# Buenas prácticas que deberías aplicar para optimizar código Java

1. Evitar concatenaciones de Strings, utilizar **StringBuilder** para tal caso. La concatenación de Strings produce cada vez un nuevo objeto y por tanto, mayor consumo de memoria y mayor recolección de basura. (esta deberías hacedla siempre).
2. Crear métodos con el menor número de parámetros posible.
3. Si se van a realizar operaciones complejas como, senos, cosenos u otras operaciones complicadas de coma flotante y se sabe de antemano cuál va a ser el resultado, es conveniente **pre-calcular** estos valores y utilizarlos como constantes.
4. Sacar fuera de los bucles las **constantes, declaración de variables, creación de nuevos objetos, comentarios, etc.** Es probablemente una de los errores que puede consumir más CPU, por supuesto sé más cuidadoso con aquellos bucles que tengan más iteraciones (Iteración significa repetir un proceso varias veces), con lo de sacar código no se trata de modificar la función final del bucle sino de optimizar al máximo su rendimiento, con esto queremos decir no incluir la declaración de una variable, no incluir comentarios, etc.
5. En los bucles, evitar el uso de las expresiones complejas en las condiciones. Usar en su lugar un valor constante pre-calculado de las condiciones.

   Por ejemplo, si tenemos

   ```
   for(int i=0;i<lista.size();i++) {
    . . .
   }
   ```

   Sustituirlo por

   ```
   int len=list.size();
   for(int i=0;i<len; i++){
    . . .
   }
   ```

6. Trata de evitar crear objetos innecesariamente.

   Por ejemplo, si tenemos

   ```
   A a = new A();   // creación innecesaria
   if(i==1){
      list.add(a);
   }
   ```

   Debe decir

   ```
   if(i==1){
      A a = new A();
      list.add(a);
   }
   ```

7. Reutilizar las variables siempre que se pueda.
8. No repetir, menos repetir y sobre todo no repetir código.
9. Codificar genéricamente, reutilizando el código.

10. No reinventar la rueda. No perdamos el tiempo codificando al que ya está hecho y funciona. Incorporar librerías o frameworks existentes, y que funcionan bien, en nuestras aplicaciones.
11. En la medida de lo posible utiliza el modificador **final**.
12. Siempre que vayamos a acceder a la misma posición de un array varias veces, es mejor guardar esa posición en una variable local y así **evitar el acceso repetido al índice del array**.
13. Eliminar código innecesario. Si tienes código innecesario elimínalo, parece obvio pero en muchas ocasiones esos pequeños trozos de código se mantienen en todas las versiones de la aplicación, si tienes que reescribir una función pero no estás del todo seguro y por lo tanto no quieres perder la versión original, haz una copia de seguridad y de ese modo no se ralentizarás la ejecución de la aplicación.
14. **Liberar recursos** tan pronto como sea posible, como conexiones de red, a streams o a ficheros. Normalmente, se suele liberar este tipo de recursos dentro de la cláusula finally para asegurarnos de que los recursos se liberan aun cuando se produzca alguna excepción.
15. **Referenciar a null** instancias de objetos que ya no se van a usar para liberar memoria.
16. **Evitar** el uso de la **sincronización** dentro de bucles.
17. Siempre que sea posible, usar arrays unidimensionales en lugar de bidimensionales.
18. **El acceso a variables locales es más rápido** que a atributos de la clase. Siempre que sea posible asignar atributos de una clase a variables locales si se va a hacer referencia a ellas varias veces dentro de un método o bucle.
19. Usar operadores como **x+=1** o **x++ en vez de x = x+1** ya que generan menos bytecode. Igualmente, si tenemos que hacer **suma = suma + valor**, usar mejor **suma + = valor.**
20. **Usar tipos escalares** (primitivos) en lugar de objetos Java equivalentes siempre que sea posible, por ejemplo, int en lugar de Integer.
21. Usar **excepciones** únicamente cuando sea necesario, ya que cada excepción lanza un nuevo objeto.
22. Minimiza y optimiza el acceso a disco. Manipular datos de los discos duros o de las memorias flash es mucho más lento que manipular datos almacenados en memoria por eso si vas a manejar archivos ten en cuenta este punto.