

Interface examples

Ethernet port

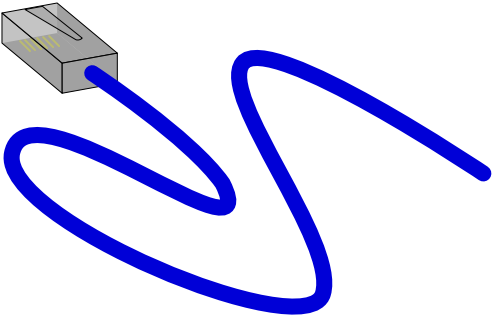


Interface examples

Ethernet port



Ethernet jack



Interface examples

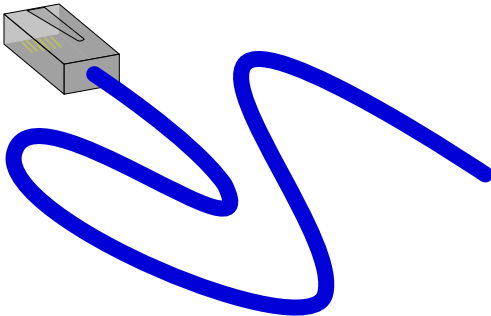
Ethernet port



Interfaces

- 8P8C
- CAT7

Ethernet jack



Interface examples

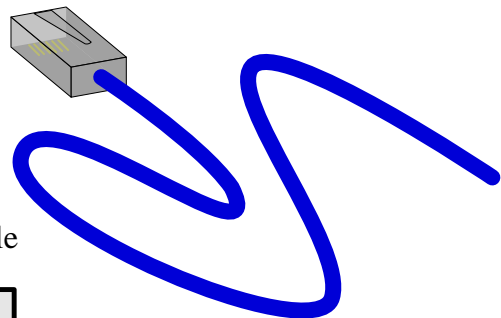
Ethernet port



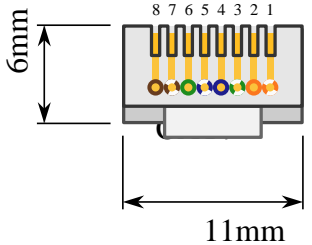
Interfaces

- 8P8C
- CAT7

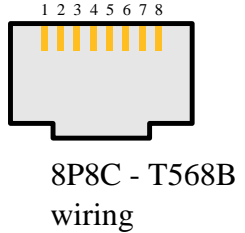
Ethernet jack



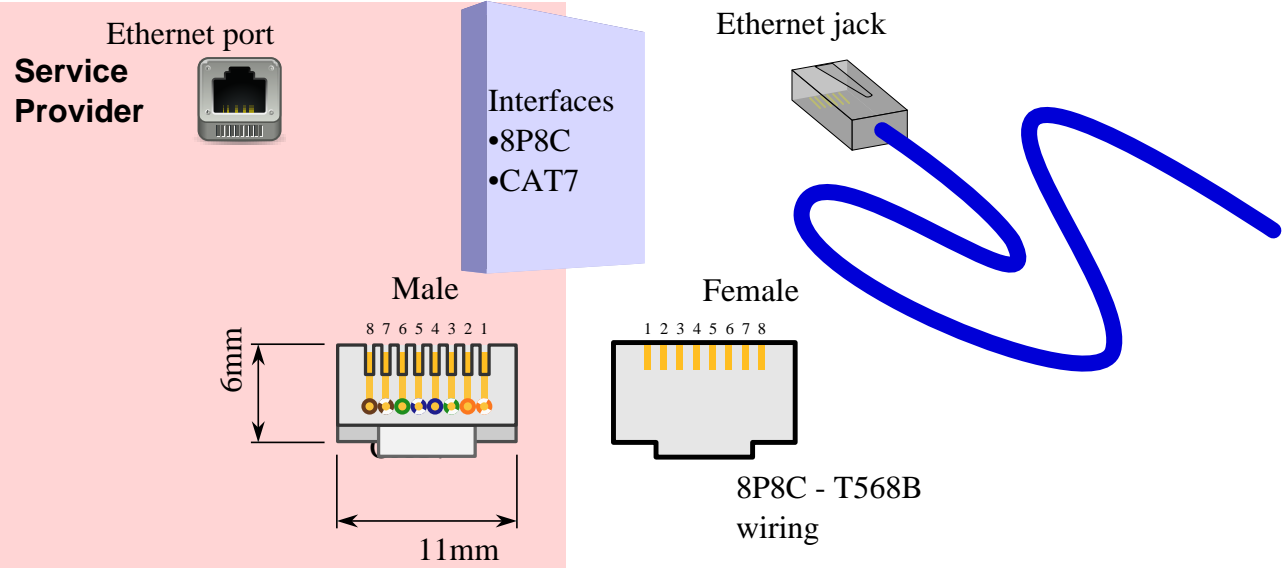
Male



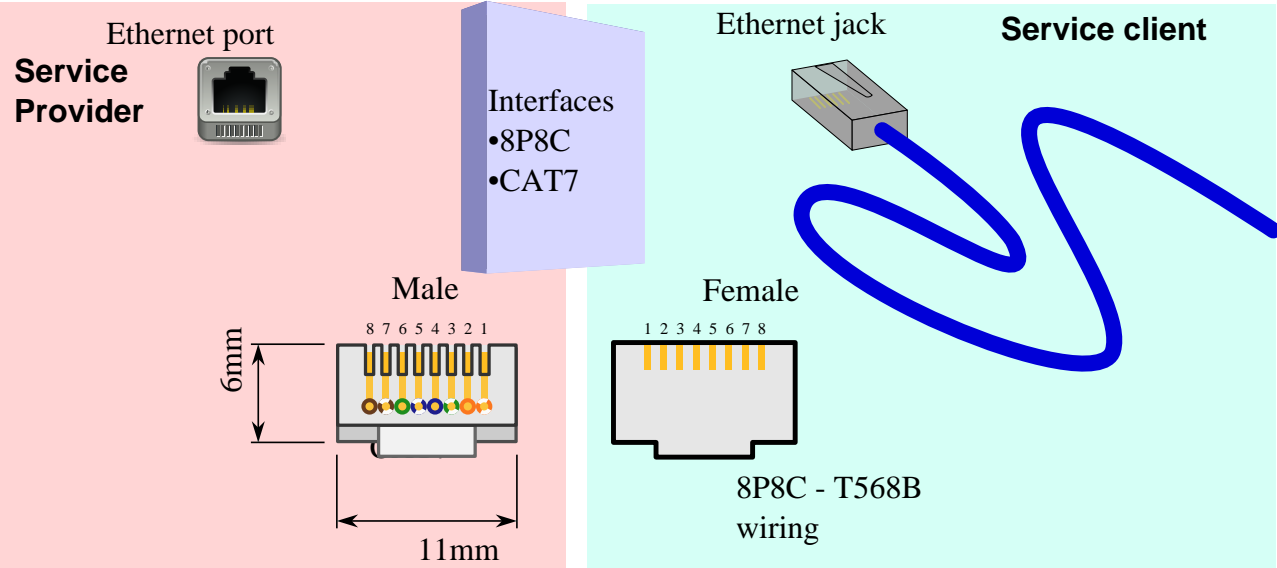
Female



Interface examples



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 Text 2File

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

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

File output.txt:

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

Possible Text File 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 outputFile =  
    "output.txt";  
  
try (final Text2File output =  
    new Text2File(outputFile)) {  
    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();
}
```

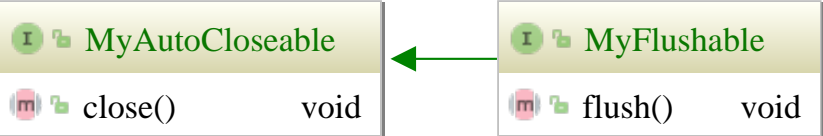
Using MyFileFlushable

