**Java 8**

1. **Difference between Lambda and anonymous class?**  
     
   Lambdas implement a functional interface.

Anonymous Inner Classes can extend a class or implement an interface with any number of methods.

*Variables* – Lambdas can only access final or effectively final.

*State* – Anonymous inner classes can use instance variables and thus can have state, lambdas cannot.

*Scope* – Lambdas can't define a variable with the same name as a variable in enclosing scope.

*Compilation* – Anonymous compiles to a class, while lambda is an invokedynamic instruction.  
  
***Ref :*** <https://dzone.com/articles/how-lambdas-and-anonymous-inner-classesaic-work>

1. **What is effectively final?**A variable which is not declared as final but whose value is never changed after initialization.
2. **What is capturing & non-capturing lambda?***Non-Capturing :* Doesn't access fields outside its body.  
   *Capturing :* Accesses final or effectively final fields outside their bodies.
3. **What is optional and how it can be used?**Optional is a new class in Java 8 that encapsulates an optional value i.e. a value that is either there or not. It is a wrapper around an object, and you can think of it as a container of zero or one element. Optional has a special Optional.empty() value instead of wrapped null. Thus it can be used instead of a nullable value to get rid of NullPointerException in many cases.  
     
   The main purpose of Optional, as designed by its creators, was to be a return type of methods that previously would return null. Such methods would require you to write boilerplate code to check the return value and sometimes could forget to do a defensive check. In Java 8, an Optional return type explicitly requires you to handle null or non-null wrapped values differently.  
     
   For instance, the Stream.min() method calculates the minimum value in a stream of values. But what if the stream is empty? If it was not for Optional, the method would return null or throw an exception. But it returns an Optional value which may be Optional.empty() (the second case). This allows us to easily handle such case:

int min1 = Arrays.stream(new int[]{1, 2, 3, 4, 5})  .min().orElse(0);

assertEquals(1, min1);

int min2 = Arrays.stream(new int[]{}) .min().orElse(0);

assertEquals(0, min2);

It's worth noting that Optional is not a general purpose class like Option in Scala. It is not recommended to be used as a field value in entity classes, which is clearly indicated by it not implementing the Serializable interface.

1. **What is predicate?**A predicate (functional interface) is commonly understood to be a **boolean-valued** function **'P: X? {true, false}'**, called the predicate on X. Informally, a strong. It can be thought of as an operator or function that returns a value that is either true or false.  
     
   **boolean test(T t)**It is majorly used for condition checking or data filtering with the help of lambda operation for stream operations.  
     
   **Ref :** <https://www.javatpoint.com/java-predicate-interface>
2. **What is supplier?  
     
   @FunctionalInterface**

**public interface Supplier<T> {**

**T get();**

**}**[Supplier](https://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html) is a functional interface; it takes no arguments and returns a result.   
  
This is majorly used to generate sort of static data or pre-computed data.

1. **What is consumer?**It contains an abstract **accept()** and a default **andThen()** method. It can be used as the assignment target for a lambda expression or method reference.   
     
   The Consumer Interface accepts a single argument and does not return any result. This is majorly used for void utility operations. Example : Sending email to multiple users using loop.  
     
   **Ref :** <https://www.javatpoint.com/java-consumer-interface>
2. **Stream Operations:**

**Ref :** <https://stackify.com/streams-guide-java-8/>