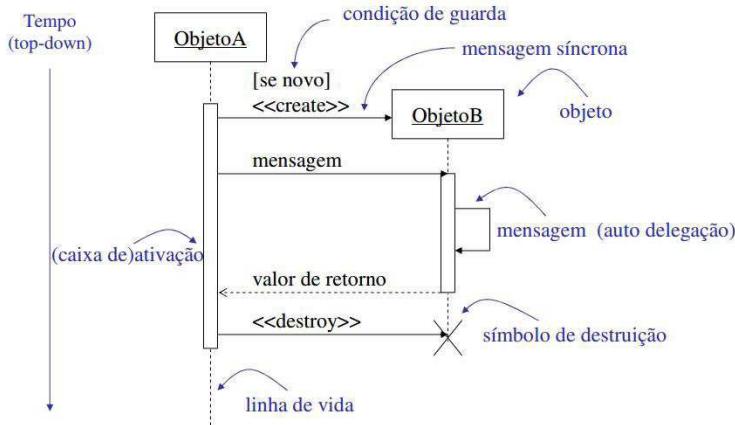


Figura 13 - Diagrama de sequência

De acordo com o diagrama da Figura 14, temos o Objeto A e Objeto B. E as linhas verticais que partem desses objetos representam o tempo, ou seja a linha de vida do objeto. Ele demonstra os objetos participando em interações de acordo com suas linhas de vida e as mensagens que trocam.

Perceba que o Objeto A tem a sua linha de vida, porém, dentro da sua linha de vida, ele tem o período de ativação que no início cria o Objeto B, e em seguida envia uma mensagem para o Objeto B, esse objeto por sua vez interpreta a mensagem de A e lhe passa um valor de retorno. O Objeto A recebe o valor e depois é destruído.



Exercícios Propostos

5.3 Diagrama de Máquina de Estado

Procura acompanhar as mudanças sofridas por um objeto dentro de um processo no sistema, dando ênfase aos estados dos objetos e as transições entre os estados. Costuma ser empregado para modelar os aspectos dinâmicos do sistema e possui o seu comportamento orientado por eventos, onde cada evento gera um novo estado no diagrama.

Temos um diagrama de estado genérico, com o único estado, representado na Figura 15 - Diagrama de máquina de estado. Observando a imagem identificamos o **estado** (como o objeto está naquele momento) representado por um retângulo com bordas arredondadas, as **transições** (mudança de estado do objeto) representadas por linhas orientadas por setas, **ponto inicial** (onde o objeto nasce) representado por um círculo totalmente hachurado e o **ponto final** (onde o objeto morre) representado por um círculo parcialmente hachurado.

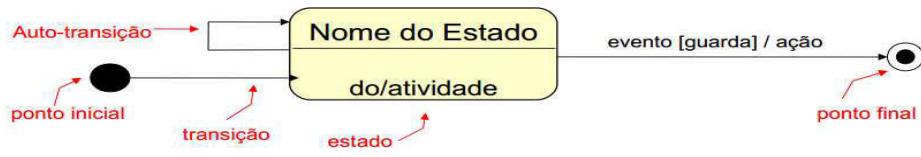


Figura 15 - Diagrama de máquina de estado

Na Figura 16 - Ciclo do diagrama de máquina de estado o objeto nasce, torna o usuário ativo, em seguida o torna inativo e depois morre.

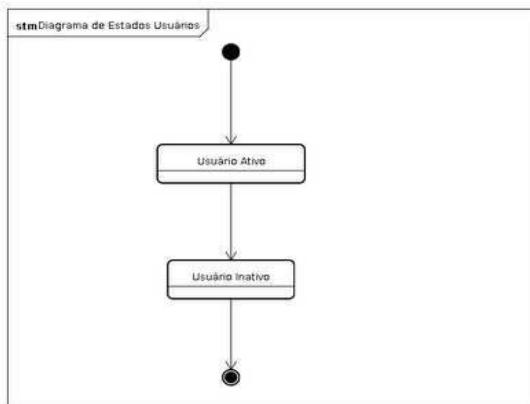


Figura 16 - Ciclo do diagrama de máquina de estado

5.4 De Atividades

Descreve os passos a serem percorridos para a conclusão de uma atividade.

O diagrama de atividade enfase as fluxo de controle de uma atividade para outra e identifica as atividades existentes no sistema. A descrição dos elementos diagramas de atividades encontra-se na Figura 17:

• Elementos



Figura 17 - Elementos diagrama de atividades

O diagrama da Figura 18 - Diagrama de atividades inicia com o círculo hachurado e o primeiro estado é solicitação do código do cliente, em seguida verifica se o cliente já existe, nesse momento cai em uma tomada de decisão, se o cliente já existe, informa para o usuário mostrando uma mensagem na tela e termina a atividade. Se não solicita os dados do novo cliente, salva os dados e termina a atividade.

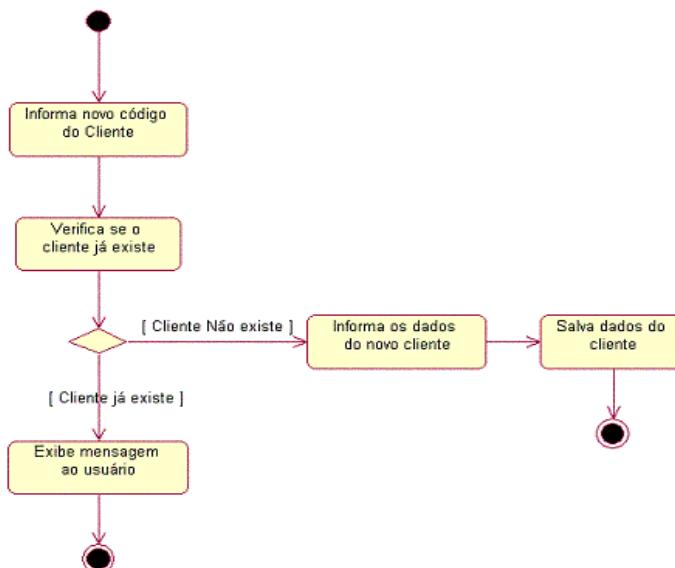


Figura 18 - Diagrama de atividades



Exercícios Propostos

6 Relacionamentos em UML

Em UML, os relacionamentos são conexões entre itens. Ou seja, temos relacionamentos em vários diagramas da UML, como relacionamento entre classes e relacionamento entre atores e casos de uso.

Em UML existem 3 tipos de relacionamento:

- ✓ Dependência
- ✓ Generalização
- ✓ Associação

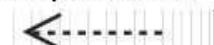
6.1 Relacionamento de Dependência

É um relacionamento de utilização, ou seja, um item é dependente do outro, dessa forma alteração de um item pode afetar o outro. Por exemplo, o relacionamento de dependência entre classes indica que os objetos de uma classe usam funcionalidades dos objetos de outra classe.

- Notação:

- Linha tracejada com seta podendo incluir um rótulo indicando o tipo de dependência.

Dependência



Veja o exemplo abaixo:



Figura 19 - Relacionamento de dependência

Nesse exemplo a classe A tem um relacionamento de dependência com a classe B do tipo friend.

6.2 Relacionamento de Associação

É um relacionamento estrutural que especifica a associação de objetos, classes e casos de uso.

Notação: Linha sólida podendo incluir:

Associação



- Nome: pode ser utilizado para descrever a natureza do relacionamento. Pode-se atribuir direção para o nome, fornecendo um triângulo de orientação que aponta a direção como nome deve ser lido
- Cardinalidade: a quantidade de objetos que podem ser conectados pela associação.



Exemplo



Figura 20 - Relacionamento de associação

Na leitura do diagrama acima dizemos que um item pode ser de nenhum ou muitos Pedido de compra, já um pedido de compra pode ser composto de um ou muitos itens.

Exemplo 2:



Figura 21 - Direção do relacionamento

Na leitura do exemplo acima podemos dizer que cliente compra produto

Simbologia para criação de cardinalidade:

Nome	Simbologia
Apenas Um	1
Zero ou Muitos	0..*
Um ou Muitos	* 1..*
Zero ou Um	0..1
Intervalo Específico	I _p ..I _q

Figura 22 - Cardinalidade dos relacionamentos

6.3 Relacionamento de Generalização

A generalização é um relacionamento de herança, ou seja, existe uma superclasse (classe pai) e uma subclasse (classe filha).

A classe filha herda características da classe pai e pode também possuir suas próprias características, dessa forma as subclasses compartilham a estrutura e comportamento das superclasses

Generalizações são relacionamentos “é-umtipo-de, ou seja, só é possível dizer que existe um relacionamento de generalização quando a subclasse é um tipo da superclasse.

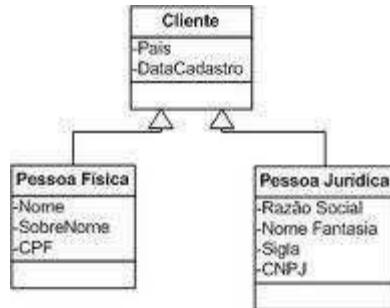
Notação:



– Linha sólida com seta em branco apontando a Superclasse

Vejamos como é possível identificar se existe um relacionamento de Generalização entre duas classes. Imagine a classe Veiculo e avião. Então devemos fazer a pergunta: avião é um tipo de veiculo?

Resposta: SIM, então significa que avião é uma subclasse de veiculo.



Exemplo 1:

Figura 23 - Relacionamento de herança

No exemplo acima a classe cliente é a superclasse e Pessoa Física e Pessoa Jurídica são suas subclasses. Note que as subclasses possuem características próprias além de herdar as características da sua superclasse.

A Generalização pode ainda ser demonstrada de duas formas:

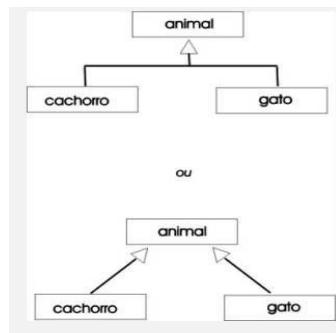


Figura 24 - Diagrama de classe com herança



Exercícios Propostos

Fase II – Programação na Linguagem Java

7 Visão Geral das Tecnologias Java

7.1 Introdução

Java é uma linguagem complexa utilizada para montar aplicações com acesso a bancos de dados, em plataforma desktop, dispositivos móveis entre outras. O código Java deve ser desenvolvido, testado e depois convertido em bytecodes através de uma operação de compilação. Durante a execução de uma aplicação desenvolvia em Java, os *bytecodes* são interpretados pela JVM (Java Virtual Machine), que é responsável pela execução dos programas Java. O Java ainda possui uma API (***)¹, ou seja, um conjunto de classes predefinidas e organizadas prontas para serem utilizados por todos os programadores em Java.



Figura 25 – Símbolo Java

O que é uma API

Uma API é uma coleção de componentes de software que já vêm prontos e o programa só precisa usar. A API do Java é uma coleção imensa, de classes reutilizáveis. Temos, por exemplo, as classes das bibliotecas JAVA. AWT e JAVA. SWING, que servem para criarmos telas gráficas de apresentação para o nosso usuário. Temos a JAVA. IO, com suas classes usadas para manipular entrada e saída de dados. Portanto as APIs nos economizam tempo, porque podemos utilizar em nossos programas as bibliotecas já existentes do Java.

Java Virtual Machine (JVM)

Vamos fazer uma comparação de um programa feito em outra linguagem e um feito na linguagem Java.

Uma linguagem tradicional cria um arquivo EXE, executável, que roda em contato direto com o sistema operacional do computador do usuário. Assim, se o EXE foi compilado para ser usado no Windows, ele não roda no Linux. Por isso os programas precisam ter várias versões, uma para Linux, uma para Windows e assim por diante.

A Máquina Virtual do Java é um software que deve ser baixado e instalado no computador do usuário para que os programas em Java rodem. Parece um problema, a princípio, mas é aqui que mora a verdadeira vantagem da linguagem Java sobre as linguagens mais tradicionais. Ou seja, no Java, o programa roda em contato direto com a JVM, e a JVM é quem fala diretamente com o Sistema Operacional. Assim, se eu tiver dois clientes, um com Linux e outro com Windows e tiverem em suas máquinas a JVM (que é gratuita), meu programa poderá ser compilado uma única vez, pois funciona em qualquer sistema operacional. Dessa forma o lema da linguagem Java é

“Write Once, Run Everywhere”, ou seja, escreva (o código) uma vez, e rode em todos os lugares.

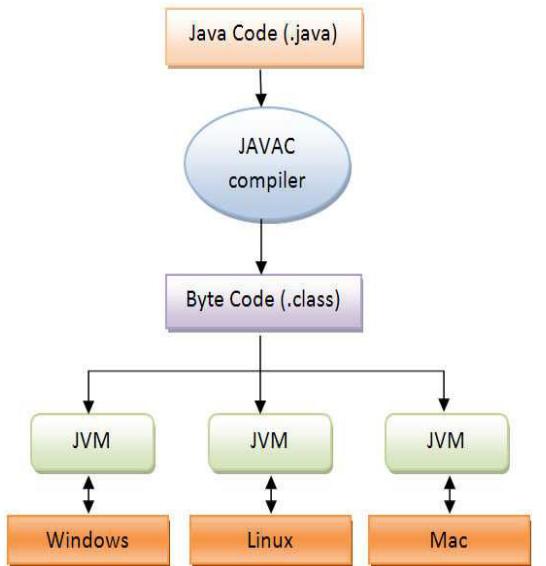


Figura 26 – Esquema de funcionamento da linguagem Java

JDK (Java Developer Kit)

O JDK é um kit de desenvolvimento Java composto pela JVM e a API de classes do Java, ele é disponível para download gratuitamente. Com o JDK instalado o programador tem tudo que precisa para desenvolver aplicações Java.

Coleta automática de lixo (Garbage Collection)

Nas linguagens de programação antiga quando não precisávamos mais de uma variável era preciso excluir a mesma da memória. No Java não, ele possui um coletor de lixo que ao perceber que uma variável não está mais sendo utilizada ele a exclui automaticamente da memória. Portanto no Java o programador não precisa se preocupar em gerenciar a memória.

O que é uma IDE (Integrates Development Environments)

As IDEs, são, Ambientes de Desenvolvimento Integrado. Ou seja, são programas que nos ajudam a desenvolver os nossos programas, nele você pode codificar suas classes e depois compilar seu programa.

Existem IDEs muito simples, como o BlueJ, e outras bem mais complexas, como o NetBeans e o Eclipse. No Eclipse, por exemplo, ao escrever o nome de uma classe ou de um objeto pertencente a uma classe, é disponibilizado os métodos daquela classe em forma de uma lista, no ponto do programador escolher. Esse recurso facilita muito a vida do programador pois ele não tem que memorizar todos os métodos de Java. Essas IDEs também corrigem nossos erros, marcando eles com sublinhados vermelhos, e ainda possuem assistentes de compilação que nos ajudam a gerar o arquivo JAR sem que tenhamos que compilar o arquivo via código.



Figura 27 – IDEs de desenvolvimento Java

7.2 Tecnologias Java

São suportes para desenvolvimento de aplicações com diferentes funcionalidades em Java. Todas se baseiam nos mesmos conceitos: bytecodes e JVM.



Figura 28 - Tecnologias Java

7.2.1 J2SE (Standard Edition)

O J2SE é utilizado para fazer softwares desktop, que operam na tela do computador normalmente. Ele consiste em codificar as classes do programa que ao ser compilado gera um arquivo JAR que funciona como executável da aplicação em qualquer computador que tenha a JVM instalada.

7.2.2 J2EE (Enterprise Edition)

O J2EE é utilizado para desenvolver aplicações que rodam na Web, por exemplo um sistema financeiro que funciona a partir de um site web. Ou seja é uma forma de criar páginas HTML que se comunicam com classes Java, formando assim a lógica da aplicação.

7.2.3 J2ME (Micro Edition)

O J2ME serve para desenvolver aplicações para telefones celulares, palmtops e outros dispositivos móveis.



Exercícios Propostos

8 Classes Java

8.1 Classe String

String é um tipo de texto que corresponde à união de um conjunto de caracteres. Em Java, uma variável do tipo String é uma instância (objeto) da classe String. Esses objetos possuem propriedades e métodos diferentes.

As Strings podem ser manipuladas de várias formas, é possível verificar seu tamanho, retirar uma parte da String ou converte-la para um formato específico.

A classe String em Java possui mais de 50 métodos que simplificam a programação com objetos Strings. Vamos estudar os principais métodos da classe String.

Método charAt

Método	Retorno	Parâmetro
charAt	char	int

O método **charAt** é usado para retornar um caractere de determinada string de acordo com um índice especificado entre parênteses. Esse índice refere-se à posição do caractere na string, sendo 0 (zero) o índice do primeiro caractere, 1 o índice do segundo caractere e assim por diante.

Esse método é útil quando, por exemplo, quisermos checar a existência de um caractere em determinada string. Veja o exemplo abaixo:

```

12  /*
13  public class PartesDaPalavra {
14      public static void main(String[] args) {
15          String palavra = JOptionPane.showInputDialog("Digite uma palavra");
16          for (int i = 0; i<palavra.length();i++){
17              char letra = palavra.charAt(i);
18              JOptionPane.showMessageDialog(null, "A " + i +"o letra da palavra " + palavra + " é " + letra );
19          }
20      }
21  }
22
23 */
24

```

Figura 29 – Exemplo de utilização método charAt

Método equal

Método	Retorno	Parâmetro
equal	boolean	String

Retorna verdadeiro se as strings forem "exatamente" iguais.

```

13
14     public class CalcRedes {
15
16     public static void main(String[] args) {
17         String nome1 = "Info";
18         String nome2 = "Info";
19
20         if ( nome1.equals(nome2) ){
21             System.out.println("Os nomes são iguais");
22         }
23     }
24 }
25

```

Figura 30 – Exemplo de utilização método equals

Método length

Método	Retorno	Parâmetro
length	int	Sem parâmetros

O método **length** é utilizado para contar a quantidade de caracteres de uma variável do tipo **String**, incluindo os espaços em branco. Esse método retorna sempre um valor do tipo inteiro.

Veja o exemplo abaixo:

```

12 */
13 public class TamanhoDaPalavra {
14     public static void main(String[] args) {
15         String palavra = JOptionPane.showInputDialog("Digite uma palavra");
16         int tamanho = palavra.length();
17         JOptionPane.showMessageDialog(null, "A palavra "+palavra+" tem "+tamanho +" caracteres");
18     }
19
20 }
21

```

Figura 31 – Exemplo de utilização método length

Linha 16 – A variável **tamanho** esta recebendo o valor numérico que representa a quantidade de caracteres da variável **palavra**.

Linha 17 – Esta sendo exibido em uma caixa do tipo **Message** a palavra e o seu tamanho.

Método toLowerCase e toUpperCase

Método	Retorno	Parâmetro
toLowerCase/toUpperCase	String	Sem Parâmetros

Os métodos **toUpperCase** e **toLowerCase** são utilizados para transformar todas as letras de uma determinada **String** em maiúsculas ou minúsculas, respectivamente. O método **toUpperCase** transforma todos os caracteres em maiúsculos e o **toLowerCase** transforma em minúsculos. Veja o exemplo abaixo:

```

12  /*
13   public class PartesDaPalavra {
14     public static void main(String[] args) {
15       String palavra = JOptionPane.showInputDialog("Digite uma palavra");
16       JOptionPane.showMessageDialog(null, "Em maiusculo " + palavra.toUpperCase());
17       JOptionPane.showMessageDialog(null, "Em minusculo " + palavra.toLowerCase());
18     }
19   }

```

Figura 32 – Exemplo método toUpperCase e toLowerCase

Método trim

Método	Retorno	Parâmetro
trim	String	Sem Parâmetros

O método **trim** remove todos os espaços em branco que aparecem no inicio e no fim de uma determinada string, porém não são removidos os espaços existentes entre as palavras.

