Principais APIs do Java 8

Para utilização nos testes de POO

José Creissac Campos

Departamento de Informática
Universidade do Minho

v.1.0

Conteúdo

1 Notas introdutórias

Este documento apresenta um resumo das APIs de algumas das principais interfaces e classes do Java 8. A listagem de variáveis construtores e métodos **não é exaustiva**. No caso das classes que implementam Interfaces, **os métodos das Interfaces não são explicitamente listados** (deverão ser consultados nas Interfaces respectivas).

Todas variáveis, construtores e métodos apresentados são `public` excepto indicação em contrário.

2 java.lang

2.1 Integer, Double, Float, etc.

`class java.lang.Integer`

Variáveis de classe
`static final int MAX_VALUE`
`static final int MIN_VALUE`
Construtores
`Integer(int value)`
`Integer(String s) throws NumberFormatException`
Métodos de classe
`static int compare(int x, int y)`
`static int parseInt(String s) throws NumberFormatException`
`static int parseInt(String s, int radix) throws NumberFormatException`
`static Integer valueOf(int i)`
`static Integer valueOf(String s) throws NumberFormatException`
`static Integer valueOf(String s, int radix) throws NumberFormatException`
Métodos de instância
`int compareTo(Integer anotherInteger)`
`int intValue()`
Comentários / Exemplos
Classes análogas existem para Double, Float, etc.

2.2 Math

`class java.lang.Math`

Variáveis de classe

```
static final double E //número de Euler (e)
static final double PI //π
```

Métodos de classe

```
static TipoNumérico abs(TipoNumérico a)
static double ceil(double a) //menor valor ≥ a
static double floor(double a) //maior valor ≤ a
static TipoNumérico max(TipoNumérico a, TipoNumérico b)
static TipoNumérico min(TipoNumérico a, TipoNumérico b)
static double pow(double a, double b) //a^b
static double random()
static double sqrt(double a) //√a
```

Comentários / Exemplos

TipoNumérico = double | float | int | long

2.3 String

`class java.lang.String`

Construtores

```
String()
String(String s)
```

Métodos de instância

```
char charAt(int index)
int compareTo(String anotherString)
int compareToIgnoreCase(String str)
String concat(String str)
boolean contains(CharSequence s)
boolean endsWith(String suffix)
boolean equals(Object anObject)
boolean equalsIgnoreCase(String anotherString)
int indexOf(String str)
int indexOf(String str, int fromIndex)
```

(continua...)

```
boolean isEmpty()
int lastIndexOf(String str)
int length()
boolean startsWith(String prefix)
String substring(int beginIndex, int endIndex)
String toLowerCase()
String toUpperCase()
String trim()
```

Comentários / Exemplos

Strings são imutáveis, podendo ser definidas utilizando aspas ("). Java suporta concatnação de Strings através do operador (+).

```
String s = "Olá" + "Mundo";
```

2.4 StringBuilder/StringBuffer

```
class java.lang.StringBuilder
class java.lang.StringBuffer
```

Construtores

```
StringBuilder()
StringBuilder(String str)
```

Métodos de instância

```
StringBuilder append(TipoPrimitivo b) // ex. append(int b)
StringBuilder append(Object str)
StringBuilder append(String str)
StringBuilder append(StringBuilder str)
StringBuffer delete(int start, int end)
int length()
StringBuffer replace(int start, int end, String str)
StringBuffer reverse()
String substring(int start, int end)
String toString()
```

Comentários / Exemplos

Construtores e métodos de instância análogos existem para StringBuffer. Ao contrário de Stringbuilder, StringBuffer é *thread-safe* (e, por isso, mais lenta).

3 java.util

3.1 Scanner

```
class java.util.Scanner
```

Construtores

```
Scanner(File f)
Scanner(InputStream is)
```

Métodos de instância

```
boolean hasNext()
boolean hasNextLine()
boolean hasNextTipoPrimitivo() // ex. boolean hasNextInt()
String next()
String nextLine()
TipoPrimitivo nextTipoPrimitivo() // ex. int nextInt()
void close()
```

Comentários / Exemplos

```
Scanner s = new Scanner(System.in); //para ler do teclado
```

3.2 Optional, OptionalInt, OptionalDouble e OptionalLong

```
class java.util.Optional<T>
```

Métodos de classe

```
static <T> Optional<T> empty()
static <T> Optional<T> of(T v) throws NullPointerException
static <T> Optional<T> ofNullable(T v) //v pode ser null
```

Métodos de instância

```
T get() throws NoSuchElementException
boolean isPresent()
T orElse(T other) //o valor do Optional ou other se null
T orElseGet(Supplier<? extends T> other)
<X extends Throwable> T
        orElseThrow(Supplier<? extends X> exceptionSupplier)
```

Comentários / Exemplos

Existem classes análogas para `int`, `double` e `long` (`OptionalInt`, `OptionalDouble` e `OptionalLong`). T é substituido pelo tipo apropriado.

