Java Performance Considerations

by
Anand Kulkarni
anand.pune38@gmail.com

Software Performance

- Now a days, software is becoming more & more complicated. Hence, maintaining the speed, response time, stability, reliability, scalability and resource usage of a software application under particular workload is a major challenge.
- IT industries perform continuous performance testing to ensure scalability of software applications.
- Thus, software performance can be optimized by:
  1. Developers adopting better coding practices
  2. Continuous performance testing conducted by testers using different tools like Jmeter, Jprofile etc.

Good coding practices for performance tuning

1) Using StringBuilder and + to concatenate strings
2) Use of primitives
3) Avoiding BigInteger and BigDecimals
4) Checking current log level
5) Caching expensive resources
6) Tuning GC

1 - Using StringBuilder and + to concatenate strings

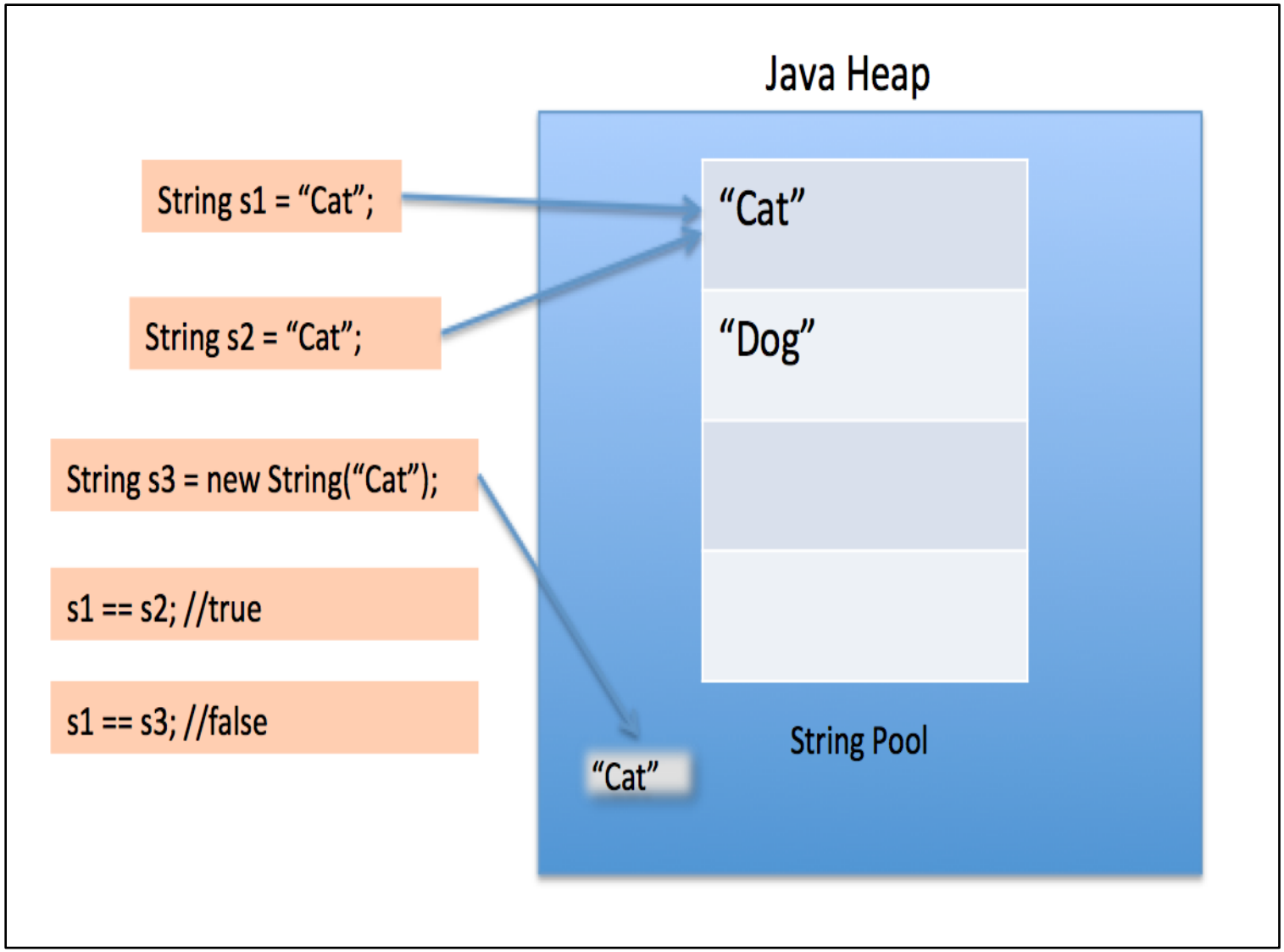
String manipulation is a common requirement of many business logics.