Veja o exemplo abaixo:

```

12  /*
13   public class MetodoTrim {
14     public static void main(String[] args) {
15       String palavra = "  Estudando Java  ";
16       JOptionPane.showMessageDialog(null, "String com espaços: " + palavra);
17       JOptionPane.showMessageDialog(null, "String sem espaços: " + palavra.trim());
18     }
19   }

```

Figura 33 – Exemplo método trim

Método replace

Método	Retorno	Parâmetro
replace	String	char, char

O método **replace** é utilizado para substituição de caracteres, ou grupo de caracteres, em uma determinada **String**. Para isso é preciso informar quais caracteres serão substituídos e os novos caracteres que irão substituir.

Veja o exemplo abaixo:

```

12  /*
13   public class MetodoReplace {
14     public static void main(String[] args) {
15       String palavra = "Estudando Java";
16       JOptionPane.showMessageDialog(null, "Substituindo a por u : " + palavra.replace("a", "u"));
17     }
18   }

```

Figura 34 – Exemplo método replace

Método valueOf

Método	Retorno	Parâmetro
valueOf	String	double
valueOf	String	float
valueOf	String	int
valueOf	String	long

O método **valueOf** é usado para converter diversos tipos de dados em strings. Esse método aceita vários tipos de argumentos e transforma-os em **String**.

Veja o exemplo de código abaixo:

```

13  public class MetodoValueOf {
14  public static void main(String[] args) {
15      int a = 1000;
16      long b = 5000;
17      float c = 20.45f;
18      double d = 15.432;
19      JOptionPane.showMessageDialog(null, "Números convertidos em Strings /n" +
20          String.valueOf(a) + " " +
21          String.valueOf(b) + " " +
22          String.valueOf(c) + " " +
23          String.valueOf(d) );
24  }
25 }
```

Figura 35 – Exemplo método valueOf

8.2 Classe Math

A classe **Math** faz parte do pacote Java.lang da linguagem Java. Ela possui diversos métodos matemáticos, todos os métodos são estáticos, isso significa que para chamar um método da classe **Math** basta digitar o nome da classe e em seguida o nome do método.

A classe **Math** define duas constantes matemáticas, que é o **Math.PI** e o **Math.E** (que é o valor da base de logaritmos naturais).

Vamos estudar os principais métodos da classe **Math**:

Método ceil e floor

Método	Retorno	Parâmetro
ceil	double	double
floor	double	double

O método **ceil** tem a função de realizar o arredondamento de um número do tipo **float** ou **double**, para o seu próximo inteiro. Já o método **floor** faz o arredondamento para o seu anterior.

Veja o exemplo abaixo:

```

13  public class MetodoCeil {
14  public static void main(String[] args) {
15      double n1 = 8.2 , n2 = -7.6;
16      System.out.println("1º número arredondado: " + n1 + " = " + Math.ceil(n1));
17      System.out.println("2º número arredondado: " + n2 + " = " + Math.ceil(n2));
18  }
19 }
20

```

Figura 36 – Exemplo dos métodos ceil

Figura 37 - Resultado da execução do método ceil

Método max e min

Método	Retorno	Parâmetro
Max	double	double, double
Max	float	float, float
Max	int	int, int
Max	long	long, long

Os métodos **max** e **min** são usados para verificar o maior e o menor valor entre dois números, que podem ser do tipo **double**, **float**, **int** ou **long**.

```

-- 13  public class MétodoMaxMin {
14  public static void main(String[] args) {
15      int n1 = 3, n2 = 6;
16      System.out.println("O maior número entre " + n1 + " e " + n2 + " é: " + Math.max(n2, n1));
17      System.out.println("O menor número entre " + n1 + " e " + n2 + " é: " + Math.min(n2, n1));
18  }
19 }

```

Figura 38 - Exemplo métodos max e min

Método sqrt

Método	Retorno	Parâmetro
sqrt	Double	double

O método **sqrt** realiza o cálculo da raiz quadrada de um número. O retorno é do tipo **double**.

```

13  public class MétodoSqrt {
14  public static void main(String[] args) {
15      int n1 = 25;
16      System.out.println("A raiz quadrada de " + n1 + " é: " + Math.sqrt(n1));
17  }
18 }
19

```

Figura 39 - Exemplo método sqrt

Método pow

Método	Retorno	Parâmetro
Pow	double	double, double

O método **pow** eleva uma base à uma potência, ele precisa de dois parâmetros de entrada o primeiro é a base e o segundo a potência.

Método random

Método	Retorno	Parâmetro
random	double	Sem Parâmetros

Esse método é utilizado para gerar valores aleatórios. Ele sorteia valores do tipo **double** entre 0.0 e 1.0. Para obtermos números maiores que um pode-se multiplicar o número gerado pelo método por 100 e convertermos para inteiro, assim é possível ter números de 0 a 99.

```
String num = (int) (Math.random()*100)
```

Exercícios Propostos

Fase III – Programação Orientada a Objetos

9 Orientação a Objetos

9.1 Introdução

A orientação a objetos é, sem dúvida alguma, um dos mais significativos conhecimentos daquele que quer efetivamente se aprofundar no universo da Ciência da Computação. Para haver uma precisa compreensão do que vem a ser tal conteúdo, torna-se indispensável, antes de mais nada, conhecer aspectos como sua origem e conceitos relacionados. Assim, neste capítulo você terá oportunidade de estudar perspectivas que facilitarão e enriquecerão seu estudo acerca do fascinante universo da Orientação a Objetos.

Como aspecto inicial, podemos dizer que a orientação a objetos é uma forma de análise, projeto e programação de sistemas de software, que tem por base a composição e interação entre diversas unidades computacionais, as quais de chamamos de objetos.

Ela surgiu com a finalidade de facilitar e garantir mais agilidade aos então existentes programas computacionais e viabilizar também um entendimento mais claro de como se conectam, interagem e trabalham os processos computacionais.

Inicialmente, os programas eram feitos de forma sequencial, onde o fluxo tinha um início, meio e fim já definidos, e era processado sequencialmente até encontrar o fim do programa, quando ele parava, simplesmente.

Com a necessidade de controlar este fluxo, surgiram os recursos da programação estruturada, como laços e desvios condicionais. Mesmo já sendo um avanço, ainda existia o problema da repetição demasiada de códigos. Numa tentativa de otimizar o desenvolvimento de sistemas, a necessidade de reutilizar trechos de códigos foi sendo estudada cada vez mais pelos profissionais e pesquisadores da área, passando pela filosofia de guerra de Sun Tzu, “Dividir para conquistar”, que dividia os programas em vários programas menores.

A Programação Orientada a Objetos surgiu como um desenvolvimento natural da programação procedural. Na verdade buscava-se dar maior agilidade, inteligência, clareza e funcionalidade aos sistemas, assim, tentou-se implementar ainda melhoramentos focando a reusabilidade de códigos, até chegar na Programação Orientada a Objetos.

Adiante, veremos como a Programação Orientada a Objetos conquistou um verdadeiro avanço na programação de computadores, garantindo assim o desenvolvimento de novos programas, linguagens e ações no mundo computacional.

9.2 Histórico

Historicamente a origem da Orientação a Objetos conta da década de 1960 na Noruega, por meio de Kristen Nygaard e Ole-Johan Dahl, pesquisadores do Centro Norueguês de Computação. De fato, por meio da chamada linguagem Simula 67, deram-se os primeiros conceitos de classe e herança. Este é considerado o primeiro conteúdo de programação orientada ao objeto, porém, precisou de mais algum tempo para alcançar mais destaque com a linguagem *Smalltalk*.

Como dito, houve grande impulso para a Orientação ao Objeto com a *Smalltalk*, nascida nos laboratórios da Empresa Norte Americana Xerox (Estados Unidos da América). Neste renomado laboratório chamado *Palo Alto Research Center*, foi desenvolvida pela equipe liderada por Alan Curtis Kay uma sequência de protótipos que resultou na linguagem *Smalltalk*, esta é considerada uma das primeiras linguagens computacionais orientadas a objeto e por isso apropriada para iniciantes. Diante deste trabalho é que o estudioso programador Alan Kay é considerado historicamente um dos criadores do termo “Programação Orientada a Objetos”.

Alan Kay, percebeu que o conceito de objetos possuia grande força como instrumento do conhecimento. De fato a partir deste conceito se pode fazer uma série de associações da maneira como percebemos pessoas, objetos e o mundo em si. Se prestarmos bem atenção, a programação orientada a objetos tem como filosofia básica simular o mundo real dentro do mundo virtual, implementando objetos para que se comportem de forma similar à realidade.

Dentro deste conceito, Alan Kay pensou em como construir um sistema de software partindo de itens autônomos que interagissem entre si, estabelecendo os seguintes princípios da orientação a objetos:

- ✓ Qualquer coisa do mundo pode ser um objeto;
- ✓ Tarefas são realizadas por objetos por meio de requisição de serviços;
- ✓ Cada objeto é originado a partir de uma classe;
- ✓ Uma classe agrupa objetos semelhantes;
- ✓ Uma classe possui comportamentos associados ao objeto;
- ✓ Classes são organizadas de forma hierárquica.

Neste sentido, após coletar, elaborar e formatar conceitos de diversos campos do saber, aliando seu conhecimento e sua experiência, Alan Kay conseguiu lançar um dos mais utilizados e precisos instrumentos de linguagem computacional, qual seja, a programação orientada a objetos. Vejamos algumas características desse paradigma (2):

- ✓ A orientação a objetos é uma tecnologia para a produção de modelos que especifiquem o domínio do problema de um sistema.
- ✓ Quando construídos corretamente, sistemas orientados a objetos são flexíveis a mudanças, possuem estruturas bem conhecidas e provém a oportunidade de criar e implementar componentes totalmente reutilizáveis.

- ✓ Modelos orientados a objetos são implementados convenientemente utilizando uma linguagem de programação orientada a objetos. A engenharia de software orientada a objetos é muito mais que utilizar mecanismos de sua linguagem de programação, é saber utilizar da melhor forma possível todas as técnicas da modelagem orientada a objetos.
- ✓ A orientação a objetos não é só teoria, mas uma tecnologia de eficiência e qualidade comprovadas usada em inúmeros projetos e para construção de diferentes tipos de sistemas.

Fonte: Apostila Lingagem de Modelagem Unificada (<http://apostilando.net.br/swf/2038.swf> em junho de 2012).

Temos a seguir alguns exemplos de linguagens orientadas a objetos:

- | | | |
|-------------|-----------------|----------|
| ✓ Simula | ✓ Eiffel | ✓ C# |
| ✓ Smalltalk | ✓ Object Pascal | ✓ Perl |
| ✓ C++ | ✓ Java | ✓ Python |
| ✓ ADA | ✓ Common Lisp | |

9.3 Fundamentos de Orientação a Objetos

9.3.1 Uma analogia

No capítulo anterior nós aprendemos que a Programação Orientada a Objetos foi criada para que se tornasse possível fazermos uma simulação do mundo real no nosso computador. Dessa forma, os objetos reais serão moldados e terão suas características e comportamentos instruídos através da programação, podendo interagir entre si, através da troca de mensagens. Essa interação entre os objetos possibilita que uma tarefa computacional seja realizada.

Cabe ao programador fazer uma análise do objeto do mundo real que será implementado, observar suas particularidades relevantes e implementá-las, definindo quais características os objetos podem assumir, e quais ações podem executar. Mas, como fazer isso?

Naturalmente, os seres humanos costumam agrupar coisas para entendê-las. Por exemplo, observemos o objeto “Carro”. Facilmente podemos concluir que existem diversas variações de modelo, forma, cor, tamanho, etc. Porém, mesmo assim reconheceríamos um carro como sendo um carro, por características que eles têm em comum, como cor, modelo, fabricante, a ação de frear, de acelerar, e enfim, uma infinidade de características similares, ainda que cada um possua suas diferenças.



Fonte: <http://vidadeprogramador.com.br/>

Figura 40 – Tirinha sobre o que é Programação Orientada a Objeto

9.3.2 Objeto x Classe x Instância

Na POO (Programação Orientada a Objetos), os objetos do mundo real são analisados de uma forma abstrata, ou seja, as características do objeto que são levadas em consideração são apenas as que são fundamentais para o sistema. E aquilo que é importante será definido na classe.

Mas, o que é uma classe?

É uma abstração das características relevantes de um grupo de coisas do mundo real. A classe, em orientação a objeto, é criada para definir no sistema como o objeto vai ser criado, quais serão suas características (atributos) e suas ações (métodos). Já o objeto é o componente de software que representa o objeto do mundo real, reunindo atributos e métodos relacionados, e se relacionando entre si através de mensagens. Uma classe pode gerar vários objetos.

Chamamos de instância o ato de a classe criar um novo objeto. Em outras palavras, um objeto é uma instância de uma classe.

Vamos voltar ao nosso exemplo e refletir como podemos criar uma classe para ele:

Classe:

Carro
placa
cor
fabricante
velocidade
acelerar()
frear()

← Atributos

← Métodos

Figura 41 – Exemplo classe Carro

Comumente os conceitos de classe e objetos são confundidos por aprendizes, mas é importante entender bem essa diferença. Uma classe não é um objeto, mas ela é usada para criar um. Usamos a classe carro para definir diversos tipos de carro, como ilustra a tabela a seguir:

Classe Carro	Objeto Fusca	Objeto Opala
Atributos de Objeto	Placa XYZ 0000	ZYX 9999
	Cor Preto	Vermelho
	Fabricante Wolks	Wolks
	Velocidade 30km/h	120km/h
Métodos de Objetos	Método Acelerar	
	Método Frear	

Observe que a classe define quais os tipos de características que o objeto vai ter, e quando construídos, cada um adquire um conjunto novo de estados. Cada objeto é único, mesmo que possuam dados idênticos.

Agora, vamos considerar como exemplo um estudante, sendo nosso objeto um aluno. Para que esse objeto possa ser instanciado, precisamos definir a classe. Fazendo uma análise abstrata, temos de relevante que um aluno possui nome, idade, matrícula, notas e média geral. Como ação, temos o ato de calcular a média geral das disciplinas. Observe:

Classe:

Aluno
nome
matricula
idade
nota_portugues
nota_matematica
nota_ciencias
media_geral
calcula_media()

← Atributos

← Métodos

Figura 42 - Exemplo de classe: Aluno

Mas, afinal, como escrever uma classe em códigos? Ao definir como a classe será usada, temos que obedecer a estrutura a seguir:

```
<modificador> class <nome_da_classe> {
    <atributos>
    <construtor>
    <métodos>
}
```

Onde:

- ✓ modificador: é a forma como a classe será acessada (estudaremos mais adiante);
- ✓ nome_da_classe: nome da classe, a ser definido pelo programador;
- ✓ atributos: declaração dos atributos do objeto;
- ✓ construtor: construtor da classe (estudaremos mais adiante);
- ✓ método: declaração dos métodos da classe.

Veja como fica em Java:

```
2  public class Aluno {
3      // Instruções: atributos, métodos, construtores...
4
5
6 }
```

Figura 43 - Criação da Classe Aluno em Java



Dicas de Programação

Pense em nomes apropriados para a sua classe. Não a chame simplesmente de classe XYZ ou qualquer outro nome aleatório.

Os nomes de classes devem ser iniciados por letra MAIÚSCULA.

O nome do arquivo de sua classe obrigatoriamente possui o MESMO NOME da sua classe pública.

- Observação: Fonte das Dicas de Programação:
Balagtas, Florence Tiu. Projeto J.E.D.I. *Students Manual.s.l..*



9.3.3 Mensagem

Mensagens são requisições enviadas de um objeto para outro para que alguma operação seja realizada. É a forma como os objetos interagem entre si, enviam e recebem mensagens e respostas. Eles manifestam essa interatividade através da

invocação de métodos que definem e implementam uma determinada responsabilidade que o objeto tem.

9.3.4 Atributo

Os objetos possuem atributos, que são informações que definem seu estado. Os atributos de cada objeto de uma classe têm seus próprios valores. Em uma classe, podemos encontrar dois tipos de atributos: atributos de objeto e atributos de classe.

- ✓ Atributos de objeto: cada objeto tem seus atributos, e não compartilham com os outros.
- ✓ Atributos de classe são aqueles que têm valores compartilhados por todos os objetos da classe.

Vamos analisar nosso exemplo, a classe Aluno. Sabemos que a classe tem como atributos de objeto o nome, a matrícula, idade, as notas das disciplinas e a média geral do aluno. Poderemos ter também um atributo que guarda a média de notas dos alunos da sala, não fazendo sentido algum essa informação constar em todos os objetos. Acrescentemos, então, o atributo (de classe) media_turma.



Figura 44 - Exemplo classe Aluno com o atributo de classe media_turma

A seguir, a estrutura para a declaração de um atributo. Escrevemos, dentro da classe:

<modificador> <tipo_do_atributo> <nome_do_atributo> [= <inicialização>];

Onde:

- ✓ modificador: forma como o atributo será acessado;
- ✓ tipo: tipo de dado;
- ✓ nome: nome escolhido pelo programador;
- ✓ inicialização: valor inicial que o atributo vai receber.

Veja como fica em Java:

```

3 public class Aluno {
4     // Atributos de objeto
5     private String nome, matricula;
6     private int idade;
7     private double nota_portugues, nota_matematica, nota_ciencias, media_geral;
8
9     // Atributos de classe
10    private double media_turma;
11 }

```

Figura 45 - – Atributos de objeto e atributos de classe da Classe aluno



Dicas de Programação

- 1 Declare todos os atributos de objeto na parte superior da declaração da classe.
- 2 Atributos de objeto, assim como qualquer outro atributo devem iniciar com letra MINÚSCULA.
- 3 Use o tipo de dado apropriado para cada atributo declarado.
- 4 Declare atributos de objetos como *private* de modo que somente os métodos da classe possam acessá-los diretamente.



9.3.5 Método

Os métodos são as ações que a objeto poderá realizar. É a descrição, em forma de código, das operações que um objeto executa quando recebe uma mensagem. Pode ser chamado por qualquer outro método para realizar alguma função específica. Por exemplo, vamos imaginar um aparelho de som. Sua função é reproduzir músicas, mas só a executa se receber uma ordem para isso. E essa ordem é passada pelo botão de ligar, que ao ser acionado, faz com que a música toque. Nesse caso, teríamos que implementar um método que tivesse a ação de fazer o som tocar. Perceba que um aparelho de som possui outros botões, cada um com suas responsabilidades diversas como parar de tocar, gravar, aumentar volume, diminuir.

Os métodos possuem as seguintes características:

- ✓ Podem retornar ou não algum valor;
- ✓ Podem aceitar ou não repasse de argumentos;
- ✓ Após o encerramento de sua execução, o fluxo de controle retorna para quem o solicitou.

A seguir, a estrutura para a declaração de um método. Escrevemos, dentro da classe:

```

<modificador> <tipo_do_retorno> <nome_do_método> (<argumentos>){
    <instruções>
}

```

Onde:

- ✓ modificador: como o método será acessado;
- ✓ tipo_do_retorno: tipo de dado do valor que o método retornará (podendo também ser **void**);
- ✓ nome_do_método: nome escolhido para o método;
- ✓ argumentos: argumentos que serão recebidos pelo método, separados por vírgula., precedido pelo seu tipo de dado.

