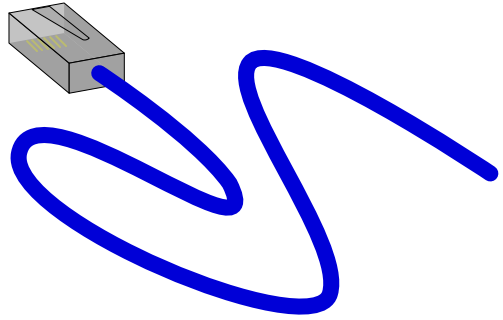
Interface examples

Ethernet port

Interface examples

Ethernet port

Ethernet jack

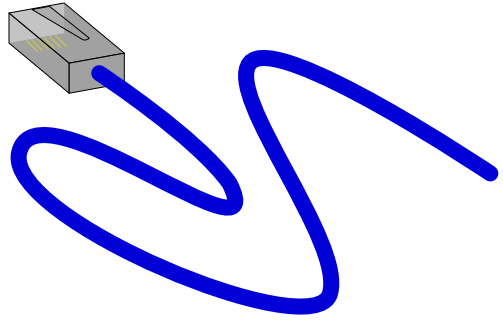


Interface examples

Ethernet port

Interfaces
- 8P8C
- CAT7

Ethernet jack

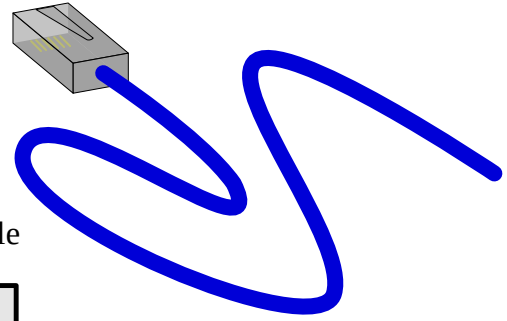


Interface examples

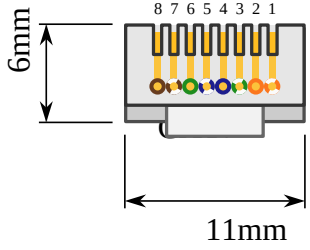
Ethernet port

Interfaces
•8P8C
•CAT7

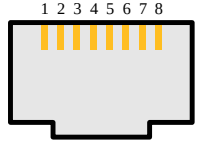
Ethernet jack



Male



Female



8P8C - T568B wiring

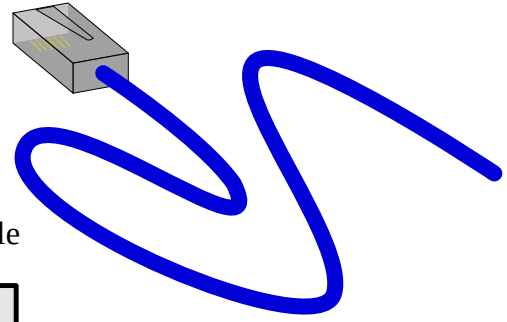
Interface examples

Ethernet port

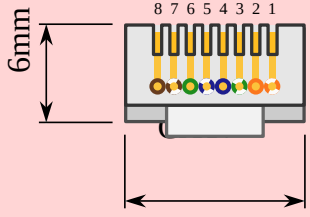
Service Provider

Interfaces
•8P8C
•CAT7

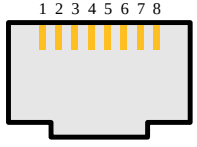
Ethernet jack



Male

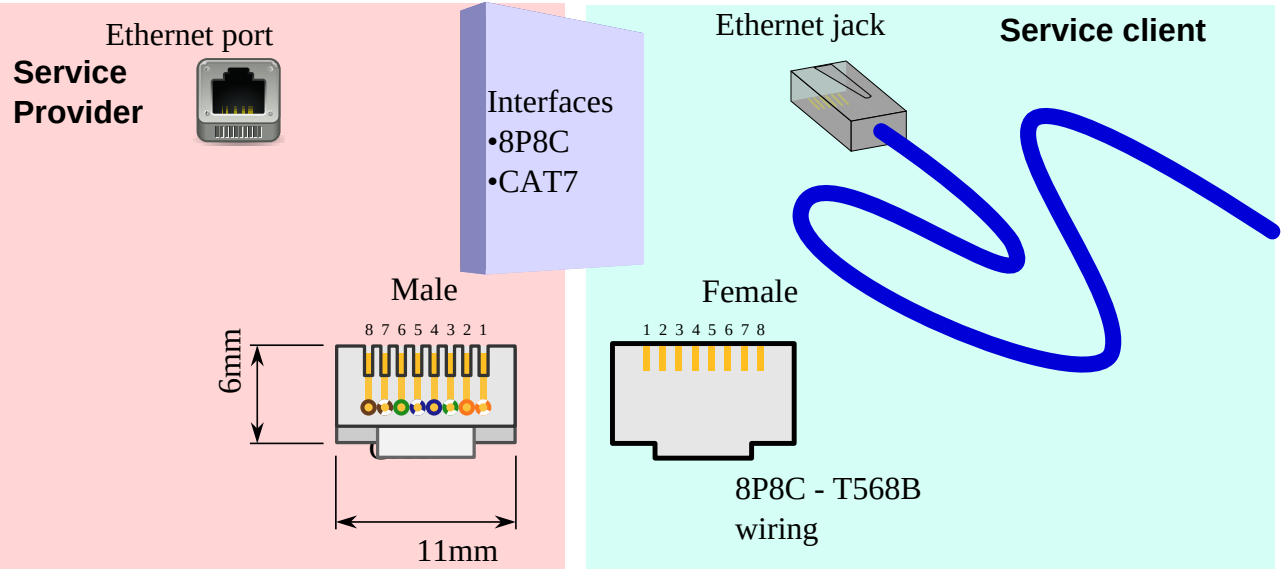


Female



8P8C - T568B wiring

Interface examples



Observations

Multiple standards involved:

8P8C	Mechanical dimensions and tolerances.
CAT7	Telecommunication performance of twisted-pair copper interconnects.

Note: Compatible hardware must obey **both** standards.

Writing strings to file

```
public class Text2File {
  private final PrintStream out;

  public Text2File(final String fileName)
     throws FileNotFoundException {
    out = new PrintStream(new File(fileName));}

  public void println(final String s) {
    out.println(s);}

  public void closeFile() {
    out.close();}
}
```

Using `Text2File`

```
final String outputFileName =
            "output.txt";
try {
  final Text2File output =