Java provides us 3 options for string manipulation:

- String
- StringBuffer
- StringBuilder

In case of programmatic string manipulation, Java recommends to use StringBuilder for string manipulation over class String & StringBuffer. It is because String is an immutable class & StringBuffer is synchronized object. Thus, both adversely affect on performance. On the other side, StringBuilder is an mutable & non-synchronized object.

String Pool


Java Heap
String s1 = "Cat";
String s2 = "Cat";
String s3 = new String("Cat");
s1 == s2; //true
s1 == s3; //false
"Cat"
"Dog"
String Pool
"Cat"

Programmatic concatenation using StringBuilder

```
StringBuilder sb = new StringBuilder("This is a test");

for (int i=0; i<10; i++) {

        sb.append(i);

        sb.append(" ");

}

log.info(sb.toString());
```

Static text concatenation using String

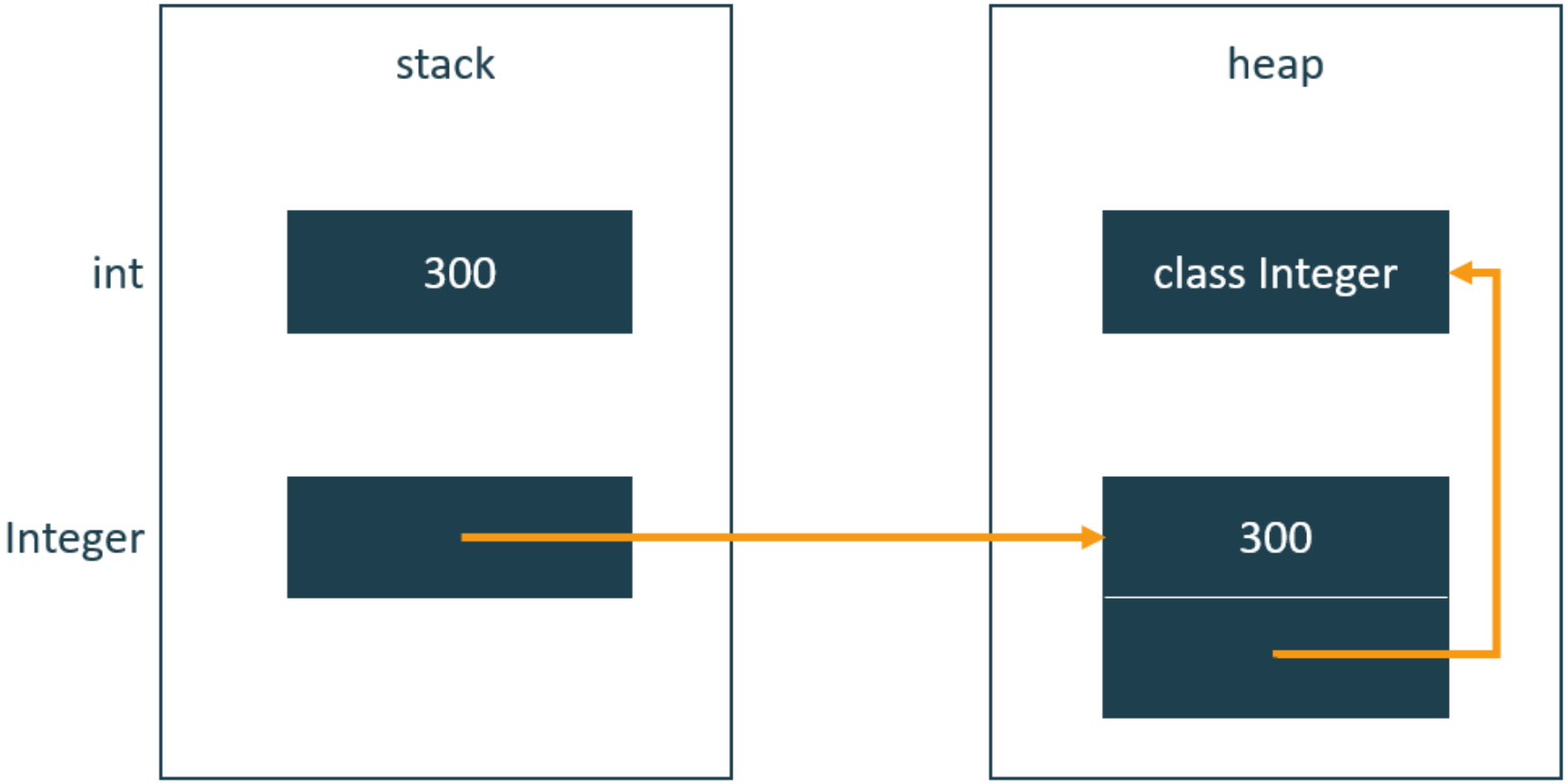
String message = “I am an Indian” + “ I stay in Maharashtra”.

