1 Maybe

```
static <T> Maybe<T> of(T val)
```

Creates a Maybe<T> with the given content val if val is not null.
Otherwise, returns the shared instance of None<?>.

```
static <T> Maybe<T> some(T val)
```

Creates a Maybe<T> with the given content val which may be null.

```
static <T> Maybe<T> none()
```

Creates a Maybe<T> without any content, this is guaranteed to return the shared instance of None<?>.

```
String toString()
```

Returns the string representation of Maybe<T>.

```
boolean equals(Object obj)
```

Maybe<T>: Returns true if the content is equal to the content of obj. Otherwise returns false.
None<T>: Returns true if obj is also None<T>. Otherwise returns false.

```
<U> Maybe<U> map
(Transformer<? super T, ? extends U> fn)
```

Maybe<T>: Create a new instance of Maybe<T> by applying the transformer fn to the content and wrapping
it in Maybe<T>. It will never return None<T> to allow for our InfiniteList<T> to contain null.
None<T>: Returns None<T>.

```
Maybe<T> filter
(BooleanCondition<? super T> pred)
```

Maybe<T>: If the content is not null and pred.test(content) returns true, we return the current instance.
Otherwise, returns None<T>.
None<T>: Returns None<T>.

```
<U> Maybe<U> flatmap
(Transformer<? super T, ? extends Maybe<? extends U>> fn)
```

Maybe<T>: Create a new instance of Maybe<T> by applying the transformer fn to the content without
wrapping it in Maybe<T> as fn already returns Maybe<U>.
None<T>: Returns None<T>.

```
T orElse(Producer<? extends T> prod)
```

Maybe<T>: Returns the content (even if it is null).
None<T>: Returns the value produced by the producer prod.

```
void ifPresent(Consumer<? super T> cons)
```

Maybe<T>: Pass the content to the consumer cons.
None<T>: Do nothing.

2 Lazy

```
static <T> Lazy<T> of(T val)
```

Creates a Lazy<T> with the given content val already evaluated.

```
static <T> Lazy<T> of(Producer<? extends T> prod)
```

Creates a Lazy<T> with the content not yet evaluated and will be evaluated using the given producer.

```
boolean equals(Object obj)
```

Returns true if the content is equal to the content of obj. Otherwise returns false.
This forces evaluation of the content.

```
<U> Lazy<U> map
(Transformer<? super T, ? extends U> fn)
```

Lazily maps the content using the given transformer.

```
Lazy<Boolean> filter
(BooleanCondition<? super T> pred)
```

Lazily test if the value passes the test or not and returns a Lazy<Boolean> to indicate the result.

```
<U> Lazy<U> flatMap
(Transformer<? super T, ? extends Lazy<? extends U>> fn)
```

Lazily creates a new instance of Lazy<T> by applying the transformer fn to the content without
wrapping it in another Lazy<..>.

```
<U, V> Lazy<V> combine
(Lazy<? extends U> lazy, Combiner<? super T, ? super U, ? extends V> fn)
```

Combine this with lazy using Combiner by invoking fn.combine(this.get(), lazy.get()).
Then we wrap the result back in Lazy.

```
T get()
```

Evaluates (if not yet evaluated, otherwise do not evaluate again) and returns the content.

3 Optional

static <T> Optional<T> empty()

Returns an empty Optional instance.

boolean equals(Object obj)

Indicates whether some other object is "equal to" this Optional.

Optional<T> filter(Predicate<? super T> predicate)

If a value is present, and the value matches the given predicate, return an Optional describing the value, otherwise return an empty Optional.

<U> Optional<U> flatMap(Function<? super T,Optional<U>> mapper)

If a value is present, apply the provided Optional-bearing mapping function to it, return that result, otherwise return an empty Optional.

T get()

If a value is present in this Optional, returns the value, otherwise throws NoSuchElementException.

void ifPresent(Consumer<? super T> consumer)

If a value is present, invoke the specified consumer with the value, otherwise do nothing.

<U> Optional<U> map(Function<? super T,? extends U> mapper)

If a value is present, apply the provided mapping function to it, and if the result is non-null, return an Optional describing the result.

static <T> Optional<T> of(T value)

Returns an Optional with the specified present non-null value.

T orElse(T other)

Return the value if present, otherwise return other.

T orElseGet(Supplier<? extends T> other)

Return the value if present, otherwise invoke other and return the result of that invocation.

String toString()

Returns a non-empty string representation of this Optional suitable for debugging.