  new Text2File(outputFileName);
  output.println("Some dumb text");
  output.println("More dumb text");
  output.closeFile();
} catch (final FileNotFoundException e){
  System.err.println("Unable to open '"
   + outputFileName + "' for writing");
}
```

File `output.txt`:

```
Some dumb text
More dumb text
```

Possible `Text2File` errors:

- Missing `output.closeFile()` call.

  Some text portion may not be flushed to disk.

- Calling `output.println(...)` after `output.closeFile()`:

```
output.closeFile();
output.println("Too late!");
```

  Last call will be silently ignored.

Employ “try-with-resources”

```
final String outputFileName =
     "output.txt";

try (final Text2File output =
     new Text2File(outputFileName)){
  output.println("Some dumb text");
  output.println("More dumb text");
} catch (FileNotFoundException e){...}
```

Compile time error:

```
Required:
   java.lang.AutoCloseable
Found:
  de.hdm_stuttgart.mi.sd1.Text2File
```

`interface` syntax

```
accessModifier interface interfaceName [throwsClause]?{
    [field]*
   [method]*
}
```

The AutoCloseable promise

```
package java.lang; ❶

public interface AutoCloseable {

/**
 * Closes this resource,
 * relinquishing any
 * underlying resources.
 */
void close() ❷;

}
```

```
public class Text2File
  implements AutoCloseable ❶{

  private ❷ PrintStream out;
...
  public void println(final String s){
    out.println(s); }

  public void close() ❸{
    out.close(); ❹
    out = null; ❺
  }
}
```

