StringBuilder and StringBuffer

In Java, both StringBuilder and StringBuffer are classes that provide a way to manipulate strings. They are similar in functionality but differ in terms of thread-safety and performance characteristics.

StringBuilder:

StringBuilder is a non-thread-safe class introduced in Java 1.5. It provides methods for efficient manipulation of strings, such as appending, inserting, or deleting characters. The StringBuilder class is mutable, meaning you can modify the contents of the string without creating a new object.

However, StringBuilder is not synchronized, which means it is not thread-safe. If multiple threads access a StringBuilder object concurrently and modify its content, you may encounter issues like data corruption or inconsistent results. Due to its non-thread-safe nature, StringBuilder generally offers better performance compared to StringBuffer.

Example usage of StringBuilder:

```
StringBuilder sb = new StringBuilder();
sb.append("Hello");
sb.append(" ");
sb.append("world!");
String result = sb.toString();  // "Hello world!"
```

StringBuffer:

StringBuffer is a thread-safe class that predates StringBuilder and has been available since the early versions of Java. It provides the same functionality as StringBuilder for string manipulation, but it ensures that multiple threads can safely access and modify its content.

StringBuffer achieves thread-safety by synchronizing access to its methods, which introduces some overhead. Consequently, StringBuffer is generally slightly slower than StringBuilder in single-threaded scenarios.

Example usage of StringBuffer:

```
StringBuffer sb = new StringBuffer();
sb.append("Hello");
sb.append(" ");
sb.append("world!");
String result = sb.toString();  // "Hello world!"
```

In summary, if you're working in a single-threaded environment or you don't require thread-safety, **StringBuilder** is recommended for better performance. On the other hand, if you're dealing with multiple threads and need to ensure thread-safety, you should use **StringBuffer**, even though it comes with a slight performance penalty.

```java
package com.hdfc.collections;

public class StringExample {

    public static void main(String[] args) {


        String s1 = "abc";
        s1 = s1.toUpperCase();

        //non synchronised method
        // not thread safe
        //single thread application
        //fast
        StringBuilder sb1 = new StringBuilder("abc");
        StringBuilder sb2 = new StringBuilder("abc");

        //StringBuilder does not override equals method
        System.out.println(sb1==sb2); //false, both reference are different
        System.out.println(sb1.equals(sb2)); //false  - StringBuilder does not override equals method, hence
Object class equals method is called and reference comparation is done.
        System.out.println(sb1==sb1); // true


        //sb.append("xyz");
        //String result = sb.toString();

        //multi thread application then use StringBuffer
        // all methods are synchronized
        //slow in multi thread env
        StringBuffer sbuffer1 = new StringBuffer("abc");
        StringBuffer sbuffer12 = new StringBuffer("abc");

        //StringBuilder does not override equals method
        System.out.println(sbuffer1==sbuffer12); //false, both reference are different
        System.out.println(sbuffer1.equals(sbuffer12)); //false  - StringBuilder does not override equals
method, hence Object class equals method is called and reference comparation is done.
        System.out.println(sbuffer1==sbuffer12); // true
    }

}
```

```java
package com.hdfc.collections;

public class StringExample {

    public static void main(String[] args) {


        String s1 = "   abc         cba         ";
        System.out.println(s1.trim());
        System.out.println(s1.replaceAll("\\s+", " "));

        String s2= "abc";
        System.out.println(s2.substring(0,2));
        //0 included,
        // 2 exclude (n-1)

       // s1 = s1.toUpperCase();

        //non synchronised method
        // not thread safe
        //single thread application
        //fast
        StringBuilder sb1 = new StringBuilder("abc");
        StringBuilder sb2 = new StringBuilder("abc");

        //StringBuilder does not override equals method
        //System.out.println(sb1==sb2); //false, both reference are different
       // System.out.println(sb1.equals(sb2)); //false  - StringBuilder does not override equals method, hence Object class
equals method is called and reference comparation is done.
        //System.out.println(sb1==sb1); // true


        //sb.append("xyz");
        //String result = sb.toString();

        //multi thread application then use StringBuffer
        // all methods are synchronized
        //slow in multi thread env
        StringBuffer sbuffer1 = new StringBuffer("abc");
        StringBuffer sbuffer12 = new StringBuffer("abc");

        //StringBuilder does not override equals method
        System.out.println(sbuffer1==sbuffer12); //false, both reference are different
        System.out.println(sbuffer1.equals(sbuffer12)); //false  - StringBuilder does not override equals method, hence
Object class equals method is called and reference comparation is done.
        System.out.println(sbuffer1==sbuffer1); // true
        System.out.println(sbuffer1.toString().equals(sbuffer12.toString())); // true

    }

}
```