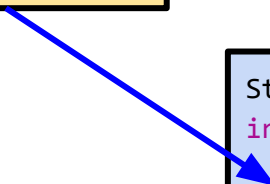


# HELPFUL TIPS FOR MODULE 1 CAPSTONE

USING  
STRING.FORMAT

# USING STRING.FORMAT

Using concatenation.



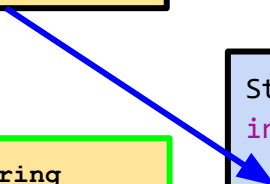
```
String name = "Yoav";  
int messageCount = 5;  
String msg = name + ", you have " + messageCount + " messages.";  
  
String msgUsingFormat = String.format("%s, you have %d messages.",  
    name, messageCount);
```

# USING STRING.FORMAT

Using concatenation.

The static `String` method `format` takes a parameter that is a `String` literal with placeholders. After the literal, the parameters to populate all the placeholders are listed. The method creates a `String` using the parameters and returns it.

Building `Strings` using `format` can be a lot cleaner and easier to read than concatenation.



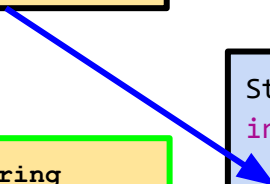
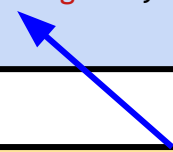
```
String name = "Yoav";  
int messageCount = 5;  
  
String msg = name + ", you have " + messageCount + " messages.";   
  
String msgUsingFormat = String.format("%s, you have %d messages.",  
    name, messageCount);
```

# USING STRING.FORMAT

Using concatenation.

The static `String` method `format` takes a parameter that is a `String` literal with placeholders. After the literal, the parameters to populate all the placeholders are listed. The method creates a `String` using the parameters and returns it.

Building `Strings` using `format` can be a lot cleaner and easier to read than concatenation.

```
String name = "Yoav";  
int messageCount = 5;  
  
String msg = name + ", you have " + messageCount + " messages.";   
  
String msgUsingFormat = String.format("%s, you have %d messages.",  
    name, messageCount); 
```

Literal with placeholders. `%s` indicates a `String`, `%d` indicates a whole number like and `int`.

# USING STRING.FORMAT

Using concatenation.

The static `String` method `format` takes a parameter that is a `String` literal with placeholders. After the literal, the parameters to populate all the placeholders are listed. The method creates a `String` using the parameters and returns it.

Building `Strings` using `format` can be a lot cleaner and easier to read than concatenation.

The `name` variable will provide the `String` for the first placeholder.

```
String name = "Yoav";  
int messageCount = 5;  
  
String msg = name + ", you have " + messageCount + " messages.";   
  
String msgUsingFormat = String.format("%s, you have %d messages.",  
    name, messageCount);
```

Literal with placeholders. `%s` indicates a `String`, `%d` indicates a whole number like and `int`.

# USING STRING.FORMAT

Using concatenation.

The static `String` method `format` takes a parameter that is a `String` literal with placeholders. After the literal, the parameters to populate all the placeholders are listed. The method creates a `String` using the parameters and returns it.

Building `Strings` using `format` can be a lot cleaner and easier to read than concatenation.

The `name` variable will provide the `String` for the first placeholder.

```
String name = "Yoav";  
int messageCount = 5;  
  
String msg = name + ", you have " + messageCount + " messages.";   
  
String msgUsingFormat = String.format("%s, you have %d messages.",  
    name, messageCount);
```

The `messageCount` variable will provide the `int` for the second placeholder.

Literal with placeholders. `%s` indicates a `String`, `%d` indicates a whole number like `int`.

# USEFUL LINKS

- 1) [Free online tool for creating diagrams](#)
- 2) [More info on String.format method](#)
- 3) [Documentation for DateTimeFormatter Java class](#)
- 4) [Mostly Useful Info on Using BigDecimal](#)