Interface Collection<E>

- **Type Parameters:**
  `E - the type of elements in this collection`

All Methods

Modifier and Type	Method and Description
`boolean`	`add(E e)` Ensures that this collection contains the specified element (optional operation).
`boolean`	`addAll(Collection<? extends E> c)` Adds all of the elements in the specified collection to this collection (optional operation).
`void`	`clear()` Removes all of the elements from this collection (optional operation).
`boolean`	`contains(Object o)` Returns `true` if this collection contains the specified element.
`boolean`	`containsAll(Collection<?> c)` Returns `true` if this collection contains all of the elements in the specified collection.
`boolean`	`equals(Object o)` Compares the specified object with this collection for equality.
`int`	`hashCode()` Returns the hash code value for this collection.
`boolean`	`isEmpty()` Returns `true` if this collection contains no elements.
`Iterator<E>`	`iterator()` Returns an iterator over the elements in this collection.
`default Stream<E>`	`parallelStream()` Returns a possibly parallel `Stream` with this collection as its source.
`boolean`	`remove(Object o)` Removes a single instance of the specified element from this collection, if it is present (optional operation).
`boolean`	`removeAll(Collection<?> c)` Removes all of this collection's elements that are also contained in the specified collection (optional operation).
`default boolean`	`removeIf(Predicate<? super E> filter)` Removes all of the elements of this collection that satisfy the given predicate.
`boolean`	`retainAll(Collection<?> c)` Retains only the elements in this collection that are contained in the specified collection (optional operation).
`int`	`size()` Returns the number of elements in this collection.
`default Spliterator<E>`	`spliterator()` Creates a `Spliterator` over the elements in this collection.
`default Stream<E>`	`stream()` Returns a sequential `Stream` with this collection as its source.
`Object[]`	`toArray()` Returns an array containing all of the elements in this collection.
`<T> T[]`	`toArray(T[] a)` Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

- ### Methods inherited from interface java.lang.Iterable

  `forEach`

Interface List<E>

- **Type Parameters:**
  `E - the type of elements in this list`

All Methods

Modifier and Type	Method and Description
`boolean`	**add**`(`**E**` e)` Appends the specified element to the end of this list (optional operation).
`void`	**add**`(int index, `**E**` element)` Inserts the specified element at the specified position in this list (optional operation).
`boolean`	**addAll**`(`**Collection**`<? extends `**E**`> c)` Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
`boolean`	**addAll**`(int index, `**Collection**`<? extends `**E**`> c)` Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
`void`	**clear**`()` Removes all of the elements from this list (optional operation).
`boolean`	**contains**`(`**Object**` o)` Returns `true` if this list contains the specified element.
`boolean`	**containsAll**`(`**Collection**`<?> c)` Returns `true` if this list contains all of the elements of the specified collection.
`boolean`	**equals**`(`**Object**` o)` Compares the specified object with this list for equality.
E	**get**`(int index)` Returns the element at the specified position in this list.
`int`	**hashCode**`()` Returns the hash code value for this list.
`int`	**indexOf**`(`**Object**` o)` Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
`boolean`	**isEmpty**`()` Returns `true` if this list contains no elements.
Iterator<**E**>	**iterator**`()` Returns an iterator over the elements in this list in proper sequence.
`int`	**lastIndexOf**`(`**Object**` o)` Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<**E**>	**listIterator**`()` Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<**E**>	**listIterator**`(int index)` Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
E	**remove**`(int index)`

	Removes the element at the specified position in this list (optional operation).
`boolean`	**remove**(Object o) Removes the first occurrence of the specified element from this list, if it is present (optional operation).
`boolean`	**removeAll**(Collection<?> c) Removes from this list all of its elements that are contained in the specified collection (optional operation).
`default void`	**replaceAll**(UnaryOperator<E> operator) Replaces each element of this list with the result of applying the operator to that element.
`boolean`	**retainAll**(Collection<?> c) Retains only the elements in this list that are contained in the specified collection (optional operation).
E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).
`int`	**size**() Returns the number of elements in this list.
`default void`	**sort**(Comparator<? super E> c) Sorts this list according to the order induced by the specified `Comparator`.
`default` Spliterator<E>	**spliterator**() Creates a `Spliterator` over the elements in this list.
List<E>	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element).