```
public class Text2FileFlushable implements MyFileFlushable {
    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;
    }
}
```

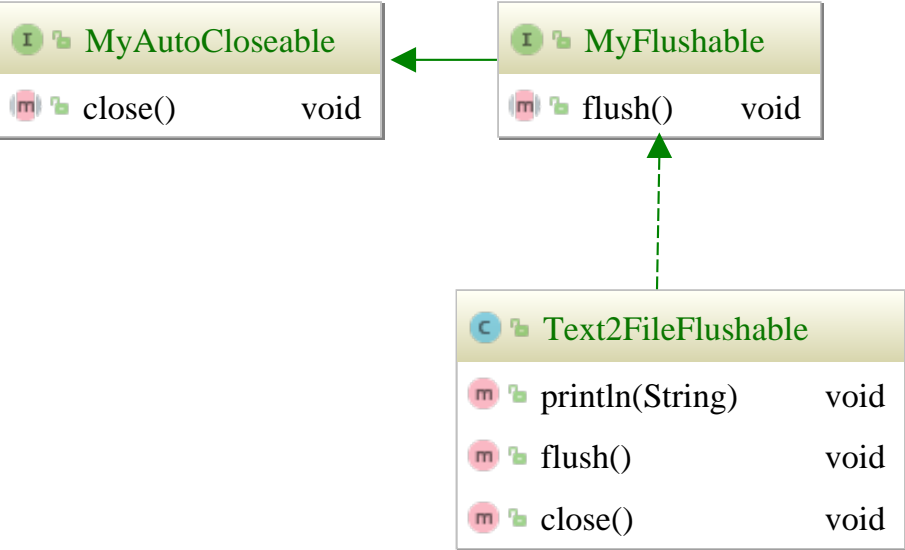
Inheritance hierarchy

 MyAutoCloseable
 close() void

Inheritance hierarchy



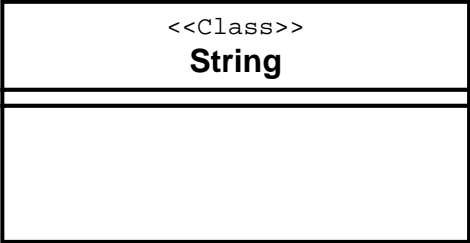
Inheritance hierarchy



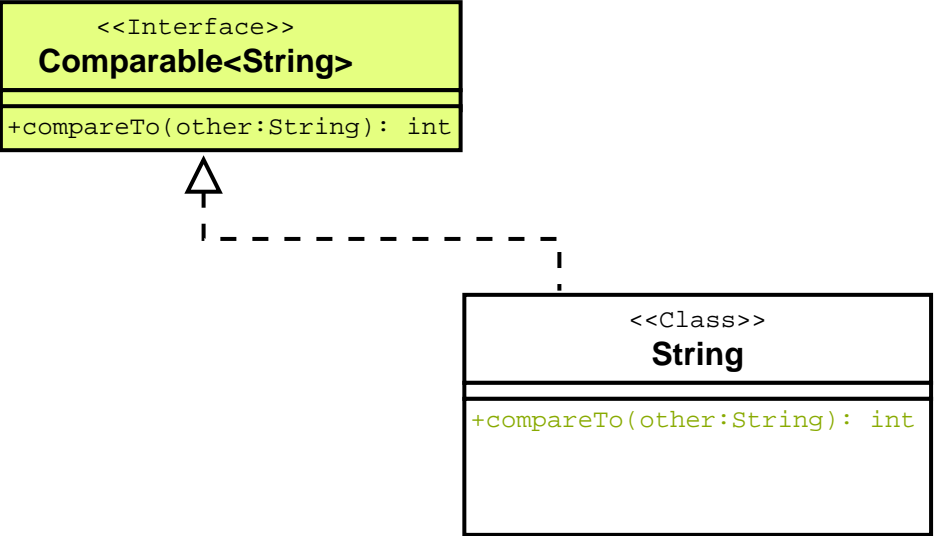
Upcoming topics

- Default methods.
- Base classes.

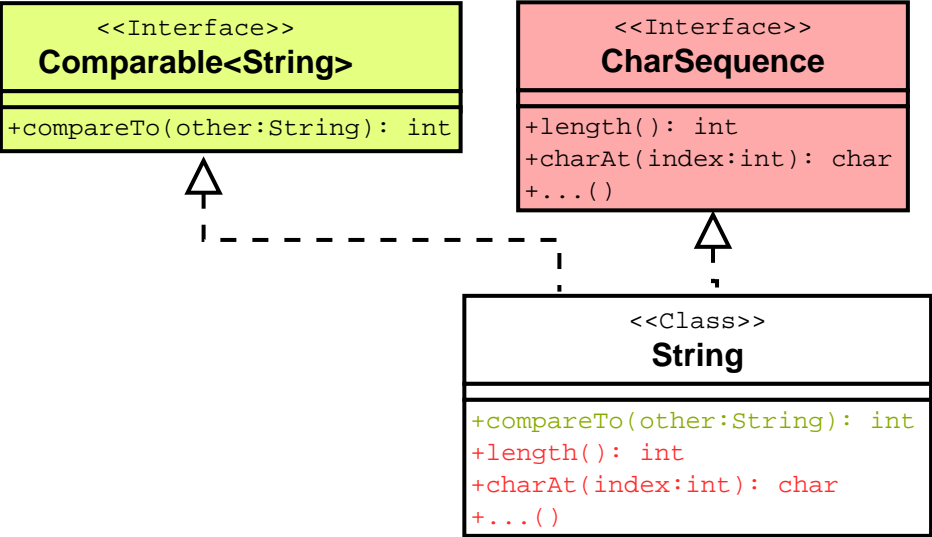
Interfaces implemented by class String



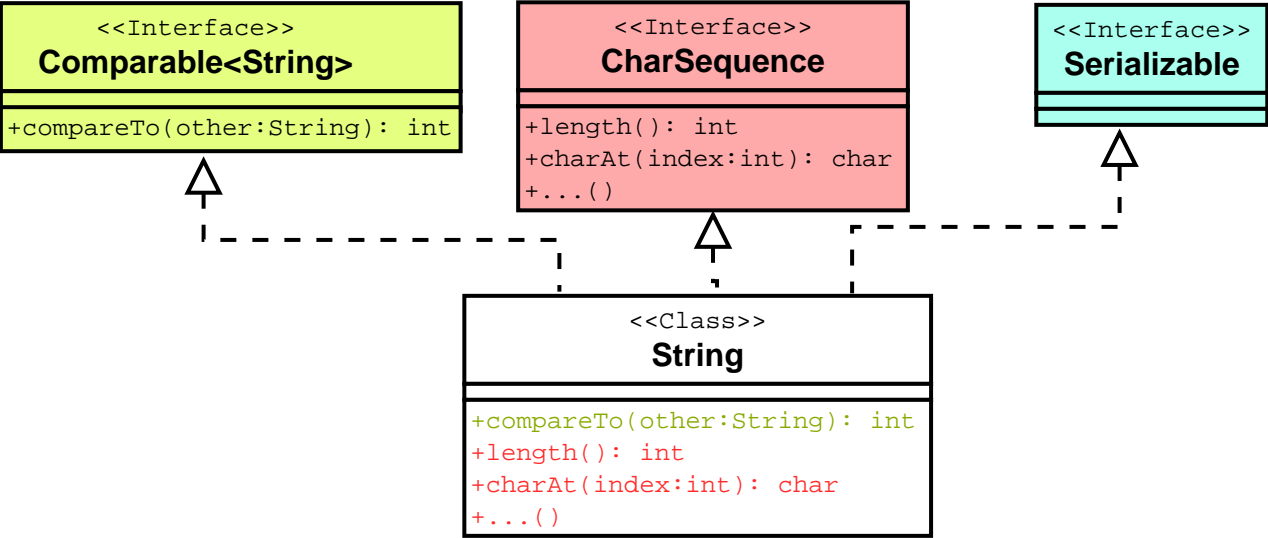
Interfaces implemented by class String



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

"Aaron"

Ascending and descending names

"Aaron"

"Bernie"

Ascending and descending names

"Aaron"

"Bernie"

"Eve"

Ascending and descending names

"Aaron"

"Bernie"

"Eve"

"Laura"

Ascending and descending names

"Aaron"

"Bernie"

"Eve"

"Laura"

"Peter"

Ascending and descending names

"Aaron"

"Bernie"

"Eve"

"Laura"

"Peter"

"Tim"

Ascending and descending names

"Aaron"

"Tim"

"Bernie"

"Peter"

"Eve"

"Laura"

"Laura"

"Eve"

"Peter"

"Bernie"

"Tim"

"Aaron"

Ascending and descending names

"Aaron"
 .compareTo() → -1

"Bernie"

"Eve"

"Laura"

"Peter"

"Tim"

"Tim"

"Peter"

"Laura"

"Eve"

"Bernie"
 .compareTo() → 1

"Aaron"

Ascending and descending names

"Aaron"
 .compareTo() → -1

"Bernie"
 .compareTo() → -3

"Eve"

"Laura"

"Peter"

"Tim"

"Tim"

"Peter"

"Laura"

"Eve"
 .compareTo() → 3

"Bernie"
 .compareTo() → 1

"Aaron"

