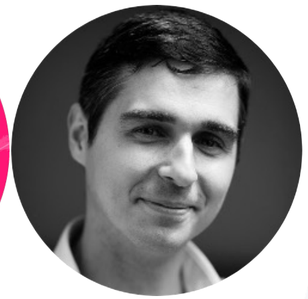
Transforming Strings



Andrejs Doronins
Software Developer in Test

Overview

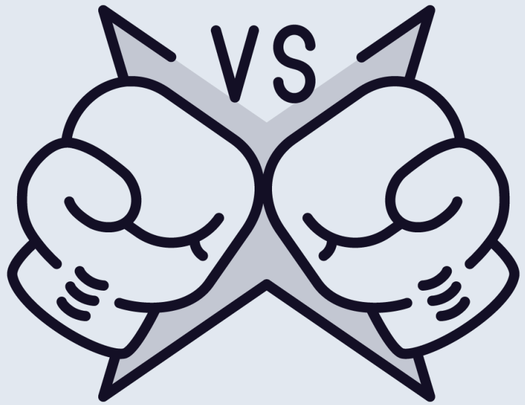
- Single string
- Compare strings
- Iterating over strings
- Tokenizing (breaking up) strings
- Putting strings back together
- Demo program

		String
int	⟷	Integer
double	⟷	Double
boolean	⟷	Boolean

// etc.

```
class String {

    String substring(...);

    String toUpperCase();

    String startsWith(...);
}
```

```
class String {

    boolean isEmpty();
    boolean isBlank();

    String trim();
    String strip();
}
```

