## Lambda Syntax:

**1. Basic Lambda Expression (Runnable)**

Runnable r = () -> System.out.println("Hello from Lambda!");

new Thread(r).start();

**Explanation**: Simplifies writing Runnable logic with concise syntax.

**2. Lambda with Comparator (Custom Sorting)**

List<String> names = Arrays.asList("John", "Alice", "Bob");

Collections.sort(names, (a, b) -> a.compareToIgnoreCase(b));

**Explanation**: Avoids writing separate Comparator class.

**3. Predicate for Filtering**

Predicate<String> isShort = s -> s.length() <= 3;

System.out.println(isShort.test("Hi")); // true

**Explanation**: Used for conditional checks in functional pipelines.

**4. Function (Transform Input to Output)**

Function<String, Integer> toLength = s -> s.length();

System.out.println(toLength.apply("Lambda")); // 6

**Explanation**: Accepts 1 input and returns an output.

**5. Consumer (Takes Input, Returns Nothing)**

Consumer<String> greeter = name -> System.out.println("Hello " + name);

greeter.accept("Students");

**Explanation**: Used for performing an action like printing, logging.

**6. Supplier (Provides Data)**

Supplier<Double> randomValue = () -> Math.random();

System.out.println(randomValue.get());

**Explanation**: Supplies data without input — often used for lazy initialization.

**7. Iterating Collections with forEach**

List<String> fruits = Arrays.asList("Apple", "Banana", "Mango");

fruits.forEach(fruit -> System.out.println(fruit));

**Explanation**: Lambda with forEach provides elegant iteration.

**8. Map Iteration using Lambda**

Map<Integer, String> map = Map.of(1, "A", 2, "B");

map.forEach((k, v) -> System.out.println(k + " : " + v));

**Explanation**: Clean iteration over key-value pairs.

**9. Stream Filter**

List<String> list = Arrays.asList("one", "two", "three");

list.stream().filter(s -> s.startsWith("t")).forEach(System.out::println);

**Explanation**: Filters data based on a condition.

**10. Stream Map (Transformation)**

list.stream().map(String::toUpperCase).forEach(System.out::println);

**Explanation**: Transforms each element in a stream.

**11. Stream Sorted**

list.stream().sorted().forEach(System.out::println);

**Explanation**: Sorts elements using natural or custom order.

**12. Stream Reduce (Aggregation)**

int sum = Arrays.asList(1, 2, 3).stream().reduce(0, (a, b) -> a + b);

System.out.println(sum);

**Explanation**: Combines all elements into a single result.

**13. Stream Collect (Result Collection)**

List<String> filtered = list.stream()

.filter(s -> s.contains("o"))

.collect(Collectors.toList());

System.out.println(filtered);

**Explanation**: Collects stream output into a list or other structures.

**14. Passing Lambda to Method**

public static void greet(Consumer<String> action) {

action.accept("Java Learner");

}

greet(name -> System.out.println("Welcome, " + name));

**Explanation**: Lambdas can be passed to methods just like variables.

**15. BiFunction (Two Inputs, One Output)**

BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;

System.out.println(add.apply(4, 5));

**Explanation**: Useful for combining two values.

**16. BiPredicate (Two Inputs, Boolean Output)**

BiPredicate<String, Integer> checkLength = (s, len) -> s.length() == len;

System.out.println(checkLength.test("Java", 4)); // true

**Explanation**: Conditional checks on two values.

**17. Optional with Lambda**

Optional<String> name = Optional.of("Lambda");

name.ifPresent(n -> System.out.println(n.toUpperCase()));

**Explanation**: Safely operate on potentially null values.

**18. Method Reference**

List<String> colors = Arrays.asList("Red", "Green");

colors.forEach(System.out::println);

**Explanation**: Concise form of lambda when calling existing methods.

**19. Custom Functional Interface**

@FunctionalInterface

interface MessagePrinter {

void print(String msg);

}

MessagePrinter printer = msg -> System.out.println("Message: " + msg);

printer.print("Lambda Rocks!");

**Explanation**: Enables domain-specific behavior using lambdas.

**20. Multiple Statements in Lambda**

Runnable r = () -> {

System.out.println("Start");

System.out.println("End");

};

new Thread(r).start();

**Explanation**: Lambdas can include multiple statements using {}.

**21. Stream anyMatch / allMatch**

List<Integer> nums = Arrays.asList(2, 4, 6);

boolean allEven = nums.stream().allMatch(n -> n % 2 == 0);

System.out.println(allEven); // true

**Explanation**: Useful for validation checks in a collection.

**22. Stream findFirst / findAny**

Optional<String> result = list.stream().filter(s -> s.startsWith("t")).findFirst();

result.ifPresent(System.out::println);

**Explanation**: Quickly retrieves matching element from stream.

**23. Grouping using Collectors**

Map<Integer, List<String>> groupedByLength = list.stream()

.collect(Collectors.groupingBy(String::length));

System.out.println(groupedByLength);

**Explanation**: Group elements using a classifier function.

**24. Sorting using Comparator + Lambda**

List<String> cities = Arrays.asList("Delhi", "Mumbai", "Chennai");

cities.sort((a, b) -> b.compareTo(a));

System.out.println(cities);

**Explanation**: Custom sorting logic using lambda.

**25. Removing Duplicates with Distinct**

List<Integer> numbers = Arrays.asList(1, 2, 2, 3, 3);

numbers.stream().distinct().forEach(System.out::println);

