**Lambda:**

**Thread has been in Java from the very beginning the class called thread is a thread of execution and it is a class you can instantiate and run and one of the things about thread is that it has a constructor that takes in an argument a runnable now what is a runnable a functional Interface.**

**This is key to working with lambdas, when Java introduced the concept of a Lambda expression and did not make a class called Lambda or an interface called Lambda for that matter instead they decided to leverage existing interfaces, that have been in Java from 1.0 where interfaces are known as functional interfaces, so if it if a Java interface has a single abstract method then that interface is called a functional interface and therefore be used as the assignment target for a Lambda expression or a method reference the idea of a functional interface is simply an interface that lets only one abstract method.**

**How above code is compiled, compiler sees that we are instantiating the thread class and it look at all the thread constructors, is there a thread constructor that takes a Lambda.**

**That means is there a thread constructor that takes in an argument as functional interface and of course its runnable and it is a functional interface. So, it checks what is the single abstract method here it is run with no arguments that returns void and it checks is this Lambda compatible with that single abstract method and it is because it takes no arguments and returns body.**

|  |
| --- |
| public static void main (String args[]) {  new Thread(new Runnable() {  @Override  public void run() {  System.*out*.println("this is runnable, anonymous class");  }  }).start();   //Expression Lambda  new Thread(() -> System.*out*.println("This is expression lambda")).start();   //Block Lambda  new Thread(() -> {  System.*out*.println("this is block lambda");  }).start();   //Assign Lambda to a variable  Runnable runnable = () -> System.*out*.println("lambda assigned to a variable");  new Thread(runnable).start(); } |

**We can take lambda itself and simply assign it to a variable that variable must be of functional interface type simply said let us take a reference of type functional interface runnable a use that to assign our Lambda now again this syntax will have to be compatible with the single abstract method inside the functional interface.**