100001111100 110110101101 110010110000 111101010011 101101010110 000010101110 110110010001 011101010111 000101101011 010110011010 101110110000 111110101011 010001111110 010011111110 101010111001 110010010111 110001100101	Wij en onze partners slaan informatie, zoals unieke identificatoren in cookies, op apparaten op en/of gebruiken deze om persoonsgegevens te verwerken. U kunt uw keuzes te kennen geven of beheren door hieronder te klikken. U kunt ze ook op elk moment wijzigen op de pagina met onze privacyverklaring. Uw keuzes worden aan onze partners doorgegeven en hebben geen effect op browsegegevens.	

public static String concatStringsWSep(Iterable<String> strings, String separator) {
StringBuilder sb = new StringBuilder();
String sep = "";
for(String s: strings) {
sb.append(sep).append(s);
sep = separator;
}
return sb.toString();
}
		Guava is a pretty neat library from Google:
		
		Joiner joiner = Joiner.on(", ");
		joiner.join(sList);

list.stream().collect(Collectors.joining(delimiter));	If you are developing for Android, there is TextUtils.join provided by the SDK.	Your approach is dependent on Java's ArrayList#toString() implementation. While the implementation is documented in the Java API and very unlikely to change, there's a chance it could. It's far more reliable to implement this yourself (loops, StringBuilders, recursion whatever you like better). Sure this approach may seem "neater" or more "too sweet" or "money" but it is, in my opinion, a worse approach.
I really like this answer b/c you can use a foreach and it is very simple, but it is also more inefficient. How can you make it a tighter loop?	Have you seen this Coding Horror blog entry? The Sad Tragedy of Micro-Optimization Theater I am not shure whether or not it is "neater", but from a performance-standpoint it probably won't matter much.	