`abstract` class replacement

```
package hdm.project; ❶

abstract public
    class AutoCloseable ❷{

/**
 * Closes this resource,
 * relinquishing any
 * underlying resources.
 */
abstract void close();❸

}
```

```
public class Text2File
  extends AutoCloseable ❶{

  private  PrintStream out;
...
  public void println(final String s){
    out.println(s); }

  @Override public void close() ❷{
    out.close();
    out = null;
  }
}
```

`interface` vs. `abstract` class

- Java™ disallows multiple inheritance.
- A class may implement an arbitrary number of interfaces:

```
public class X implements I1, I2, I3 {...}
```

interface MyAutoCloseable

```
/**
 * Support auto-closing of resources
 */
public interface MyAutoCloseable {
  /**
   * close resource in question. Example: Terminate
   * a database connection or a file stream.
   */
  public void close();
}
```

Extending `MyAutoCloseable` to flush

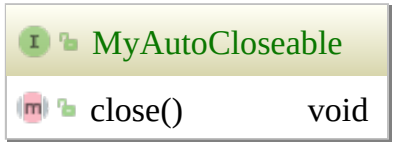
```
/**
 * Flush pending values.
 */
public interface MyFlushable extends MyAutoCloseable {

  /**
   * Save pending i.e. buffered values.
   */
  public void flush();
}
```

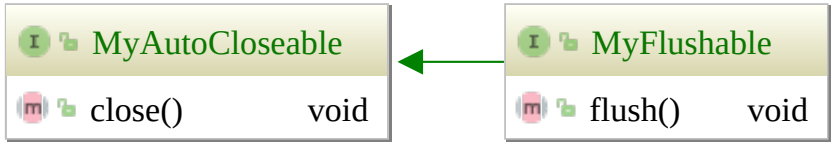
Using MyFlushable

```
public class Text2FileFlushable implements MyFlushable {
  private PrintStream out;
...
  /**
   * Flushing pending output to underlying file.
   */
  public void flush(){
    out.flush();
  }
  /**
   * Closing file thereby flushing buffer. Caution: Further calls
   * to {@link #println(String)} will fail!.
   */
  public void close() {
    out.close();
    out = null;
  }
}
```

