1. ArrayList is not synchronized
2. ArrayList supports dynamic array which can grow as needed
3. Size of ArrayList can be dynamically increased or decreased
4. ArrayList are created with initial size = 10
5. ArrayList contains duplicate elements. ArrayList maintain insertion order of the element
6. ArrayList is not synchronized collection, hence it is not suitable to be used between multiple Threads concurrently.

If you want to use ArrayList data structure in multithreaded environment, then you need to use the copyonwritearraylist or use collection.synchronised() to create a synchronized list

1. former is a part of concurrent collections package and is much more stable than the second one, but only useful when they are many readers and writers

A new copy of ArrayList is created every time a write happens, it can be overkill if used in write heavy environment

Second option is strictly synchronized collection, much like vector and hashtable, but is not scalable because once a number of threads increases drastically contention become a huge issue

1. copyonwritearraylist is recommended for concurrent multithreading environment as it is optimized for multiple concurrent reads and creates copy for write operation.

this was added in JDK 1.5 Java.util.concurrent package along with concurrenthashmap

1. When ArrayList get full, it creates another array using system.Arraycopy to copy all elements from one array to another array. this is where array insertion takes a lot of time.
2. Iterator and ListIterator of Java ArrayList are fail fast. it means if ArrayList is structurally modified at any time after the Iterator is created in any way except the iterator’s own add (), remove () methods, the iterator will throw a concurrentmodificationexception

Thus, in the face of concurrent modification, the iterator fails quickly and clearly that's why it's called fail fast

1. concurrent modification exception is not guaranteed and it is only thrown at best effort.
2. If you are creating synchronized list, it is recommended to create it while creating an instance of underlying ArrayList to prevent accidental non-synchronized access to the list
3. An application can increase the capacity of the ArrayList instance before adding a large number of elements using the ensureCapacity () operation. this may reduce the amount of incremental reallocation due to incremental filling of ArrayList
4. The size (), isEmpty (), get (), set (), iterator () and listIterator () operation run in constant time because the list is based on Array but adding and removing element is costly as compared to linklist
5. ArrayList class is enhanced in JDK 1.5 to support Generics which adds extra type safety on ArrayList.

It is recommended to use generic version of ArrayList to ensure that's your ArrayList contain only specific type of elements and avoid any class cast exception

1. If we set ArrayList reference to null in Java, all the elements inside ArrayList becomes eligible for garbage collection in Java provided that there are no strong references exist for those objects
2. Always use isEmpty () method to check if ArrayList is empty or not, instead of using size () = 0, check former one as it is more readable.