Veja como fica em Java:

```

14 // Método: calcula a média geral do aluno
15 public double calculaMediaAluno(){
16     media_geral = (nota_portugues + nota_matematica + nota_ciencias) / 3;
17     return media_geral;
18 }
```

Figura 46 – Método que calcula a média geral do aluno



Dicas de Programação

- 1 Nomes de métodos devem iniciar com letra MINÚSCULA.
- 2 Nomes de métodos devem conter verbos.
- 3 Sempre faça documentação antes da declaração do método. Use o estilo javadoc para isso.



9.3.6 Construtores

O construtor de uma classe é utilizado para a construção de um objeto pertencente àquela classe. Ao ser chamado, todas as inicializações do objeto são declaradas de forma organizada. Deve ser criado com o mesmo nome da classe, e pode ou não receber um argumento, podendo inicializar, assim, já com algum tipo de informação atribuída.

Definindo um construtor:

```

<modificador> <nome_da_classe> (<argumento>){
    <instrução>
}
```

Vejamos agora exemplo de construtor sem parâmetros. Um objeto instanciado através desse construtor terá os valores dos atributos de notas inicializados com 0.

```

18 // Construtor sem parâmetro
19 Aluno(){
20     nota_portugues = 0;
21     nota_matematica = 0;
22     nota_ciencias = 0;
23     media_geral = 0;
24 }
```

Figura 47 – Exemplo de construtor para a classe Aluno sem parâmetros

Vejamos agora exemplo de construtor com parâmetros. Nesse caso, ao ser chamado, o construtor irá inicializar os atributos de nome e matrícula com os valores dos parâmetros.

```

26 // Construtor com parâmetro
27 Aluno(String nm, String mat){
28     nome = nm;
29     matricula = mat;
30 }
```

Figura 48 – Exemplo de construtor para a classe Aluno com parâmetros



Exemplo

Considerando que você já conhece os comandos básicos, vamos ilustrar nossos exemplos na linguagem de programação Java. A seguir, a implementação a classe Aluno com definição de métodos, atributos e objetos instanciados. Observe e acompanhe os comentários explicativos no código.

```

3 public class Aluno {
4     // Atributos de objeto
5     private String nome, matricula;
6     private int idade;
7     private double nota_portugues, nota_matematica, nota_ciencias, media_geral;
8
9     // Atributos de classe
10    private double media_turma;
11
12    // Método: calcula a média geral do aluno
13    public double calculaMediaAluno(){
14        media_geral = (nota_portugues + nota_matematica + nota_ciencias) / 3;
15        return media_geral;
16    }
17
18    // Construtor sem parâmetro
19    Aluno(){
20        nota_portugues = 0;
21        nota_matematica = 0;
22        nota_ciencias = 0;
23        media_geral = 0;
24    }
25
26    // Construtor com parâmetro
27    Aluno(String nm, String mat){
28        nome = nm;
29        matricula = mat;
30    }
31
32 }
```

Figura 49– Implementação da classe Aluno com atributos, métodos e construtores

Agora, vejamos a criação de objetos (instância) da classe Aluno na classe principal:

```

3 public class ProjetoAluno {
4
5     public static void main(String[] args) {
6         // Instância: criação de dois objetos da classe Aluno, utilizando o construtor com parâmetros
7         Aluno aluFabricio = new Aluno("Fabrício Nascimento", "000-1");
8         Aluno aluCamila= new Aluno("Camila Dantas", "000-2");
9     }
10 }
```

Figura 50 - Instância e utilização de método da classe Aluno



Exercícios Propostos

10 Encapsulamento

10.1 O que é Encapsulamento?

Encapsulamento é um princípio da programação orientada a objeto que permite que determinados elementos de uma classe possam ser ocultados de outras classes. Ou seja, encapsulando, conseguimos esconder dados contidos em uma classe, podendo ser acessados apenas por intermédio de métodos próprios.

Para que alguma funcionalidade seja utilizada, não é necessário que a forma como ela foi implementada seja conhecida. Vamos imaginar o funcionamento de um telefone celular. O usuário faz ligações, recebe, usa a calculadora e o alarme sem ter conhecimento de como aquelas funções foram desenvolvidas, mas, para que ele as utilize, basta apenas acessar da forma correta que o aparelho atenderá suas expectativas, independente de ter sido implementado de uma forma ou de outra.

10.1.1 Modificadores de Acesso

Agora já sabemos que o encapsulamento nos dá a opção de esconder dados de objetos. Mas como? Podemos definir esse acesso aos dados de três formas: Público, Protegido e Particular.

10.1.2 Acesso Público (Public)

No acesso público, os elementos da classe poderão ser acessados tanto de dentro como de fora da classe. Ou seja, qualquer objeto que interage com a classe poderá acessar seus elementos que estão públicos.

10.1.3 Acesso Privado (Private)

Já no acesso Privado, o atributo ou o método só poderá ser acessado internamente, apenas na classe que o definiu.

10.1.4 Acesso Protegido (Protected)

Somente classes do mesmo pacote podem ter acesso aos atributos e métodos no modo protegido.

10.1.5 Métodos Get e Set

Vimos que podemos definir a forma como os nossos dados serão acessados e modificados por outros objetos, utilizando os modificadores de acesso. Muitas vezes o acesso aos dados se dá de forma privada (*private*), porém, em algumas situações se faz necessário que outros objetos acessem esses dados. Devemos ter muita cautela ao declarar esses métodos, só devendo ser declarados em caso de real utilidade. Mas de qual forma ele fará este acesso, se o dado foi definido como privado?

Set

Utilizamos o método *Set* para modificar o valor de um atributo. Ele necessita de um argumento, qual seja, o novo dado a ser atribuído, devendo ser do mesmo tipo de dado do atributo declarado.

Veja como fica em Java:

```

33  /** Método SET dos atributos de Aluno
34.
35.
36  // SET idade
37  public void setIdade(int id){
38      idade = id;
39  }
40.
41  // SET nota de português
42  public void setNotaPortugues(double nt_port){
43      nota_portugues = nt_port;
44  }
45.
46  // SET nota de matemática
47  public void setNotaMatematica(double nt_mat){
48      nota_matematica = nt_mat;
49  }
50.
51  // SET nota de ciências
52  public void setNotaCiencias(double nt_cien){
53      nota_ciencias = nt_cien;
54  }

```

Figura 51 – Implementação dos métodos *Set* para atributos da classe Aluno

Agora, veja como o método *Set* é chamado na classe principal:

```

10  // Utilização dos métodos SET
11  // Atribuição de valores aos atributos do objeto: aluno Fabrício
12  aluFabrício.setIdade(18);
13  aluFabrício.setNotaPortugues(7.3);
14  aluFabrício.setNotaMatematica(9.0);
15  aluFabrício.setNotaCiencias(7.7);
16.
17  // Utilização dos métodos SET
18  // Atribuição de valores aos atributos do objeto: aluna Camila
19  aluCamila.setIdade(17);
20  aluCamila.setNotaPortugues(9.2);
21  aluCamila.setNotaMatematica(7.0);
22  aluCamila.setNotaCiencias(9.6);

```

Figura 52 – Chamada do método *Set* na classe principal para atribuição de valores

Get

Podemos utilizar o método *Get* para ler valores de atributos (de objeto ou de classe). O método deverá retornar um valor, que é a informação contida no atributo, obviamente com o mesmo tipo de dado. Ao método *Get*, não é necessário passarmos nenhum argumento.

Veja como fica em Java:

```

57 	/** Método GET dos atributos de Aluno
58
59 	// GET de nome
60 	public String getNome(){
61 		return nome;
62 	}
63
64 	// GET de matrícula
65 	public String getMatricula(){
66 		return matricula;
67 	}
68
69 	// GET de idade
70 	public int getIdade(){
71 		return idade;
72 	}
73
74 	// GET de nota de português
75 	public double getNotaPortugues(){
76 		return nota_portugues;
77 	}
78
79 	// GET de nota de matemática
80 	public double getNotaMatematica(){
81 		return nota_matematica;
82 	}
83
84 	// GET de nota de ciências
85 	public double getNotaCiencias(){
86 		return nota_ciencias;
87 	}
88
89 	// GET de média geral do aluno
90 	public double getMediaGeral(){
91 		return media_geral;
92 	}
93
94 	// GET de média da turma
95 	public double getMediaTurma(){
96 		return media_turma;
97 }

```

Figura 53 – Implementação dos métodos Get para atributos da classe Aluno

Veja abaixo como o método *Get* é chamado na classe principal. Observe a chamada do método *calculaMediaAluno()* definido anteriormente.

```

25 // Utilização dos métodos GET
26 System.out.println("Matrícula: " + aluFabricio.getMatricula() + " Nome: " + aluFabricio.getNome());
27 System.out.println("Nota Português: " + aluFabricio.getNotaPortugues());
28 System.out.println("Nota Matemática: " + aluFabricio.getNotaMatematica());
29 System.out.println("Nota Ciências: " + aluFabricio.getNotaCiencias());
30 // Utilização do método que calcula a média do aluno
31 System.out.println("MÉDIA: " + aluFabricio.calculaMediaAluno());
32 System.out.println("*****");
33
34 // Utilização dos métodos GET
35 System.out.println("Matrícula: " + aluCamila.getMatricula() + " Nome: " + aluCamila.getNome());
36 System.out.println("Nota Português: " + aluCamila.getNotaPortugues());
37 System.out.println("Nota Matemática: " + aluCamila.getNotaMatematica());
38 System.out.println("Nota Ciências: " + aluCamila.getNotaCiencias());
39 // Utilização do método que calcula a média do aluno
40 System.out.println("MÉDIA: " + aluCamila.calculaMediaAluno());
41 System.out.println("*****");

```

Figura 54 - Chamada do método Get na classe principal

Resultado do processamento do programa:

```
Matrícula: 000-1 Nome: Fabrício Nascimento
Nota Português: 7.3
Nota Matemática: 9.0
Nota Ciências: 7.7
MÉDIA: 8.0
*****
Matrícula: 000-2 Nome: Camila Dantas
Nota Português: 9.2
Nota Matemática: 7.0
Nota Ciências: 9.6
MÉDIA: 8.6
*****
```

Figura 55 - Saída do programa



Exercícios Propostos

11 Herança

11.1 O que é herança?

Uma das grandes vantagens do uso da orientação a objetos é justamente a utilização do conceito de herança. Com a herança o trabalho do programador pode ser otimizado, pois proporciona uma eficiente e segura política de reutilização de códigos, evitando assim o retrabalho.

A herança nos permite que características que são comuns a diversas classes sejam reunidas em uma única classe base. A partir desta, outras classes herdam suas especificações, e nelas apenas é implementado o que lhes falta, a diferença.

11.2 Superclasse e Subclasse

Essa classe base, que reúne informações comuns a outras classes, é chamada de Superclasse. Já a Subclasse é uma classe mais específica, que herda as funcionalidades da Superclasse e ainda adiciona funcionalidades específicas que por ventura venha a ter.

Esse conceito de Superclasses e Subclasses faz com que exista uma estrutura de hierarquia entre as classes.

11.3 Herança múltipla

Algumas linguagens de programação suportam a herança múltipla, que permite que uma subclasse tenha capacidade de herdar características de duas ou mais superclasses. Assim, uma única classe pode agrupar atributos e métodos de várias classes. Não é o caso de Java, que não permite essa funcionalidade.

11.4 Classe Object

No Java, a classe *Object* é a raiz principal, a classe suprema, e a partir dela todas as outras subclasses são criadas. Ainda que você não expresse que sua classe está herdando de outra, o Java considera que você está herdando de *Object*. Sem exceção, todas as classes herdam de *Object*, de forma direta ou indireta.

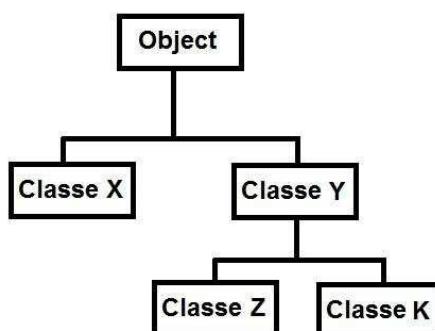


Figura 56 – Hierarquia entre classes



Exemplo:

Vimos o exemplo da implementação da classes Aluno. Imagine agora que desejamos criar objetos Alunos, mas com uma particularidade: é aluno de uma EEEP. Por estar em uma Escola Profissional, o aluno, além das notas de português, matemática e ciências, tem também nota de TESE, FPC e PPE. Vamos utilizar o conceito de herança para adicionar esses dados, sem precisar repetir todas as demais informações que já constam na classe Aluno. Nesse caso, criaremos a classe AlunoEEE, que herda funcionalidades da classe Aluno, e podemos acrescentar o que está faltando.

Veja baixo a implementação da subclasse de Aluno, a AlunoEEE. A primeira coisa que devemos fazer é mudar na classe Aluno o modificador de acesso de *private* para *protected*, para que os atributos sejam visualizados também pela subclasse.

```

3  public class Aluno {
4      // Atributos de objeto
5      protected String nome, matricula;
6      protected int idade;
7      protected double nota_portugues, nota_matematica, nota_ciencias, media_geral;
8
9      // Atributos de classe
10     protected double media_turma;

```

Figura 57 -Modificador de acesso “protected” para ser acessado também pelas subclasses

Agora vejamos a implementação da classe AlunoEEE:

```

3  public class AlunoEEE extends Aluno {
4      // Atributos de objeto adicionais
5      private double nota_TESE, nota_FPC, nota_PPE, media_geral_tecnica;
6
7      // Atributo de classe: média geral apenas das disciplinas técnicas
8      public double media_turma_tecnica;
9
10     // Método: calcula média das disciplinas técnicas
11     public double CalculaMediaTecnicaAluno(){
12         media_geral_tecnica = (nota_TESE + nota_FPC + nota_PPE) / 3;
13         return media_geral_tecnica;
14     }
15
16     // Construtor com parâmetro
17     AlunoEEE(String nm, String mat){
18         nome = nm;
19         matricula = mat;
20     }
21 }

```

Figura 58 – Implementação da subclasse AlunoEEE, que herda da superclasse Aluno

Implementação dos métodos *SETs* e *GETs* na classe AlunoEEEP:

```

16     //*** Método SET dos atributos de AlunoEEEP
17
18     // SET nota de tese
19     public void setNotaTESE(double nt_TESE){
20         nota_TESE = nt_TESE;
21     }
22
23     // SET nota de fpc
24     public void setNotaFPC(double nt_FPC){
25         nota_FPC = nt_FPC;
26     }
27
28     // SET nota de PPE
29     public void setNotaPPE(double nt_PPE){
30         nota_PPE = nt_PPE;
31     }
32
33     //*** Método GET dos atributos de AlunoEEEP
34
35     // GET de nota de TESE
36     public double getNotaTESE(){
37         return nota_TESE;
38     }
39
40     // GET de nota de FPC
41     public double getNotaFPC(){
42         return nota_FPC;
43     }
44
45     // GET de nota de PPE
46     public double getNotaPPE(){
47         return nota_PPE;
48     }

```

Figura 59– Métodos *SETs* e *GETs* da classe AlunoEEEP

Agora, vejamos a criação de objetos (instância) da classe AlunoEEEP e atribuição de valores na classe principal:

```

10    // Instância: criação de dois objetos da subclasse de Aluno, a AlunoEEEP
11    AlunoEEEP aluMilena = new AlunoEEEP("Milena Rebouças", "EP-100");
12    AlunoEEEP aluRafaela = new AlunoEEEP("Rafaela Alves", "EP-101");
13
14    // Utilização dos métodos SET
15
16    // Atribuição de valores aos atributos do objeto: aluna Milena
17    aluMilena.setIdade(17);
18    aluMilena.setNotaPortugues(8.3);
19    aluMilena.setNotaMatematica(9.7);
20    aluMilena.setNotaCiencias(8.0);
21    aluMilena.setNotaTESE(10.0);
22    aluMilena.setNotaFPC(8.9);
23    aluMilena.setNotaPPE(7.4);
24
25    // Atribuição de valores aos atributos do objeto: aluna Milena
26    aluRafaela.setIdade(17);
27    aluRafaela.setNotaPortugues(9.4);
28    aluRafaela.setNotaMatematica(7.7);
29    aluRafaela.setNotaCiencias(7.0);
30    aluRafaela.setNotaTESE(10.0);
31    aluRafaela.setNotaFPC(9.9);
32    aluRafaela.setNotaPPE(8.4);

```

Figura 60 - Instância de objetos e chamada do método Set na classe principal para atribuição de valores

Veja a chamada dos métodos *GETs* na classe principal. Observe como o conceito de herança é praticado pelo objeto da subclasse AlunoEEEP. Ele utiliza métodos de sua própria classe (*calculaMediaTecnicaAluno()*) e herda métodos de sua superclasse (*calculaMediaAluno()*).

```

67      // Utilização dos métodos GET
68      System.out.println("Matrícula: " + aluMilena.getMatricula() + " Nome: " + aluMilena.getNome());
69      System.out.println("Nota Português: " + aluMilena.getNotaPortugues());
70      System.out.println("Nota Matemática: " + aluMilena.getNotaMatematica());
71      System.out.println("Nota Ciências: " + aluMilena.getNotaCiencias());
72      System.out.println("Nota TESE: " + aluMilena.getNotaTESE());
73      System.out.println("Nota FPC: " + aluMilena.getNotaFPC());
74      System.out.println("Nota PPE: " + aluMilena.getNotaPPE());
75      // Utilização do método que calcula a média do aluno
76      System.out.println("MÉDIA Base Comum: " + aluMilena.calculaMediaAluno());
77      System.out.println("MÉDIA Técnica: " + aluMilena.CalculaMediaTecnicaAluno());
78      System.out.println("*****");
79
80      // Utilização dos métodos GET
81      System.out.println("Matrícula: " + aluRafaela.getMatricula() + " Nome: " + aluRafaela.getNome());
82      System.out.println("Nota Português: " + aluRafaela.getNotaPortugues());
83      System.out.println("Nota Matemática: " + aluRafaela.getNotaMatematica());
84      System.out.println("Nota Ciências: " + aluRafaela.getNotaCiencias());
85      System.out.println("Nota TESE: " + aluRafaela.getNotaTESE());
86      System.out.println("Nota FPC: " + aluRafaela.getNotaFPC());
87      System.out.println("Nota PPE: " + aluRafaela.getNotaPPE());
88      // Utilização do método que calcula a média do aluno
89      System.out.println("MÉDIA Base Comum: " + aluRafaela.calculaMediaAluno());
90      System.out.println("MÉDIA Técnica: " + aluRafaela.CalculaMediaTecnicaAluno());
91      System.out.println("*****");

```

Figura 61 – Instância de objetos e chamada do método SET na classe principal para atribuição de valores**Resultado do processamento do programa:**

```

Matrícula: EP-100 Nome: Milena Rebouças
Nota Português: 8.3
Nota Matemática: 9.7
Nota Ciências: 8.0
Nota TESE: 10.0
Nota FPC: 8.9
Nota PPE: 7.4
MÉDIA Base Comum: 8.666666666666666
MÉDIA Técnica: 8.766666666666666
*****
Matrícula: EP-101 Nome: Rafaela Alves
Nota Português: 9.4
Nota Matemática: 7.7
Nota Ciências: 7.0
Nota TESE: 10.0
Nota FPC: 9.9
Nota PPE: 8.4
MÉDIA Base Comum: 8.033333333333333
MÉDIA Técnica: 9.43333333333332
*****

```

Figura 62 - Saída do programa**Exercícios Propostos**

12 Polimorfismo

12.1 O que é polimorfismo?

A palavra Polimorfismo vem do grego, e significa muitas formas. Em orientação a objetos, é a capacidade de uma referência mudar de comportamento de acordo com o objeto a que se refere. Significa que um mesmo tipo de objeto, sob certas condições, pode se comportar de formas distintas ao receber uma mensagem. Ou seja, dependendo do contexto da execução, o sistema decidirá qual método será executado.

