String vs stringbuffer vs stringbuilder (java)

1. String:

- **Immutability:**

  `String` objects are immutable. Once a `String` object is created, its value cannot be changed. Any operation that appears to modify a `String` actually creates a new `String` object in memory.

- **Thread-Safety:**

  `String` is inherently thread-safe due to its immutability. Multiple threads can access a `String` object concurrently without causing data inconsistency.

- **Performance:**

  Due to the creation of new objects on every modification, `String` operations can be less efficient than `StringBuffer` or `StringBuilder` for frequent modifications, especially in loops.

- **Use Case:**
  Ideal for representing fixed, unchanging text data.

2. StringBuffer:

- **Mutability:**

  `StringBuffer` objects are mutable, meaning their content can be modified after creation without creating new objects.

- **Thread-Safety:**

  `StringBuffer` is thread-safe as its methods are synchronized. This ensures that only one thread can access and modify the `StringBuffer` at a time, preventing data corruption in multi-threaded environments.

- **Performance:**

  The synchronization overhead makes `StringBuffer` slower than `StringBuilder`.

- **Use Case:**
  Suited for mutable strings in multi-threaded environments where thread-safety is crucial.

3. StringBuilder:

- **Mutability:**

  `StringBuilder` objects are also mutable, similar to `StringBuffer`.

- **Thread-Safety:**

  `StringBuilder` is not thread-safe as its methods are not synchronized. This makes it faster than `StringBuffer` because it avoids the overhead of synchronization.

- **Performance:**

  Offers better performance than `StringBuffer` in single-threaded environments due to the lack of synchronization.

- **Use Case:**
  Recommended for mutable strings in single-threaded environments where performance is a priority and thread-safety is not required.