3.3 Iterator

```
interface java.util.Iterator<E>
```

Métodos de instância

```
default void forEachRemaining(Consumer<? super E> action)
boolean hasNext()
E next()
default void remove()
```

4 java.io

4.1 ObjectOutputStream

```
class java.io.ObjectOutputStream
```

Construtores

```
ObjectOutputStream(OutputStream out)
```

Métodos de instância

```
void close()
void flush()
void writeObject(Object obj) throws IOException
```

Comentários / Exemplos

Existem métodos de escrita análogos a `writeObject(Object obj)` para os tipos primitivos: `writeInt(int val)`, `writeDouble(double val)`, etc.

```
//Abrir ficheiro para escrita em binário
ObjectOutputStream os =
            new ObjectOutputStream(new FileOutputStream(f));
os.writeObject(...);
os.flush();
os.close();
```

4.2 ObjectInputStream

```
class java.io.ObjectInputStream
```

Construtores

```
ObjectInputStream(InputStream out)
```

Métodos de instância

```
void close()
Object readObject(Object obj) throws ClassNotFoundException,
                                     IOException
```

Comentários / Exemplos

Existem métodos de leitura análogos a `readObject()` para os tipos primitivos: `readInt()`, `readDouble()`, etc.

```
//Abrir ficheiro para leitura em binário
ObjectInputStream is =
            new ObjectInputStream(new FileInputStream(f));
```

5 Collection

```
interface java.util.Collection<E> extends java.lang.Iterable<E>
```

Métodos de instância

```
boolean add(E e)
boolean addAll(Collection<? extends E> c)
void clear()
boolean contains(Object o)
boolean containsAll(Collection<?> c)
boolean equals(Object o)
default void forEach(Consumer<? super E> action)
boolean isEmpty()
Iterator<E> iterator()
boolean remove(Object o)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
default boolean removeIf(Predicate<? super E> filter)
int size()
default Stream<E> stream()
```

(continua...)

```
Object[] toArray()
```

Comentários / Exemplos

```
// Estes exemplos são válidos para todas as Collection
// Considere a seguinte variável de instância de uma classe
Collection<Aluno> alunos;
// Remover alunos com menos que certa nota (iterador interno)
public void removerPorNota(int nota) {
  lstAlunos.removeIf(a->a.getNota()<nota);
}
// Remover alunos com menos que certa nota (iterador externo)
public void removerPorNota(int nota) {
  Iterator<Aluno> it = lstAlunos.iterator();
  while(it.hasNext())
     if(it.next().getNota()<nota)
         it.remove();
}
```

5.1 Set

```
interface java.util.Set<E> extends java.util.Collection<E>
```

Métodos de instância

```
boolean add(E e)
boolean addAll(Collection<? extends E> c)
void clear()
boolean contains(Object o)
boolean containsAll(Collection<?> c)
boolean equals(Object o)
default void forEach(Consumer<? super E> action)
boolean isEmpty()
Iterator<E> iterator()
boolean remove(Object o)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
default boolean removeIf(Predicate<? super E> filter)
int size()
```

(continua...)

```
default Stream<E> stream()
Object[] toArray()
```

Comentários / Exemplos

A mesma API que Collection.

