**Why Java 8?**

* Concise and minimal Code.
* To utilize functional programing.
* To enable parallel programming.

**Features of Java 8?**

* Lambda Expression
* Stream API
* Date and Time API
* Base64 Encode Decode
* Method reference Constructor reference
* Default method and Static method in Interface
* Functional Interface
* Optional Class
* Java IO Improvements
* Collection API Improvements

**Lambda Expression**

* It is an Anonymous Function

**Functional Interface**

* Interface having only and only one Single Abstract method and number of Defaults and Static Methods.
* We can invoke Lambda Expression by using Function Interface.
* Functional Interface act as Type for Lambda Expression.

**Default Methods in Interface**

* If avoid ambiguity in this so there is two ways
* Override those methods and write own implementation
* Else in this call super method

**Static Method in Interface**

* It contain the complete definition of the function.
* Cannot be override or change in the implementation class.

**Predicate<T>** - T – Generic Type return Boolean

boolean test(T t);

**Function<T,R>** - T – One Generic Argument , R - return Result type

R apply(T t);

**Consumer<T> -** Accept one Argument and No Retune.

void accept(T t);

**Supplier<T> -** Can’t take any argument only Return.

T get();

BiPredicate<String,Integer> biPredicate1 = (str,x) -> str.length() == x;  
System.*out*.println(biPredicate1.test("ABCDE",5));  
  
BiFunction<String,String,Integer> biFunction = (x,y) -> x.length() + y.length();  
System.*out*.println(biFunction.apply("Hi","Hello"));  
  
BiConsumer<Integer,Integer> biConsumer = (x,y) -> System.*out*.println(x + y);  
biConsumer.accept(1,2);

BiFunction<String,String,String> biFunction = (str1,str2) -> str1 + str2;  
BinaryOperator<String> binaryOperator = (str1,str2) -> str1 + str2;  
System.*out*.println(binaryOperator.apply("Hi","Hello"));

UnaryOperator<Integer> unaryOperator = x -> x \* x;  
UnaryOperator<String> unaryOperator1 = str -> str.toLowerCase();