In this situation, you should concatenate your static Strings with a simple + instead of StringBuilder. Your Java compiler will optimize this and perform the concatenation at compile time. So, at runtime, your code will just use 1 String, and no concatenation will be required.

2 - Using Primitives

- Another quick and easy way to avoid any overhead and improve the performance of your application is to use primitive types instead of their wrapper classes.
- So, it's better to use an int instead of an Integer, or a double instead of a Double. That allows your JVM to store the value in the stack instead of the heap to reduce memory consumption and overall handle it more efficiently.
- Also, wrapper classes come up with the overhead of boxing & unboxing which in turn reduce the performance.

2 - Using Primitives



3 - Avoiding BigInteger and BigDecimals

- BigInteger & BigDecimal classes are used for mathematical operation which involves very big integer/double calculations that are outside the limit of all available primitive data types.
- BigInteger and BigDecimal require much more memory than a simple long or double and slow down all calculations dramatically.
- Hence try to avoid using these classes if your data lies in the range if primitive data types.

4 - Checking current log level

➢ Before you create a log message, you should always check the current log level first. Otherwise, you might create a String with your log message that will be ignored afterward.

```
//Poor performance
logger.debug("User [" + userName + "] called method X with [" + i + "]");
//Good performance
if (logger.isDebugEnabled()) {
        logger.debug("User [" + userName + "] called method X with [" + i + "]");
}
```

5 - Caching expensive resources

- Caching is a popular solution to avoid the repeated execution of expensive or frequently used code snippets.
- A typical example is caching database connections in a pool. The creation of a new connection takes time, which you can avoid if you reuse an existing connection.
- When you think about caching, keep in mind that your caching implementation also creates an overhead. You need to spend additional memory to store the reusable resources, and you need to manage your cache to make the resources accessible or to remove outdated ones.
- So, before you cache any resources, make sure that you use them often enough to outweigh the overhead of your cache implementation.

6 – Tuning GC

- Garbage Collection(GC) tuning is the process of adjusting the startup parameters of your JVM-based application to match the desired results.
- With poor GC tuning, it is observed that around 33% of the time GC was running its job & application was not doing any job. This can be a major setback to overall application's performance. Hence, it is essential to tune your GC.
- Note that GC tuning is a last option. If you really feel that application's performance is badly suffering then think of GC tuning.

How to tune GC?

- Java provides us various parameters to be passed to JVM while starting any Java application. Thus, tuning GC means setting correct value of these parameters.
- There are various parameters offered by JVM. However, we are going to look into few important:

-Xms<size> - Sets initial Java heap size

-Xmx<size> - Sets maximum Java heap size

-XX:PermSize<size> - Sets initial PermGen Size.

-XX:MaxPermSize<size> - Sets the maximum PermGen Size.

-Xss<size> - Sets java thread stack size

-verbose:gc - Activates the logging of garbage collection information.

Thank you!!