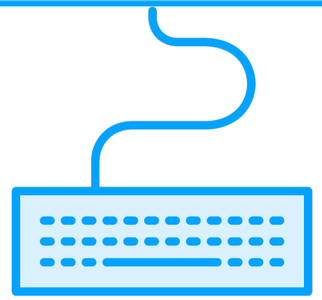


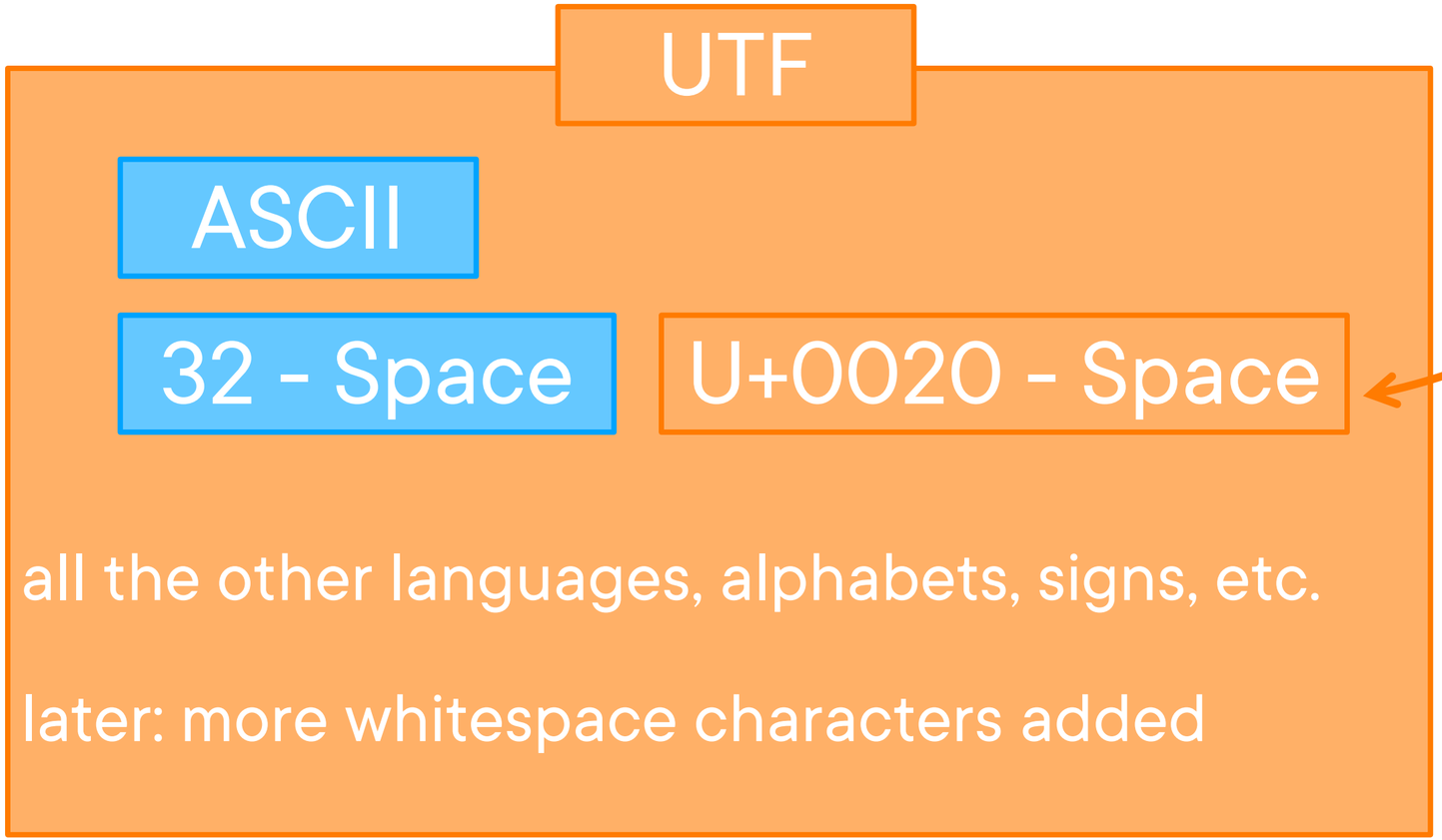
ASCII

0-31 : control chars (tab, line feed, space, etc.)

32-126: punctuation, numbers, letters

127: delete






```
/**
 * whitespace as any
character
 * whose codepoint is less
 * than or equal to 'U+0020'
 */
  trim();

 (removes \s, \t, \r...)
```

```
class String {

   String trim();

   String strip();
}
```

Change the implementation and break half of the world…

Unicode-aware evolution of trim()

```
class String {

    boolean isEmpty(); // “”.length == 0
    // “   ” <- not “empty”
}
```

```
String result  = someStr
                   .toLowerCase()
                   .replaceAll("[a-z]", "")
                   .strip();
```

compares references in memory

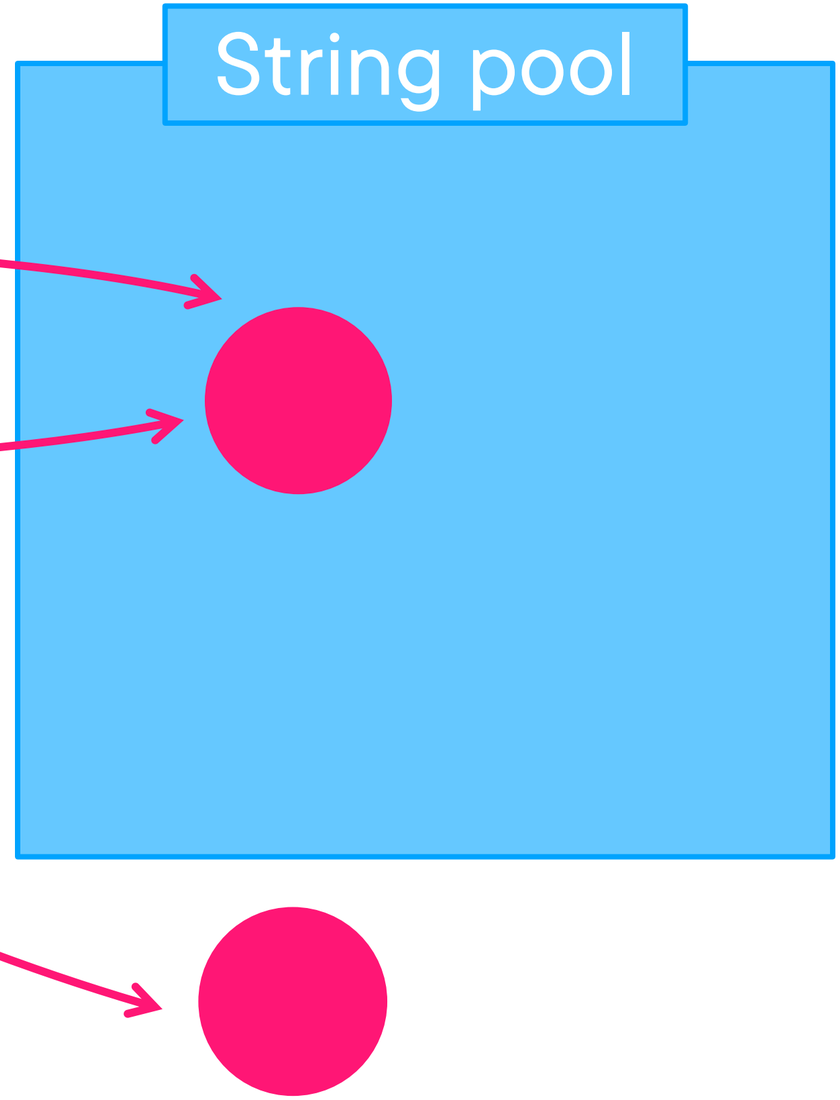
```
“str1” == “str2”
```

```
“str1”.equals(“str2”)
“str1”.equalsIgnoreCase(“str2”)
```