Interface Map<K,V>

- **Type Parameters:**
  `K` - the type of keys maintained by this map

  `V` - the type of mapped values

All Methods

Modifier and Type	Method and Description
`void`	`clear()` Removes all of the mappings from this map (optional operation).
`default V`	`compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)` Attempts to compute a mapping for the specified key and its current mapped value (or `null` if there is no current mapping).
`default V`	`computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)` If the specified key is not already associated with a value (or is mapped to `null`), attempts to compute its value using the given mapping function and enters it into this map unless `null`.
`default V`	`computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)` If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
`boolean`	`containsKey(Object key)` Returns `true` if this map contains a mapping for the specified key.
`boolean`	`containsValue(Object value)` Returns `true` if this map maps one or more keys to the specified value.
`Set<Map.Entry<K,V>>`	`entrySet()` Returns a `Set` view of the mappings contained in this map.
`boolean`	`equals(Object o)` Compares the specified object with this map for equality.
`default void`	`forEach(BiConsumer<? super K,? super V> action)` Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
`V`	`get(Object key)` Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.
`default V`	`getOrDefault(Object key, V defaultValue)` Returns the value to which the specified key is mapped, or `defaultValue` if this map contains no mapping for the key.
`int`	`hashCode()` Returns the hash code value for this map.
`boolean`	`isEmpty()` Returns `true` if this map contains no key-value mappings.
`Set<K>`	`keySet()` Returns a `Set` view of the keys contained in this map.
`default V`	`merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)`

	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V	put(K key, V value) Associates the specified value with the specified key in this map (optional operation).
void	putAll(Map<? extends K,? extends V> m) Copies all of the mappings from the specified map to this map (optional operation).
default V	putIfAbsent(K key, V value) If the specified key is not already associated with a value (or is mapped to `null`) associates it with the given value and returns `null`, else returns the current value.
V	remove(Object key) Removes the mapping for a key from this map if it is present (optional operation).
default boolean	remove(Object key, Object value) Removes the entry for the specified key only if it is currently mapped to the specified value.
default V	replace(K key, V value) Replaces the entry for the specified key only if it is currently mapped to some value.
default boolean	replace(K key, V oldValue, V newValue) Replaces the entry for the specified key only if currently mapped to the specified value.
default void	replaceAll(BiFunction<? super K,? super V,? extends V> function) Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
int	size() Returns the number of key-value mappings in this map.
Collection<V>	values() Returns a `Collection` view of the values contained in this map.

Class HashSet<E>

- **Type Parameters:**
  `E - the type of elements maintained by this set`

Constructors

Constructor and Description
HashSet`()` Constructs a new, empty set; the backing `HashMap` instance has default initial capacity (16) and load factor (0.75).
HashSet`(`Collection`<? extends `E`> c)` Constructs a new set containing the elements in the specified collection.
HashSet`(int initialCapacity)` Constructs a new, empty set; the backing `HashMap` instance has the specified initial capacity and default load factor (0.75).
HashSet`(int initialCapacity, float loadFactor)` Constructs a new, empty set; the backing `HashMap` instance has the specified initial capacity and the specified load factor.

All Methods

Modifier and Type	Method and Description
`boolean`	add`(`E` e)` Adds the specified element to this set if it is not already present.
`void`	clear`()` Removes all of the elements from this set.
Object	clone`()` Returns a shallow copy of this `HashSet` instance: the elements themselves are not cloned.
`boolean`	contains`(`Object` o)` Returns `true` if this set contains the specified element.
`boolean`	isEmpty`()` Returns `true` if this set contains no elements.
Iterator<E>	iterator`()` Returns an iterator over the elements in this set.
`boolean`	remove`(`Object` o)` Removes the specified element from this set if it is present.
`int`	size`()` Returns the number of elements in this set (its cardinality).
Spliterator<E>	spliterator`()` Creates a ***late-binding*** and *fail-fast* **`Spliterator`** over the elements in this set.

