

Iterator Types
fail-fast
<ul style="list-style-type: none">collections maintain an internal counter called modCount.<ul style="list-style-type: none">each time an item is added or removed from the Collection, counter gets incremented.when iterating, on each next() call, the current value of modCount gets compared with the initial value.if there's a mismatch, it throws ConcurrentModificationException which aborts the entire operation.however, this check is done without synchronization, so there is a risk of seeing a stale value of the modification count and therefore that the iterator does not realize a modification has been made.this was a deliberate design tradeoff to reduce the performance impact of the concurrent modification detection codeConcurrentModificationException can arise in single-threaded code as well; this happens when objects are removed from the collection directly.if during iteration over a Collection, an item is removed using Iterator's remove() method, that's entirely safe and doesn't throw an exception.example: Default iterators on Collections from java.util package such as ArrayList, HashMap, etc.
weakly consistent
<ul style="list-style-type: none">reflects some but not necessarily all of the changes that have been made to their backing collection since they were created.<ul style="list-style-type: none">e.g., if elements in the collection have been modified or removed before the iterator reaches them, it definitely will reflect these changes, but no such guarantee is made for insertions.the default iterator for the ConcurrentHashMap is weakly consistent.<ul style="list-style-type: none">this means that this Iterator can tolerate concurrent modification, traverses elements as they existed when Iterator was constructed and may (but isn't guaranteed to) reflect modifications to the Collection after the construction of the Iterator.example: Default iterators on Collections from java.util.concurrent package such as ConcurrentHashMap, ConcurrentSkipListSet etc.
fail-safe
<ul style="list-style-type: none">these iterators create a clone of the actual Collection and iterate over it. If any modification happens after the iterator is created, the copy still remains untouched. Hence, these Iterators continue looping over the Collection even if it's modified.disadvantage is the overhead of creating a copy of the Collection, both regarding time and memory.example: CopyOnWriteArrayList, CopyOnWriteArraySet

