Object class & Hashcode:

Object class: The Object class is the parent class of all the classes in Java by default which implements serialize interface.

The Object class is beneficial if we want to refer any object whose type is not known. Notice that parent class reference variable can refer the child class object, known as upcasting.

The Object class provides some common behaviours to all the objects such as Object can be compared, object can be cloned, object can be notified.

public final class getClass() → returns the Class class object of this object. Also can be used to get metadata of this class.

public int hashCode() → returns the hashcode number for this object.

public boolean equals(Object obj) → compares given object to this object.

protected Object clone() throws CloneNotSupportedException → Creates and returns the exact copy (clone) of this object.

public String toString() → Returns string representation of this object

public final void wait() throws InterruptedException ⟶ Cause the current thread to wait until another thread notifies.

protected void finalize throws Throwable → is invoked by the garbage collector before object is being garbage collected.

HashCode:

hashCode in Java is a function that returns the hashCode value of an object on calling. It returns an integer or 4 byte value which is generated by the hashing algorithm.

The process of assigning a unique value to an object or attribute using an algorithm, which enables quicker access is known as hashing.

Eg: We can consider dictionary as a hashing implementation. like A would have both Apple & Anaconda by it will be arranged accordingly.

Tubin's bday 9/5 → 9+5 → 13 → 4
Mallu's bday 4/4 → 2+4 → 6
Neetha's bday 4/2 → 2+4 → 6

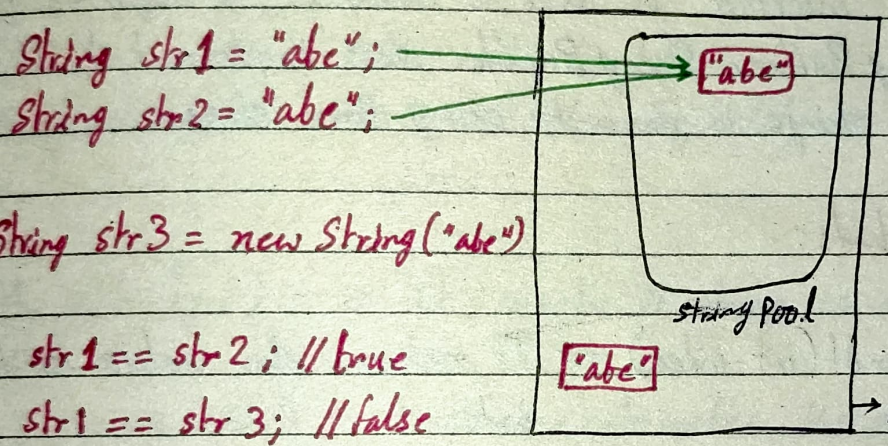
So if we need to find with hashcode 6 Tubin will not be possible solution.

Either its Mallu or Neetha.

String & String builders and its uses:

String: A String is a sequence of characters. In Java, objects of string are immutable which means a constant and cannot be changed once created.

Any String we create will be stored in our RAM inside a string pool which is also inside Heap memory.



If we create any string which has same value then only one instance is created in string pool where value is stored & both variable i.e str1 & str2 are pointing to same memory location.

But when we create a String a an object i.e with help of new keyword that object get created outside the string pool in heap memory.

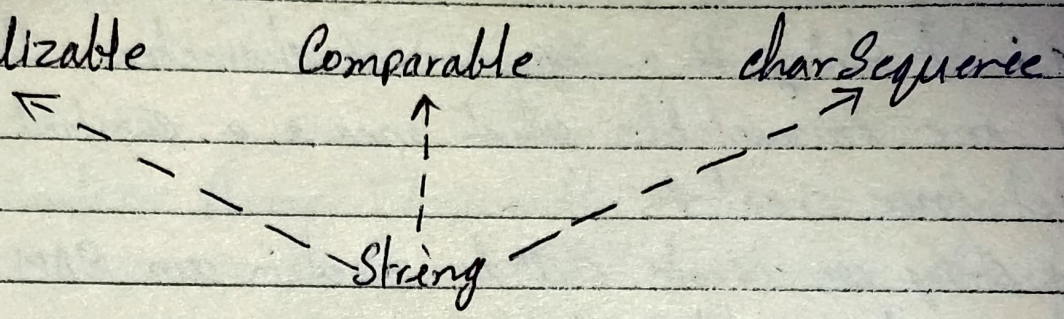
Then why str1 == str2 true & str1 == str3 is false?