5.1.1 HashSet

```
class HashSet<E> implements Set<E>
```

Construtores

```
HashSet()
HashSet(Collection<? extends E> c)
HashSet(int initialCapacity)
```

Métodos de instância (+ todos os métodos em `java.util.Set<E>`)

```
String toString()
```

5.1.2 TreeSet

```
class TreeSet<E> implements Set<E>
```

Construtores

```
TreeSet()
TreeSet(Collection<? extends E> c)
TreeSet(Comparator<? super E> comparator)
```

Métodos de instância (+ todos os métodos em `java.util.Set<E>`)

```
Comparator<? super E> comparator()
E first()
SortedSet<E> headSet(E toElement)
E last()
SortedSet<E> subSet(E fromElement, E toElement)
SortedSet<E> tailSet(E fromElement)
String toString()
```

5.2 List

```
interface java.util.List<E> extends java.util.Collection<E>
```

Métodos de instância

```
boolean add(E e)
boolean add(int index, E element)
boolean addAll(Collection<? extends E> c)
boolean addAll(int index, Collection<? extends E> c)
void clear()
boolean contains(Object o)
boolean containsAll(Collection<?> c)
boolean equals(Object o)
default void forEach(Consumer<? super E> action)
E get(int index)
int indexOf(Object o)
boolean isEmpty()
Iterator<E> iterator()
int lastIndexOf(Object o)
E remove(int index)
boolean remove(Object o)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
default boolean removeIf(Predicate<? super E> filter)
default void replaceAll(UnaryOperator<E> operator)
E set(int index, E element)
int size()
default void sort(Comparator<? super E> c)
default Stream<E> stream()
List<E> subList(int fromIndex, int toIndex)
Object[] toArray()
```

5.2.1 ArrayList

```
class ArrayList<E> implements List<E>
```

Construtores

```
ArrayList()
ArrayList(Collection<? extends E> c)
ArrayList(int initialCapacity)
```

Métodos de instância (+ todos os métodos em `java.util.List<E>`)

```
protected void removeRange(int fromIndex, int toIndex)
void ensureCapacity(int minCapacity)
protected void removeRange(int fromIndex, int toIndex)
String toString()
void trimToSize()
```

6 Map

```
interface java.util.Map<K,V>
```

Métodos de instância

```
void clear()
boolean containsKey(Object key)
boolean containsValue(Object value)
Set<Map.Entry<K,V>> entrySet()
boolean equals(Object o)
default void forEach(BiConsumer<? super K,? super V> action)
V get(Object key)
default V getOrDefault(Object key, V defaultValue)
int hashCode()
boolean isEmpty()
Set<K> keySet()
default V merge(K key, V value,
       BiFunction<? super V,? super V,? extends V> remapFun)
V put(K key, V value)
void putAll(Map<? extends K,? extends V> m)
default V putIfAbsent(K key, V value)
V remove(Object key)
```

(continua...)

```
default boolean remove(Object key, Object value)
default V replace(K key, V value)
default boolean replace(K key, V oldValue, V newValue)
default void replaceAll(
            BiFunction<? super K,? super V,? extends V> fun)
int size()
Collection<V> values()
```

`interface Map.Entry<K,V>`

Métodos de instância
`boolean equals(Object o)`
`K getKey()`
`V getValue()`
`V setValue(V value)`

6.1 HashMap

`class java.util.HashMap<K,V>`

Construtores
`HashMap()`
`HashMap(int initialCapacity)`
`HashMap(Map<? extends K,? extends V> m)`
Métodos de instância
(todos os métodos em `java.util.Map<K,V>`)

6.2 TreeMap

`class java.util.HashMap<K,V>`

Construtores
`TreeMap()`
`TreeMap(Comparator<? super K> comparator)`
`TreeMap(Map<? extends K,? extends V> m)`
Métodos de instância (+ todos os métodos em `java.util.Map<K,V>`)
`Comparator<? super K> comparator()`
`Map.Entry<K,V> firstEntry()`
(continua...)

```
K firstKey()
Map.Entry<K,V> higherEntry(K key)
K higherKey(K key)
Map.Entry<K,V> lastEntry()
K lastKey()
Map.Entry<K,V> lowerEntry(K key)
K lowerKey(K key)
SortedMap<K,V> subMap(K fromKey, K toKey)
SortedMap<K,V> tailMap(K fromKey)
```