String a = “str1”;

String b = “str1”;

String c = new String(“str1”);




```
input.equals(“myStr”);
```

```
if(input != null) {
    boolean result = input.equals(“myStr”);
}
```

```
input.equals(“myStr”);
```

```
“myStr”.equals(input);
```

Check if a String contains only {chars}

Find and replace a String?

```
“ “ “
    Common task
    Iterate over multiple lines
    from a file
” ” ”
```

.lines()

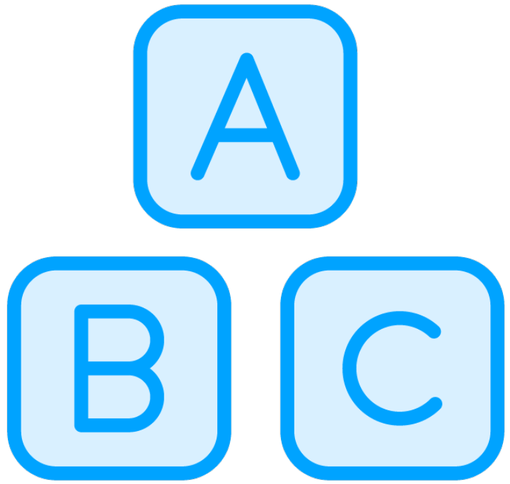
```
class String {
 /**
   * Returns a stream of lines […] separated by line
   * terminators […]
   * a line feed character "\n" (U+000A), a carriage return
    * character "\r" (U+000D), or a carriage return followed
    * immediately by a line feed "\r\n" (U+000D U+000A)
    * /
    Stream<String> lines()
}
```

Tokenizing
(breaking up a string into tokens)

Tokenize by comma or semicolon?

```
class String {

    String split(String regex);
}
```

```
var sb = new StringBuilder();
for(String line : lines) {
    sb.append("-> ").append(line);
}
```



Strings are immutable

New operation - new String object

Million operations - million new objects

StringBuilder optimizes this

```
class StringBuilder {
    delete();
    insert();
    replace();
    reverse();
    // etc.
}
```

String Requirements

- Secure
- Vary in size
- Specific set of characters
- Generation speed
- Thread-safe?

