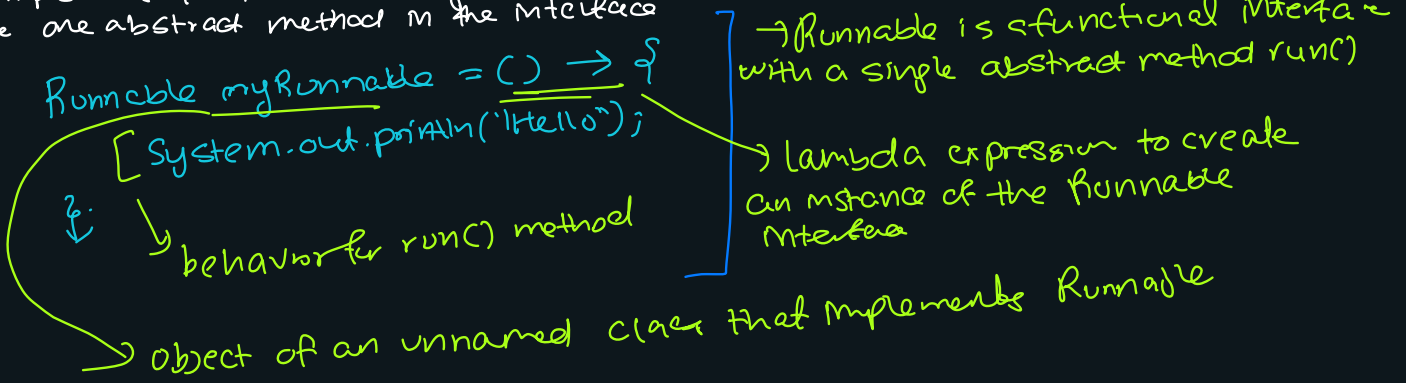
Lambda Expressions

- **Lambda expression**: a special syntax for creating an object that implements an interface that only has one abstract method (known as functional interface) that closely match a method in the Object class.

- **Functional Interfaces**: can be reduced to a 2-step process by using lambda expression syntax. It has only one abstract method but it can have static or default methods

- **Lambda Expression Syntax**:

1) instantiate – use a lambda expression to create an object of an unnamed class that implements the interface by defining what the override behavior should be for the one abstract method in the interface



2) Interact using the interface: use a variable with the interface as its type so that your code works with any object of a class that implements the interface

```
// Driver.java (assume proper package and import statements)
public class Driver {

    public static void forEach(String[] strings, Consumer<String> consumer) {
        for (int i = 0; i < strings.length; i++) {
            String str = strings[i];
            consumer.accept(str);
        } // for
    } // forEach

    public static void main(String[] args) {
        Consumer<String> shout = (String t) -> System.out.println(t.toUpperCase());
        Driver.forEach(args, shout);
    } // main

} // Driver
```

functional interface

lambda

→ lambda expression

`Consumer<String> shout = (String t) -> System.out.println(t.toUpperCase());`

ref variable shout declared with the type Consumer<String>

assigns the object's ref to the variable

A lambda expression is used to create an object that has one method by defining what that method should do. We match the method's type layout to match the abstract method accept in the Consumer<String> and it does (accept in it has (T t) so (String t))

→ How to create a lambda expression?

1) Assign your intended method override to a variable of interface type

```
Consumer<String> shout = @Override public void accept(String t) {
        return System.out.println(t.toUpperCase());
    };
```

2) Remove @Override annotation, visibility modifier, return type & method name & add arrows between parameter list & opening curly-braces

```
Consumer<String> shout = (String t) -> {
        return System.out.println(t.toUpperCase());
    };
```

→ Room for Improvement

- If method body contains only 1 statement, then omit curly braces

  `Consumer<String> shout = (String t) -> System.out.println(t.toUpperCase());`

- Specifying parameter types is optional: automatic conversion (t) to (String t)

  `Consumer<String> shout = (t) -> System.out.println(t.toUpperCase());`

- If there is exactly one method parameter, then the parantheses for parameter list are optional

  `Consumer<String> shout = t -> System.out.println(t.toUpperCase());`