ROTEIRO DO CURSO DE PROGRAMAÇÃO EM JAVA

Estêvão Monteiro
Setembro de 2015

PRÉ-REQUISITOS

Lógica de programação
Algoritmos
Estruturas de dados

INTRODUÇÃO

Plataforma Java
- JVM
- JRE
- JDK
- bytecode
- compilador JIT

Eclipse IDE
- project explorer
- editor de código
    - autocompletar
    - detecção de erros de sintaxe
    - documentação
- console
- criação de projeto

SINTAXE JAVA

Variáveis
- instância x classe (estática)
- local
- parâmetro
- case-sensitive, convenção
- primitivas
    - byte: inteiro 8 bits (-128 a 127); default 0
    - short: inteiro 16 bits (-32.768 a 32.767); default 0
    - char: inteiro 16 bits positivo (0 a 65.535); default '0'
    - int: inteiro 32 bits (-2.147.483.648 a 2.147.483.647); default 0
    - long: inteiro 64 bits (-9,2 a 9,2 quintilhões: 10^{18}); default 0L
    - float: ponto flutuante 32 bits ($4,9x10^{-324}$ a $1,8x10^{308}$, com 6-7 dígitos significativos); default 0.0f
    - double: ponto flutuante 64 bits ($1,4x10^{-45}$ a $3,4x10^{38}$, com 14-15 dígitos significativos); default 0.0
    - boolean: true/false 1 bit; default false
- String
- caracteres: '\b', '\t', '\n', '\f', '\r', '\"', '\'', '\\',
- vetores
    - 0 a n-1

Operadores
= + - * / %
+= -= *= /= %=
++ -- !
== != > >= < <=
& && | || ?:
instanceof
()

Declaração: ;
Bloco: {}

Controle de fluxo
- if-then-else
- while
- do-while
- for
- switch-case

Convenções e boas práticas
- trabalho em equipe: clareza, manutenção, facilitar a aprendizagem do colega, evitar introdução de bugs
- formatação
    - capitalização
    - endentação
    - blocos
    - largura do código: 80 ou 100
    - ordem dos membros da classe
    - codificação de caracteres: UTF-8
    - documentação: comentários, JavaDoc

Exceções
- try-catch-finally
- throws
- checked vs. unchecked
- Throwable > Error | Exception > RuntimeException
- Separa código normal de tratamento de erros
- Propaga erros pela pilha de chamadas
- Agrupamento e diferenciação de erros
- sempre tratar exceções
- se abafar exceção, inserir comentário justificando
- não retornar códigos de erro

CLASSES E OBJETOS

Orientação a objetos
- estado: campos
- comportamento: métodos/operações
- vantagens
    - encapsulamento
    - modularidade/granularidade
    - reuso
    - substituição de componentes
    - depuração e testes

Classe
- tipo de objeto
- objeto é uma instância da classe
- pacote
- modificadores de acesso: public, private, (package)
- métodos
    - retorno: void, tipo primitivo ou classe
    - sobrecarga
    - sobrescrita
- construtores
- passagem de parâmetros: por valor
- imports
- herança

Objeto
- instanciação e referência
- acesso a campos e métodos
- casting: moldagem/fundição de referências
- coletor de lixo

Membros
- acessos package (default), public, private, protected
- palavra-chave this
- modificador static: refere-se à classe, não a um objeto
- modificador final: definição inalterável
- modificador abstract: deverá ser implementado em subclasse
- método main

Interface
- composição e delegação (padrão de projeto)

Enum
- constantes cujos valores são ignorados

Padrões de projeto
- programar para interfaces
- programar para abstrações se for necessária funcionalidade herdada
- evitar vícios procedurais no mundo orientado a objetos
- encapsular: manter sempre a mínima visibilidade possível aos membros da classe
- esconder detalhes de implementação
- granularidade de classes: classes são especialistas
- limitar subclasses
- preferir composição em vez de herança
- projetar e documentar para herança, senão proibí-la
- granularidade de métodos
- isolar código que pode mudar do que não deve mudar
- evitar classes sem métodos
- reduzir argumentos nos métodos
- considerar o padrão Factory em vez de construtores públicos
- considerar o padrão Builder se o construtor precisar de muitos parâmetros
- garantir Singleton ou classe não-instanciável através de construtores privados
- não invocar em construtor métodos que possam ser sobrescritos
- usar padrões de projeto comuns: Singleton, Factory, Bean, ValueObject, DTO, POJO....
- sobrecarga em vez de ramificações lógicas
- nomes auto-explicativos para variáveis e métodos

