`List` and `ArrayList`

List

```
import java.util.List;

import java.util.ArrayList;
```

Return	Method	Description
`boolean`	`add(T item)`	append the **`item`** to the end of the list
`boolean`	`addAll(Collection<? extends E c)`	appends all of the elements in the specified collection **`c`** to the end of the list, in the order that they are returned by the specified collection's iterator (*optional operation*)
`boolean`	`contains(T item)`	returns **`true`** if an item is in the list
`T`	`get(int index)`	retrieve the item at the specified index without removing the item (*the first element has an index of 0*)
`T`	`remove(int index)`	retrieve the item at the specified index ***and*** remove the item (*the first element has an index of 0*)
`boolean`	`remove(Object item)`	removes the passed **`item`** from the list
`int`	`size()`	return the number of elements in the list

Note

- List is an interface and cannot be instantiated.
- You can create a new empty **`ArrayList`** with **`new ArrayList<T>()`**.

Stream

Data Source

Modifier & Return	Method	Description
`static <T> Stream<T>`	`of(T t)`	returns a sequential stream containing a single element **`t`**
`static <T> Stream<T>`	`of(T... values)`	returns a sequential *ordered* stream whose elements are the specified **`values`**
`static <T> Stream<T>`	`generate(Supplier<? extends T> s)`	returns an infinite sequential unordered stream where each element is generated by the provided **`Supplier`**

`static <T> Stream<T>`	`iterate(`
	`  T seed,`
	`  UnaryOperator<T> f`
	`)`
		returns an infinite sequential unordered stream produced by iterative application of a function **`f`** to an initial element **`seed`**, producing a stream consisting of **`seed, f(seed), f(f(seed)),`** *etc*

`static <T> Stream<T>`	`iterate(`
	`  T seed,`
	`  Predicate<? super T> hasNext,`
	`  UnaryOperator<T> next`
	`)`
		returns an infinite sequential unordered stream produced by iterative application of a function **`next`** to an initial element **`seed`**, conditioned on satisfying the given **`hasNext`** predicate

Stream

Intermediate Operations

Modifier & Return	Method	Description

<R> Stream<R>	map(
	  Function<
	  ? super T,
	  ? extends R
	  > fn
	)
		returns a stream consisting of the results of *applying* the given function **fn** to the elements of this stream

<R> Stream<R>	flatMap(
	  Function<
	  ? super T,
	  ? extends Stream<? extends R>
	  > fn
	)
		returns a stream consisting of the results of *replacing* each element of this stream with the contents of a mapped stream produced by applying the provided mapping **fn** to each elements

Stream<T>	filter(Predicate<? super T> pr)	returns a stream consisting of the elements of this stream, that match the given predicate **pr**
Stream<T>	limit(long maxSize)	returns a stream consisting of the elements of this stream, truncated to be no longer than **maxSize** in length
Stream<T>	takeWhile(Predicate<? super T> pr)	returns, *if this stream is ordered*, a stream consisting of the longest prefix of elements taken from this stream that match the given predicate **pr**
Stream<T>	takeWhile(Predicate<? super T> pr)	returns, *if this stream is ordered*, a stream consisting of the remaining elements of this stream after dropping the longest prefix that match the given predicate **pr**

Stream

Terminal Operations

Modifier & Return	Method	Description
void	forEach(Consumer<? super T> action)	performs an **action** for each element of this stream
boolean	allMatch(Predicate<? super T> pr)	returns whether *all* elements of this stream match the provided predicate **pr**
boolean	anyMatch(Predicate<? super T> pr)	returns whether *any* elements of this stream match the provided predicate **pr**
boolean	noneMatch(Predicate<? super T> pr)	returns whether *no* elements of this stream match the provided predicate **pr**
long	count()	returns the count of elements in this stream
<R,A> R	collect(Collector<? super T, A, R> coll)	performs a mutable reduction operation on the elements of this stream using a **Collector coll**

T	reduce(
	  T identity,
	  BinaryOperator<T> acc
	)
		performs a reduction on the elements of this stream, using the provided **identity** value and an associative accumulation function **acc**, and returns the reduced value

<U> U	reduce(
	  T identity,
	  BiFunction<U, ? super T U> acc,
	  BinaryOperator<U> combiner
	)
		performs a reduction on the elements of this stream, using the provided **identity** value and an associative accumulation function **acc**, and combining functions **combiner**

Note

- **Collectors.toList()** will be useful and has been demonstrated in Lecture.
- To use **Collectors**, you need to import **java.util.stream.Collectors;**.