String Requirements

Cryptography

Pseudo Random Data Generation

Collision

URL safety

Session Management

Use an appropriate library

Use reliable libraries

Apache Commons

```
class StringUtils {


    isAllLowerCase(...);
    abbreviate(...);
    capitalize(...)
    containsAny(...);
    containsNone(...);
}
```

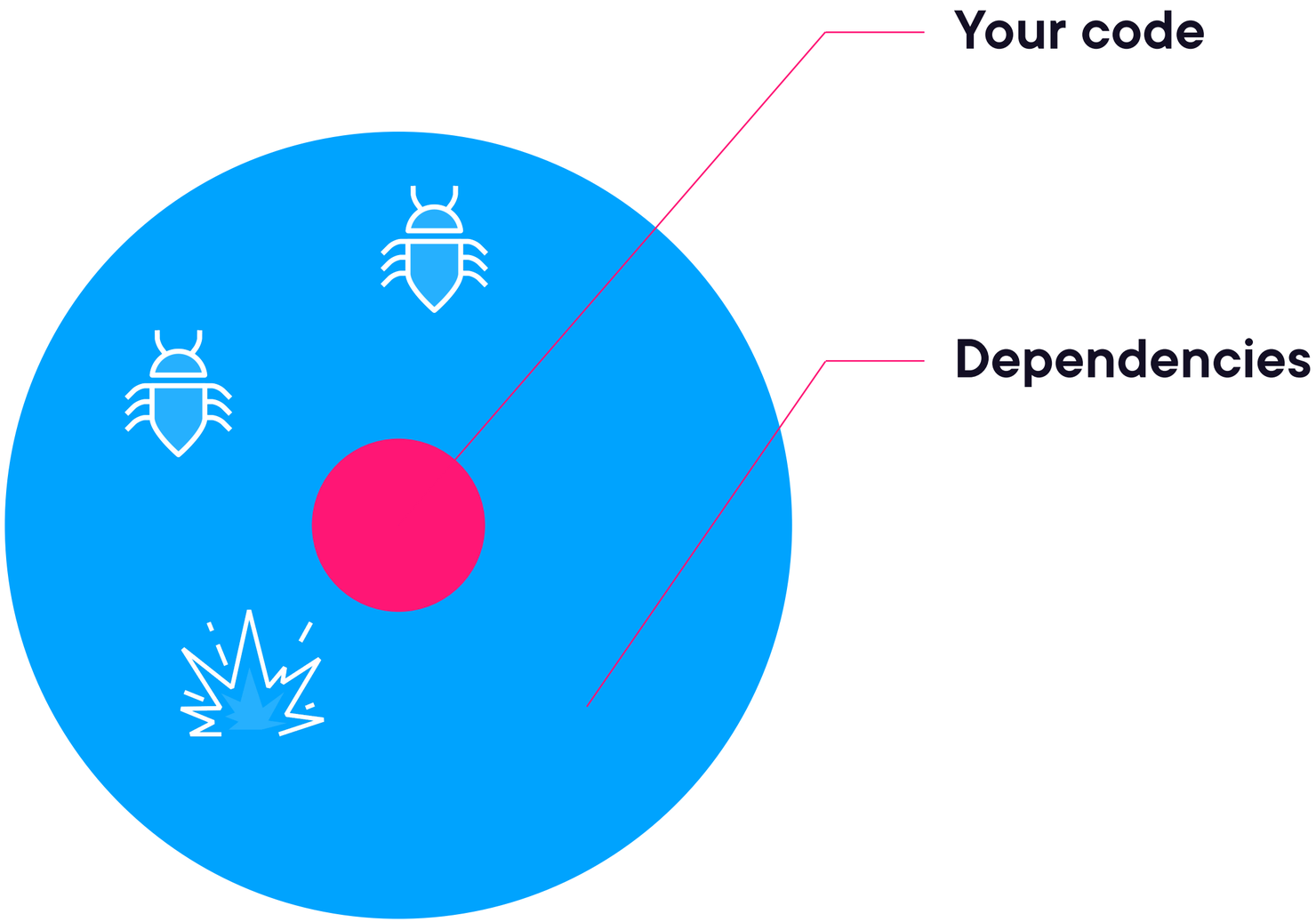
Dependency Management

Beware of frivolously including dependencies

Every dependency has a maintenance cost and even a security risk

I'm not saying: "don't include dependencies"

I'm saying: "think twice before you include one"


Your code
Dependencies

Summary

Most common String operations

Up Next:

Working with Numbers