Factory
- Classe que instancia outra de acordo com argumentos recebidos
- Esconde qual implementação da abstração foi instanciada
- Polimorfismo
- Exemplos na API do Java

JavaBean
- serializável
- construtor sem argumentos
- métodos getters e setters
- nenhuma restrição de API específica
    - não estende classe
    - não implementa interface
    - não possui anotações, exceto aquelas que não modifiquem o comportamento da classe em si

POJO - plain old Java object
- nenhuma restrição de API específica
- costuma ser um bean

DTO - data transfer object
- padrão de projeto
- apenas carrega dados
- usado em comunicação entre interfaces remotas (cliente-servidor, aplicação-banco)
- não possui comportamento exceto o necessário para guardar e recuperar dados
- evita grande volume de parâmetros
- serializável
- costuma ser um bean

VO - value object
- entidade simples
- igualdade de valores, não de identidade
- imutável: não pode ser bean, POJO nem DTO

Projetando exceções
- anunciar explicitamente todas as exceções lançadas
- fazer suas próprias exceções
- implementar exceções com argumento de construtor do tipo String
- criar suas próprias exceções
- capturar exceções de baixo nível e encapsular em exceções de negócio

Boas práticas de implementação
- variáveis no menor escopo possível
- liberar objetos para o coletor de lixo
- não criar mais objetos que o necessário
- evitar lógica muito aninhada ou complexa
- membros estáticos referidos pela classe, não por variável
- evitar o método clone
- não reinventar a roda: preferir bibliotecas existentes, que já estão maduras

CLASSES UTILITÁRIAS DA PLATAFORMA JAVA

Object
- toString()
- equals()
    - chave de hashtable
    - reflexivo, simétrico, transitivo, consistente, nulos
- hashcode()
    - consistente na instancia e no equals

API Java
- Javadoc

Classes numéricas (wrappers/invólucros)
- valueOf(String) / parseInt(String)
- autoboxing/unboxing

String
- 16 bits
- imutável: String pool
- String.valueOf(Number)
- Number.toString()
- substring()
- split()
- trim()
- toLowerCase()
- toUpperCase()
- indexOf()

- lastIndexOf()
- contains()
- replaceFirst()
- replaceAll()
- endsWith()
- startsWith()
- compareTo()
- compareToIgnoreCase()
- equals()
- equalsIgnoreCase()
- StringBuilder

Math
- abs()
- ceil()
- floor()
- rint()
- round()
- min()
- max()
- pow()
- sqrt()
- random(): 0.0 a 1.0

Date
- Ano = ano - 1900
- Mês = 0 a 11
- Dia = 1 a 31
- Hora = 0 a 23
- Minuto = 0 a 59
- Definir campos: usar Calendar
    - Internacionalização (fuso horário)
    - Sincronização de threads
    - Ano referencial de 1900
- SimpleDateFormat
    - format(Date): String
    - parse(String): Date
    - y ano
    - Y ano da semana (primeira e última semanas do ano)
    - M mês
    - D dia do ano
    - d dia do mês
    - H hora (0-23)
    - h hora am/pm (0-11)
    - k hora (1-24)
    - K hora am/pm (1-12)
    - m minuto
    - s segundo
    - S milissegundo

I/O com fluxos (streams)
- java.io.*
- aplicável a:
    - sistema de arquivos
    - dispositivos
    - outros programas
    - vetores de memória
- tipos de dados suportados:
    - bytes
    - tipos primitivos