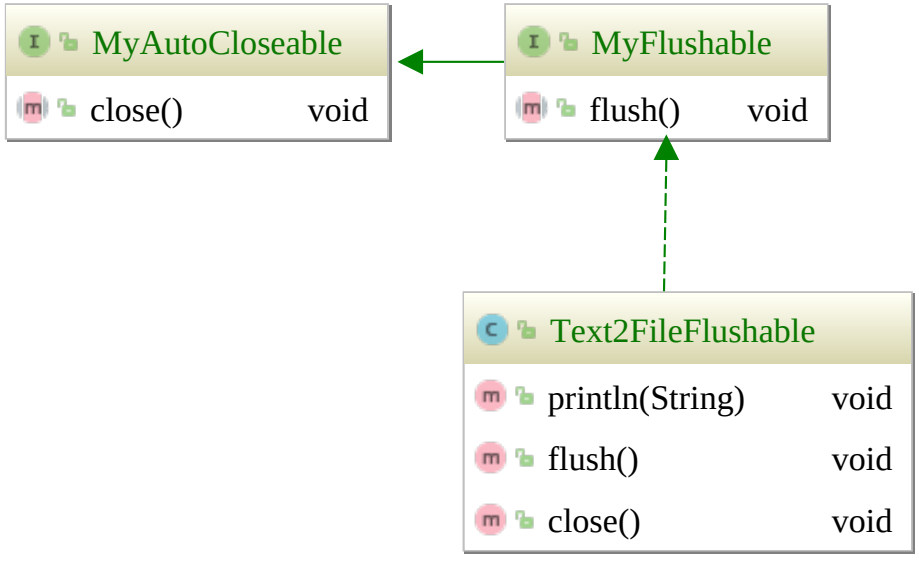
Inheritance hierarchy



Inheritance hierarchy



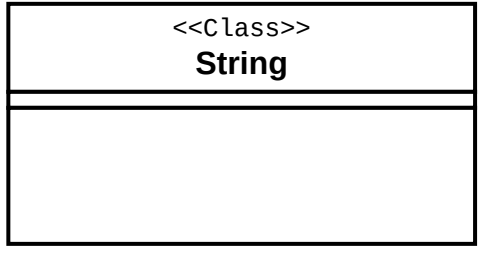
Inheritance hierarchy



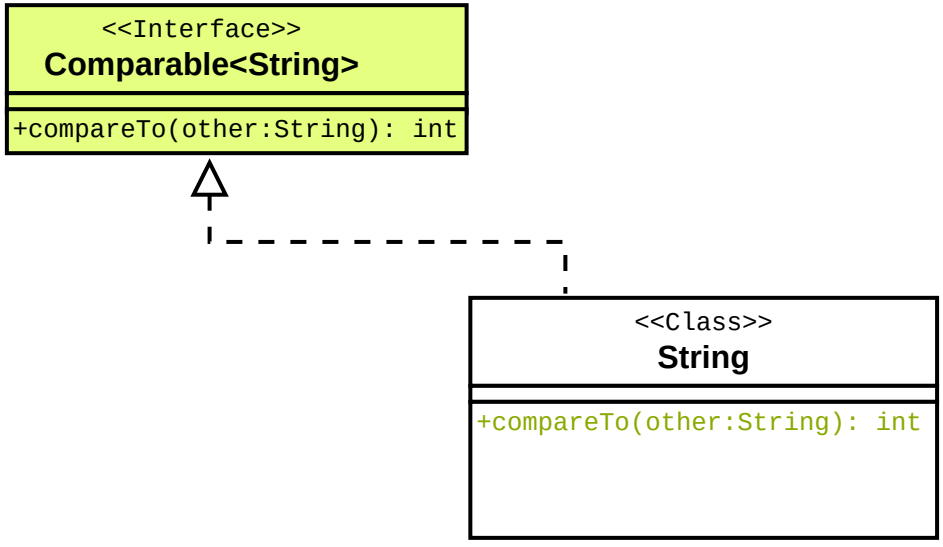
Upcoming topics

- Default methods.
- Base classes.

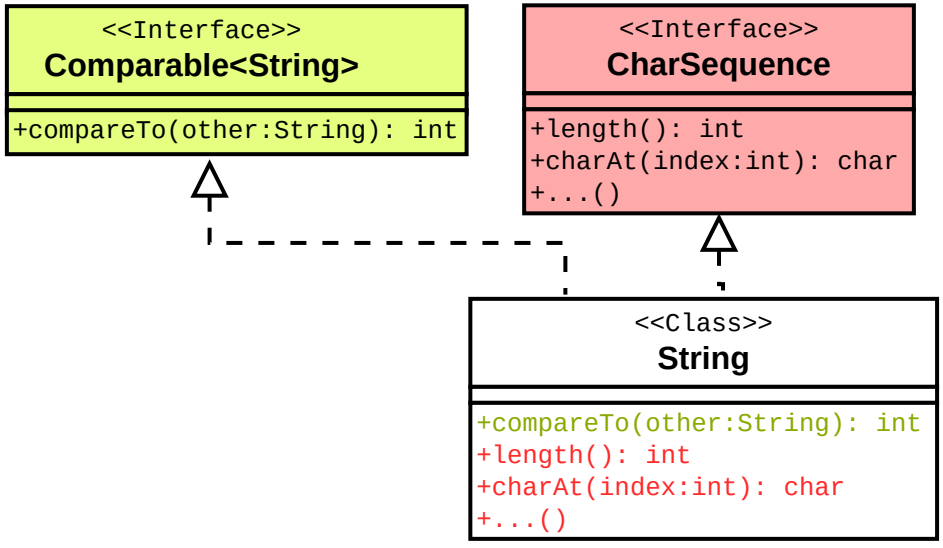
Interfaces implemented by class `String`