Através do polimorfismo, a aplicação dos métodos se dá de forma automática, de acordo com o tipo do objeto. Por exemplo, já implementamos a superclasse Aluno e criamos objetos “aluFabricio” e “aluCamila”. Depois, criamos uma subclasse AlunoEEEP, para os estudos de escola profissionalizante, que herda funcionalidades de Aluno. Dessa classe, criamos os objetos AluMilena e AluRafaela.

Na classe Aluno, criamos o método “calculaMediaAluno()” que faz a média aritmética das notas de português, matemática e ciências. Já na classe AlunoEEEP, criamos o método “calculaMediaTecnicaAluno()”, que calcula a média aritmética das disciplinas técnicas, quais sejam TESE, FPC e PPE.

Vamos refletir sobre a seguinte situação: no momento em que um objeto da classe AlunoEEEP chama o método “calculaMediaAluno()”, ele se comporta da mesma forma como para um objeto da classe Aluno: calcula as médias de português, matemática e ciências. Mas, e se nós quisermos que ele calcule a métrica aritmética de todas as disciplinas, inclusive as técnicas, como fazemos? Utilizando o polimorfismo, conseguimos definir a forma como queremos que nosso objeto se comporte.

Analise o código a seguir e observe que o mesmo método “calculaMediaAluno()” é implementado de forma a atender as particularidades de cada uma das classes.

Método “calculaMediaAluno()” na classe Aluno:

```

12 // Método: calcula a média geral do aluno
13 public double calculaMediaAluno(){
14     media_geral = (nota_portugues + nota_matematica + nota_ciencias) / 3;
15     return media_geral;
16 }
```

Figura 63 – Método para calcular a média do aluno na classe Aluno

Método “calculaMediaAluno()” na classe AlunoEEEP:

```

21 // Método: calcula a média geral do aluno
22 public double calculaMediaAluno(){
23     media_geral = (nota_portugues + nota_matematica + nota_ciencias + nota_TESE + nota_FPC + nota_PPE) / 6;
24     return media_geral;
25 }
```

Figura 64 - Método para calcular a média do aluno na classe AlunoEEEP

Agora, na classe principal, vamos ver como objetos das classes Aluno e AlunoEEEP se comportam ao chamarem o método “calculaMediaAluno()”:

```

116 // Atribuição das notas: aluno Fabrício (classe Aluno)
117 aluFabricio.setNotaPortugues(7.3);
118 aluFabricio.setNotaMatematica(9.0);
119 aluFabricio.setNotaCiencias(7.7);
120
121 // Atribuição das notas: aluna Camila (classe Aluno)
122 aluCamila.setIdade(17);
123 aluCamila.setNotaPortugues(9.2);
124 aluCamila.setNotaMatematica(7.0);
125 aluCamila.setNotaCiencias(9.6);
126
127 // // Atribuição das notas: aluna Milena (classe AlunoEEEP)
128 aluMilena.setNotaPortugues(8.3);
129 aluMilena.setNotaMatematica(9.7);
130 aluMilena.setNotaCiencias(8.0);
131 aluMilena.setNotaTESE(10.0);
132 aluMilena.setNotaPFC(7.6);
133 aluMilena.setNotaPPE(8.4);
134
135 // // Atribuição das notas: aluna Rafaela (classe AlunoEEEP)
136 aluRafaela.setNotaPortugues(9.4);
137 aluRafaela.setNotaMatematica(7.7);
138 aluRafaela.setNotaCiencias(7.0);
139 aluRafaela.setNotaTESE(10.0);
140 aluRafaela.setNotaPFC(9.9);
141 aluRafaela.setNotaPPE(8.4);
142
143
144 // Utilização do método calculaMediaAluno()
145
146 // Objetos da classe Aluno
147 System.out.println("Média do Fabrício, aluno de escola CONVENCIONAL: " + aluFabricio.calculaMediaAluno());
148 System.out.println("Média da Camila, aluna de escola CONVENCIONAL: " + aluCamila.calculaMediaAluno());
149 // Objetos da subclasse AlunoEEEP
150 System.out.println("Média da Milena, aluno de escola PROFISSIONAL: " + aluMilena.calculaMediaAluno());
151 System.out.println("Média da Rafaela, aluno de escola PROFISSIONAL: " + aluRafaela.calculaMediaAluno());
152

```

Figura 65 - Chamada do método “calculaMediaAluno()” de objetos de classes distintas

Observe a saída do sistema. Para os objetos da classe Aluno, o sistema calculou a média aritmética entre as disciplinas de português, matemática e ciências, como definido em Aluno. Já para os objetos da classe AlunoEEEP, calculou a média aritmética das notas de português, matemática, ciências, TESE, PFC e PPE.

```

Média do Fabrício, aluno de escola CONVENCIONAL: 8.0
Média da Camila, aluna de escola CONVENCIONAL: 8.6
Média da Milena, aluno de escola PROFISSIONAL: 8.666666666666666
Média da Rafaela, aluno de escola PROFISSIONAL: 8.733333333333333

```

Figura 66– Saída do programa: médias calculadas de acordo com o tipo de objeto

12.2 Sobrecarga de métodos

Sobrecarga de métodos é a propriedade que torna possível a criação de métodos com o mesmo nome, que executam funções diferentes. Ele saberá de qual maneira deverá se comportar através da quantidade de argumentos informada. A sobrecarga de métodos facilita a implementação, pois muitas vezes a mesma operação tem implementações diferentes para cada situação, e ao invés de criarmos nomes para cada uma delas, podemos diferenciar apenas através dos argumentos enviados.

Vamos refazer o método “calculaMediaAluno()” para que funcione de outra forma. Agora, vamos passar as notas a serem calculadas por parâmetro.

```

19     // Método: calcula a média geral do aluno com parâmetros
20     public double calculaMediaAluno(double nt_port, double nt_mat, double nt_cien){
21         media_geral = (nt_port + nt_mat + nt_cien) / 3;
22         return media_geral;
23     }

```

Figura 67 - Método “calculaMediaAluno()” implementado com passagem de parâmetro

Chamada do método na classe principal:

```

156     // Objetos da classe Aluno
157     System.out.println("Média do Fabrício, aluno de escola CONVENCIONAL: " + aluFabricio.calculaMediaAluno(7.3, 9.0, 7.7));
158     System.out.println("Média da Camila, aluna de escola CONVENCIONAL: " + aluCamila.calculaMediaAluno(9.2, 7.0, 9.6));
159     // Objetos da subclasse AlunoEEEP
160     System.out.println("Média da Milena, aluno de escola PROFISSIONAL: " + aluMilena.calculaMediaAluno(10.0, 7.6, 8.4));
161     System.out.println("Média da Rafaela, aluno de escola PROFISSIONAL: " + aluRafaela.calculaMediaAluno(10.0, 9.9, 8.4));

```

Figura 68 - Chama do método “calculaMediaAluno()” na classe principal

A saída do sistema é a média calculada das três notas enviadas por parâmetro:

```

Média do Fabrício, aluno de escola CONVENCIONAL: 8.0
Média da Camila, aluna de escola CONVENCIONAL: 8.6
Média da Milena, aluno de escola PROFISSIONAL: 8.666666666666666
Média da Rafaela, aluno de escola PROFISSIONAL: 9.433333333333332

```

Figura 69 - Saída do programa

Agora vamos refletir sobre a seguinte situação: em alguns momentos a média do aluno de uma escola profissionalizante deve ser calculada apenas com as disciplinas da base comum, porém em outros momentos é necessário que a nota seja calculada também incluindo as disciplinas técnicas. Usando o conceito de polimorfismo, podemos implementar o método “calculaMediaAluno()” na classe AlunoEEEP de forma que, automaticamente, o programa saiba de que forma calcular: somente com as notas da base comum, ou incluindo, também, as disciplinas técnicas. Esse reconhecimento é possível devido a quantidade de argumentos passados. Se a média deve ser calculada apenas com as notas da base comum, o método deve ser chamado e enviado a ele apenas os três argumentos. Se é para ser calculada incluindo também as disciplinas técnicas, então o método é chamado e a ele enviado os seis parâmetros. Vejamos:

```

21     // Método: calcula a média geral do aluno
22     public double calculaMediaAluno(){
23         media_geral = (nota_portugues + nota_matematica + nota_ciencias + nota_TESE + nota_FPC + nota_PPE) / 6;
24         return media_geral;
25     }

```

Figura 70 – Implementação do método calculaMediaAluno na classe AlunoEEEP, com seis parâmetros

Agora, um mesmo objeto da classe AlunoEEEP chama o método “calculaMediaAluno()” de duas formas diferentes: uma passando três (03) parâmetros, e outra, seis (06).

```

167      // Objetos da classe AlunoEEEP
168      // Calculo da média apenas das disciplinas da base comum
169      System.out.println("Média geral (base comum): " + aluMilena.calculaMediaAluno(8.3, 9.7, 8.0));
170
171      // Calculo da média incluindo as disciplinas técnicas
172      System.out.println("Média geral (incluindo disciplinas técnicas): " + aluMilena.calculaMediaAluno(8.3, 9.7, 8.0, 8.4, 7.6, 7.1));

```

Figura 71 – Chamada do método calculaMediaAluno pelo mesmo objeto

O programa retornará resultados diferentes. O conceito de polimorfismo, sobrecarga de métodos é o que torna possível esse reconhecimento automático de como o método deve se comportar através os parâmetros informados.

```

Média geral (base comum): 8.666666666666666
Média geral (incluindo disciplinas técnicas): 8.183333333333334

```

Figura 72– Resultados diferentes

12.3 Classes e métodos abstratos

Uma classe abstrata é uma superclasse criada apenas para representar entidades e conceitos abstratos, e dela não são gerados objetos, ou seja, a classe abstrata não é instanciada.

Geralmente criamos uma classe com um propósito específico, porém, a classe abstrata tem outra função, ela é criada com a intenção de que outras classes herdem funcionalidades delas. É definida como um modelo genérico a ser utilizado pelas suas subclasses.

A utilização de classes abstratas nos proporciona uma redução de código e um ganho considerável na utilização do polimorfismo, pois com elas podemos criar métodos mais genéricos que se adaptam a diversos tipos de objetos.

Vejamos um outro exemplo em que se aplica a utilização de classes abstratas. Imagine uma superclasse “Pessoa” e duas subclasses que herdam suas funcionalidades, quais sejam “PessoaFisica” e “PessoaJuridica”. Quando cadastramos um cliente, temos que defini-lo como pessoa física ou pessoa jurídica. Sendo assim, podemos criar a Pessoa como sendo abstrata para herdar funcionalidades e ganharmos polimorfismo, não fazendo sentido ser instanciada.

Agora vamos imaginar a nossa classe Aluno contendo os dados básicos comuns a todos os alunos da rede estadual de ensino. A superclasse Aluno poderá ser abstrata, tendo como subclasses AlunoConv, AlunoEEEP e AlunoSuperior, que são, respectivamente, alunos de escolas convencionais, alunos de escolas profissionalizantes e alunos de ensino superior. Se todos os alunos se encaixam em um desses três tipos, não faz sentido instanciar a classe Aluno, sendo ela, portanto, uma classe abstrada.

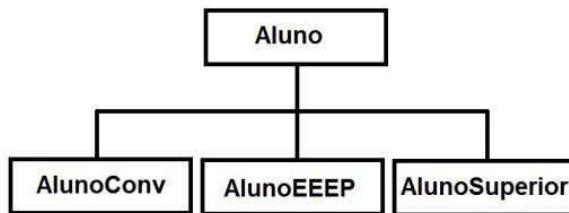


Figura 73 - Classe abstrata Aluno e subclasses

Vejamos como implementar uma classe abstrata em Java:

```

3  public class AlunoAbstract {
4      // Atributos de objeto
5      protected String nome, matricula;
6      protected int idade;
7      protected double nota_portugues, nota_matematica, nota_ciencias, media_geral;
8
9      // Método: calcula a média geral do aluno com parâmetros
10     public double calculaMediaAluno(double nt_port, double nt_mat, double nt_cien){
11         media_geral = (nt_port + nt_mat + nt_cien) / 3;
12         return media_geral;
13     }
14 }
  
```

Figura 74 – Implementação da classe abstrata Aluno

Métodos abstratos são métodos da classe abstrata que não têm implementação. Eles são declarados apenas com a assinatura, sem o corpo e com a palavra-chave *abstract*.



Dicas de Programação (9)

Use classes abstratas para definir muitos tipos de comportamentos no topo de uma hierarquia de classes de programação orientada a objetos. Use suas subclasses para prover detalhes de implementação da classe abstrata.



Exercícios Propostos

Fase IV – Projeto Orientado a Objetos

13 Conhecendo e Organizando um Projeto no Padrão MVC

13.1 Entendo o Padrão Model-View-Controller – MVC

Em um sistema grande e complexo composto por várias classes, não é interessante ter todo o código da aplicação “misturado”, ou seja, termos em um único pacote: a codificação da tela gráfica de interação do usuário, as ações do sistema, os códigos de negócio da aplicação e os códigos de acesso a um banco de dados. Essa metodologia de desenvolvimento de software torna muito difícil realizar manutenção e reuso de código em outros sistemas.

Para solucionar nosso problema podemos utilizar o padrão Model-View-Controller – MVC, que é um padrão de arquitetura de software utilizado para separar partes da aplicação a fim de melhorar o entendimento e aproveitamento das partes.

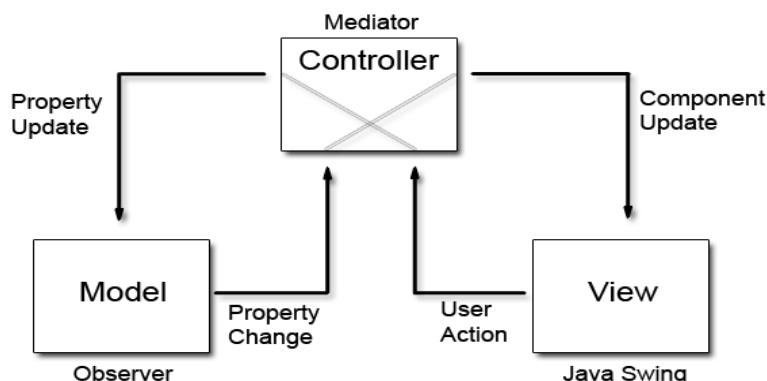


Figura 75 - Estrutura padrão MVC

A Figura 75 - Estrutura padrão MVC, acima, demonstra como ocorre a comunicação entre os módulos do padrão MVC, que consiste em dividir as classes do sistema em três pacotes: Model, View e Controller. Perceba, a classe de visão chama a classe de controle que por sua vez manipula a classe de negócio (Model), quando a operação é realizada a classe de negócio retorna para a classe de controle que chama a classe de visão para exibir o resultado da operação.

O pacote **Model** será constituído das classes de negócio do sistema. Nesse momento você pode se perguntar como identificar as classes de negócio do meu sistema? O negócio é aquilo que trata o sistema, ou seja, é razão do sistema existir a estrutura que será manipulada pelo sistema. Por exemplo, em um sistema de cadastro de alunos qual é razão do sistema existir? O que ele irá manipular? Se você observar para todas as perguntas, a resposta é alunos, o sistema existe para manipular alunos, e armazenar informações sobre eles. Então aluno será uma classe de negócio (Model) da sua aplicação.

Já o pacote **Controller** terá as classes de manipulação sobre o negócio, ou seja, as ações que serão realizadas com o negócio da aplicação. Por exemplo, no nosso sistema de cadastro de alunos pode ser necessário as ações cadastrar, excluir, alterar e visualizar os alunos, essas ações serão a nossa classe de controle.

E por fim o pacote **View** será a interface de comunicação com o usuário, ela permitirá que ele escolha uma opção a ser realizada, essa escolha chamará a funcionalidade correspondente da classe de controle que manipulará a classe de negócio. O resultado da operação também será mostrado para usuário através da classe de visão.

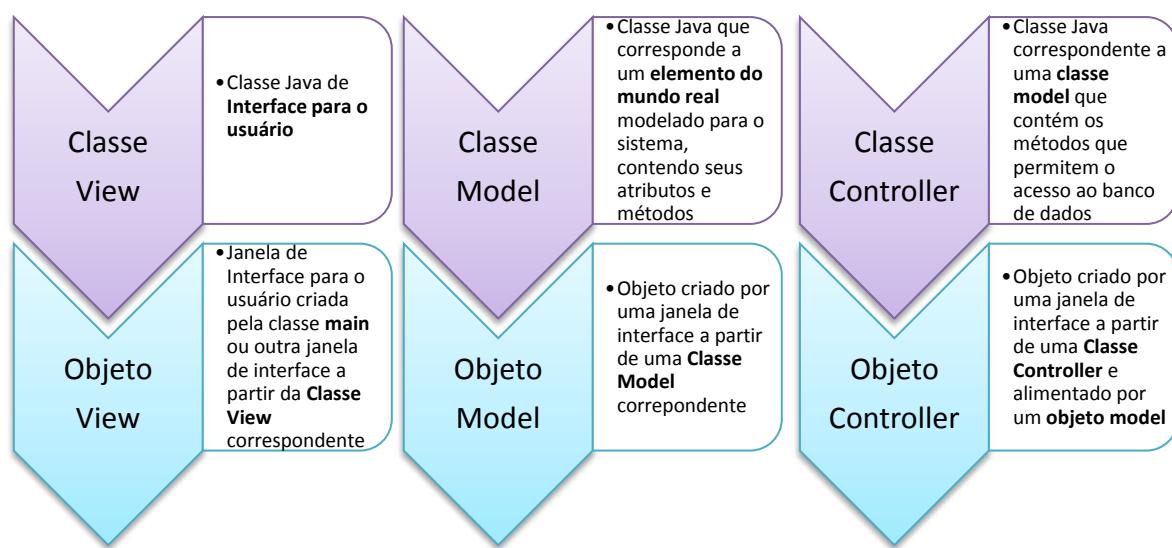


Figura 76 - Padrão MVC



Exercícios Propostos

14 Implementação do Projeto

14.1 Identificação do Projeto

Turma chegamos à parte final da nossa apostila, nessa fase vamos criar um sistema passo a passo utilizando o projeto MVC criado anteriormente. Portanto você irá aprender muito sobre a interação entre as classes em um projeto MVC, também irá conhecer os componentes para criar interfaces gráficas do Java e utilizar um vetor de objetos como banco de dados. Tudo isso implementando os conceitos de orientação a objeto apresentados na fase anterior.

Agora vamos aprender como organizar nosso projeto no padrão MVC, durante todo o nosso curso iremos utilizar a IDE Netbeans para desenvolver nossos projetos.

14.2 Criando o Projeto

Você já deve ter o Java e Netbeans instalado em sua máquina.

Primeiro abra a IDE Netbeans.

