CourseCube
(Formerly Java Learning Center)

Java 8

New Features

Author
Srinivas Dande

Java Learning Center

1. Introduction

- Java 8 is a revolutionary release of the **World's #1 development platform**.
- It includes a huge upgrade to the Java programming model and a coordinated evolution of the JVM, Java language, and libraries.
- Java 8 includes features for productivity, ease of use, security and improved performance.
- Welcome to the latest iteration of the largest, open, standards-based, community-driven platform.

- Java8 Release comes with Many new features as Listed

1) Default Methods in Interface
2) Static Methods in Interface
3) Lambda Expressions
4) Method References
5) Functional Interfaces
6) Functional Programming
7) Streams API
8) Joda Date API
9) Optional Class
10) Miscellaneous Features

1. Default Methods in Interface

- **Concrete methods (Methods with Body)** Defined in the Interface with **default keyword** are called as **Default Methods**.
- Default methods are also known as **defender methods** or **virtual extension methods**
- Default Methods are public by default.
- Default Methods will be inherited to Sub classes.
- Sub class can override the Interface Default Methods.
- We can't write default methods inside a class. Even when we are overriding the default method in sub class, we should not use default keyword for sub class method.
- We can't override Object class methods as default methods inside Interface

Why Default Methods:

- We can define new functionality in the interfaces without breaking down the implementing classes
- We can avoid writing separate utility classes

Demo1: Files Required:

1. Animal.java	2. Dog.java
3. Cat.java	4. Demo1.java

1)Animal.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface Animal {
	public abstract void eating();
	public abstract void sleeping();

	default void running() {
		System.out.println("Animal is Running");
	}

	default void thinking() {
		System.out.println("Animal is Thinking");
	}
}
```

2)Dog.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Dog  implements Animal{

	@Override
	public void eating() {
		System.out.println("Dog is eating");
	}
	@Override
	public void sleeping() {
		System.out.println("Dog is sleeping");
	}
	@Override
	public void running() {
		System.out.println("Dog is running");
	}
}
```

3)Cat.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Cat  implements Animal{

	@Override
	public void eating() {
		System.out.println("Cat is eating");
	}
	@Override
	public void sleeping() {
		System.out.println("Cat is sleeping");
	}
	@Override
	public void thinking() {
		System.out.println("Cat is thinking");
	}
}
```

4)Demo1.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo1 {
	public static void main(String[] args) {

		Dog mydog= new Dog();
		mydog.eating();  //Overriden method
		mydog.sleeping();  //Overriden method
		mydog.running();  //Overriden method
		mydog.thinking();  //Inherited default method

		Cat mycat= new Cat();
		mycat.eating();  //Overriden method
		mycat.sleeping();  //Overriden method
		mycat.running();  //Inherited default method
		mycat.thinking();   //Overriden method
	}
}
```

Demo2: Files Required:

1. A.java	2. B.java
3. Hello.java	4. Demo2.java

1)A.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface A {

	default void show() {
		System.out.println("A- show() ");
	}
}
```

2)B.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface B {

	default void show() {
		System.out.println("B- show()");
	}
}
```

3)Hello.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Hello  implements A,B{

	@Override
	public  void show() {
		System.out.println("Hello- show() ");
	}
	public void test() {
		System.out.println("Hello- test() ");
		show();
		A.super.show();
		B.super.show();
	}
}
```

4)Demo1.java

```
package com.jlcindia.demo2;

public class Demo2 {
	public static void main(String[] args) {
	Hello hello=new Hello();
	hello.test();
	}
}
```

Demo3: Files Required:

1. A.java	2. Hello.java
3. Demo3.java	

1)A.java

```
package com.jlcindia.demo3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface A {

	default void m1() {
		System.out.println("A- m1() ");
	}

	default void m2() {
		System.out.println("A- m2() ");
		m1();
	}

	/*
	default boolean equals(Object obj) {
		System.out.println("A- equuals() ");
	}
	*/
}
```

2)Hello.java

```
package com.jlcindia.demo3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Hello  implements A{

}
```

3)Demo3.java

```
package com.jlcindia.demo3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo3 {
	public static void main(String[] args) {
		Hello hello=new Hello();
		hello.m1();
		hello.m2();
	}
}
```

Demo4: Files Required:

1. A.java	2. B.java
3. Hello.java	4. Demo4.java

1)A.java