    - caracteres localizados
    - objetos
- input stream & output stream
- File
- FileInputStream, FileOutputStream
- sempre fechar streams
- caracteres: Reader & Writer
    - InputStreamReader > FileReader
    - OutputStreamWriter > FileWriter
- linhas: \r\n (retorno do cartucho, nova linha)
    - BufferedReader & PrintWriter
- BufferedReader & BufferedWriter
    - flush()
- PrintWriter
    - print(...)
    - println(...)
    - format(...): %d, %f, %s, %n

Collection e Map
- List: índice
    - ArrayList: ordem de inserção, iteração rápida, acesso aleatório rapido
    - LinkedList: ligação dupla para pilhas e filas, inserção e deleção posicionais rápidas
- Set: elementos únicos (equals())
    - HashSet: sem ordem, hashcode()
    - LinkedHashSet: como LinkedList, sem elementos repetidos
    - TreeSet: ordem natural (Comparable/Comparator.compareTo())
- Map: chave e valor
    - HashMap: sem ordem, hashcode()
    - LinkedHashMap: como LinkedHashSet
    - TreeMap: ordem natural (Comparable/Comparator.compareTo())
- Ordem interna (ordered) x ordem natural (sorted)
- Métodos de Collection
    - size()
    - isEmpty()
    - contains(Object)
    - add(Object)
    - remove(Object)
    - clear()
    - addAll(Collection)
    - removeAll(Collection)
    - retainAll(Collection)

RECURSOS AVANÇADOS

API JDBC - Java DataBase Connectivity
- java.sql, javax.sql
- DriverManager ou DataSource do JNDI
- Oracle JDBC driver
- Connection
- Statement > PreparedStatement > CallableStatement
    - execute: retorna múltiplos ResultSet
    - executeQuery: retorna só um ResultSet
    - executeUpdate: não retorna ResultSet (insert, delete, update)
- Fechar conexão
- SQLException
    - getMessage
    - getSQLState
    - getErrorCode

Genéricos
- Erros de execução
- Verificação de tipos mais forte
- Evita casts
- Algoritmos mais genéricos
- Tipos genéricos
    - Convenções
        - E: elemento de coleção
        - K: chave
        - N: número
        - T: tipo (qualquer classe)
        - V: valor
        - S, U, V...: sequências de tipos
- Métodos genéricos
    - Tipos genéricos no escopo do metodo
- Parâmetros de tipo vinculados
    - <T extends classe & interface & interface...>
- Herança de tipos genéricos
- Inferência de tipo em métodos genéricos: BoxDemo.<Integer>addBox(Integer.valueOf(10), listOfIntegerBoxes);
- Inferência de tipo na instanciação (diamante)

Anotações
- Acrescenta restrições de compilação para evitar erros
- Processamento durante compilação ou implantação
    - Gerar código
    - Gerar XML
- Processamento durante execução
- Múltiplas anotações de tipos diferentes
- Múltiplas anotações de mesmo tipo (JDK8)
- Aplicadas em declarações: classes, campos, métodos
- @Deprecated
    - Gera aviso na compilação
    - Usar também JavaDoc @deprecated
- @Override
- @SuppressWarnings({“deprecation”, “unchecked”})

Classes aninhadas
- Dependência íntima com a aninhadora
- Ex.: tratador de eventos em interface gráfica
- Tipos
    - Estática
        - Classe avulsa, sem qualquer privilégio especial com a aninhadora
        - Se não houver justificativa para outro tipo, usar este
    - Normal
        - Membro da instância aninhadora
        - Acesso aos membros da instância aninhadora
        - Aninhadora.this
    - Local
        - Dentro de bloco
        - Acesso aos membros da aninhadora
        - Visibilidade somente no método
        - Acesso somente às variáveis finais locais e da instância
    - Anônima
        - Definida durante sua própria instanciação, portanto instância única
        - Estende uma classe ou implementa uma única interface
        - Destinada para uso polimórfico
    - Expressão lambda
        - Permite um único método
        - Mais enxuta que a anônima