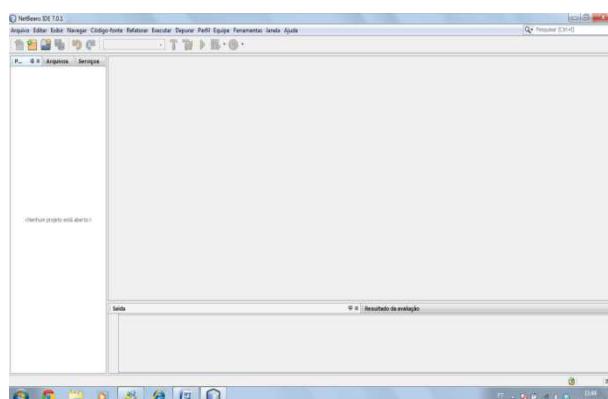


Figura 77 - Tela inicial NetBeans

Crie um novo projeto clicando em Arquivo – Novo projeto

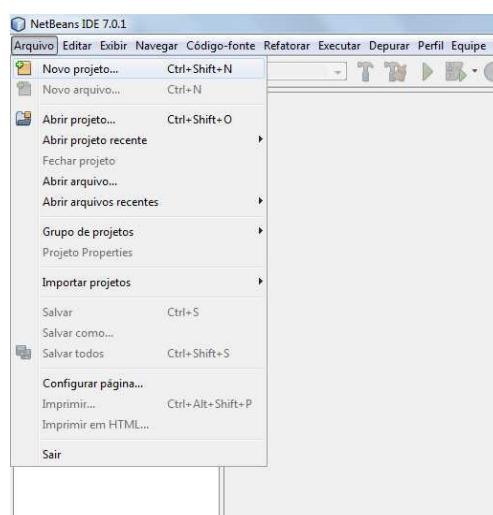


Figura 78 - Criando um novo projeto

Na tela para criação do novo projeto escolha a opção Java – Aplicativo Java e clique em próximo.

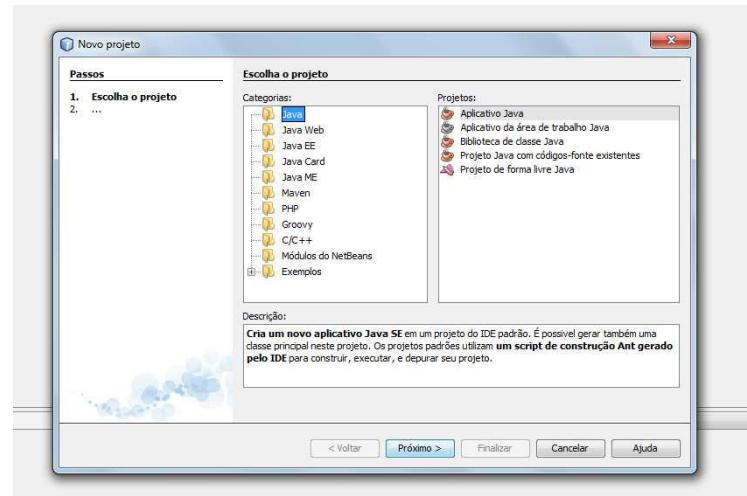


Figura 79 - Escolha o tipo do projeto

Na tela seguinte coloque o nome do projeto de ProjetoMVC e clique em finalizar

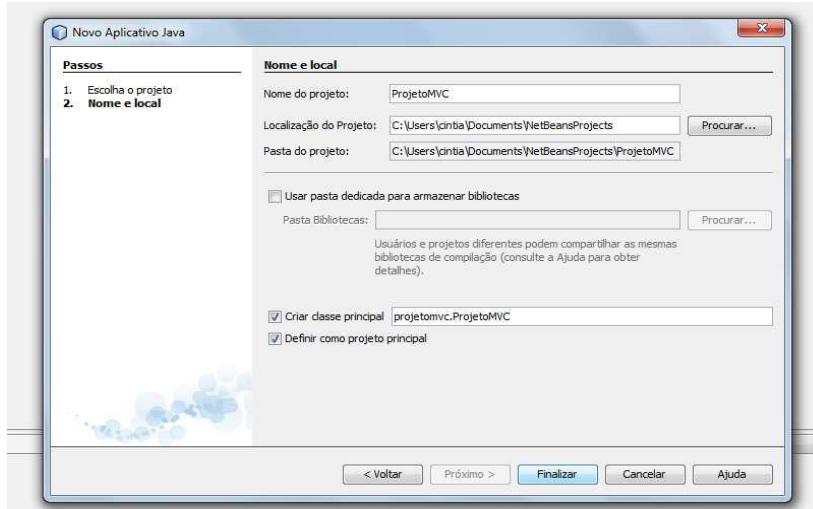


Figura 80 - Nome do projeto

Pronto nosso projeto foi criado.

Perceba que temos um pacote com o nome de projetomvc que possui uma classe chamada ProjetoMVC.java.



Figura 81 - Projeto criado

14.3 Criando os Pacotes MVC

Clique com o botão direito no pacote projetomvc, escolha a opção Novo depois Pacote Java.

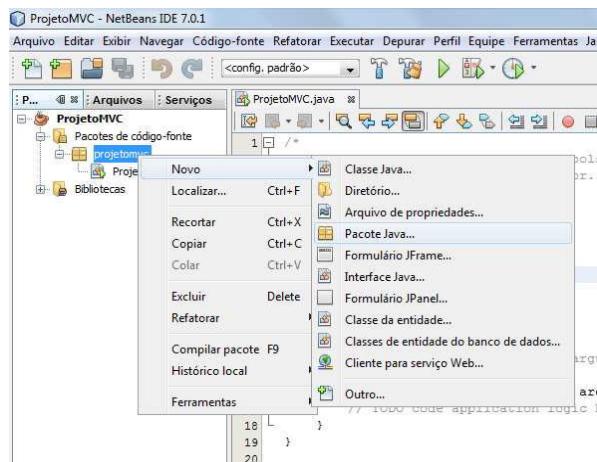


Figura 82 - Criando um novo pacote

Na tela seguinte coloque o nome do pacote de projetomvc.model e clique em finalizar. Em seguida crie mais dois pacotes chamados projetomvc.controller e projetomvc.view.

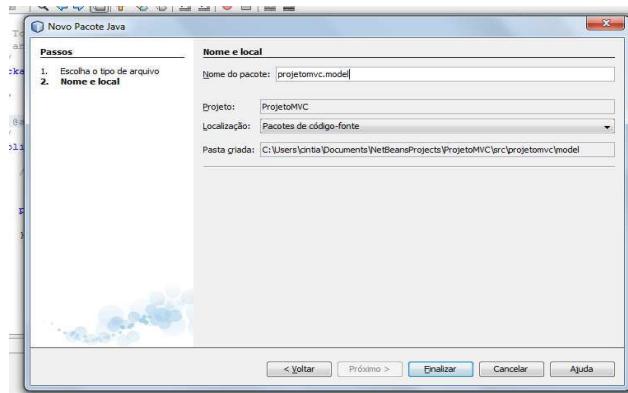


Figura 83 - Nomeando o pacote

Seu projeto deve estar da seguinte forma.



Figura 84 - Pacotes criados

Agora exclua o pacote 'projetomvc.java', para isso clique com o botão direito no projeto e escolha a opção excluir. Seu projeto deve ficar com a seguinte aparência.

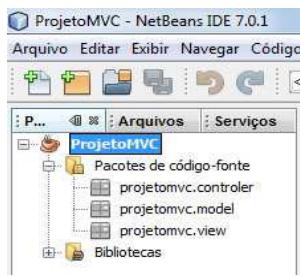


Figura 85 - Projeto criado

Nosso projeto agora está organizado no padrão MVC.

Primeiro vamos definir sobre o que será o nosso sistema. Iremos construir um sistema de controle de produtos que terá como funcionalidades básicas; cadastrar, excluir, alterar e visualizar. Tudo manipulando um vetor de objetos de produtos.

14.4 Pacote Model

Precisamos definir nossa classe de negócio (Model) da aplicação. Se o sistema será para manipulação de produtos está claro que o negócio da nossa aplicação são produtos, então devemos criar uma classe chamada Produto.

Para isso clique com o botão direito no projetomvc.model, escolha a opção Novo em seguida Classe Java, coloque o nome da classe de Produto(Lembre-se nome de classe inicia com letra maiúscula) e clique em Finalizar .

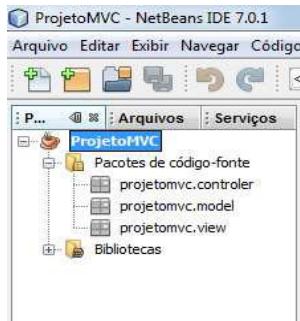


Figura 86 - Projeto MVC

Veja se o projeto está dessa forma:

A classe Produto foi criada e está pronta para ser programada.

```

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package projetomvc.model;

public class Produto {
}

```

Figura 87 - Criação da classe produto

Agora vamos definir quais são as informações que precisamos ter sobre o produto no momento que for cadastrado.

Vamos solicitar que seja informado o nome, código, o preço e data de validade do produto.

Tudo isso serão variáveis da nossa classe produto, então vamos criá-las? Lembre-se de colocar todas como private, pois vamos implementar o conceito de encapsulamento, não queremos que as nossas variáveis sejam vistas por todos.

O código será o seguinte:

```
private String nome;
private int código;
private double preço;
private String data;
```

Veja se ficou da seguinte forma:

```
/*
 * To change this template, choose
 * and open the template in the editor.
 */
package projetomvc.model;

public class Produto {

    private String nome;
    private int código;
    private double preço;
    private String Data;
}
```

Figura 88 - Variáveis classe produto

Agora precisamos criar os métodos de acesso as variáveis (métodos get e set). Vamos usar um atalho do Netbeans.

Dentro da classe pressione as teclas ALT e INSERT e escolha a opção Getter e Setter.

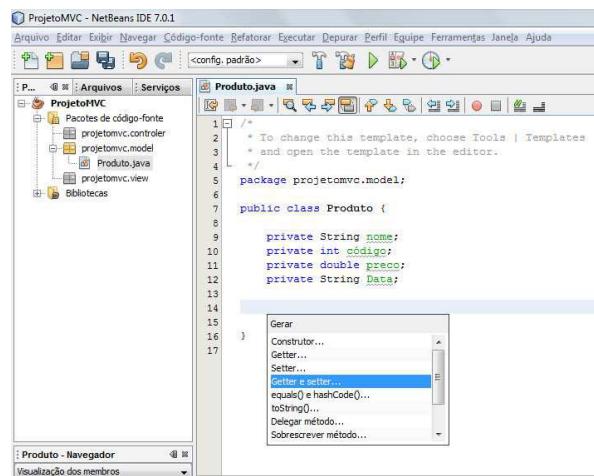


Figura 89 - Atalho para get e set

Depois marque todas as variáveis que deseja gerar os métodos get e set e clique em gerar.

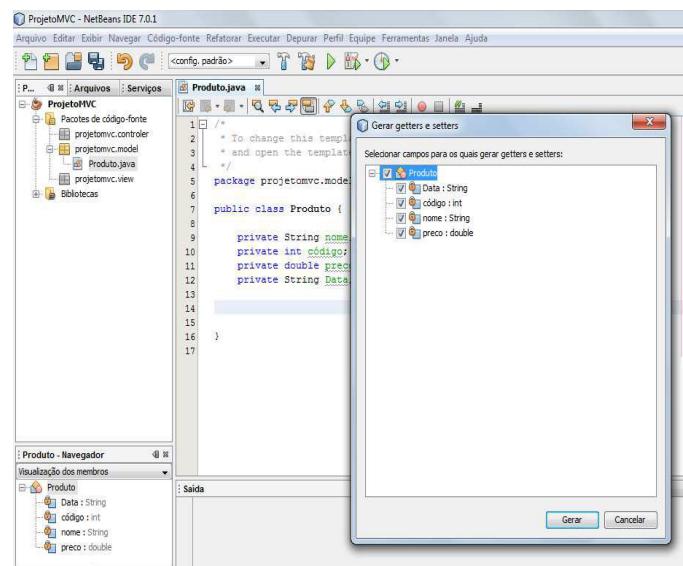
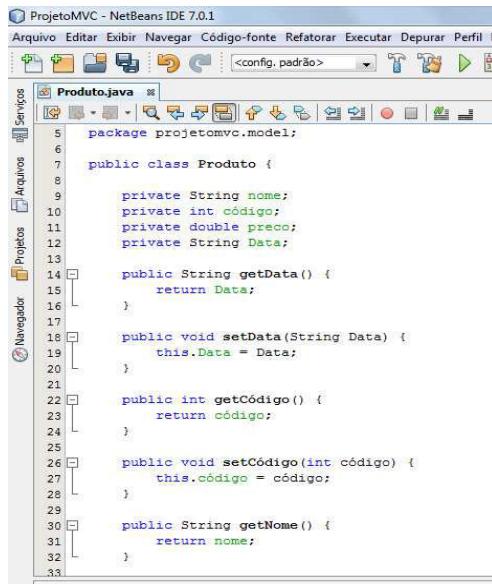


Figura 90 - Gerando métodos get e set

Pronto os métodos Getters e Setters detodas as variáveis foram gerados.



```

5 package projetomvc.model;
6
7 public class Produto {
8
9     private String nome;
10    private int código;
11    private double preço;
12    private String Data;
13
14    public String getData() {
15        return Data;
16    }
17
18    public void setData(String Data) {
19        this.Data = Data;
20    }
21
22    public int getCódigo() {
23        return código;
24    }
25
26    public void setCódigo(int código) {
27        this.código = código;
28    }
29
30    public String getNome() {
31        return nome;
32    }
33}

```

Figura 91 - Classe produto pronta

14.5 Pacote View

Terminado a classe de negócio da nossa aplicação, vamos agora criar a interface gráfica do sistema que será no nosso pacote View.

14.5.1 Tela de Login

Vamos aprender como criar uma interface gráfica em nosso projeto. Clique com o botão direito no projetomvc.view, Escolha a opção Novo – Formulário JFrame.

Coloque o nome de JfLogin e clique em finalizar.

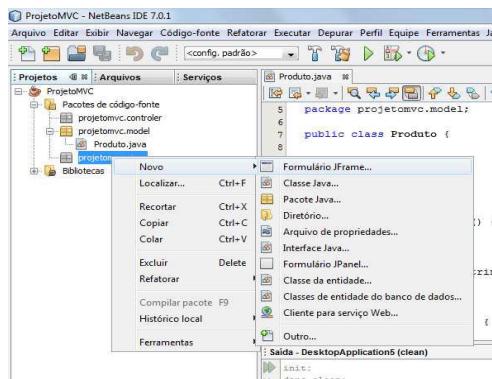


Figura 92 - Criando a tela de login

Veja se ficou como na figura abaixo.

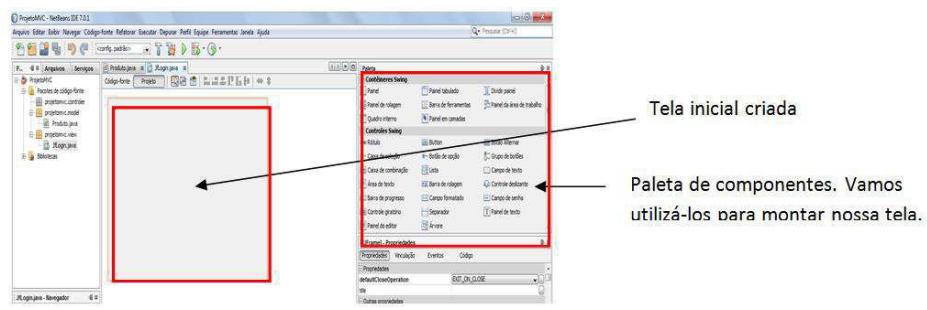


Figura 93 - Iniciando a tela de login

Para que possamos montar nossas telas precisamos conhecer a funcionalidade da paleta de componentes. Portanto vamos estudá-los agora.

Cada componente é uma classe do pacote Swing (`javax.swing`) do Java, portanto cada componente possui métodos (ações) prontos para que possamos utilizar em nossos sistemas.

Vamos estudar os componentes mais comuns com os seus métodos mais importantes.

Componente JLabel



Figura 94 - Componete JLabel

O JLabel é utilizado para mostrar um texto na tela que normalmente oferece informação para o usuário de forma a tornar a tela mais comprehensível.

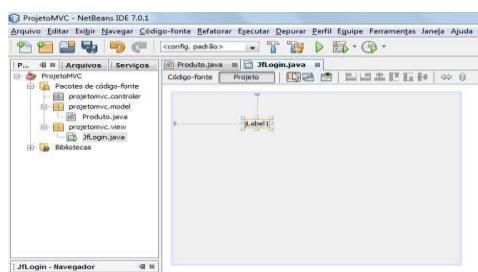


Figura 95 - Inserindo componente JLabel

Vamos agora inserir um JLabel em nossa tela inicial. Clique no componente arrate e solte em cima de sua tela. O resultado desta operação pode ser visto na Figura 95. É possível redimensionar o componente, como fazemos para redimensionar uma imagem.

Vamos aprender agora algumas propriedades do nosso componente JLabel. Você pode utilizar essas propriedades através da barra ao lado

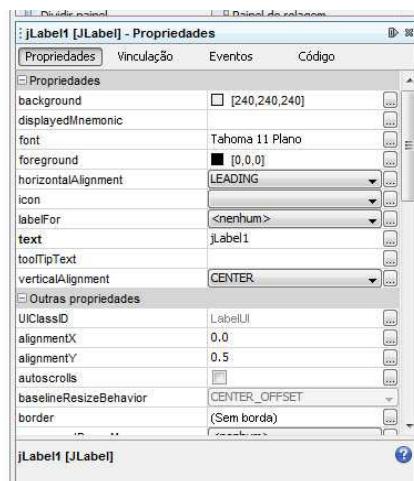


Figura 96- propriedades JLabel

- ✓ **Background:** Altera a cor de fundo.
- ✓ **Font:** Altera o tamanho e o tipo da letra.
- ✓ **Foreground:** Altera a cor do texto.
- ✓ **Text:** Altera o texto exibido dentro do componente.

Agora altere a propriedade Text para Sistema Controle de Produtos, coloque a font High Tower Text, tamanho 24 e altere a cor do texto para vermelho. Sua tela deve ficar como ostrado na Figura 97:



Figura 97 - Início da tela de Login

Componente JTextField

Um textfield é como um label, só que pode ser editado e modificado pelo usuário. Textfields são usados geralmente para entrada de dados pelo usuário. Na paleta de componentes você encontra o componente com o nome Campo de Texto.

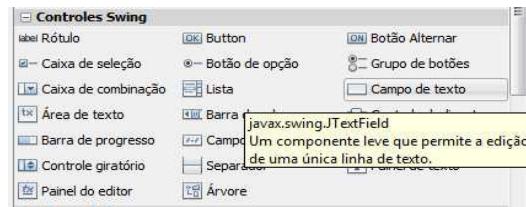


Figura 98 - Componente JTextField

Deixe sua tela agora com a seguinte aparência



Figura 99 - Adicionando JTextField a tela de Login

Para isso adicione dois JLabel e altere seus textos para Login e Senha. Coloque tamanho da letra 14. Em seguida adicioneum JTextField. Para retirar o nome que aparece dentro do textFieldbasta apagar o texto dentro da sua propriedade text.

Componente JPasswordField

Vamos utilizar esse componente paraque ao digitar a senha ninguém possa visualizá-la. Ele irá adicionar uma máscara na senha. Na paleta de componentes ele é chamado de Campo de senha. Adicione um JPasswordField na tela de Login, no Campo senha.

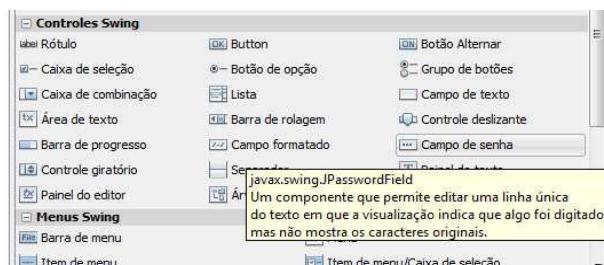


Figura 100 - Componete JPasswordField

Componente JButton

Um botão é uma região clicável coma qual o usuário interage de forma a disparar uma ação. Na paleta de componentes ele é chamado Button.

