s1220006 Cristian Kamia

# Java II report

## Difference Between String , StringBuilder And StringBuffer Classes

### String

String is *immutable* ( once created can not be changed )object . The object created as a String is stored in the Constant String Pool .
Every immutable object in Java is thread safe ,that implies String is also thread safe . String can not be used by two threads simultaneously.
String once assigned can not be changed.

String demo = " hello " ;
// The above object is stored in constant string pool and its value can not be modified.

demo="Bye" ; //new "Bye" string is created in constant pool and referenced by the demo variable

// "hello" string still exists in string constant pool and its value is not overrided but we lost reference to the "hello"string

### StringBuffer

StringBuffer is mutable means one can change the value of the object . The object created through StringBuffer is stored in the heap . StringBuffer has the same methods as the StringBuilder , but each method in StringBuffer is synchronized that is StringBuffer is thread safe .

Due to this it does not allow two threads to simultaneously access the same method . Each method can be accessed by one thread at a time .

But being thread safe has disadvantages too as the performance of the StringBuffer hits due to thread safe property . Thus StringBuilder is faster than the StringBuffer when calling the same methods of each class.

StringBuffer value can be changed , it means it can be assigned to the new value . Nowadays its a most common interview question ,the differences between the above classes .
String Buffer can be converted to the string by using
toString() method.

StringBuffer demo1 = new StringBuffer("Hello") ;
// The above object stored in heap and its value can be changed .
demo1=new StringBuffer("Bye");
// Above statement is right as it modifies the value which is allowed in the StringBuffer

## StringBuilder

StringBuilder  is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that StringBuilder is also not thread safe.
StringBuilder is fast as it is not thread safe .

```
StringBuilder demo2= new StringBuilder("Hello");
// The above object too is stored in the heap and its value can be modified
demo2=new StringBuilder("Bye");
// Above statement is right as it modifies the value which is allowed in the StringBuilder
```

# Multi-Thread of Java

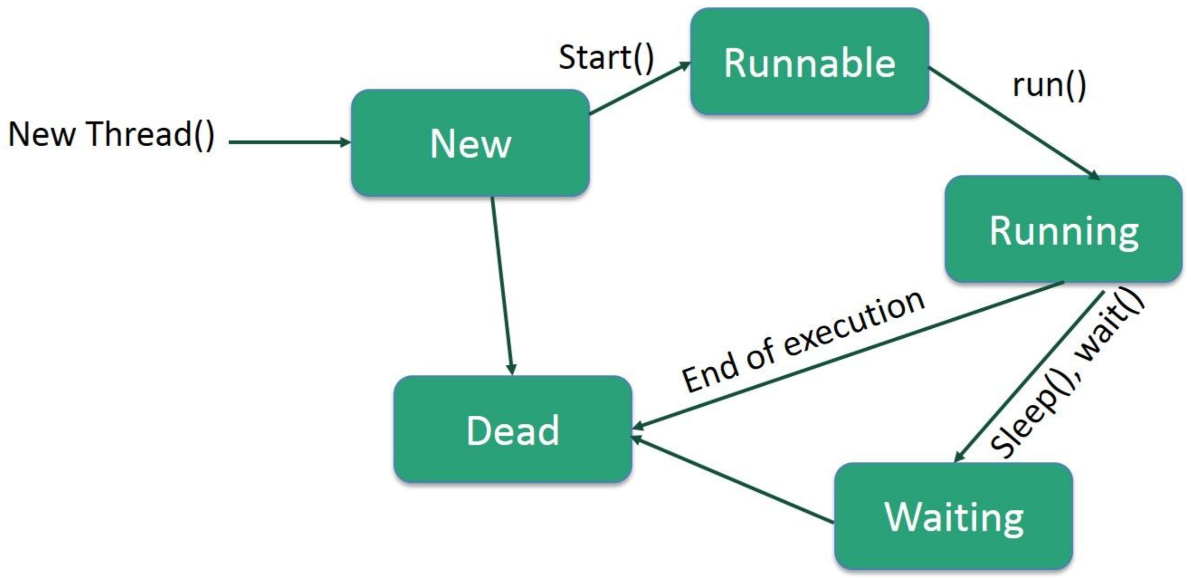
Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

## Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle −

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.