**Explanation**: Filters out duplicate elements in a stream.

1. 1️⃣ Which of the following is a checked exception in Java?  
   a) NullPointerException  
   b) ArrayIndexOutOfBoundsException  
   c) IOException  
   d) ArithmeticException
2. 2️⃣ What is the correct order of catch blocks when handling multiple exceptions?  
   a) Specific to general  
   b) General to specific  
   c) Doesn’t matter  
   d) Only one catch block is allowed
3. 3️⃣ What happens if you don’t handle a checked exception in your code?  
   a) Code compiles but throws at runtime  
   b) Compilation error  
   c) Runs normally  
   d) JVM shuts down
4. 4️⃣ What is the use of the finally block in Java exception handling?  
   a) Executes only if exception occurs  
   b) Executes only if no exception occurs  
   c) Executes regardless of exception  
   d) Never executes
5. 5️⃣ Which of these classes allows duplicate elements and maintains insertion order?  
   a) HashSet  
   b) TreeSet  
   c) ArrayList  
   d) HashMap
6. 6️⃣ Which collection should you use when you need to store unique elements in sorted order?  
   a) HashSet  
   b) LinkedHashSet  
   c) TreeSet  
   d) ArrayList
7. 7️⃣ What is the main difference between ArrayList and LinkedList?  
   a) Both are same internally  
   b) ArrayList is faster for random access, LinkedList is faster for inserts/deletes  
   c) LinkedList is thread-safe, ArrayList is not  
   d) None of the above
8. 8️⃣ Which interface does Map extend?  
   a) Collection  
   b) Iterable  
   c) Map doesn’t extend any collection interfaces  
   d) Set
9. 9️⃣ The compareTo() method is defined in which interface?  
   a) Comparator  
   b) Comparable  
   c) Collections  
   d) Map.Entry
10. 🔟 When would you prefer to use a Comparator over Comparable?  
    a) When you want a single natural ordering  
    b) When you want multiple different sorting orders  
    c) When you want no sorting  
    d) When using HashMap
11. what does the `static` keyword in Java mean?  
    a) Variable belongs to the class, not to instances  
    b) Variable is immutable  
    c) Variable is private  
    d) Variable is destroyed after method call

2️⃣ True or False: A `static` method can access instance variables of a class directly.

3️⃣ What will happen if you declare a `static` block in a class?  
a) It runs once when the class is loaded.  
b) It runs every time an object is created.  
c) It runs every time the JVM starts.  
d) It never runs.

4️⃣ Which of these is NOT allowed with a `final` variable?  
a) Reassigning the value  
b) Declaring it without initialization and assigning later (only for instance variables)  
c) Using it as a constant  
d) Passing it to a method as a parameter

5️⃣ True or False: A `final` method cannot be overridden in a subclass.

6️⃣ What does `final` mean when applied to a class?

7️⃣ In which memory area are Java objects created?  
a) Stack  
b) Heap  
c) Method Area  
d) Native Method Stack

8️⃣ Which part of memory stores class-level metadata (like static variables & bytecode)?  
a) Heap  
b) Stack  
c) Method Area (Metaspace in newer JVMs)  
d) Registers

9️⃣ What is garbage collection in Java? Briefly explain.

🔟 Which of the following is NOT a way to create a thread in Java?  
a) Extending `Thread` class  
b) Implementing `Runnable` interface  
c) Using `ExecutorService`  
d) Declaring a method as `synchronized`

1️⃣1️⃣ What does the `synchronized` keyword do?

1️⃣2️⃣ True or False: Multiple threads can enter a synchronized block of the same object at the same time.

1️⃣3️⃣ What is the purpose of `wait()` and `notify()` in Java?

1️⃣4️⃣ What is the output of:  
```java  
int[] arr = new int[5];  
System.out.println(arr[0]);  
```  
a) 0  
b) null  
c) Compilation error  
d) Undefined

1️⃣5️⃣ Can the size of an array be changed after it is created? (Yes/No)

1️⃣6️⃣ Which of the following are immutable in Java?  
a) String  
b) StringBuilder  
c) StringBuffer  
d) Both a and c

String str = new String(“abc”);

String str1 = “abc”

Str1=”abcd”

Q1: **Which of the following is true about abstraction in Java**?  
a) Abstract classes can have both abstract and concrete methods.  
b) Interfaces cannot have any method implementations.  
c) Abstract classes cannot have constructors.  
d) Abstract classes cannot have variables.

Q2: **Which type of polymorphism is achieved through method overloading**?  
a) Runtime polymorphism  
b) Compile-time polymorphism  
c) Both a and b  
d) None of the above

Q3: **What is required for a method to override another method?**a) Same name, same parameters, and subclass relationship  
b) Same name but different parameters  
c) Method must be static  
d) Method must be private

Q4: **Which keyword is used in Java to inherit a class?**a) extends  
b) implements  
c) inherit  
d) super

Q5: **Can a Java class inherit from multiple classes? Why or why not?**

Q6**: Which of the following best describes encapsulation?**a) Wrapping data and methods together and restricting direct access to data  
b) Hiding implementation details and showing only functionality  
c) Using a superclass to define common behavior  
d) None of the above

Q7: **Which access specifier allows a member to be accessible within the same package and also by subclasses in different packages?**a) private  
b) default  
c) protected  
d) public

Q8: **What is the purpose of a constructor in Java?**a) To create an object  
b) To initialize an object  
c) Both a and b  
d) None of the above

Q9**: Which of the following is not allowed in method overloading?**a) Changing the number of parameters  
b) Changing the type of parameters  
c) Changing the method name  
d) Having same parameters but different return type

**Q10: Match the following:**- Abstraction → b  
- Encapsulation → d  
- Inheritance → c  
- Polymorphism → a  
  
a) One interface, many implementations  
b) Hiding internal details  
c) Reusing code  
d) Binding data and behavior