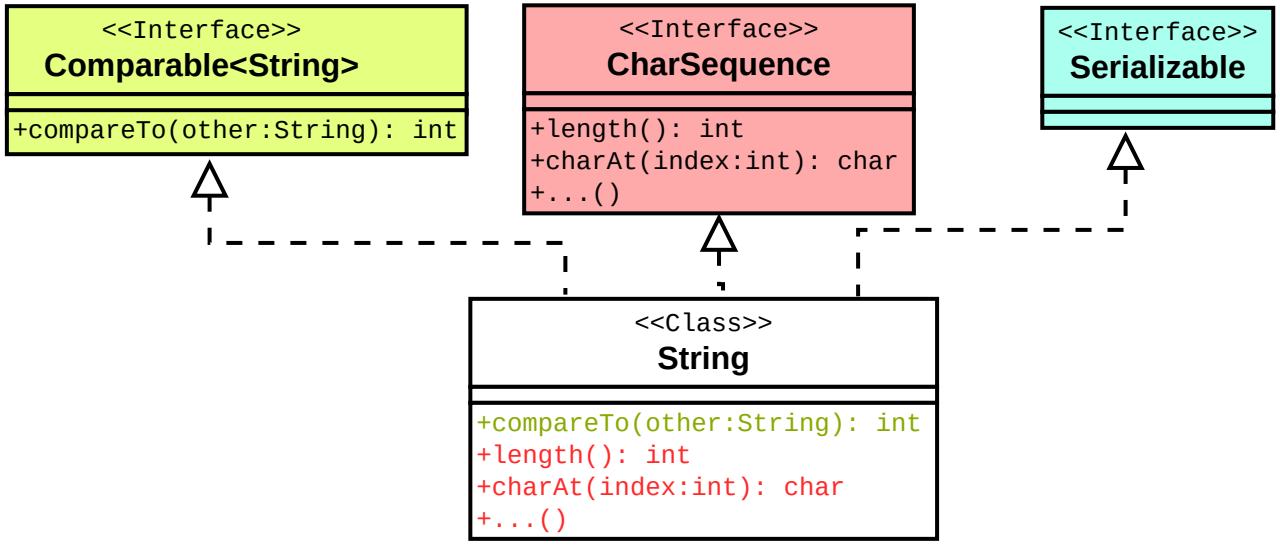
Interfaces implemented by class `String`



Interfaces implemented by class `String`



Interfaces implemented by class `String`



The `Comparable` interface

```
interface Comparable<T> {
                    ┌──┘
                    ▼
  int compareTo(T o);

}
```

class `String` and `Comparable`

```
public class String implements Comparable <String ❶>, ... {
  ...
  @Override
  public int compareTo(final String ❷ other) {
     ...
    return ...;
  }
}
```

Comparison examples

```
System.out.println("Eve".compareTo("Paul"));      ❶     -11 ❶
System.out.println("Victor".compareTo("Andrew")); ❷     21  ❷
System.out.println("Hannah".compareTo("Hannah")); ❸     0   ❸
```

Ascending and descending names

"Aaaron"

Ascending and descending names

"Aaaron"

"Bernie"

Ascending and descending names

"Aaaron"

"Bernie"

"Eve"

Ascending and descending names

"Aaaron"

"Bernie"

"Eve"

"Laura"

Ascending and descending names

"Aaaron"

"Bernie"

"Eve"

"Laura"

"Peter"

Ascending and descending names

"Aaaron"

"Bernie"

"Eve"

"Laura"

"Peter"

"Tim"

Ascending and descending names

"Aaaron"

"Bernie"

"Eve"

"Laura"

"Peter"

"Tim"

"Tim"

"Peter"

"Laura"

"Eve"

"Bernie"

"Aaaron"

Ascending and descending names

"Aaaron"
.compareTo() → -1

"Bernie"

"Eve"

"Laura"

"Peter"

"Tim"

"Tim"

"Peter"

"Laura"

"Eve"

"Bernie"
.compareTo() → 1

"Aaaron"

Ascending and descending names

"Aaaron"
.compareTo() → -1

"Bernie"
.compareTo() → -3

"Eve"

"Laura"

"Peter"

"Tim"