Processamento concorrente em threads (linhas)
- otimiza processamento de IO (arquivos, conexões de rede)
- não depende do processador ter múltiplos núcleos
- processo
    - ambiente de execução auto-contido
    - recursos próprios (caro)
    - aplicações podem ser vários processos cooperando com Inter Process Communication (IPC)
    - JVM roda como processo único, mas oferece ProcessBuilder
- thread
    - “processos leves”
    - requer menos recursos que um processo novo
    - existe dentro de um processo, todo processo tem pelo menos 1 thread
    - JVM tem várias threads de sistema e uma thread MAIN para a aplicação
    - a thread main pode criar mais threads
    - cria com a classe Thread ou um Executor
- Interface Runnable: método run()
    - Thread implementa Runnable, então outra opção é estendê-la
- Classe Thread
    - start(), sleep(long), stop()
    - InterruptedException
- Interrupção
    - métodos que lançam InterruptedException já tratam
    - Thread.interrupted(): boolean verifica e reseta o flag
    - método de instância isInterrupted() verifica sem resetar o flag
- Join
- Sincronização
    - Evitar interferências por entrelaçamento e inconsistência memória
    - modificador de método synchronized
        - todos os métodos synchronized da classe fazem fila
- Executor
    - (new Thread(r)).start(); é equivalente a e.execute(r)
- ExecutorService
    - submit(Runnable) e submit(Callable)
    - Callable dá retorno
    - pool de threads
- ScheduledExecutorService
    - schedule(Runnable) e schedule(Callable)
    - scheduleAtFixedRate(), scheduleWithFixedDelay()
- Thread pools
    - threads trabalhadoras, que são recicladas

Reflexão
- Recursos de extensibilidade: carregar classes do usuário em tempo de execução
- Visualização de classes, ambientes de desenvolvimento
- Depuração e testes
- Custo de processamento: impossível usar otimizações de máquina virtual
- Restrições de segurança (applets etc.)
- Pode quebrar encapsulamento e abstração
- objeto.getClass()
- java.lang.Class: imutável
- As classes do pacote java.lang.reflect não possuem construtores, exceto ReflectPermission
- .class
    - boolean.class vs. Boolean.TYPE
- Class.forName()
- classe.getSuperClass()

Testes com JUnit
- ...

BIBLIOTECAS

Java Standard Edition
- propósito geral
    - java.lang
    - java.io
    - java.nio
    - java.math
    - java.net
    - java.text
    - java.util
- propósito especial
    - java.applet
    - java.beans
    - java.awt (Abstract Window Toolkit, nativo)
    - java.rmi
    - java.security
    - java.sql
    - javax.rmi
    - javax.swing (widgets independentes de plataforma)
    - javax.swing.text.html.parser
    - javax.xml.bind.annotation
    - org.omg.CORBA
    - org.omg.PortableInterceptor

Java Enterprise Edition
- arquitetura para servidor de aplicação
    - JBoss
    - GlassFish
    - WebLogic
    - WebSphere
    - Apache Geronimo
- cliente/servidor
- web
- mapeamento objeto-relacional
- arquitetura distribuída
- arquitetura multi-camadas
- convenção > configuração
- configuração anotada (ou XML)
- pacotes
    - javax.servlet (HTTP, JSP)
    - javax.websocket
    - javax.faces (JSF)
    - javax.faces.component
    - javax.el (expressão de linguagem: JSP, JSF)
    - javax.enterprise.inject (injeção de contexto e dependência)
    - javax.enterprise.context
    - javax.ejb
    - javax.validation
    - javax.persistence (JPA, JPQL, Hibernate-HQL)
    - javax.transaction (JTA)
    - javax.security.auth.message (Java Authentication Service Provider Interface)
    - javax.enterprise.concurrent
    - javax.jms
    - javax.batch.api
    - javax.resource (Java EE Connector Architecture - JCA)

Projetos Apache
- Gerenciador de dependências Maven
- Bibliotecas Apache Commons

REFERÊNCIAS

http://docs.oracle.com/javase/tutorial/
http://docs.oracle.com/javase/8/docs/api/
http://www.oracle.com/technetwork/java/codeconvtoc-136057.html
http://google.github.io/styleguide/javaguide.html
http://commons.apache.org/
http://search.maven.org/
Joshua Bloch: Effective Java http://www.oracle.com/technetwork/java/effectivejava-136174.html
Sierra & Bates: Sun Certified Programmer & Developer for Java