This is because "==" operator checks if both variables are pointing to same location in memory. This do not compare the value of string.

Thats why we should always use .equals() methods to check if content of two strings are same instead of "==" operator.

str1.equals(str2) OR str1.equals(str3) returns true.

wa.lang.String class implements Serializable, Comparable, charSequence terfaces.



CharSequence interface is used to represent the sequence of characters String, stringBuffer, StringBuilder class implement it. It means, We can create strings in Java by using three classes.

String methods:

1. str1.charAt(int index) → It returns char value for the particular index
2. str1.length() → It returns the string length.
3. str1.format(String format, Object... args) → It returns a formatted string

eg:

```
String sf3 = String.format("value is %.32.12f", 32.33434);
// returns 12 char fractional part filling with 0
```

4. str1.subString(int beginIndex, int endIndex) → It returns substring for given begin & end index.

Also if end index is not mentioned returns from start index to rest of the string.

5. str1.Contains(charSequence s) → It returns true or false after matching the sequence of char value.

6. static String join(charSequence delimiter, charSequence... elements)
→ It returns the joined string of given elements with given delimiter in between.
eg: System.out.println(String.join("/", "12", "5", "2021"));
output: 12/5/2021

7. str1.equals(str2) → It checks equality of a string with given

8. str1.isEmpty() → It checks if string is empty.

9. str1.concat(str2) → It concatenates the specified string.

10. str1.replace(char old, char new) → It replaces all occurrences of the specified char value. It can also replaces all occurrences of specified charSequence.

11. ~~static~~ String.equalsIgnoreCase(String another) → It compares another string. It doesn't check case.

eg: str1.equalsIgnoreCase(str2); → // true.

12. str1.toLowerCase() / str1.toUpperCase() → respectively change the string to lower & upper case.

13. Str1.trim() → It remove begining and ending spaces of this string.

14. static String valueOf(int value) → It converts given type into string. It is an overloaded method.

StringBuffer : StringBuffer class is used to create mutable string objects. The StringBuffer class in Java is the same as string class except it is mutable.

★ Java StringBuffer class is thread-safe i.e. multiple threads can not access it simultaneously. So, it is safe and will result in an order.

StringBuffer() → Empty string buffer with capacity 16
StringBuffer(String Str) → stringBuffer with specified string
StringBuffer(int capacity) → empty StringBuffer with sp. capacity

Methods in String Buffer:

1) .append(String s)
2) .insert(int offset, String s)
3) .replace(int startIndex, int endIndex, String str)
4) .delete(int startIndex, int endIndex)
5) .capacity()
6) .length()
7) .substring(int beginIndex, int endIndex)

String Builder: String Builder in Java: represents a mutable sequence of characters. Since the String class in Java creates immutable character sequence.

String Builder class & String Buffer class both provide mutable strings however, both differ in basis of synchronization String Builder does not guarenty the synchronization where as String Buffer does.

constructors:

String Builder (): initialized with 16 char capacity & no char
String Builder (int Capacity): No char & given capacity.
String Builder (CharSequence ch): with given character sequence.
String Builder (String str): with given string.

Methods:

1) .append("...");
2) .capacity();
3) .charAt(i);
4) .delete(i, j);
5) .index of (" ")
6) .Last Index of ()
7) .length();
8) .replace (int str, int end, string str)
9) .reverse()
10) .set CharAt (index, char)
11) .subString ()
12) .toString ()