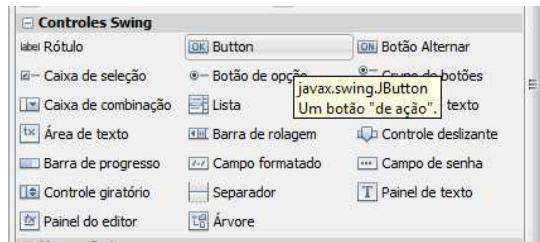


Figura 101 - Componente JButton

Insira um componente JButton em nossa tela inicial e altere a sua propriedade text para Login. A tela deve ficar da seguinte forma:



Figura 102 - Tela de Login finalizada



Alterando o nome da variável dos componentes

Cada um dos componentes adicionados precisa ser identificado por um nome internamente, como sendo uma variável. Isso ajuda o programador na hora de identificar os componentes.



Para alterar o nome da variável de um componente clique sobre ele com o botão direito e escolha a opção Alterar o nome da variável.

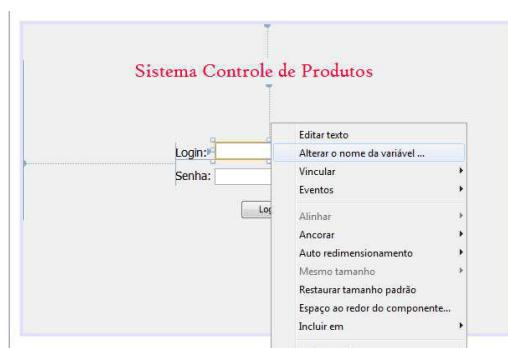


Figura 103 - Alterando o nome das variáveis

Por convenção, costumamos nomear os componentes de acordo com o tipo de componente. Por exemplo, um JTextField começamos o nome com Jtf um JButton com Jb, JPasswordField com Jpf e assim por diante.

Agora altere o nome da variável do JTextField para JtfLogin e do JPasswordField para JpfSenha e o nome do JButton para JbLogin. Dessa forma saberemos que tipo de componente é, e qual a sua funcionalidade no sistema.



14.5.2 Tela de Cadastro de Produtos

Vamos criar a tela para cadastro dos produtos, para isso crie um nova tela (Formulário JFrame) no pacote View coloque o nome de JfCadastro e adicione a ele os seguintes componentes:

Componente	Nome da Variável	Texto
jLabel	Alteração desnecessária	Cadastro de Produto
	Alteração desnecessária	Nome do Produto
	Alteração desnecessária	Preço do Produto
	Alteração desnecessária	Data de Validade
jTextField	JtfNome	Sem texto
	JtfPreco	Sem texto
	JtfData	Sem texto

Componente	Nome da Variável	Texto
jButton	JbCadastrar	Cadastrar
	JbBuscar	Buscar
	JbExcluir	Excluir
	JbAlterar	Alterar

A tela deve ficar no final com a aparência da Figura 104:



Figura 104 - Tela cadastro de produto

14.5.3 Codificando o Botão OK da Tela de Login

Já criamos nossas telas gráficas, mas elas ainda não têm nenhuma funcionalidade. Por exemplo, na tela de Login o usuário deverá digitar um usuário e senha e depois clicar no botão Login, se o usuário e a senha estiverem corretos será permitido o acesso ao sistema.

Porém, nosso botão ainda não consegue ter nenhuma funcionalidade. Para resolver esse problema iremos chamar o evento ActionPerformed do botão tudo que for programado dentro desse evento será executado no momento que o usuário clicar no botão. Ou seja, o evento funcionará como um método sendo invocado.

Evento ActionPerformed componente JButton

Vamos agora programar nossa tela de Login. Para chamar o evento actionPerformed, clique com o botão direito no botão Login escolha a opção Eventos – Action – actionPerformed.

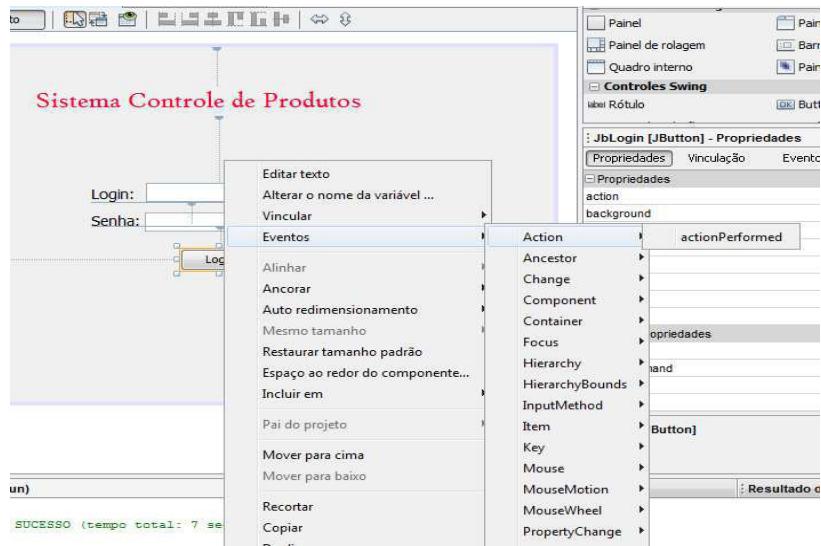


Figura 105 - Evento actionPerformed do botão Login

Nesse momento você será direcionado para o código fonte da sua interface, não se preocupe com todos os códigos existentes, vamos nos preocupar somente com o evento de nosso botão.

```

105
106 private void JbLoginActionPerformed(java.awt.event.ActionEvent evt) {
107     // TODO add your handling code here:
108 }
109 /**
110 */

```

Figura 106 -Método do botão login

Como pode perceber na imagem acima o método de ação do nosso botão foi gerado. Agora precisamos inserir o código para validação do usuário e senha. Digite o código abaixo.

```

107
108 private void JbLoginActionPerformed(java.awt.event.ActionEvent evt) {
109
110     String usuario = JtfLogin.getText();
111     String senha = new String(JpfSenha.getPassword());
112     if (senha.equals("") || usuario.equals("")){
113         JOptionPane.showMessageDialog(null, "Digite um usuário e senha");
114     } else {
115         if (usuario.equals("informatica") && senha.equals("java")) {
116             JfCadastro cad = new JfCadastro();
117             cad.setVisible(true);
118         } else {
119             JOptionPane.showMessageDialog(null, "Usuário ou Senha inválidos");
120         }
121     }
122     JtfLogin.setText("");
123     JpfSenha.setText("");
124 }
125

```

Figura 107 - Código fonte do botão Login

Nesse exemplo estamos utilizando um usuário e senha padrão, ou seja todas as pessoas que desejarem fazer login em nosso sistema devem usar o usuário informática e a senha java.

Vamos entender o código.

Linha 110 – Criação da variável usuário que recebe o valor capturado pelo método getText() do campo JtfLogin.

Linha 111 – Criação de um objeto da classe String chamado senha que recebe como valor a senha capturada pelo método getPassword() do campo JpfSenha

Linha 112 – Verifica se um dos dois campos estão vazios, se sim, retorna uma mensagem de erro pedindo ao usuário que digite os valores.

Linha 115 – Se os campos não estiverem vazios, é verificado se o usuário e a senha digitados são “informática” e “java”, se sim, o login é realizado.

Linha 116 – Criação do objeto da tela de cadastro, é necessário criar objeto para que a tela seja chamada.

Linha 117 – Setando a tela de cadastro como visível.

Linha 118 – Caso o usuário ou a senha não sejam compatíveis com o esperado, retornará uma mensagem de erro para o usuário.

Linhas 122 e 123 – Após todas as operações os campos de usuário e senha são limpos.

14.5.4 Codificando o Botão Cadastrar da Tela de Produto

Primeiro passo é ativar o evento ActionPerformed de cada botão. Faça isso! O resultado deve ser esse abaixo. Todos os botões com seus eventos ativos.

```
private void JbCadastrarActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}  
  
private void JbBuscarActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}  
  
private void JbExcluirActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}  
  
private void JbAlterarActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```

Figura 108 - Eventos ActionPerformed botões cadastro de produtos

Vamos agora começar pelo botão cadastrar. Para realizar o cadastro é necessário capturar os dados do produto da tela de cadastro. Para isso no evento do botão cadastrar digite o seguinte código:

```
private void JbCadastrarActionPerformed(java.awt.event.ActionEvent evt) {
    String nomePro = JtfNome.getText();
    double precoPro = Double.parseDouble(JtfPreco.getText());
    String data = JtfData.getText();
}
```

Figura 109 - Captura de dados do botão cadastrar

Pronto, agora já capturamos os valores. Precisamos agora armazenar os dados do produto. Mas temos um problema. Onde vamos armazenar os dados? Você lembra? Isso mesmo em vetor de objetos. Porém esse vetor ainda não foi criado, e muito menos a classe de controle que irá controlar as ações no vetor. Então o primeiro passo é criar a classe de controle.

Crie uma classe chamada ControleProduto no pacote projetomvc.controler. Deverá ficar da seguinte forma:



Figura 110 - Criação da classe ControleProduto

Na classe de controle vamos criar nosso vetor. Lembre-se que o vetor deve ser do tipo de Produto para que possamos armazenar objetos de produto. Teremos que informar o tamanho do vetor vamos colocar 5. Vamos também criar uma variável do tipo de produto, uma variável chamada i inicializada com 0 e uma variável cod também inicializada com 0, e uma variável com nome achou do tipo boolean. Veja como ficou.

```
5 package projetomvc.controler;
6
7 import projetomvc.model.Produto;
8
9 public class ControleProduto {
10
11     Produto[] produto = new Produto[5];
12     Produto pro = null;
13     int i = 0;
14     int cod = 0;
15     boolean achou;
```

Figura 111 - Variáveis da classe de Controle

Agora vamos codificar o nosso método cadastrar na classe ControleProduto.

```

15
16     public void cadastrar(String nome, String data, double preco){
17         cod++;
18         pro = new Produto (nome, data, preco, cod);
19         produto[i] = pro;
20         i++;
21     }
22
23
24     j
25

```

Figura 112 - Método cadastrar classe ControleProduto

Vamos entender o código.

Linha 16 – Declaração do método cadastrar que irá receber como parâmetro os dados para cadastro da tarefa, ou seja esses dados serão repassados a partir da classe de interface.

Linha 17 - Incremento da variável cod, essa variável será o controle do código de cada produto, portanto cada vez que o método cadastrar for chamado o valor dela será alterado gerando um novo código para o produto.

Linha 18 – Criação do novo objeto de produto passando como parâmetro os valores do produto que serão armazenado nas suas respectivas variáveis na classe produto.

Linha 19 – Armazenando o objeto gerado no vetor.

Linha 20 – Incrementando a variável i. Essa variável funcionará como controle para as posições do vetor.

Agora vamos ver como fica o restante da implementação do evento no botão cadastrar. Utilizamos um objeto chamado pro da classe ControleProduto para chamarmos o método cadastrar.

```

162 private void jbCadastrarActionPerformed(java.awt.event.ActionEvent evt) {
163     String nomePro = JtfNome.getText();
164     double precoPro = Double.parseDouble(JtfPreco.getText());
165     String data = JtfData.getText();
166     pro.cadastrar(nomePro, data, precoPro);
167     JOptionPane.showMessageDialog(null, "Produto cadastrado com sucesso");
168     limpar();
169 }

```

Figura 113 - Método cadastrar na classe JFCadastrar

Linha 168 – Nessa linha estamos chamando o método limpar(). O código dele esta demonstrado logo abaixo a sua função é limpar os campos após o cadastro.

```

153     public void limpar() {
154         JtfNome.setText("");
155         JtfPreco.setText("");
156         JtfData.setText("");
157     }

```

Figura 114 - Método Limpar classe JFCadastro

14.5.5 Codificando o Botão Buscar da Tela de Produto

Para implementar a busca usaremos a seguinte lógica: o usuário irá informar o nome do produto que deseja buscar e depois clicar no botão buscar. A classe de controle irá percorrer o vetor e localizar o objeto, realizando assim o retorno do mesmo. Se a classe não encontrar o objeto ela irá retornar null, e o programa irá mostrar uma mensagem que o produto não existe. Vamos ver como fica a implementação do evento do botão Buscar.

```

170
171     private void JbBuscarActionPerformed(java.awt.event.ActionEvent evt) {
172         nomeBusca = JtfNome.getText();
173         produto = pro.buscar(nomeBusca);
174         if (produto == null) {
175             JOptionPane.showMessageDialog(null, "Produto não existe");
176             limpar();
177         } else {
178             JtfNome.setText(produto.getNome());
179             JtfData.setText(produto.getData());
180             JtfPreco.setText(String.valueOf(produto.getPreco()));
181         }
182     }

```

Figura 115 - Método Buscar tela JfCadastro

Linha 171 – Declaração do método de ação para o botão Buscar

Linha 172 – A variável nomeBusca está recebendo o nome para busca digitado no campo nome.

Linha 173 – O objeto da classe ControleProduto chamado pro esta chamando o método buscar da classe de controle(ControleProduto) e esta passando como parâmetro o nome do produto para busca. O retorno do método buscar será um objeto contendo as informações do produto buscado.

Linha 174 – Se o produto não existir irá retornar null, e a mensagem informando que o produto não existe irá aparecer, em seguida os campos serão limpos.

Linha 178 – O objeto produto retornado esta capturando o nome do produto e setando no campo de texto do nome.

Linha 179 – O objeto produto retornado esta capturando a data do produto e setando no campo de texto da data.

Linha 180 – O objeto produto retornado esta capturando o preço do produto e setando no campo de texto referente ao preço. A utilização do comando String.valueOf é para converter o preço do produto de double para String, dessa forma é possível exibir o valor no campo de texto.

Vamos ver agora como fica o método buscar na classe ControleProduto.

```

24
25     public Produto buscar(String nomeBusca) {
26         achou = false;
27         for (int a = 0; a < produto.length; a++) {
28             String nome = produto[a].getNome();
29             if (nome.equals(nomeBusca)) {
30                 pro = produto[a];
31                 achou = true;
32             } else if (achou == false) {
33                 pro = null;
34             }
35         }
36         return pro;
37     }
38

```

Figura 116 - Método buscar classe ControleProduto

Linha 25 – Criação do método para realizar a busca, com parâmetro chamado nomeBusca do tipo String, o tipo de retorno do método é Produto, pois no final irá retornar um objeto de produto.

Linha 26 – Inicialização da variável achou como false, esse valor será importante caso não seja encontrado o objeto da busca.

Linha 27 – Criação do for para realizar o loop até a variável a ser menor que o vetor chamado produto, o método length serve para analisar o tamanho do vetor.

Linha 28 – A variável nome recebe o nome do objeto armazenado no vetor produto

Linha 29 – Verifica se o nome retornado é igual ao nome que foi passado na busca.

Linha 30 – Se os nomes forem iguais o objeto do vetor será guardado em objeto do tipo produto.

Linha 31 – Como o objeto foi encontrado a variável achou passa a ser true.

Linha 32 – Verifica se a variável achou é igual a false, se for na linha 33 o objeto de retorno recebe null.

Linha 36 – O objeto é retornado para o método buscar da nossa tela.

14.5.6 Codificando o Botão Alterar da Tela de Produto

Para fazer uma alteração em um produto, o usuário precisa realizar primeiro a busca dos dados, só então ele pode alterar um dado e em seguida clicar no botão alterar para que a modificação seja salva.

Vamos ver como fica a implementação do evento do botão Alterar.

```

195  private void JbAlterarActionPerformed(java.awt.event.ActionEvent evt) {
196      String nomeAltera = JtfNome.getText();
197      double precoPro = Double.parseDouble(JtfPreco.getText());
198      String data = JtfData.getText();
199      pro.alterar(nomeBusca, nomeAltera, data, precoPro);
200      JOptionPane.showMessageDialog(null, "Produto alterado com sucesso");
201      limpar();
202  }
203

```

Figura 117 - Método alterar na classe JfCadastro

Linha 195 – Declaração o evento actionPerformed do botão alterar.

Linha 196 – A variável nomeAltera recebe o conteúdo do campo nome.

Linha 197 – A variável precoPro recebe o conteúdo do campo preço convertido para double.

Linha 198 – A variável data recebe o conteúdo do campo data

Linha 199 – O objeto pro chama o método alterar da classe de controle e passa como parâmetro o nomeBusca, essa variável contém o nome do objeto que será alterado, e as variáveis que contém os novos valores que serão armazenados no objeto.

Linha 200 e 201 – Exibe uma mensagem de sucesso na alteração e limpa os campos.

Vamos ver agora como fica o método alterar na classe de Controle

```

39  public void alterar(String nomeBusca, String nomeAltera, String data, double preco) {
40      for (int a = 0; a < produto.length; a++) {
41          String nome = produto[a].getNome();
42          if (nome.equals(nomeBusca)) {
43              pro = produto[a];
44              pro.setNome(nomeAltera);
45              pro.setData(data);
46              pro.setPreco(preco);
47          }
48      }
49  }
50

```

Figura 118 - Método alterar na classe ControleProduto

Linha 39 – Declaração do método alterar que recebe 4 parâmetros, o primeiro será o nome do objeto que servirá como referência para alteração e os outros são os dados do produto que serão alterados.

Linha 40 – O for para percorrer o vetor.

Linha 41 – A captura do nome de cada objeto para verificar se é o objeto certo.

Linha 42 – Verifica se é o objeto certo para ser alterado.

Linha 43 – Captura o objeto e guarda na variável pro.

Linha 44, 45 e 26 – Seta os novos valores para o objeto.

14.5.7 Codificando o Botão Excluir na Tela de Produto

O método excluir vai partir do mesmo princípio do alterar, ou seja, é necessário primeiro realizar a busca do produto e só depois clicar em excluir.

Vamos ver como fica a implementação do evento do botão Excluir.

