Null-hostile collections

Many JDK collection types permit null elements:

- `ArrayList`
- `LinkedList`
- `Hash{Set,Map}`
- `LinkedHash{Set,Map}`
- `Tree{Set,Map}` (with suitable comparator)
- `IdentityHashMap`
- `EnumMap` (values)
- `CopyOnWriteArray{List,Set}`

But many don't:

- `EnumSet`
- `EnumMap` (keys)
- `ConcurrentHashMap`
- `ConcurrentSkipList{Set,Map}`
- All ten `Queue` implementations except `LinkedList`

Likewise in Guava we have many general-purpose collections that permit null:

- `ArrayListMultimap`
- `HashBiMap`
- `HashMulti{set,map}`
- `LinkedHashMulti{set,map}`
- `TreeMulti{set,map}` (with suitable comparator)
- `LinkedListMultimap`
- `MutableClassToInstanceMap`
- `HashBasedTable`
- `Sets.union/intersection/difference`

but many that do not:

- `ConcurrentHashMultiset`
- `EnumBiMap`
- `EnumMultiset`
- `MinMaxPriorityQueue`
- `Interners`
- `MapMaker`-made maps
- `Sets.cartesianProduct/powerSet`
- All implementations of `ImmutableCollection` and `ImmutableMap`

But what if?

What if you find yourself wanting to put a null element into one of these null-hostile beasts?

- If in a Set or as a key in a Map -- don't; it's clearer (less surprising) if you explicitly special-case null during lookup operations

- If as a value in a Map -- leave out that entry; keep a separate Set of non-null keys (or null keys)
- If in a List -- if the list is sparse, might you rather use a `Map<Integer, E>` ?
- Consider if there is a natural "null object" that can be used. There isn't always. But sometimes.
  - example: if it's an enum, add a constant to mean whatever you're expecting null to mean here.
- Just use a different collection implementation, for example `Collections.unmodifiableList(Lists.newArrayList())` instead of `ImmutableList`.
- Mask the nulls (this needs more detail)
- Use `Optional<T>`

See also: using and avoiding null.