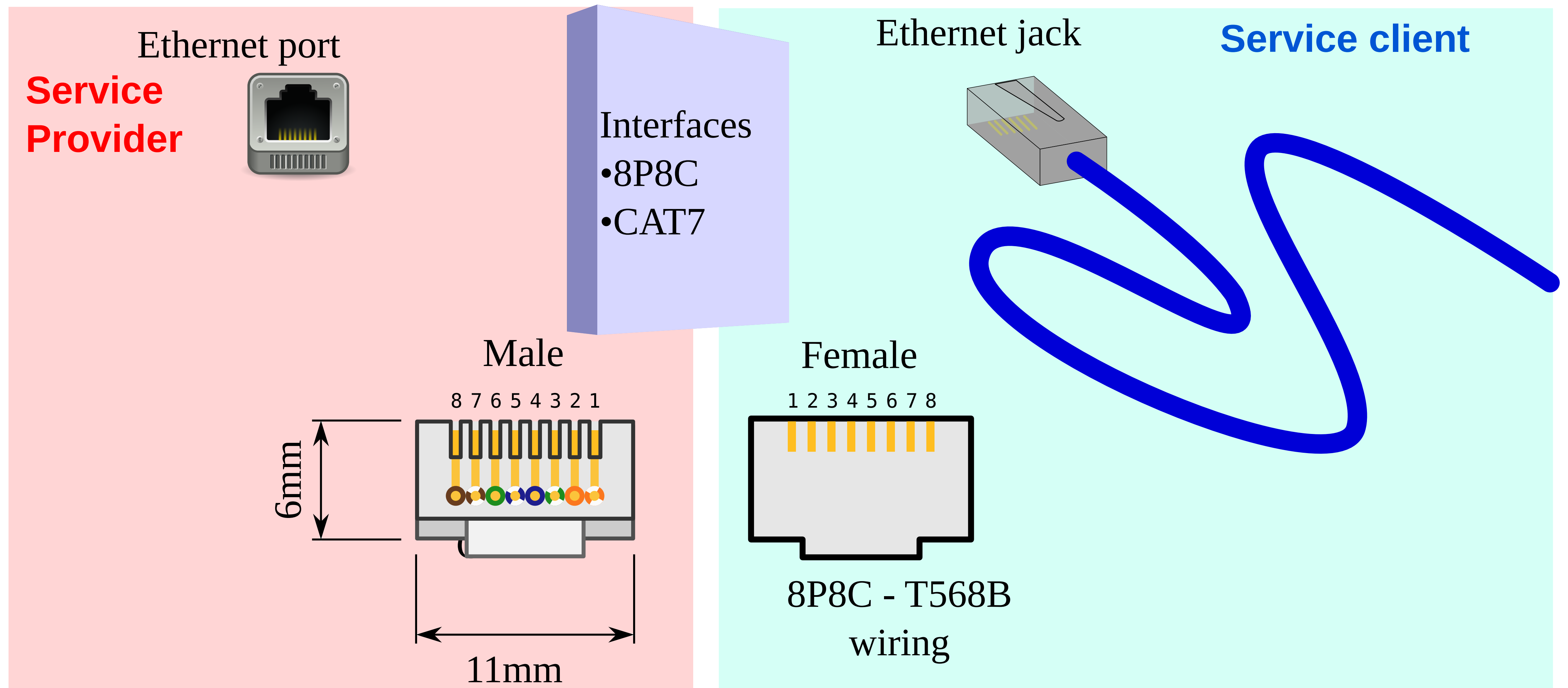


interface definitions and abstract Classes

Interface examples



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

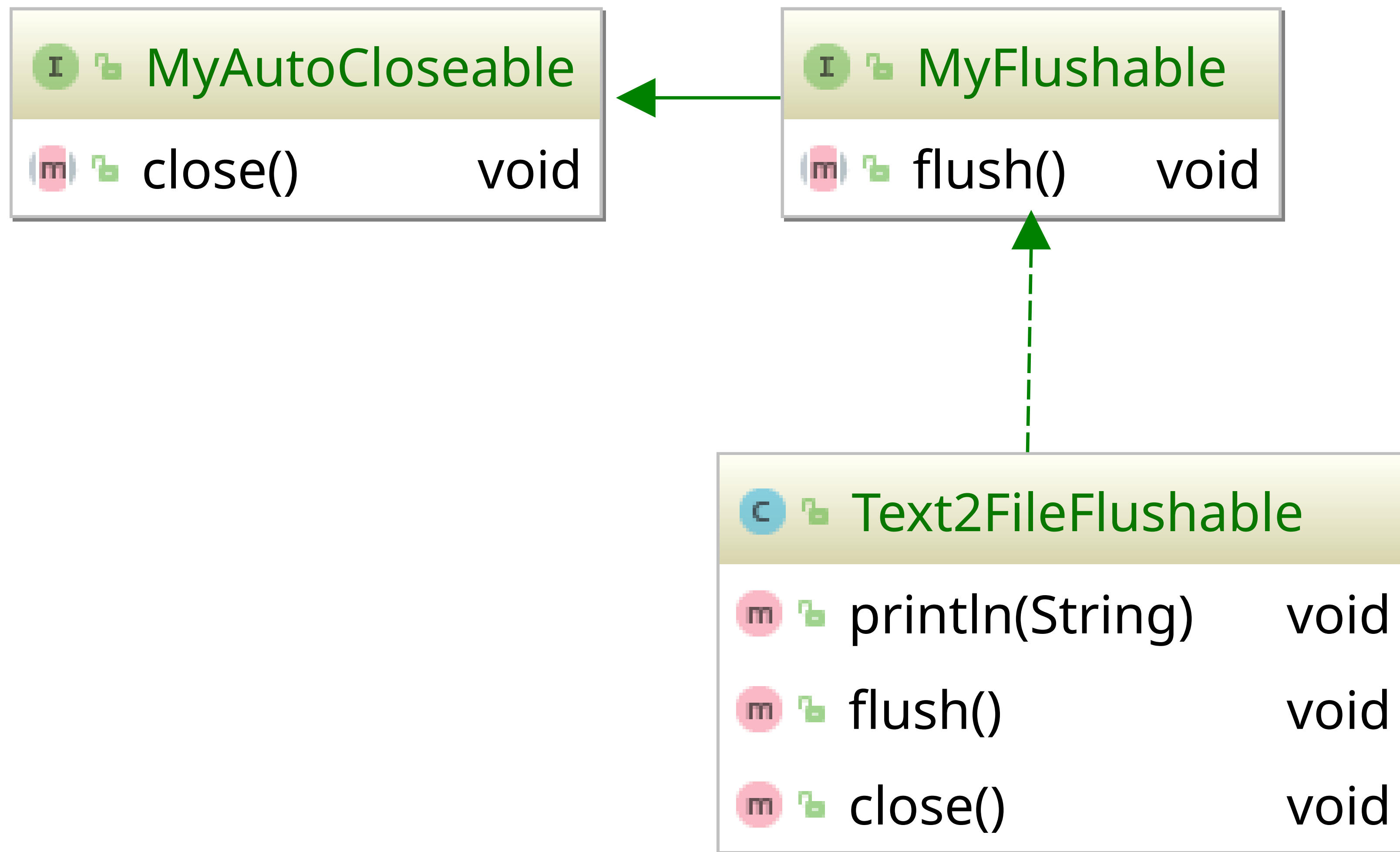
```
/**
 * Flush pending values.
 */
public interface MyFlushable extends MyAutoCloseable {

    /**
     * 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