7 java.util.stream

interface java.util.stream.Stream<T>

Métodos de instância

```
boolean allMatch(Predicate<? super T> predicate)
boolean anyMatch(Predicate<? super T> predicate)
<R,A> R collect(Collector<? super T,A,R> collector)
long count()
Stream<T> distinct()
Stream<T> filter(Predicate<? super T> predicate)
Optional<T> findAny()
Optional<T> findFirst()
void forEach(Consumer<? super T> action)
Stream<T> limit(long maxSize)
<R> Stream<R> map(Function<? super T,? extends R> mapper)
DoubleStream mapToDouble(ToDoubleFunction<? super T> mapper)
IntStream mapToInt(ToIntFunction<? super T> mapper)
LongStream mapToLong(ToLongFunction<? super T> mapper)
Optional<T> max(Comparator<? super T> comparator)
Optional<T> min(Comparator<? super T> comparator)
boolean noneMatch(Predicate<? super T> predicate)
T reduce(T identity, BinaryOperator<T> accumulator)
Stream<T> sorted()
Stream<T> sorted(Comparator<? super T> comparator)
```

(continua...)

Comentários / Exemplos

```
List<Aluno> lstAlunos;
// Quantos alunos passam?
public long aguaBenta(int bonus) {
  return lstAlunos.stream().filter(Aluno::passa).count();
}
// Média da turma
public double media() {
  return stream().mapToDouble(Aluno::getNota)
                 .average().orElse(0);
}
```

7.1 IntStream, LongStream, DoubleStream

`interface java.util.stream.IntStream`

Métodos de instância

```
OptionalDouble average()
long count()
OptionalInt max()
OptionalInt min()
int sum()
```

(todos os métodos de `Stream` adaptados para trabalhar com `int`)

Comentários / Exemplos

Interfaces análogas existem para long e double.

7.2 Collectors

A classe `Collectors` fornece um conjunto de implementações de `Collector`

`class java.util.stream.Collectors`

Métodos de classe

```
static <T> Collector<T,?,Long> counting()
static <T,K> Collector<T,?,Map<K,List<T>>>
       groupingBy(Function<? super T,? extends K> classifier)
static <T,K,D,A,M extends Map<K,D>> Collector<T,?,M>
       groupingBy(Function<? super T,? extends K> classifier,
                  Supplier<M> mapFactory,
                  Collector<? super T,A,D> downstream)
static <T,U,A,R> Collector<T,?,R>
          mapping(Function<? super T,? extends U> mapper,
                  Collector<? super U,A,R> downstream)
static <T,C extends Collection<T>> Collector<T,?,C>
                  toCollection(Supplier<C> collectionFactory)
static <T> Collector<T,?,List<T>> toList()
static <T,K,U> Collector<T,?,Map<K,U>>
          toMap(Function<? super T,? extends K> keyMapper,
                Function<? super T,? extends U> valueMapper)
static <T,K,U,M extends Map<K,U>> Collector<T,?,M>
          toMap(Function<? super T,? extends K> keyMapper,
                Function<? super T,? extends U> valueMapper,
                BinaryOperator<U> mergeFunction,
                Supplier<M> mapSupplier)
static <T> Collector<T,?,Set<T>> toSet()
```

Comentários / Exemplos

```
Map<String, Aluno> alunos
// Lista dos alunos que passam
public List<Aluno> passam() {
  return alunos.values().stream().filter(Aluno::passa)
          .map(Aluno::clone).collect(Collectors.toList());
}
```

(continua...)

```
// Calcular Map de nota para lista de cópias dos alunos
public Map<Double, List<Aluno>> nomesPorNota() {
  return alunos.values().stream().map(Aluno::clone)
          .collect(Collectors.groupingBy(Aluno::getNota));
}
// Calcular TreeMap de nota para Set de nomes dos alunos
// Assume-se import static de Collectors.groupingBy,
// Collectors.mapping e Collectors.toSet.
public TreeMap<Double, Set<String>> nomesPorNota() {
  return alunos.values().stream()
          .collect(groupingBy(Aluno::getNota, TreeMap::new,
                              mapping(Aluno::getNome, toSet())));
}
```

8 Interfaces funcionais

Uma interface funcional representa um método ou expressão lambda.

Consumer<**T**> aceita um valor (de tipo T) e não retorna nada. Ao contrário da marioria das interfaces funcionais, `Consumer` actua via efeitos laterais.

Predicate<**T**> aceita um T e retorna um booleano.

UnaryOperator<**T**> aceita um T e retorna um T.

BiConsumer<**T, U**> aceita dois valores e não retorna nada.

BiFunction<**T, U, R**> aceita dois valores (T e U) e retorna um resultado (R).

Supplier<**T**> devolve um resultado (T).