"Tim"

"Peter"

"Laura"

"Eve"
.compareTo() → 3

"Bernie"
.compareTo() → 1

"Aaaron"

Ascending and descending names

"Aaaron"
.compareTo() → -1

"Bernie"
.compareTo() → -3

"Eve"
.compareTo() → -7

"Laura"

"Peter"

"Tim"

"Tim"

"Peter"

"Laura"
.compareTo() → 7

"Eve"
.compareTo() → 3

"Bernie"
.compareTo() → 1

"Aaaron"

Ascending and descending names

"Aaaron"
.compareTo() → -1

"Bernie"
.compareTo() → -3

"Eve"
.compareTo() → -7

"Laura"
.compareTo() → -4

"Peter"

"Tim"

"Tim"

"Peter"
.compareTo() → 4

"Laura"
.compareTo() → 7

"Eve"
.compareTo() → 3

"Bernie"
.compareTo() → 1

"Aaaron"

Ascending and descending names

"Aaaron"
.compareTo() → -1

"Bernie"
.compareTo() → -3

"Eve"
.compareTo() → -7

"Laura"
.compareTo() → -4

"Peter"
.compareTo() → -4

"Tim"

"Tim"
.compareTo() → 4

"Peter"
.compareTo() → 4

"Laura"
.compareTo() → 7

"Eve"
.compareTo() → 3

"Bernie"
.compareTo() → 1

"Aaaron"

API requirements

1. Antisymmetric: sgn(x.compareTo(y)) == -sgn(y.compareTo(x))
2. Transitive: `x.compareTo(y)` > 0 and `y.compareTo(z) > 0` $\Rightarrow$ `x.compareTo(z) > 0`.
3. `x.compareTo(y)==0` $\Rightarrow$ that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all z.
4. Recommendation: `(x.compareTo(y)==0) == (x.equals(y))`

Sorting strings alphabetically

```
final String[] names = { ❶
  "Laura", "Aaron", "Tim", "Peter", "Eve", "Bernie"
};

Arrays.sort(names); ❷

for (final String n: names) { ❸
  System.out.println(n);
}
```

```
Aaron
Bernie
Eve
Laura
Peter
Tim
```

Related exercises

Exercise 179: Understanding `Arrays.sort()`
Exercise 180: Sorting `Rectangle` instances by width
Exercise 181: Sorting `Rectangle` instances by width and height

Situation dependent sorting criteria

Unsorted	Case sensitive	Case insensitive	Descending
UK	ATM	ATM	sign
quick	UK	hello	quick
hello	hello	quick	hello
sign	quick	sign	UK
ATM	sign	UK	ATM

Implementing flexible sorting

Solution: Provide your own `Comparator`!

```
import java.util.Comparator;

public class SortCaseInsensitive implements Comparator<String> {

  @Override
  public int compare(final String a, final String b) {
    return a.toLowerCase().compareTo(b.toLowerCase());
  }
}
```

`Comparator` in action

```
System.out.println("hello".compareTo("UK")); ❶

System.out.println(new SortCaseInsensitive(). ❷
   compare("hello", "UK"));
```

```
19 ❶
-13 ❷
```

Case insensitive sort

```
final String[] names = {
"UK", "quick", "hello", "sign", "ATM"
};

Arrays.sort(names, new SortCaseInsensitive());

for (final String n: names) {
  System.out.println(n);
}
```

```
ATM
hello
quick
sign
UK
```

Sort descending by lambda expression

```
final String[] names = {
"UK", "quick", "hello", "sign", "ATM"
};
Arrays.sort(names, (a, b) -> b.compareTo(a)); ❶

for (final String n: names) {
  System.out.println(n);
}
```

```
sign
quick
hello
UK
ATM
```

Related exercises

Exercise 182: Adding flexibility in sorting rectangles
Exercise 183: A nonsense generator
Exercise 184: An interface based plotter
Exercise 185: Various integer array algorithms
Exercise 186: A command line version computing a sample's average and median
Exercise 187: Adding line numbers to text files
Exercise 188: A partial implementation of GNU UNIX **wc**