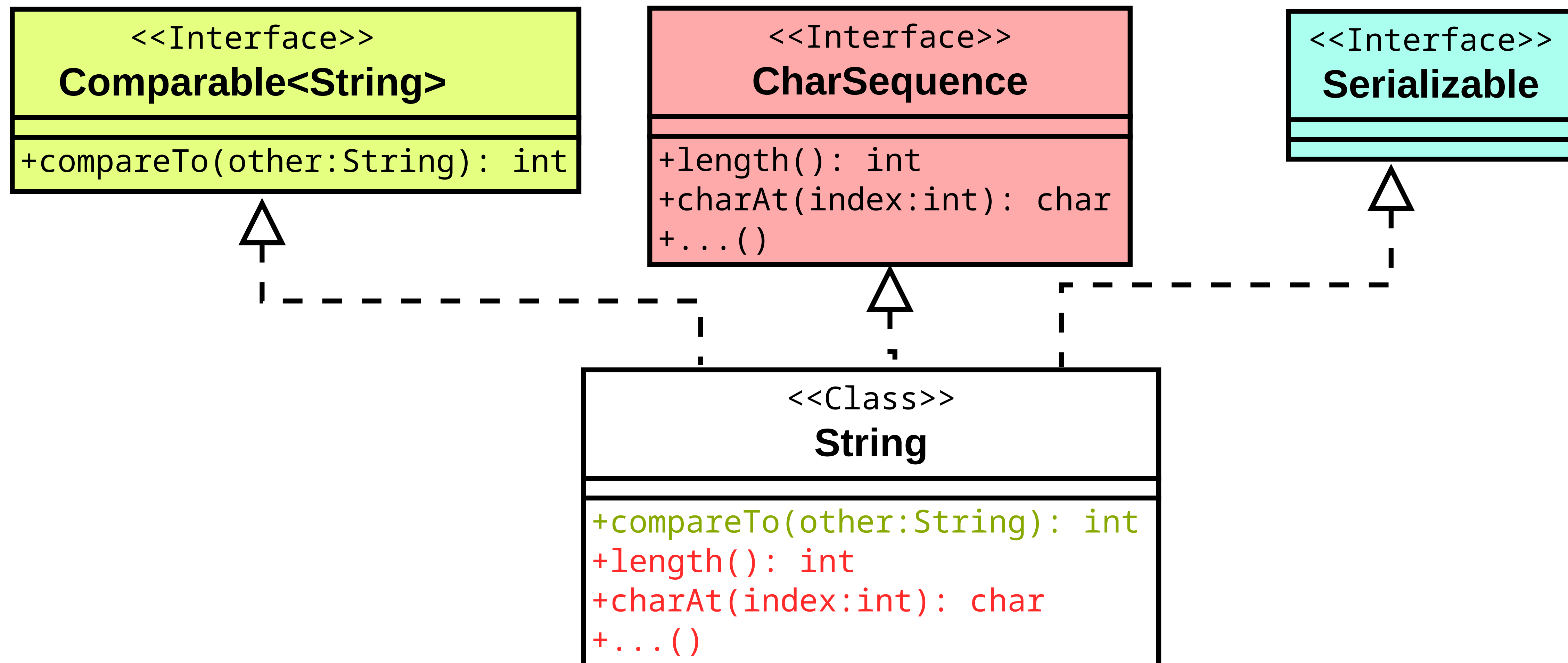


Upcoming topics

- Default methods.
- Base classes.

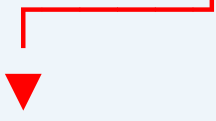
interface definitions and abstract Classes
 ⇒ Interfaces and sorting

Interfaces implemented by class **String**



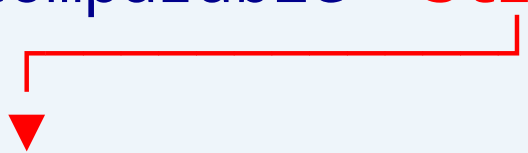
The Comparable interface

