**Java 8 Concepts & Features**

Java 8 introduced several new features that made Java more functional, readable, and efficient. Below are the key concepts:

**🔹 1. Lambda Expressions**

* Introduced functional programming in Java
* Syntax: (parameters) -> expression
* Example:

java

CopyEdit

interface Sayable {

void say(String msg);

}

public class LambdaExample {

public static void main(String[] args) {

Sayable s = (msg) -> System.out.println("Message: " + msg);

s.say("Hello, Java 8!");

}

}

**🔹 2. Functional Interfaces**

* An interface with only **one** abstract method
* Example: Runnable, Callable, Predicate, Consumer, Supplier, Function
* Custom Functional Interface:

java

CopyEdit

@FunctionalInterface

interface MyFunctionalInterface {

void display();

}

public class Test {

public static void main(String[] args) {

MyFunctionalInterface obj = () -> System.out.println("Functional Interface Example");

obj.display();

}

}

**🔹 3. Default & Static Methods in Interfaces**

* Default methods allow adding new functionality to interfaces without breaking existing implementations.
* Example:

java

CopyEdit

interface Vehicle {

default void show() {

System.out.println("This is a default method in an interface.");

}

}

class Car implements Vehicle {}

public class Test {

public static void main(String[] args) {

Car car = new Car();

car.show(); // Output: This is a default method in an interface.

}

}

**🔹 4. Streams API (Java.util.stream)**

* Used for processing collections efficiently (e.g., filtering, mapping, reducing)
* Example:

java

CopyEdit

import java.util.Arrays;

import java.util.List;

public class StreamExample {

public static void main(String[] args) {

List<String> names = Arrays.asList("John", "Jane", "Mark", "Emily");

names.stream().filter(name -> name.startsWith("J")).forEach(System.out::println);

}

}

**🔹 5. Method References**

* A shorthand notation of lambda expressions
* Types: **Static Method Reference, Instance Method Reference, Constructor Reference**
* Example:

java

CopyEdit

import java.util.function.Consumer;

class Printer {

static void print(String msg) {

System.out.println(msg);

}

}

public class MethodRefExample {

public static void main(String[] args) {

Consumer<String> printer = Printer::print;

printer.accept("Hello from Method Reference!");

}

}

**🔹 6. Optional Class (java.util.Optional)**

* Helps avoid NullPointerException by handling null values gracefully
* Example:

java

CopyEdit

import java.util.Optional;

public class OptionalExample {

public static void main(String[] args) {

Optional<String> str = Optional.ofNullable(null);

System.out.println(str.orElse("Default Value"));

}

}

**🔹 7. New Date & Time API (java.time package)**

* Introduced to replace the outdated java.util.Date and java.util.Calendar
* Example:

java

CopyEdit

import java.time.LocalDate;

import java.time.LocalTime;

import java.time.LocalDateTime;

public class DateTimeExample {

public static void main(String[] args) {

LocalDate date = LocalDate.now();

LocalTime time = LocalTime.now();

LocalDateTime dateTime = LocalDateTime.now();

System.out.println("Date: " + date);

System.out.println("Time: " + time);

System.out.println("Date & Time: " + dateTime);

}

}

**🔹 8. Collectors Class (java.util.stream.Collectors)**

* Used to perform operations like grouping, joining, and reducing data
* Example:

java

CopyEdit

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class CollectorsExample {

public static void main(String[] args) {

List<String> names = Arrays.asList("John", "Jane", "Mark", "Emily");

String result = names.stream().collect(Collectors.joining(", "));

System.out.println(result);

}

}

**🔹 9. CompletableFuture & Asynchronous Programming**

* Used for handling asynchronous tasks efficiently
* Example:

java

CopyEdit

import java.util.concurrent.CompletableFuture;

public class AsyncExample {

public static void main(String[] args) {

CompletableFuture.runAsync(() -> System.out.println("Running asynchronously"));

}

}

**🔹 10. Parallel Streams**

* Used for parallel processing of collections
* Example:

java

CopyEdit

import java.util.Arrays;

import java.util.List;

public class ParallelStreamExample {

public static void main(String[] args) {

List<String> names = Arrays.asList("John", "Jane", "Mark", "Emily");

names.parallelStream().forEach(System.out::println);

}

}

**🔥 Key Takeaways**

✔ **Lambda Expressions** → Functional programming  
✔ **Functional Interfaces** → Single abstract method  
✔ **Default & Static Methods** → Interface enhancements  
✔ **Streams API** → Processing collections efficiently  
✔ **Method References** → Shorthand notation for lambdas  
✔ **Optional Class** → Handling null values  
✔ **Date & Time API** → New, improved date handling  
✔ **Collectors Class** → Data grouping & aggregation  
✔ **CompletableFuture** → Asynchronous programming  
✔ **Parallel Streams** → Multi-threaded data processing

**🎯 Conclusion**

Java 8 introduced a more functional, concise, and powerful approach to programming. Learning these concepts will help you write **better, cleaner, and more efficient code**! 🚀