```
import java.io.*
import java.util.*
```

STRINGS

128 ASCII Characters are covered by Strings.

Java uses UTF-16 (16 bits) } Unicode Format.

Java Strings
- → char Array (like in C)
- → String Class (like in C++) (Immutable).
- → StringBuilder (Mutable not thread safe)
- → StringBuffer (Mutable, thread safe but overhead).

Examples

1)
```
char x = 'a';
System.out.println((int)(x));
```
O/P → 97.

2) Array to String.

Method 1
```
char[] arr = {'p','q','r','s'};
String str1 = String.valueOf(arr);
```

Method 2
Use `Arrays.toString(arr);`

3) String to Array
```
x = "abcdef"
char[] char1 = x.toCharArray();
```

4) for char Array ——→ Use x[i]
for Strings ——→ Use x.charAt[i] } to access an element in them.

Q) print freq of char (in sorted order).

```
String str = "geeksforgeeks";
int[] count1 = new int[26]

for (int i=0; i<str.length(); i++)
  { count1[str.charAt(i) - 'a']++;
  }
for (int i=0; i<26; i++)        → constant time as only 26 loops not O(n)
  { if (count1[i]>0)
      { System.out.println ((char)(i+'a') + " " + count1[i]);
      }
  }
```

Note :— charAt cannot be used on left side i.e

str1.charAt(3) = 'a'; } not valid as strings are immutable.

→ str = 'geeks';
str.substring(2,4) ⟶ ek
  ↳ not counted.

Imp fns (String)
contains
equals
compareTo → for lexi
indexOf

Q. Palindrome Check.

Naive → take rev of str

Compare rev == str & Return true/false.

~~Prog~~

```
bool isPal (String str)
{
  StringBuilder str_rev = StringBuilder (str);

  str_rev.reverse();

  System.out.println( str.equals(str_rev.toString()));
}
```

O(n) & O(n).

Efficient → Compare first & last characters.

Prog.

```
⇒ {int begin = 0;
   int end = 0;

   while(begin < end)
   {
      if(str.charAt (begin) != str.charAt(end)){
         return false;
      }
      begin++;
      end--;
   }
   return true;
}
```

Q. Check if a String is subseq of other.

Note → Substring is continuous, not subsequence.
→ 2^n subseq possible for n-letter word.

* Naive → Generate all subseq & compare all.
$O(2^n * n)$.

Efficent ⟹ 2 pointer approach.

```
if ( s1[i] == s2[j] ) { i++; j++; }
else { i++; }
return ( j == s2.length() )
```

Q Check for Anagram

i/p → s1 = "listen" s2 = "silent"
o/p → Yes

Naive → Sort both and compare.

In Java sort →
```
char a[] = x.toCharArray();
Arrays.sort[a];
s1 = new String(a);
```

Efficent → Counting freq like in first question.

count ++ for letter in str1
count -- for letter in str2 (same)
} At end count array for all char must be 0.

Q. Reverse words in a String

I/p → "welcome to gfg"

O/p → "gfg to welcome"

I/p → "abc"

O/p → "abc"

Naive → Use Stack. , strings seperated by space.

(Auxillary space)

Efficient → Reverse individual words

(Constant space) & then reverse the whole string.

1) → abc bda

2) → cba adb

3) → bda abc