```
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"));  
System.out.println("Victor".compareTo("Andrew"));  
System.out.println("Hannah".compareTo("Hannah"));
```

1

2

3

-11
21
0

1

2

3

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. $x.\text{compareTo}(y) == 0 \Rightarrow$ 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
```

Followup exercises

- 186. Understanding Arrays . sort ()
- 187. Sorting Rectangle instances by width
- 188. Sorting Rectangle instances by width and height

Situation dependent sorting criteria

Unsorted

UK
quick
hello
sign
ATM

Case sensitive

ATM
UK
hello
quick
sign

Case insensitive

ATM
hello
quick
sign
UK

Descending

sign
quick
hello
UK
ATM

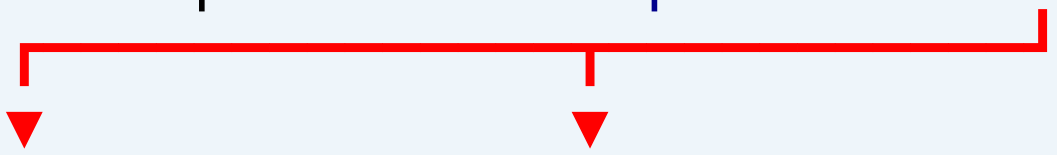
Implementing flexible sorting

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());
    }
}
```



The diagram illustrates the relationship between the `Comparator` interface and the `String` parameters in the `compare` method. A red line connects the `String` type in the `Comparator<String>` declaration to the `String` parameters `a` and `b` in the `compare` method signature. Two red arrows point from the line to the `a` and `b` parameters, indicating that they are of type `String`.

Comparator in action

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

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

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
```

Followup exercises

- 189. Adding flexibility in sorting rectangles
- 190. A nonsense generator
- 191. An interface based plotter
- 192. Various integer array algorithms
- 193. A command line version computing a sample's average and median
- 194. Adding line numbers to text files
- 195. A partial implementation of GNU UNIX **wc**