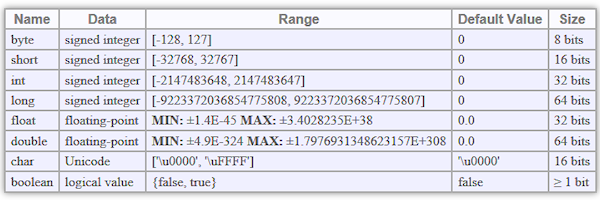
**EBAC – Especialista Back-end Java**

**Conceitos não triviais:**

Durante o curso alguns assuntos nada triviais e específicos do Java são passados de forma extremamente rápida e superficial. Para fixar bem esses conteúdos estou fazendo esse resumo/revisão. Não vou me concentrar em conceitos muito gerais da programação que sejam de entendimento básico, como lógica, mas no que for mais específico do Java e naquilo que de alguma forma eu não sentir confiança mesmo após as aulas e exercícios.

**Tipos Primitivos:**

Os tipos primitivos na linguagem Java são:



As variáveis de instância de tipo primitivo são inicializadas por padrão, as variáveis dos tipos byte, char, short, int, long, float e double são inicializadas como 0, e as variáveis do tipo boolean são inicializadas como false. Esses tipos podem especificar seu próprio valor inicial para uma variável do tipo primitivo atribuindo à variável um valor na sua declaração.

Variáveis que não são dos tipos primitivos são inicializadas com null.

**Wrappers:**

Wrappers são uma maneira de "envolver" um tipo primitivo ou um objeto em outro objeto. No Java, os tipos primitivos (como int, char, boolean, etc.) não possuem métodos, o que pode dificultar a programação e o uso dessas variáveis em Collections, por exemplo. A solução para isso é "envolver" esses tipos em um Wrapper, que efetivamente transforma esses tipos primitivos em objetos que têm propriedades e métodos.

Dessa forma, podemos usar o polimorfismo, utilizando essas variáveis em funções que só aceitam objetos e temos muito mais opções de processamento para elas. Os Wrappers fornecem métodos úteis, como conversões entre tipos e operações adicionais.

Porém, é necessário tomar cuidado com as comparações entre Wrappers, pois como são objetos, tratam-se de tipos de referência. Dois Wrappers do tipo Integer com o mesmo valor, se comparados com o operador ==, retornarão false, a menos que sejam a mesma instância. Para compará-los corretamente, deve-se usar o método equals.

Além disso, os Wrappers fornecem algumas funcionalidades adicionais, como a possibilidade de converter strings em tipos primitivos (e vice-versa) e a manipulação de valores nulos, o que não é possível com tipos primitivos.

**Quando utilizar os wrappers?**

**1. Uso em Collections:**  
- As Collections em Java só aceitam objetos, não tipos primitivos. Portanto, para armazenar tipos primitivos em uma Collection, é necessário usar Wrappers.

**2. Necessidade de métodos utilitários:**  
- Wrappers fornecem métodos utilitários para conversões e operações que não estão disponíveis para tipos primitivos.

**3. Trabalhando com valores nulos:**  
- Diferente dos tipos primitivos, os Wrappers podem ser usados para representar valores nulos. Isso pode ser útil em situações onde é necessário distinguir entre um valor não inicializado e um valor válido.

**Quando não utilizar wrappers?**

**1. Desempenho Crítico & Memória Limitada**  
- Wrappers introduzem uma sobrecarga de desempenho devido à autoboxing e unboxing, além do uso de memória adicional para armazenar objetos. Em operações intensivas de processamento ou em sistemas com requisitos de desempenho rigorosos, é preferível usar tipos primitivos.

**2. Uso em Arrays**  
- Arrays de tipos primitivos são mais eficientes em termos de desempenho e uso de memória em comparação com arrays de Wrappers.

**3. Evitar NullPointerException**  
- Wrappers podem ser nulos, o que pode levar a NullPointerException se não houver um tratamento adequado. Se o valor nulo não for uma necessidade, usar tipos primitivos pode evitar esses problemas.  
- Esse caso pode acontecer em: Atributos de Classe com Valor Padrão Conhecido, Operações Matemáticas e Lógicas, Estruturas de Controle e Valores de Configuração ou Constantes.

**4. Contexto de Variáveis Locais e Atribuições Temporárias**  
- Para variáveis locais e temporárias, que não precisam ser armazenadas em coleções ou passadas para métodos que exigem objetos, os tipos primitivos são preferíveis.

**Estruturas de dados:**

Um tópico importante antes de se estudar as Collections do Java é compreender quais são as estruturas de dados, como funcionam e quais são seus algoritmos. De uma forma bem resumida as estruturas de dados são:

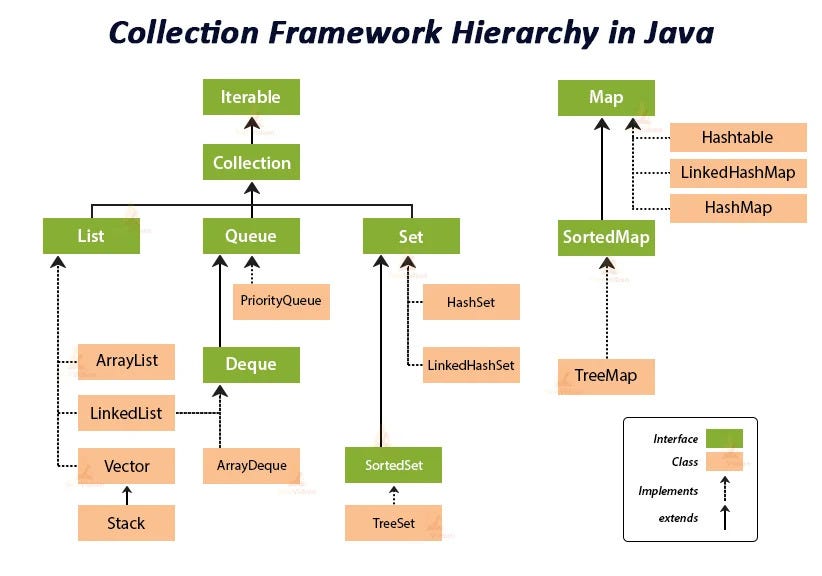
* Pilha
* Fila
* Listas
* Árvores
* Tabelas de Dispersão

Vamos resumir brevemente cada uma dessas para que possamos seguir para as Collections.

Pilha:

**Java Collection Framework:**

A Collections Framework é um conjunto de Interfaces, Implementações e Algoritmos que tem como objetivo representar, tratar e manusear estruturas de dados em Java.



Interfaces:

Antes de entrar no framework em si, precisamos definir o que são interfaces. Interfaces são contratos para a implementação de Classes. De uma forma bem resumida, se uma classe implementa uma interface ela necessariamente precisa implementar todos os métodos contidos naquela interface também.

Desse modo, todas as Classes que implementarem a interface terão os métodos descritos da interface. Isso torna extremamente útil o “programar para interfaces e não para implementações”, uma vez que independente de qual implementação você estiver utilizando, se estiver implementando a interface correta, a implementação irá funcionar.

As interfaces do Collection Framework:

Iterable:

Collection:

List:

Queue:

Deque:

Set:

Map:

Implementações:

ArrayList:

LinkedList:

PriorityQueue:

ArrayDeque:

HashSet:

LinkedHashSet:

TreeSet:

HashMap: