charAt

```
public char charAt(int index)
```

Returns the character at the specified index. An index ranges from `0` to `length() - 1`. The first character of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Specified by:
`charAt` in interface `CharSequence`

Parameters:
`index` - the index of the character.

Returns:
the character at the specified index of this string. The first character is at index `0`.

Throws:
`IndexOutOfBoundsException` - if the `index` argument is negative or not less than the length of this string.

---

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position `k` in the two string -- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length()-anotherString.length()
```

Parameters:
`anotherString` - the `String` to be compared.

Returns:
the value `0` if the argument string is equal to this string; a value less than `0` if this string is lexicographically less than the string argument; and a value greater than `0` if this string is lexicographically greater than the string argument.

equals

```
public boolean equals(Object anObject)
```

Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.

Overrides:
`equals` in class `Object`

Parameters:
`anObject` - the object to compare this `String` against.

Returns:
`true` if the `String` are equal; `false` otherwise

---

indexOf

```
public int indexOf(String str)
```

Returns the index within this string of the first occurrence of the specified substring. The integer returned is the smallest value *k* such that:

```
    this.startsWith(str, k)
```

is true.

Parameters:
`str` - any string.

Returns:
if the string argument occurs as a substring within this object, then the index of the first character of the first such substring is returned; if it does not occur as a substring, `-1` is returned.

---

indexOf

```
public int indexOf(String str,
                   int fromIndex)
```

Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. The integer returned is the smallest value `k` for which:

```
    k >= Math.min(fromIndex, str.length()) && this.startsWith(str, k)
```

If no such value of *k* exists, then -1 is returned.

Parameters:
`str` - the substring for which to search.
`fromIndex` - the index from which to start the search.

Returns:
the index within this string of the first occurrence of the specified substring, starting at the specified index.

substring

```
public String substring(int beginIndex)
```

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

```
"unhappy".substring(2) returns "happy"
"Harbison".substring(3) returns "bison"
"emptiness".substring(9) returns "" (an empty string)
```

Parameters:

`beginIndex` - the beginning index, inclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if `beginIndex` is negative or larger than the length of this `String` object.

---

substring

```
public String substring(int beginIndex,
                        int endIndex)
```

Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is:

`endIndex-beginIndex`.

Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

Parameters:

`beginIndex` - the beginning index, inclusive.
`endIndex` - the ending index, exclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.