```

187
188 [- private void JbExcluirActionPerformed(java.awt.event.ActionEvent evt) {
189
190     pro.excluir(nomeBusca);
191     JOptionPane.showMessageDialog(null, "Produto excluido com sucesso");
192     limpar();
193 }
```

Figura 119 - Método excluir na classe JfCadastro

Linha 188 – Declaração do método actionPerformed do botão excluir.

Linha 190 – O objeto pro chama o método da classe de controle excluir e passa como parâmetro o nome do produto que será excluído, esse nome foi capturado devido a busca que foi realizado no produto.

Linha 191 e 192 – exibe uma mensagem de sucesso e limpa os campos após a exclusão.

Vejamos como ficou o método excluir na classe de controle

```

51
52 [- public void excluir(String nomeExclui) {
53     for (int a = 0; a < produto.length; a++) {
54         String nome = produto[a].getNome();
55         if (nome.equals(nomeExclui)) {
56             produto[a].setData("");
57             produto[a].setNome("");
58             produto[a].setPreco(0);
59
60         }
61     }
62 }
```

Figura 120 - Método excluir na classe ControleProduto

Linha 52 – Declaração do método excluir que recebe como parâmetro o nome do produto que será excluído.

Linha 53 – O for que irá percorrer o vetor.

Linha 54 – Captura do nome de cada produto para verificação

Linha 55 – Verifica se o nome do produto é igual ao nome do produto que será excluído.

Linha 56, 57 e 58 – Se o nome for compatível com o nome do produto para exclusão, as variáveis do produto são setadas como vazia e a variável preço como 0.

14.6 Gerando um arquivo executável do sistema

Em Java não temos arquivos exe, como tínhamos em outras linguagens de programação como VB ou Delphi. Então como fazemos para executar nossa aplicação para qualquer lugar, sem ter que abrir o código fonte dela?

O Java preza pela portabilidade, por isso não cria arquivos exe, ele gera arquivos com a extensão .jar, isso possibilita que qualquer máquina com qualquer sistema operacional possa executar os aplicativos feitos em java, basta que tenha o kit de desenvolvimento Java instalado.

Um arquivo fonte Java (.java) é visto da mesma forma por qualquer sistema Operacional, o mesmo arquivo compilado no Windows, pode ser compilado no Linux, no MacOs, no Unix e assim por diante, que sempre irá ter o mesmo resultado e a mesma GUI(interface). Já os conjuntos de bytecodes (.class), que são os arquivos que a Máquina Virtual Java (JVM), usa para rodar os programas Java, também podem ser lidos pela JVM de qualquer Sistema Operacional.

Um arquivo .jar (Executable Jar File), tem o mesmo funcionamento de um arquivo executável .exe, ele esconde a implementação (código) do usuário, e roda o programa sozinho, basta clicar duas vezes sobre o mesmo, que ele dispara o programa. A principal vantagem de arquivo .jar é que ele é portável, como toda a linguagem Java. Isto quer dizer que você pode criar um arquivo executável (.jar) que vai funcionar em qualquer Sistema Operacional, sem você ter que mudar nada, nem mesmo uma linha ou vírgula.

Para gerar o arquivo jar clique com o botão direito do mouse em cima do projeto e escolha a opção limpar e construir. Veja a imagem abaixo:

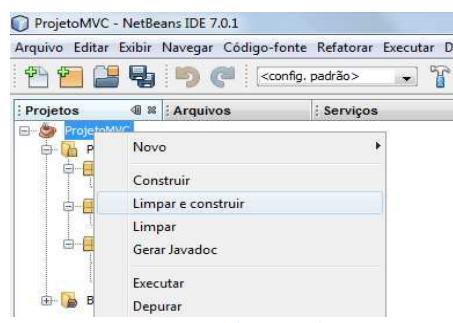


Figura 121 - Gerando o jar do projeto

Pronto o arquivo jar está criado, para localizá-lo vá até a pasta onde está seu projeto e localize a subpasta dist.

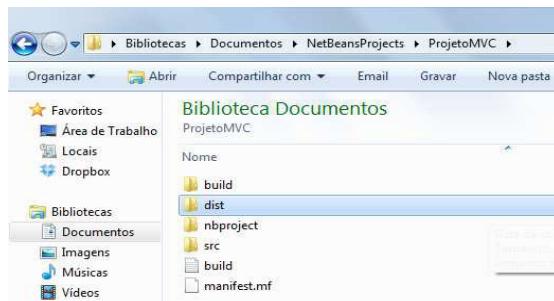


Figura 122 - Pasta dist do projeto

Dentro da pasta dist está o arquivo .jar que você poderá levar para qualquer lugar e executar o seu sistema bastando apenas dar dois cliques.

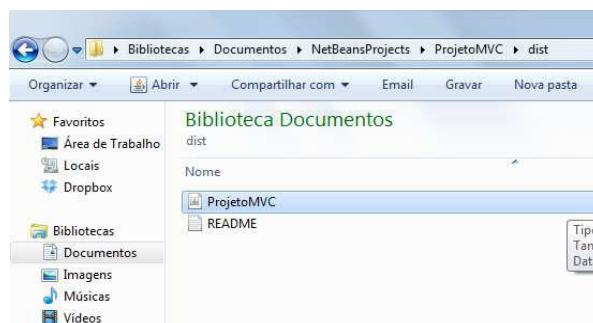


Figura 123 - Arquivo jar do projeto



Saiba

Para criar uma aplicação que seja inicializada automaticamente junto com o sistema operacional Windows, coloque seu arquivo .jar na pasta inicializar do sistema.



Chegamos ao fim do nosso projeto, é certo que ele não está 100% implementado, e que algumas funcionalidades e componentes poderiam ser adicionadas. Espero que tenha gostado de entender como funciona o padrão MVC e como podemos construir um “sisteminha” capaz de manipular informações sem precisar de um banco de dados.

Posteriormente, iremos otimizar o nosso sistema, migrando para um banco de dados, adicionando novas funcionalidades e utilizando mais recursos da linguagem Java.

15 Assuntos Complementares

15.1 Tratamento de Exceções em Java

“Exceção é um evento que acontece durante a execução de um bloco de código e, por algum motivo indevido (um erro), altera o curso normal de sua execução.”

Uma exceção indica um problema que ocorre durante a execução de um programa. O tratamento de exceções permite aos programadores criar aplicativos que podem tratar os problemas, permitindo que o programa continue funcionando.

O tratamento de exceções existe para processar erros que ocorrem quando uma instrução é executada, como índice de array inexistente, divisão de numero por zero, parâmetros inválidos de métodos e etc.

O Java possui uma hierarquia de classes que realizam tratamento das exceções mais comuns. As classes de exceções do Java herdam da classe `Exception`. Estendendo essa classe é possível criar suas próprias classes de exceções.

Veja abaixo a hierarquia de classe de exceções em Java:

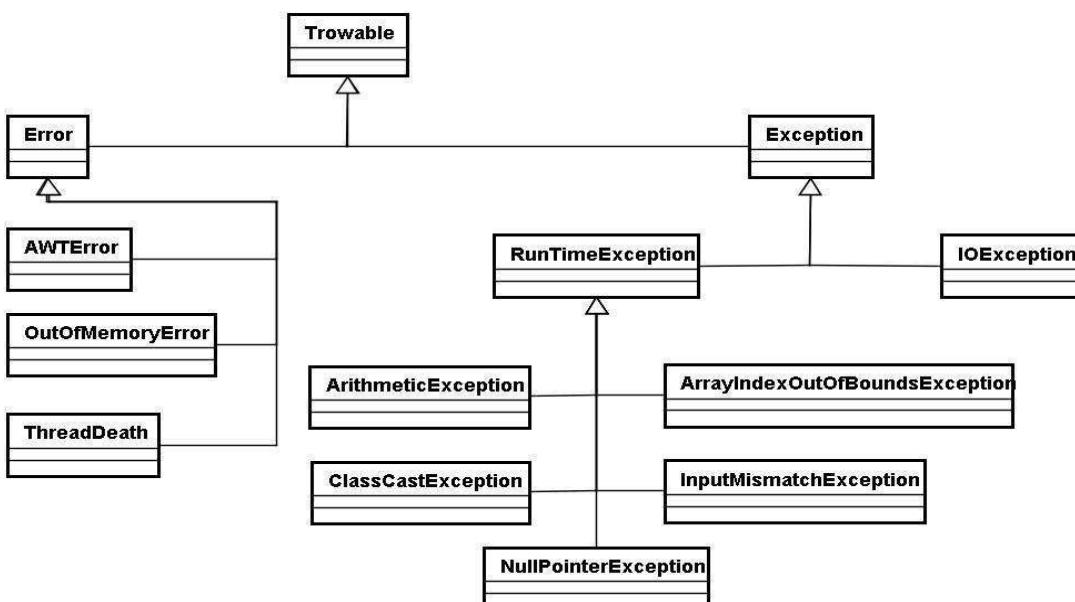


Figura 124 - Hierarquia de exceções em Java

- ✓ **Throwable**: superclasse de todas as exceptions.
- ✓ **Error**: erros irreparáveis, onde não adianta utilizar o mecanismo de tratamento de exceção. Exemplos: estouro de memória, erros na JVM;
- ✓ **Exception**: exceções tratadas pelo programador e o compilador obriga o programador a tratar os possíveis erros. Exemplos: Divisão por zero, índice do array inexistente.
- ✓ **RuntimeException**: exceções causadas por erros de lógica, ou seja, erros do programador. Exemplo: `NullPointerException`.

Iremos estudar as exceções do tipo `RunTimeException`.

AritmeticException: Esse tipo de exceção é disparada sempre que ocorre problemas relacionados a aritmética . Ex: divisão de um número por zero.

```

public class TestAritmeticaPrimitivos {
    public static void main(String[] args) {
        int numeroA = 30;
        int numeroB = 0;
        AritmeticaPrimitivos ariPri = new AritmeticaPrimitivos();
        System.out.println("Resultado Suma:" + ariPri.sumar(numeroA, numeroB));
        System.out.println("Resultado Resta:" + ariPri.restar(numeroA, numeroB));
        System.out.println("Resultado Multiplicación:" + ariPri.multiplicar(numeroA,
        System.out.println("Resultado División:" + ariPri.dividir(numeroA, numeroB))
    }
}

```

Figura 125 - Possível exceção do tipo ArithmeticException

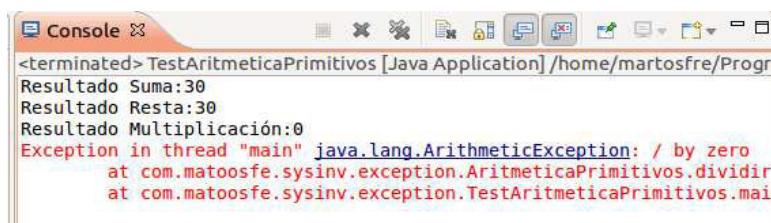


Figura 126 - Exceção ArithmeticException lançada

No exemplo acima perceba que é disparada uma exceção quando tentamos dividir um número por zero, e que a classe ArithmeticException informa o erro com a mensagem by zero.

InputMismatchException: ocorre quando o usuário digita um valos que não é do tipo do valor esperado. Ex: quando o método nextInt da classe Scanner recebe uma String.

ClassCastException: é um erro que ocorre quando tentamos fazer um cast (conversão explícita) de uma classe para outra classe diferente desta ou que não seja uma superclasse desta. Ou seja, as classes não possuem nenhuma relação portanto não pode haver a conversão de objetos.

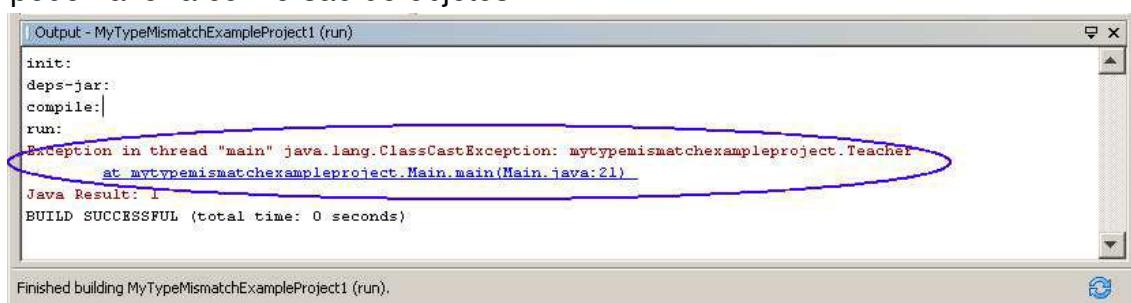
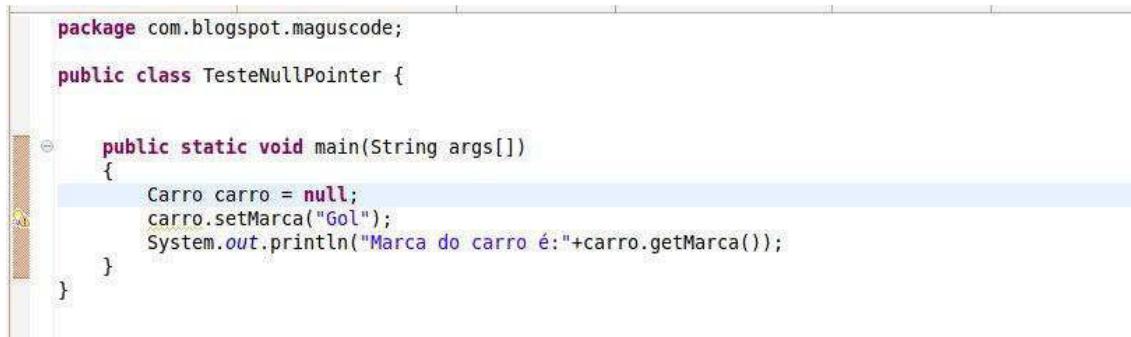


Figura 127 - Exceção do tipo ClassCastException lançada

ArrayIndexOutOfBoundsException: esse tipo de exceção ocorre quando tentamos acessar um elemento de um array inexistente, portanto deve-se lembrar que o array inicia sempre em 0 e estar atento ao limite das posições do array.

NullPointerException: ocorre quando tenta-se utilizar um atributo ou método de um objeto que não fora inicializado, ou seja, que esteja no estado **null**.



```

package com.blogspot.maguscode;
public class TesteNullPointerException {
    public static void main(String args[]) {
        Carro carro = null;
        carro.setMarca("Gol");
        System.out.println("Marca do carro é:" + carro.getMarca());
    }
}

```

Figura 128 - Possível exceção do tipo NullPointerException



```

Console > <terminated> TesteNullPointerException (1) [Java Application] /usr/local/jdk1.6.0_24/bin/java (03/02/2012 12:55:47)
Exception in thread "main" java.lang.NullPointerException
at com.blogspot.maguscode.TesteNullPointerException.main(TesteNullPointerException.java:9)

```

Figura 129 - Exceção do tipo NullPointerException lançada

No exemplo acima perceba que o objeto carro não foi instanciado por isso é gerado uma exceção do tipo NullPointerException.

A estrutura try-catch-finally

Essa estrutura tem como função desviar a execução do programa caso ocorra certos erros, predefinidos durante a codificação.

Veja abaixo como funciona a estrutura try-catch-finally

```

try {
    // código que pode lançar uma exceção
} catch (Exception1 e) {
    // tratamento da exceção
} catch (Exception2 e) {
    // tratamento da exceção
} finally {
    // código sempre executado
}

```

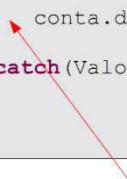
Figura 130 - Estrutura try-catch-finally

1. **Try:** bloco de código que tentará ser executado, mas que pode causar uma exceção;
2. **catch:** declara o bloco de código usado para tratar uma exceção;

3. **finally:** bloco de código, após um try-catch, que é executado independentemente do fluxo de programa seguido ao lidar com uma exceção, ou seja, o finally sempre é executado.

Regras no uso do try-catch-finally

```
void depositar(float valor) {
    conta.depositar(valor);
}
catch(ValorNegativoException ex) { }
```



Onde está o try ?

- ✓ não é permitido ter um block catch e finally sem um block try.

Figura 131 - Erro, catch sem try

- ✓ não é permitido inserir qualquer código entre os blocos try e catch.

Figura 132 - Erro, código entre o try e catch

- ✓ não é permitido ter um bloco try sem ter um bloco catch ou finally.

Figura 133 - Erro, try sem catch ou finally

Lançamento de exceções

Uma exceção é lançada usando-se a palavra chave throw seguida da referência à exceção.

```
void depositar(float valor) {
    try {
        void depositar(float valor) {
            try {
                conta.depositar(valor);
            }
        }
    }
}
```



Onde está o catch ou o finally?

No exemplo abaixo se a variável s for null será disparado uma exceção do tipo NullPointerException, se o tamanho da variável for igual a 0 será uma exceção do tipo IllegalArgumentException, se não for nenhuma das duas será disparada uma exceção padrão de exception.

```

try {
    if (s == null)
        throw new NullPointerException();
    else if (s.length() == 0)
        throw new IllegalArgumentException();
    else
        throw new Exception();
} catch (NullPointerException | IllegalArgumentException e) {
    e.printStackTrace();
}

158 private void JbCadastrarActionPerformed(java.awt.event.ActionEvent evt) {
159
160
161     try{
162         String nomePro = JtfNome.getText();
163         double precoPro = Double.parseDouble(JtfPreco.getText());
164         String data = JtfData.getText();
165         pro.cadastrar(nomePro, data, precoPro);
166         JOptionPane.showMessageDialog(null, "Produto cadastrado com sucesso");
167
168     }catch(NumberFormatException e){
169         JOptionPane.showMessageDialog(null, "Entrada inválida, cadastro não realizado!");
170
171     }finally{
172         limpar();
173     }
174
175 }
176

```

Figura 134 - Lançamento de exceção

```

try {
    if (s == null)
        throw new NullPointerException();
    else if (s.length() == 0)
        throw new IllegalArgumentException();
    else
        throw new Exception();
} catch (NullPointerException | IllegalArgumentException e) {
    e.printStackTrace();
}

```

Figura 135 - Lançamento de exceção

Se o programador desejar que a exceção assim lançada seja tratada fora do método que a gerou, ele deve explicitar isto usando a palavra chave throws seguida do tipo de exceção, na declaração do método.

No exemplo abaixo o método calcularFolha está informando que pode disparar uma exceção do tipo `IllegalArgumentException` para isso ele está utilizando o comando `throws`.

```
void calcularFolha(float valor)
    throws IllegalArgumentException {

    if (valor < 0) {

        throw new IllegalArgumentException(" +
            "Número negativo não permitido");
    }
}
```

Figura 136 - Passando uma exceção

Não confundir throws e throw.

throw: usado para lançar uma exceção para o método que o chamou;

throws: indica que um método pode passar uma exceção para o método que o chamou.

Veja como fica o método cadastrar na interface do nosso projeto, utilizando o tratamento de exceções:

Perceba que agora estamos implementando o tratamento de exceções e estamos tratando uma exceção do tipo `NumberFormatException`, que será disparada caso o usuário insira uma letra no campo do preço, dessa forma não será possível realizar a conversão para Double e a mensagem de erro será disparada, consequentemente o produto não será cadastrado e os campos serão limpos, para digitar novamente os dados do produtos.



Exercícios

15.2 Convenções de código Java

É importante programar seguindo convenções de código, isso ajuda a deixar o código mais legível, facilitando assim o entendimento do mesmo por parte dos integrantes da equipe de desenvolvimento.

No processo de desenvolvimento de software a maior parte é dedicada a manutenção do mesmo, portanto ter um código fácil de entender, facilita o processo de manutenção.

- ✓ 80% do custo do tempo de vida de uma peça de software vai para manutenção.
- ✓ Dificilmente qualquer software é mantido por toda a sua vida pelo autor original.
- ✓ Convenções de código melhorar a legibilidade do software, permitindo que os engenheiros para entender o novo código mais rapidamente e completamente.

As convenções de código para o documento Linguagem de Programação Java foi revisado e atualizado em 20 de abril de 1999.

Arquivos de código Java

A organização básica de um arquivo de código Java deve obedecer à seguinte ordem:

1. Comentários iniciais
2. Declarações de pacotes e importações
3. Declaração da classe/interface

Comentários iniciais

Todo arquivo de código deve iniciar com comentários de início, que devem conter informações a respeito da classe/interface, como por exemplo, nome da classe, informações de versão, data e autor.

```
/*
 * NewJFrame.java
 * Created on 02/01/2013, 09:41:08
 * @author cintia
 */
```

Figura 137 - Comentários inicio da classe

Declarações de pacotes e importações

A primeira linha de código após os comentários iniciais deve ser a declaração do pacote e em sequência as declarações de importações.

```
package pacotel;

import java.util.Scanner;
import java.util.ArrayList;
```

Figura 138 - Declarando pacote e importando

Declaração da classe/interface

A tabela abaixo mostra cada parte de um classe/interface, na ordem em que elas devem aparecer.

1	Comentário de documentação (** *)	Observações
2	Declaração da classe/interface	
3	Comentários de implementação da classe (* *)	Se necessário aqui pode ser inserido algum comentário a respeito da classe
4	Atributos estáticos	Inicialmente os públicos, depois os protegidos e por último os privados
5	Demais atributos	Inicialmente os públicos, depois os protegidos e por último os privados
6	Construtores	
7	Demais métodos	Agrupe os métodos por funcionalidade, exemplo: <i>getters</i> , <i>setters</i> , métodos de validação, métodos de cálculos, etc.

Figura 139 - Ordem de declaração na classe

Edentação (recesso de código)

Os recuos são utilizados para alinhar visualmente comandos pertencentes a blocos de código. Um bloco é o código envolvido pelos delimitadores { e }, como por exemplo o if. A regra geral é abrir o bloco na linha do comando e fechar alinhado a ele.