Class Date

Constructors

Constructor and Description
`Date()` Allocates a `Date` object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.
`Date(int year, int month, int date)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(year + 1900, month, date)` or `GregorianCalendar(year + 1900, month, date)`.
`Date(int year, int month, int date, int hrs, int min)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(year + 1900, month, date, hrs, min)` or `GregorianCalendar(year + 1900, month, date, hrs, min)`.
`Date(int year, int month, int date, int hrs, int min, int sec)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(year + 1900, month, date, hrs, min, sec)` or `GregorianCalendar(year + 1900, month, date, hrs, min, sec)`.
`Date(long date)` Allocates a `Date` object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.

All Methods

Modifier and Type	Method and Description
`boolean`	`after(Date when)` Tests if this date is after the specified date.
`boolean`	`before(Date when)` Tests if this date is before the specified date.
`Object`	`clone()` Return a copy of this object.
`int`	`compareTo(Date anotherDate)` Compares two Dates for ordering.
`boolean`	`equals(Object obj)` Compares two dates for equality.
`static Date`	`from(Instant instant)` Obtains an instance of `Date` from an `Instant` object.
`int`	`getDate()` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.get(Calendar.DAY_OF_MONTH)`.
`int`	`getDay()` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.get(Calendar.DAY_OF_WEEK)`.
`int`	`getHours()` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.get(Calendar.HOUR_OF_DAY)`.
`int`	`getMinutes()` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.get(Calendar.MINUTE)`.

`int`	**getMonth**`()` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.get(Calendar.MONTH).`
`int`	**getSeconds**`()` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.get(Calendar.SECOND).`
`long`	**getTime**`()` Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this `Date` object.
`int`	**getYear**`()` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.get(Calendar.YEAR) - 1900.`
`int`	**hashCode**`()` Returns a hash code value for this object.
`static long`	**parse**`(`**String**` s)` **Deprecated.** As of JDK version 1.1, replaced by `DateFormat.parse(String s).`
`void`	**setDate**`(int date)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(Calendar.DAY_OF_MONTH, int date).`
`void`	**setHours**`(int hours)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(Calendar.HOUR_OF_DAY, int hours).`
`void`	**setMinutes**`(int minutes)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(Calendar.MINUTE, int minutes).`
`void`	**setMonth**`(int month)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(Calendar.MONTH, int month).`
`void`	**setSeconds**`(int seconds)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(Calendar.SECOND, int seconds).`
`void`	**setTime**`(long time)` Sets this `Date` object to represent a point in time that is `time` milliseconds after January 1, 1970 00:00:00 GMT.
`void`	**setYear**`(int year)` **Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(Calendar.YEAR, year + 1900).`
Instant	**toInstant**`()` Converts this `Date` object to an `Instant`.
String	**toString**`()` Converts this `Date` object to a `String` of the form:

Class Time

Constructors

Constructor and Description
`Time(int hour, int minute, int second)` **Deprecated.** Use the constructor that takes a milliseconds value in place of this constructor
`Time(long time)` Constructs a `Time` object using a milliseconds time value.

All Methods

Modifier and Type	Method and Description
`int`	getDate`()` **Deprecated.**
`int`	getDay`()` **Deprecated.**
`int`	getMonth`()` **Deprecated.**
`int`	getYear`()` **Deprecated.**
`void`	setDate`(int i)` **Deprecated.**
`void`	setMonth`(int i)` **Deprecated.**
`void`	setTime`(long time)` Sets a `Time` object using a milliseconds time value.
`void`	setYear`(int i)` **Deprecated.**
Instant	toInstant`()` This method always throws an UnsupportedOperationException and should not be used because SQL `Time` values do not have a date component.
LocalTime	toLocalTime`()` Converts this `Time` object to a `LocalTime`.
String	toString`()` Formats a time in JDBC time escape format.
`static` Time	valueOf`(`LocalTime` time)` Obtains an instance of `Time` from a **`LocalTime`** object with the same hour, minute and second time value as the given `LocalTime`.
`static` Time	valueOf`(`String` s)` Converts a string in JDBC time escape format to a `Time` value.