This document defines the type rules that need to occur to have our checker work as intended.

Note: Duplicated in git repository

Type Hierarchy (LUB and GLUB rules)

These can be found via the User Manual in our repository (https://github.com/gyhughes/checker-framework/tree/master/checker/manual)

Intro Rules

Type	Rules
@Unknown	All other ints
@NonNegative	Literals >= 0
@LTLength("a")	a.getIndexOf(char c) // string
@IndexOrHigh("a")	a.length, a.size(), a = new int[i] (i becomes @IndexOrHigh("a"))
@IndewOrLow("a")	
@IndexFor("a")	a = new int[i + 1] (i becomes @IndexFor("a")) Literals >= 0 && < x such that x is the value in MinLen(x) a
@MinLen(len) a	Implicitly created arrays (i.e. {1, 3, 4} is @MinLength(3)) a = new type[@MinLen(len) a2]
@IndexBottom	Null objects

Transfer Rules (operations)
The types of the values being added/subtracted do not change

Any types here can be substituted with any of their subtypes and still be valid.
Note that if a more specific relationship is defined, that one takes precedence.

<u>Addition</u>
-commutative

Addend 1	addend 2	New type
@Unknown	@Unknown	@Unknown
@NonNegative	@NonNegative	@NonNegative
@IndexOrLow(a)	1	@IndexOrHigh(a)
Anything	0	Type of(Anything)

<u>Subtraction</u>
-Non commutative

Minuen	Subtrahend	New type
@Unknown	@Unknown	@Unknown
@LTLength(a)	@NonNegative	@LTLength(a)
@IndexOrHigh(a)	1	@IndexOrLow(a)
Anything	0	Type of(Anything)

No special cases need to be added here involving indexes of different arrays. In practice, this is just never (or extremely rarely) done. For complete correctness, notice that from the table there are no rules that involve two different index for array types. This means that from our systems perspective, any type information related to the bounds of an array do not matter for any subtrahends or at least one of the addends.

<u>Math.min(…)</u>
Here, the resulting annotated type has the annotated type of the int with the weakest lower bound. In the In the case of both having the same lower bound, the resulting annotated type is the type with the strongest upper bound. In the case of annotations being indices for different arrays, the left type is preser

Dataflow Rules (comparisons)

- Compare applies to all types listed below the comparison symbol
- No name for array means it doesn't matter

Type	Comparison	New type
Anything	<= @Unknown >= @Unknown	Type of(Anything)
@Unknown	> @IndexOrLow @NonNegative @IndexOrHigh @IndexFor >= @NonNegative @IndexOrHigh @IndexFor	@NonNegative
@Unknown	< @IndexOrHigh(a) @IndexFor(a) @IndexOrLow(a) @LTLength(a) <= @LTLength(a) @IndexFor(a) @IndexOrLow(a)	@LTLength(a)
@IndexOrHigh @NonNegative	< @IndexOrHigh(a) @IndexFor(a) @IndexOrLow(a) @LTLength(a)	@IndexFor(a)

	<= @LTLength(a) @IndexFor(a) @IndexOrLow(a)	
@IndexOrLow(a) @LTLength(a)	> @IndexOrLow @NonNegative @IndexOrHigh @IndexFor >= @NonNegative @IndexOrHigh @IndexFor	@IndexFor(a)
@IndexOrHigh(a)	!= a.length (array) != a.size() (List)	@IndexFor(a)
@IndexOrLow(a)	!= -1	@IndexFor(a)
@MinLen(x) a	a.length > literal y	@MinLen(max(x, y + 1))
@MinLen(x) a	a.length >= literal y a.length == literal y	@MinLen(max(x, y))
@Unknown a	a.length != 0	@MinLen(1) a
Literal x	literal x == a.length	@IndexOrHigh(a) x

Rules for side-effecting/re-assignment

Compare these stretch goal behavior table below and implement them if it doesn’t look to difficult

All types not mentioned are unaffected.

Action	Calls	Transformations
Re-assignment	a = (some array or List)	{@IndexFor(“a”), @IndexOrHigh(“a”)} → @NonNegative {@LTLength(“a”), @IndexOrLow(“a”)} → @Unknown
Add to list	add(E e) add(int index, E element) addAll(Collection<? extends E> c) addAll(int index, Collection<? extends E> c)	No Transformations

Remove from list	remove(Object o)
	remove(int index)
	clear()
	removeAll(Collection<?> c)
	retainAll(Collection<?> c)
		{@IndexFor(“a”), @IndexOrHigh(“a”)} → @NonNegative
		{@LTLength(“a”), @IndexOrLow(“a”),} → @Unknown

Stretch goal behavior

Action	Calls	Transformations
Re-assignment	a = (some array or List)	{@IndexFor(“a”), @IndexOrHigh(“a”)} → @NonNegative {@LTLength(“a”), @IndexOrLow(“a”)} → @Unknown

Add one to list	add(E e)
	add(int index, E element)
		@IndexOrHigh(“a”) → @IndexFor(“a”)

Add arbitrary amount to list	addAll(Collection<? extends E> c) addAll(int index, Collection<? extends E> c)	No Transformations

Remove one from list	remove(Object o)
	remove(int index)
		@IndexFor(“a”) → @IndexOrHigh(“a”)
		@IndexOrHigh(“a”) → @NonNegative
		{@LTLength(“a”),
		@IndexOrLow(“a”),} → @Unknown

Remove arbitrary amount from list	clear()
	removeAll(Collection<?> c)
	retainAll(Collection<?> c)
		{@IndexFor(“a”),
		@IndexOrHigh(“a”)} → @NonNegative
		{@LTLength(“a”),
		@IndexOrLow(“a”),} → @Unknown

Type Rules

Only an @IndexFor(“a”) int i is allowed to do the following operations

If a is an array
```
a[i]
```
If a in a List
```
a.get(i)
```
If a is a string
```
a.chatAt(i)
```

arr[arr.length -1] is safe iff arr has that annotated type @MinLen(x), where x > 0