**Q 17:** What is the main difference between a String and a StringBuffer class?

*String* is **immutable**: you can’t modify a string object but can replace it by creating a new instance. Creating a new instance is rather

expensive.

**//Inefficient version using immutable String**

String output = “Some text”

Int count = 100;

for(int I =0; i<count; i++) {

output += i;

}

return output;

The above code would build 99 new String

objects, of which 98 would be thrown away

immediately. Creating new objects is not

efficient.

StringBuffer is **mutable**: use StringBuffer or StringBuilder when you want to modify the contents. ***StringBuilder*** was added in Java 5 and it is identical in all respects to ***StringBuffer*** except that it is not synchronised, which makes it slightly faster at the cost of not being thread-safe.

**//More efficient version using mutable StringBuffer**

StringBuffer output = new StringBuffer(**110)**;

Output.append(“Some text”);

for(int I =0; i<count; i++) {

output.append(i);

}

return output.toString();

The above code creates only two new objects, the *StringBuffer* and the final *String* that is returned. StringBuffer expands as needed, which is costly however, so it would be better to initilise the *StringBuffer* with the correct size from the start as shown.

Another important point is that creation of extra strings is not limited to ‘overloaded mathematical operators’ (“+”) but

there are several methods like **concat(), trim(), substring(),** and **replace()** in String classes that generate new

string instances. So use StringBuffer or StringBuilder for computation intensive operations, which offer better

performance.