Ascending and descending names

"Aaron"
 .compareTo() → -1

"Bernie"
 .compareTo() → -3

"Eve"
 .compareTo() → -7

"Laura"

"Peter"

"Tim"

"Tim"

"Peter"

"Laura"
 .compareTo() → 7

"Eve"
 .compareTo() → 3

"Bernie"
 .compareTo() → 1

"Aaron"

Ascending and descending names

"Aaron"
 .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

"Aaron"

Ascending and descending names

"Aaron"
 .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

"Aaron"

API requirements

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

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 178: Understanding Arrays. `sort()`

Exercise 179: Sorting `Rectangle` instances by width

Exercise 180: Sorting `Rectangle` instances by width and height

Situation dependent sorting criteria

Unsorted

UK
qui ck
hel l o
si gn
ATM

Case sensitive

ATM
UK
hel l o
qui ck
si gn

Case insensitive

ATM
hel l o
qui ck
si gn
UK

Descending

si gn
qui ck
hel l o
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
```

19 ❶

-13 ❷

```
System.out.println(new SortCaseInsensitive().  
    compare("hello", "UK")); 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 181: Adding flexibility in sorting rectangles

Exercise 182: A nonsense generator

Exercise 183: An interface based plotter

Exercise 184: Various integer array algorithms

Exercise 185: A command line version computing a sample's average and median

Exercise 186: Adding line numbers to text files

Exercise 187: A partial implementation of GNU UNIX **wc**