4 Stream

boolean allMatch(Predicate<? super T> predicate)
boolean anyMatch(Predicate<? super T> predicate)
boolean noneMatch(Predicate<? super T> predicate)

Returns whether all / any / none elements of this stream match the provided predicate.

static <T> Stream<T> concat(Stream<? extends T> a, Stream<? extends T> b)

Creates a lazily concatenated stream, all of first then all of second.

long count()

Returns the count of elements in this stream.

Stream<T> distinct()

RReturns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.

static <T> Stream<T> empty()

Returns an empty sequential Stream.

Stream<T> filter(Predicate<? super T> predicate)

Returns a stream consisting of the elements of this stream that match the given predicate.

Optional<T> findAny()
Optional<T> findFirst()

Returns an Optional describing some / first element of the stream, or an empty Optional if the stream is empty.

<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)

Returns a stream consisting of the results of applying the provided mapping function to each element.

void forEach(Consumer<? super T> action)

Performs an action for each element of this stream.

static <T> Stream<T> generate(Supplier<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 ordered Stream produced by iterative application of a function f to an initial element seed.

Stream<T> limit(long maxSize)

Returns a stream consisting of the elements of this stream, truncated to be no longer than maxSize in length.

<R> Stream<R> map(Function<? super T,? extends R> mapper)

Returns a stream consisting of the results of applying the given function to the elements of this stream.

Optional<T> max(Comparator<? super T> comparator)
Optional<T> min(Comparator<? super T> comparator)

Returns the maximum / minimum element of this stream according to the provided Comparator.

static <T> Stream<T> of(T... values)

Returns a sequential ordered stream whose elements are the specified values.

Stream<T> peek(Consumer<? super T> action)

Returns a stream consisting of the elements of this stream, performing the provided action on each element.

T reduce(T identity, BinaryOperator<T> accumulator)
<U> U reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)

Performs a reduction on the elements of this stream, using the provided identity, accumulation and combining functions.

Stream<T> skip(long n)

Returns a stream consisting of the remaining elements of this stream after discarding the first n elements of the stream.

Stream<T> sorted(Comparator<? super T> comparator)

Returns a stream consisting of the elements of this stream, sorted according to the provided Comparator.

List<Object> toList()

Returns a List containing the elements of this stream.

5 List

boolean add(E e)

Appends the specified element to the end of 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 order.

void clear()

Removes all of the elements from this list (optional operation).

boolean contains(Object o)
boolean containsAll(Collection<?> c)

Returns true if this list contains the / all specified element.

static <E> List<E> copyOf(Collection<? extends E> coll)

Returns an unmodifiable List containing the elements of the given Collection, in its iteration order.

boolean equals(Object o)

Compares the specified object with this list for equality.

E get()

Returns the element at the specified position in this list.

int indexOf(Object o)

Returns the index of the first occurrence of the specified element in this list, or -1 if not found.

boolean isEmpty()

Returns true if this list contains no elements.

static <E> List<E> of()
static <E> List<E> of(E... elements)

Returns an unmodifiable list containing an arbitrary number of elements.

E remove(int index)

Removes the element at the specified position in this list (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 Stream<E> stream()

Returns a sequential Stream with this collection as its source.

Read the question

Figure 1: Read the question

6 Producer

```
Producer<T>
T produce(CreationalContext<T> ctx)

Causes an instance to be produced via the Producer.
```

7 Consumer

```
Consumer<T>
void accept(T t)

Performs this operation on the given argument.
```

8 Bifunction

```
Bifunction<T, U, R>
R apply(T t, U u)

Applies this function to the given arguments.
```

9 Functional Interface

```
@FunctionalInterface

An interface with only one abstract method.
```

10 Comparable

```
int compareTo(T o)

Compares this object with the specified object for order.
```

11 Comparator

```
int compare(T o1, T o2)

Compares its two arguments for order.
```

12 Generics and Wildcards

```
public class Pair<S extends Comparable<S>, T> implements Comparable<Pair<S, T>>

Defining a class with Generics.

public static <T> T findLargest(T[] array)

Defining a class with Generics.

@SuppressWarnings("unchecked")
T[] tmp = (T[]) new Object[size];
Wrapper<T>[] tmp = (Wrapper<T>[]) new Wrapper<?>[size];

Making a generic array (with wildcards).

Transformer<? super T, ? extends U>

Producer Extends, Consumer Super for best wildcard flexibility.
```