```
if (condicao) {
    //comandos
} else{
    //comandos
}
```

Figura 140 - Exemplo de identação

Normalmente se utiliza 4 espaços como medida padrão da tabulação, porém algumas vezes é necessário utilizar 8 espaços para obter uma visualização melhor do código. Devem ser evitadas linhas com mais de 80 caracteres.

Quebras de linha

Quando uma expressão não couber numa única linha, quebrá-la de acordo com as seguintes regras básicas:

- ✓ Após uma vírgula
- ✓ Antes de um operador
- ✓ Alinhar a nova linha com o início da expressão da linha anterior

Se as regras acima gerarem código confuso, ou se a linha de baixo ficar colada na margem, use uma tabulação de 8 espaços. Exemplos de quebra de linha:

```
executar Metodo(parametro1, parametro2, parametro3,
    parametro4, parametro5); //usando 8 espacos
int res = executar Metodo(parametro1, parametro2, parametro3,
    parametro4, parametro5); //alinhando com o inicio da
expressao anterior
```

Figura 141 - Quebra de linha em um método

```
double res = valor1 * (valor2 + valor3)
+ valor4 / valor5;
```

Figura 142 - Quebra de linha expressão aritmética

Declarações de variáveis

Ao declarar variáveis, observe as seguintes regras:

- ✓ Faça apenas uma declaração por linha, de modo a incentivar o uso de comentários.
- ✓ Inicialize apenas uma variável por linha.
- ✓ Use letras minúsculas e evite caracteres especiais.

Convenções de nomenclatura

Pacotes	Letras minúsculas sem caracteres especiais	gui
Classes e interfaces	Os nomes de classe devem ser substantivos, com a primeira letra de cada palavra interna em maiúscula. Tente manter seus nomes de classe simples e descritivos. Evite siglas e abreviações (a menos que a sigla seja muito mais usada do que a forma longa, como a URL ou HTML) .	class Pessoa class ContaCorrente
Métodos	Métodos devem ser verbos, em maiúsculas e minúsculas com a primeira letra minúscula, com a primeira letra de cada palavra interna em maiúscula.	calcularAlgo()
Variáveis	Devem iniciar com minúscula. Palavras internas começam com letras maiúsculas. Não deve começar com underline ou \$, mesmo que ambos sejam permitidos. Use nomes curtos, mas significativos. O nome deve indicar a intenção da utilização da variável. Os nomes comuns para variáveis temporárias são i, j, k, m, n e para inteiros, c, d, e e para caracteres.	idade nomeCompleto
Constantes	Constantes devem ter todas as letras maiúsculas separadas por underline	MAX_SIZE VALOR_PADRAO

Figura 143 - Convenções de nomenclatura

16 Referências Bibliográficas

1. **Davenport, T H e Prusak, L.** *Conhecimento empresarial*. Rio de Janeiro : Campus, 1988.
2. **ScriBD.** *Desenvolvimento de Software Orientado a Objetos*.
[<http://pt.scribd.com/doc/56735259/2/Desenvolvimento-de-Softwares-orientado-a-objetos>]
01/06/2012.
3. **Wikipedia.** *Java (linguagem de programação)*.
[http://pt.wikipedia.org/wiki/Java_%28linguagem_de_programa%C3%A7%C3%A3o%29 – 10/05/2012]
2012.
4. **Caelum Ensino e Inovação.** *Java e Orientação a Objetos*. [www.caelum.com - 10/05/2012]. São Paulo, São Paulo, Brasil : www.caelum.com, 10 de 05 de 2012.
5. **Wikipedia.** *Java Runtime Environment*. [<http://pt.wikipedia.org/wiki/Jre> – 11/05/2012] 2011.
6. —. *Java Development Kit*. [<http://pt.wikipedia.org/wiki/Jre> – Em: 11/05/2012] 2011. a.
7. **School of Information System.** *Java Virtual Machine*.
[https://wiki.smu.edu.sg/is200/Java_Virtual_Machine - 09/05/2012] s.l. : Singapore Management University.
8. **Kathy Sierra, Bert Bates.** *Use a Cabeça - Java*. Rio de Janeiro : O'Reilly, 2005.
9. **Antonio Manso, Célio Marques, Pedro Dias.** <http://www.dei.estt.ipt.pt/portugol>. s.l. : Instituto Politécnico de Tomar.
10. **Balagtas, Florence Tiu.** Projeto J.E.D.I. *Students Manual*. s.l. : <http://edu.netbeans.org/contrib/jedi/Intro-to-Programming-1/student-manual.pdf>, 2006.
11. **de Souza, Marcos Antônio Furlan e Gonçílio, Marcelo Marques.** *Algoritmos e Lógica de Programação*. s.l. : Thomson.
12. **Deitel, Harvey M. e Deitel, Paul J.** *Java como Programar 8ª Ed.* s.l. : Pearson.
13. **Forbellone, André Luiz Villar.** *Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados*. São Paulo : Makron, 1993.
14. **Puga, Sandra e Rissetti, Gerson.** *Lógica de Programação e Estruturas de Dados*. s.l. : Pearson.
15. **Gomes, Ana Fernanda e Veneruchi, Edilene Aparecida.** *Fundamentos da Programação de Computadores – Algoritmos, Pascal, C/C++, JAVA 2 Edição*. s.l. : Pearson Prentice Hall.
16. **Villar, André Luiz e Eberspacher, Henri F.** *Lógica de Programação a Construção de Algoritmos e Estruturas de Dados 3ª Ed.* s.l. : Pearson.
17. **Paterlini, Roberto Ribeiro.** *Fórmula versus algoritmo da resolução de um problema - RPM n.º 27, 1.º quadrimestre de 1995*.

18. **Echeverría, M.D.P.P. e Pozo, J.I.** Aprender a resolver problemas e resolver problemas para aprender.
[A. do livro] Juan Ignacio Pozo. Porto Alegre : Artmed, 1998.
19. **Pólya, G. A .** *Arte de Resolver Problemas*. Rio de Janeiro : Interciênciac, 1978.
20. **Soares, M. T. C e N.B., Pinto.** *Metodologia da resolução de Problemas. 24ª Reunião Anual da ANPED*.
Caxambu : ANPED, 2001.
21. **Brandão, Mirna Geyla Lopes.** Minha apostila. 2012.

17 Exercícios Propostos

FASE I – A UML COMO FERRAMENTA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

1 INTRODUÇÃO A SISTEMAS

2 Requisitos

3 História e Evolução da UML

1. Explique porque é importante modelarmos os sistemas antes de construí-los.

2. Explique o que é a UML e como ela surgiu.

3. Detalhe como é composta e dividida a UML.

4 Diagramas Estruturais da UML

4. Explique porque o diagrama de classe é um dos diagramas mais importante na modelagem de sistemas.

5. Descreva com suas palavras o que são métodos e atributos de uma classe.

6. Desenhe as seguintes classes com seus métodos e atributos.

- a) Pessoa
- b) Veiculo
- c) Aluno

7. Qual a importância dos diagramas estruturais na modelagem de sistemas?

8. Faça uma comparação detalhando as características de cada um dos diagramas abaixo:

- a) Diagrama de classe:

- b) Diagrama de Objeto:

- c) Diagrama de componentes:

d) Diagrama de Implantação:

e) Diagrama de pacotes:

f) Diagrama de estrutura:

5 Diagramas Comportamentais

9. Qual o objetivo dos diagramas de caso de uso?

10. Defina o que significa um ator? E qual a notação utilizada para representar um ator e um caso de uso?

11. Como podemos identificar os atores do nosso sistema?

12. Construa um modelo de caso de uso para a seguinte situação fictícia:

"Estamos criando um serviço de entregas. Nossos clientes podem requisitar a entrega de volumes. Alguns volumes são considerados de maior valor por nossos clientes, e, portanto, eles querem ter tais volumes segurados durante o transporte. Contratamos uma companhia de seguros para segurar volumes de valor.

13. Diferencie cada um dos diagramas abaixo, detalhando suas características.

- a) Diagrama de máquina de estados:

- b) Diagrama de atividades:

- c) Diagrama de sequência:

14. Identifique e explique cada um dos diagramas abaixo:

Diagrama 1:

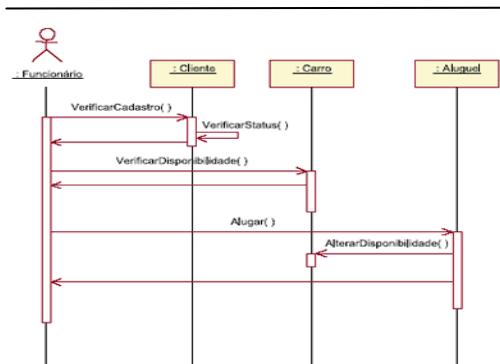
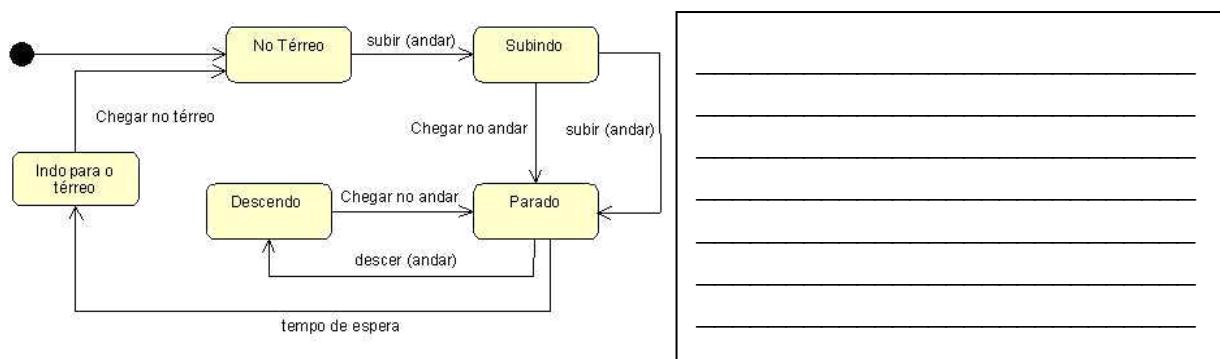


Diagrama 2:



6 Relacionamentos em UML

15. Explique cada um dos relacionamentos abaixo, demonstrando sua notação gráfica.

- a) Relacionamento de Dependência:

- b) Relacionamento de Associação:

- c) Relacionamento de Generalização:

16. Desenhe um diagrama de classes com relacionamentos, e cardinalidade para as seguintes situações.

- a) Uma pessoa pode ser casada com outra pessoa
- b) Uma disciplina é pré-requisitos para outra disciplina
- c) Uma peça pode ser composta de diversas outras peças

17. Construa um diagrama de atividade e sequência para a ação de Login em um sistema.

18. Construa um diagrama de caso de uso para a ação de emitir nota Fiscal em um sistema.

Fase II – Programação Orientada a Objetos

7 Visão Geral das Tecnologias Java

19. Explique com suas palavras porque o Java é uma das linguagens mais usadas no mundo atualmente.

20. O que é a API do Java? E como ela facilita a programação de softwares.

21. Porque o conceito da Maquina Virtual Java é tão importante para o desenvolvimento de aplicações.

22. Descreva como ocorre a compilação de um programa escrito em Java.

23. Conceitue:

JDK:

GarbageCollection:

IDE:

24. Porque não podemos utilizar as palavras reservadas do Java para nomear as variáveis do programa.

25. Diferencie as seguintes tecnologias Java

J2SE:

J2EE:

J2ME:

8 Classes Java

26. Escreva um programa para ler uma palavra e imprimir cada letra existente na palavra (uma em cada linha).
27. Crie uma classe Nome. Esta classe possui um objeto String inicializado na construção.
 - a. Crie um método tudoMinusculo, que coloque todos os caracteres da string em minúsculo (caixa baixa).
 - b. Crie um método que retorne o tamanho, em número de caracteres, da string armazenada.
 - c. Crie um método que conte o número de ocorrências de um determinado caractere recebido como parâmetro.
28. Crie uma classe para fazer o arredondamento dos seguintes valores:
5.2, 5.6 e -5.8 para o valor inteiro mais próximo.
29. Crie uma classe para calcular a raiz quadrada dos números 900 e 30.25.
30. Crie uma classe para calcular a potência de 5.5 elevado a 2 e 25 elevado a 0.5.
31. Crie uma classe que gere 5 cartões de loteria com seis números em cada um.

Fase III – Programação Orientada a Objetos

9 Orientação a Objetos

32. Explique com suas palavras o que é orientação a objetos.

33. Descreva de forma resumida como surgiu a programação orientada a objetos.

34. Descreva quais as diferenças existentes entre objeto, classe e instância.

35. Pense em um sistema de uma video locadora, quais classes deveriam ser criadas? E quais os atributos pertencentes a cada classe.

36. Qual a importancia da troca de mensagens entre os objetos.

37. Qual a diferença entre um atributo de objeto em atributo de classe.

38. O que são métodos na orientação a objetos. Dê pelo menos um exemplo de criação de método em java.

39. Qual a função do construtor de uma classe

40. Crie uma classe em java chamada Carro, e defina no mínimo 3 métodos e 3 atributos.

10 Encapsulamento

41. Porque é importante utilizar o conceito de encapsulamento na programação.

42. O que são modificadores de acesso, e quais as diferenças entre eles.

43. Qual a função dos métodos get e set.

44. Implemente o encapsulamento e os métodos get e set de uma classe chamada Aluno com no mínimo 2 atributos.

11 Herança

45. Enumere os benefícios de utilizarmos o conceito de herança no desenvolvimento de software.

46. Explique o que é herança múltipla.

47. Qual a relação de herança existente entre a classe object e a linguagem Java?

12 Polimorfismo

48. Sua tarefa é criar uma classe que contenha um Registro de Agenda. A tabela abaixo descreve as informações que um Registro de Agenda deve conter:

Atributos/Propriedades	Descrição
Nome	Nome da pessoa
Endereço	Endereço da pessoa
Número de Telefone	Número de telefone da pessoa
email	Endereço eletrônico da pessoa

Crie os seguintes métodos:

- Forneça todos os métodos acessores e modificadores necessários para todos os atributos
- Construtores

49. Crie uma classe Agenda que possa conter entradas de objetos tipo Registro de Agenda (utilize a classe criada no primeiro exercício). Devem ser oferecidos os seguintes métodos para agenda:

- Adicionar registro
- Excluir registro
- Visualizar registros
- Modificar registro

50. Neste exercício, queremos criar um registro mais especializado de Student que contém informações adicionais sobre um estudante de Informática. Sua tarefa é estender a classe StudentRecord que foi implementada nas lições anteriores e acrescentar atributos e métodos que são necessários para um registro de um estudante de Informática. Utilize override para modificar alguns métodos da superclasse StudentRecord, caso seja necessário.

51. Crie uma classe abstrata chamada Shape com os métodos getArea() e getName(). Escreva duas de suas subclasses Circle e Square. E acrescente métodos adicionais a estas subclasses.

Fonte: **Balagtas, Florence Tiu.** Projeto J.E.D.I. *Students Manual.s.l..*

Fase IV – Projeto Orientada a Objetos

13 Conhecendo e Organizando um Projeto no Padrão MVC

52. Com suas palavras escreva um resumo de no mínimo 10 linhas sobre o padrão MVC, qual a sua importância e como estruturar um projeto nesse padrão.

14 Implementação do Projeto

53. Implemente as seguintes mudanças no sistema:

- a. Modifique o método cadastrar para que seja possível cadastrar um novo produto na posição do vetor que tinha um produto, mas foi excluído.
- b. Codifique o tratamento de exceções do sistema.
- c. Escreva comentários no padrão Javadoc em todo o sistema, e em seguida gere a documentação (Para gerar automaticamente no netbeans, clique em Executar – Gerar Javadoc)
- d. Crie um manual do usuário para utilização do sistema.

15 Assuntos Complementares

15.1 Tratamento de Exceções

54. Liste quatro exemplos de exceções comuns.

55. Se nenhuma exceção é lançada em bloco try, onde o controle do programa procede quando o bloco try completa a execução?

56.Um aplicativo convencional deve capturar objetos Error? Explique.

57.Qual é o propósito de utilizar o bloco finally

58.Descreva a funcionalidade de cada uma das subclasses de RunTimeException estudadas.

59.Digite o código abaixo, trate a exceção (ArrayIndexOutOfBoundsException) com um bloco try/catch/finally.

```
import java.util.Arrays;
public class TesteTratandoEx {
    public static void main(String[] args) {
        int[] a = { 6, 7, 8, 9, 2, 3 };

        Arrays.sort(a, 0, 10);
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }

        System.out.println("O programa terminou");
    }
}
```


Hino Nacional

Ouviram do Ipiranga as margens plácidas
De um povo heróico o brado retumbante,
E o sol da liberdade, em raios fúlgidos,
Brilhou no céu da pátria nesse instante.

Se o penhor dessa igualdade
Conseguimos conquistar com braço forte,
Em teu seio, ó liberdade,
Desafia o nosso peito a própria morte!

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, um sonho intenso, um raio vívido
De amor e de esperança à terra desce,
Se em teu formoso céu, risonho e límpido,
A imagem do Cruzeiro resplandece.

Gigante pela própria natureza,
És belo, és forte, impávido colosso,
E o teu futuro espelha essa grandeza.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada, Brasil!

Deitado eternamente em berço esplêndido,
Ao som do mar e à luz do céu profundo,
Fulguras, ó Brasil, florão da América,
Iluminado ao sol do Novo Mundo!

Do que a terra, mais garrida,
Teus risonhos, lindos campos têm mais flores;
"Nossos bosques têm mais vida",
"Nossa vida" no teu seio "mais amores."

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, de amor eterno seja símbolo
O lábaro que ostentas estrelado,
E diga o verde-louro dessa flâmula
- "Paz no futuro e glória no passado."

Mas, se ergues da justiça a clava forte,
Verás que um filho teu não foge à luta,
Nem teme, quem te adora, a própria morte.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada, Brasil!

Hino do Estado do Ceará

Poesia de Thomaz Lopes
Música de Alberto Nepomuceno
Terra do sol, do amor, terra da luz!
Soa o clarim que tua glória conta!
Terra, o teu nome a fama aos céus remonta
Em clarão que seduz!
Nome que brilha esplêndido luzeiro!
Nos fulvos braços de ouro do cruzeiro!

Mudem-se em flor as pedras dos caminhos!
Chuvas de prata rolem das estrelas...
E despertando, deslumbrada, ao vê-las
Ressoa a voz dos ninhos...
Há de florar nas rosas e nos cravos
Rubros o sangue ardente dos escravos.
Seja teu verbo a voz do coração,
Verbo de paz e amor do Sul ao Norte!
Ruja teu peito em luta contra a morte,
Acordando a amplidão.
Peito que deu alívio a quem sofria
E foi o sol iluminando o dia!

Tua jangada afoita enfune o pano!
Vento feliz conduza a vela ousada!
Que importa que no seu barco seja um nada
Na vastidão do oceano,
Se à proa vão heróis e marinheiros
E vão no peito corações guerreiros?

Se, nós te amamos, em aventuras e mágoas!
Porque esse chão que embebe a água dos rios
Há de florar em meses, nos estios
E bosques, pelas águas!
Selvas e rios, serras e florestas
Brotam no solo em rumorosas festas!
Abra-se ao vento o teu pendão natal
Sobre as revoltas águas dos teus mares!
E desfraldado diga aos céus e aos mares
A vitória imortal!
Que foi de sangue, em guerras leais e francas,
E foi na paz da cor das hóstias brancas!



GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação