Lambda Mini Challenges

This challenge video is going to be a little different, and consist of several small tasks, to help you really practice creating and using lambda expressions.

I'll assign a task for you to try on your own, and then we can solve it together.

Then I'll present the next one.

Mini Challenge 1

Write the following anonymous class that you can see on screen, as a lambda expression.

Try to do it manually on your own, and don't rely on IntelliJ's tools to do it for you.

This will help you understand lambdas better.

```java
Consumer<String> printTheParts = new Consumer<String>() {

    @Override
    public void accept(String sentence) {
        String[] parts = sentence.split(" ");
        for (String part : parts) {
            System.out.println(part);
        }
    }
};
```

Mini Challenge 2

Write the following method as a lambda expression.

```
public static String everySecondChar(String source) {

    StringBuilder returnVal = new StringBuilder();
    for (int i = 0; i < source.length(); i++) {
        if (i % 2 == 1) {
            returnVal.append(source.charAt(i));
        }
    }
    return returnVal.toString();
}
```

In other words, create a variable, using a type that makes sense for this method.

Don't worry about executing it though.

Mini Challenge 3

The lambda expression we created in Challenge 2 doesn't do anything right now.

I want you to write the code to execute this lambda expression's functional method, using 1234567890, as the argument to that method, and print the result out.

```java
UnaryOperator<String> everySecondChar = source -> {
    StringBuilder returnVal = new StringBuilder();
    for (int i = 0; i < source.length(); i++) {
        if (i % 2 == 1) {
            returnVal.append(source.charAt(i));
        }
    }
    return returnVal.toString();
};
```

Mini Challenge 4

Instead of executing this function directly, suppose that we want to pass it to a method.

Write another method on your class, called everySecondCharacter.

This method should accept a Function, or UnaryOperator, as a parameter, as well as a second parameter that lets us pass, "1234567890".

In other words, don't hard code that string in your method code.

The method code should execute the functional method on the first argument, passing it the value of the string passed, from the enclosing method.

It should return the result of the call to the functional method.

Lambda Expressions, What you need to understand

1) Declare lambda variables, or pass lambdas directly to methods that are targets.

I show two examples here.

Local Variable Declaration	**Method argument**
`Function<String, String> myFunction = s -> s.concat(" " + suffix);`	`list.forEach(s -> System.out.println(s));`

2) Create methods that can be targets for lambda expressions.

Mini Challenge 5

Call the method you created from Challenge 4, passing the lambda variable we created earlier, and the string 1234567890, then print the result returned from the method.

Mini Challenge 6

Write a lambda expression that is declared with the Supplier interface.

This lambda should return the String, "I love Java", and assign it to a variable called, `iLoveJava`.

Mini Challenge 7

As with the Function example, the Supplier lambda won't do anything until we use it.

Remember, lambdas represent deferred execution of snippets of code.

Use this Supplier to assign a String, "I love Java", to a variable called supplierResult.

Print that variable to the console.