```
package com.jlcindia.demo4;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface A {
	default void m1() {
		System.out.println("A- m1() ");
	}
}
```

2)B.java

```
package com.jlcindia.demo4;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface B extends A {
	default void m2() {
		System.out.println("B- m2() ");
		m1();
	}
}
```

3)Hello.java

```
package com.jlcindia.demo4;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Hello  implements B{

}
```

4)Demo4.java

```
package com.jlcindia.demo4;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo4 {
	public static void main(String[] args) {
		Hello hello=new Hello();
		hello.m1();
		hello.m2();
	}
}
```

Demo5: Files Required:

1. A.java	2. B.java
3. Hello.java	4. Demo4.java

1)A.java

```
package com.jlcindia.demo5;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface A {
	default void m1() {
		System.out.println("A- m1() ");
	}
}
```

2)B.java

```
package com.jlcindia.demo5;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface B extends A {

	default void m1() {
		System.out.println("B- m1() ");
	}

	default void m2() {
		System.out.println("B- m2() ");
		m1();
	}
}
```

3)Hello.java

```
package com.jlcindia.demo5;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Hello  implements B{

}
```

4)Demo4.java

```
package com.jlcindia.demo5;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo4 {
	public static void main(String[] args) {
		Hello hello=new Hello();
		hello.m1();
		hello.m2();
	}
}
```

Interview Questions:

Q1) What are Interface Default Methods?

Ans:

Q2) Will Default Methods be inherited to Sub Class?

Ans:

Q3) Can I Override the Default Methods in Sub Class?

Ans:

Q4) Can I mark the Default Methods as Protected?

Ans:

Q5) Can I mark the Regular Java Class Methods as Default?

Ans:

Q6) Can I mark the Overriden Default Methods as Default in the Sub Class?

Ans:

Q7) Can I Override Object class methods as Default Methods inside Interface?

Ans:

Q8) Why we need Default Methods inside Interface?

Ans:

Q9) What happens when Sub Class is implementing two interfaces which are having same default method?

Ans:

Q10) How Can I access Default Methods inside Sub Class?

Ans:

Q11) Can I have multiple Default Methods in interface?

Ans:

Q12) Can I call one Default Method from another Default Methods of same Interface?

Ans:

Q13) Can I Override Interface Default Method extended from other Interface?

Ans:

2. Static Methods in Interface

- **Concrete methods (Methods with Body)** Defined in the Interface with **static keyword** are called as **Static Methods**.
- Static Methods are public by default.
- Static Methods will not be inherited to Sub classes.
- Sub class can not override the Interface Static Methods.
- If we write the Static Method of Interface in Sub Class then That will be treated as New Method in Sub Class.

Why Static Methods:

- We can avoid writing separate utility classes

Demo1: Files Required:

1. A.java	2. Hello.java
3. Demo1.java	

1)A.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface A {

	int P=101;
	public final static int Q=102;

	void m1();
	public abstract void m2();

	 default void m3() {
		System.out.println("A - m3()");
	}

	 default void m4() {
		System.out.println("A - m4()");
	}

	 static void m5() {
		System.out.println("A - m5()");
	}
```

```
	static void m6() {
		System.out.println("A - m6()");
	}
}
```

2)Hello.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Hello implements A {

	public void test(){

		System.out.println(P);  //Inherited
		System.out.println(Q); //Inherited
		m1();    //Overriden
		m2();   //Overriden

		m3();     //Overriden
		A.super.m3();

		m4();   //Inherited
		A.super.m4();

		A.m5();
		A.m6();
		//A.super.m6();
	}

	@Override
	public void m1() {
		System.out.println("Hello -m1");
	}

	@Override
	public void m2() {
		System.out.println("Hello -m2");
	}
```

```
	@Override
	public void m3() {
		System.out.println("Hello -m3");

	}
	/*
	@Override
	public static void m5() {
		System.out.println("Hello -m5");
	}
	*/
}
```

3)Demo1.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo1 {
	public static void main(String[] args) {
		Hello hello=new Hello();
		hello.test();
	}
}
```

Demo2: Files Required:

1. A.java	2. B.java
3. Hello.java	4. Demo2.java

1)A.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface A {
	 static void m1() {
		System.out.println("A - m1()");
	}
}
```

2)B.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface B {

	static void m1() {
		System.out.println("B - m1()");
	}

}
```

3)Hello.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Hello implements A,B {

	public void test(){

		m1();
		A.m1();
		B.m1();
	}

	 static void m1() {
			System.out.println("Hello- m1()");
	}

	 static void show() {
			System.out.println("Hello- show()");
	}
}
```

4)Demo2.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo2 {

	public static void main(String[] args) {

	Hello hello=new Hello();
	hello.test();

	//1. Calling Static Method with Ref. Variable having Null
	// A aobj = null;
	//aobj.m1();

	Hello hello1=null;
	hello1.show();


	//2. Calling Static Method with Ref. Variable having Object address
	 //A aobj = new Hello();
	//aobj.m1();

	Hello hello2=new Hello();
	hello2.show();

	//3. Calling Static Method with Class Name
	A.m1();
	Hello.show();

	// Interface Static Methods must called with Interface name always
	  }

}
```

Demo3: Files Required:

1. A.java	

1)A.java

```
package com.jlcindia.demo3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public interface A {

	static void m1() {
		System.out.println("A - m1()");
		//m2();  // Can not call Instance Method in Static
	}

	default void m2() {
			System.out.println("A - m2()");
	}

	public static void main(String[] args) { //Standard Main Method

		System.out.println("main method");
		m1();
		//m2();  // Can not call Instance Method in Static

	}

}
```

Interview Questions:

Q1) What are Interface Static Methods?

Ans:

Q2) Will Static Methods be inherited to Sub Class?

Ans:

Q3) Can I Override the Static Methods in Sub Class?

Ans:

Q4) Can I mark the Static Methods as Protected?

Ans:

Q5) Can I mark the Regular Java Class Methods as Static?

Ans:

Q6) Why we need Static Methods inside Interface?

Ans:

Q7) What happens when Sub Class is implementing two interfaces which are having same Static method?

Ans:

Q8) How Can I access Static Methods inside Sub Class?

Ans:

Q9) Can I have multiple Static Methods in Interface?

Ans:

Q10) Can I call one Static Method from another Static Methods of same Interface?

Ans:

Q11) Can I call one Static Method from another Default Methods of same Interface?

Ans:

Q12) Can I call one Default Method from another Static Methods of same Interface?

Ans:

Q13) Can I call Write Standard Main Method in Interface?

Ans:

Q14) Can I mark Default Method as Static?

Ans:

Q15) Can I mark Default Method as Abstract?

Ans:

Q16) Can I mark Static Method as Abstract?

Ans:

3. Lambda Expressions

- Functional Programming importance has been increasing in the recent days because it is well suited for concurrent and event driven (or Reactive) programming.
- That doesn't mean that objects are bad Instead, the winning strategy is to blend object oriented and functional programming
- Lambda Expressions are introduced to support functional programming in Java
- Lambda Expression is an anonymous functions
- .i.e Lambda Expression is a function which doesn't have the name, return type and access modifiers
- Lambda Expressions are used heavily inside the Collections, Streams libraries from Java 8
- We need **Functional interfaces** to write lambda expressions.

Why Lambda Expressions:

- Reduce length of the code
- Readability will be improved
- Complexity of anonymous inner classes can be avoided

Demo1: Files Required:

1. Hello.java	2. Demo1.java

1)Hello.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hello {

	void display();

	default void m1() {
		System.out.println("Hello - m1()");
		display();
	}

	static void m2() {
		System.out.println("Hello - m2()");
	}
}
```

2)Demo1.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo1 {

public static void main(String[] args) {

Hello hello1 =  () -> {
System.out.println("Hello Guys!!!");
};

hello1.display();
hello1.m1();
//hello1.m2();
Hello.m2();
```

```
Hello hello2 =  () -> System.out.println("Welcome to Lambda World!!!");

hello2.display();
hello2.m1();
//hello2.m2();
Hello.m2();

}
}
```

Demo2: Files Required:

1. Hello.java	2. Demo2.java

1)Hello.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hello {

        void display(String name);
}
```

2)Demo2.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo2 {

public static void main(String[] args) {

Hello hello1=  (name) -> {
System.out.println("Hello "+name +"  Welcome to Lambda World!!!");
};

hello1.display("Srinivas");
```

```
Hello hello2=  (name) -> System.out.println("Hello "+name +"  Welcome to Lambda World!!!");

hello2.display("Sri");

Hello hello3=  name -> System.out.println("Hello "+name +"  Welcome to Lambda World!!!");

hello3.display("Vas");
}
}
```

Demo3: Files Required:

1. Hello.java	2. Demo3.java

1)Hello.java

```
package com.jlcindia.demo3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hello {

        void test(int a,int b);
}
```

2)Demo3.java

```
package com.jlcindia.demo3;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo3 {

public static void main(String[] args) {

Hello hello1=  (a,b) -> {
int sum = a+b;
System.out.println("Sum :  "+sum);
};
```

```
hello1.test(100,50);

Hello hello2=  (a,b) -> System.out.println("Sum :  "+ ( a+b));

hello2.test(95,45);

Hello hello3=  (a,b) -> {
int sub = a-b;
System.out.println("Sub :  "+sub);
};

hello3.test(100,50);

Hello hello4=  (a,b) -> System.out.println("Sub :  "+ ( a-b));

hello4.test(95,45);

}
}
```

Demo4: Files Required:

1. Hello.java	2. Demo4.java

1)Hello.java

```
package com.jlcindia.demo4;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hello {
        int test(int a,int b);
}
```

2)Demo4.java

```
package com.jlcindia.demo4;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo4 {

public static void main(String[] args) {

        Hello hello1 = (a, b) -> {
        int sum = a + b;
        return sum;
        };

        int sum1 = hello1.test(100, 50);
        System.out.println("Sum :  " + sum1);

        Hello hello2 = (a, b) -> {
        return a + b;
        };

        int sum2= hello2.test(95, 45);
        System.out.println("Sum :  " + sum2);

        Hello hello3 = (a, b) ->  a + b;

        int sum3= hello3.test(90, 40);
        System.out.println("Sum :  " + sum3);

}
}
```

Interview Questions:

Q1) What is Lambda Expression?

Ans:

Q2) What is Annonymous Method?

Ans:

Q3) What is the Functional Interface?

Ans:

Q4) What are the uses Lambda Expressions?

Ans:

Q5) How Lambda Expressions are better than Annonymous Inner Classes?

Ans:

Q6) How Can I Reuse the Lambda Expressions?

Ans:

Q7) How Can I use lambda expression with functional interface?

Ans:

Q8) How the Lambda Expression Parameter Type will be verified?

Ans:

Q9) How the Lambda Expression Return Type will be verified?

Ans:

Q10) How target type is inferred for the lambda expression?

Ans:

Q11) Which of the following Lambda Expressions are Valid for Hello Functuon Interface given?

```
@FunctionalInterface
public interface Hello {
        int test(int a,int b);
}
```

1)	Hello hello = (int a,int b) -> {
	int sum = a + b;
	return sum;
	};	

2)	Hello hello = (a, b) -> {
	System.out.println(a + b);
	};	

3)	Hello hello = (a, b) -> {
	return a + b;
	};	

4)	Hello hello = (a, b) -> a + b;	
5)	Hello hello = (a, b,c) -> a + b-c;	
6)	Hello hello = (int a, b) -> a + b;	
7)	Hello hello = a, b -> a + b;	

8)	Hello hello = (a, b) => {
	return a + b;
	};	

4. Method References

- Sometimes, You may have method already available with some logic which you want write in the Lambda Expression. In this case, you can use existing method instead of duplicating the code again.
- You can use the Existing Method References to re-use the logic.
- Method references are a special type of lambda expressions which are used to create simple lambda expressions by referencing existing methods.
- There are 3 types of method references.
    1) Static Method Reference
    2) Instance method Reference
    3) Constructor Reference
- A new operator **::(double colon)** called as Method Reference Delimiter to specify the Method References

Static Method Reference:

- Allows to access existing Static Methods

Syntax:

```
Class::staticMethod
```

Ex:

```
Hello hello = MyInteger::findSum;
```

Instance Method Reference:

- Allows to access existing Instance or Non Static Methods

Syntax:

```
ObjRef::instanceMethod
```

Ex:

```
MyInteger myIntRef = new  MyInteger();
Hello hello2 = myIntRef::findSum;
```

Constructor Method Reference:

- Allows to access constructor of Class to Create the Object

Syntax:

```
Class::new
```

Ex:

```
Hello hello= Course::new;
```

Demo1: Files Required:

1. Hello.java	2. MyInteger.java
3. Demo1.java	

1)Hello.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hello {
	public int test(int a, int b);
}
```

2)MyInteger.java

```
package com.jlcindia.demo1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class MyInteger {
	public static int findSum(int a, int b) {
		return a + b;
	}
}
```

3)Demo1.java

```
package com.jlcindia.demo1;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo1 {

public static void main(String[] args) {

Hello hello1 = (a, b) -> {
int sum = a+b;
return sum;
};
```

```
int sum1 = hello1.test(100, 50);
System.out.println("Sum : " + sum1);

Hello hello2 = MyInteger::findSum;

int sum2 = hello2.test(100, 50);
System.out.println("Sum : " + sum2);

Hello hello3 = Integer::sum;

int sum3 = hello3.test(100, 50);
System.out.println("Sum : " + sum3);

Hello hello4 = Integer::max;

int max = hello4.test(100, 50);
System.out.println("Max : " + max);

Hello hello5 = Integer::min;

int min = hello5.test(100, 50);
System.out.println("Min : " + min);

System.out.println("Done!!!");
}
}
```

Demo2: Files Required:

1. Hello.java	2. Hai.java
3. MyInteger.java	4. Demo2.java

1)Hello.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

@FunctionalInterface
public interface Hello {
        public int test(int a, int b);
}
```

2)Hai.java

```
package com.jlcindia.demo2;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hai {
	public void test(String str);
}
```

3)MyInteger.java

```
package com.jlcindia.demo2;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class MyInteger {
	public int findSum(int a, int b) {
		return a + b;
	}
}
```

4)Demo2.java

```
package com.jlcindia.demo2;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo2 {

public static void main(String[] args) {


Hello hello1 = (a, b) -> {
int sum = a+b;
return sum;
};
```

```
int sum1 = hello1.test(100, 50);
System.out.println("Sum : " + sum1);

MyInteger myIntRef = new  MyInteger();
Hello hello2 = myIntRef::findSum;

int sum2 = hello2.test(100, 50);
System.out.println("Sum : " + sum2);

Hai hai1 = (msg) -> {
System.out.println(msg);
};

hai1.test(" Hai Guys!!! ");

Hai hai2 = System.out :: println;

hai2.test(" Hey Guys !!! ");

}
}
```

Demo3: Files Required:

1. Hello.java	2. Course.java
3. Demo3.java	

1)Hello.java

```
package com.jlcindia.demo3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hello {
        public Course test(int a,String b,String c,String d);
}
```

2)Course.java

```
package com.jlcindia.demo3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Course {
private int courseId;
private String courseName;
private String duration;
private String trainer;

public Course() {
System.out.println("Course - 0 arg Con");
}
public Course(int courseId, String courseName, String duration, String trainer) {
System.out.println("Course - 4 arg Con");
this.courseId = courseId;
this.courseName = courseName;
this.duration = duration;
this.trainer = trainer;
}
//Setters and Getters
//toString() method
}
```

3)Demo3.java

```
package com.jlcindia.demo3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo3 {

public static void main(String[] args) {

//1.Lambda Style
Hello hello1 = (a, b, c, d) -> {
Course course = new Course(a, b, c, d);
return course;
};
```

```
Course course1 = hello1.test(101, "DevOps","60 Hrs","Srinivas Dande");
System.out.println(course1);

//2.Method Refernce Style
Hello hello2= Course::new;

Course course2 = hello2.test(102, "Boot - MicroServices","100 Hrs","Srinivas Dande");
System.out.println(course2);

System.out.println("Done!!!");
}
}
```

Demo4: Files Required:

1. Hello.java	2. Demo4.java

1)Hello.java

```
package com.jlcindia.demo4;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

@FunctionalInterface
public interface Hello {
        public void test(int[] arr);
}
```

2)Demo4.java

```
package com.jlcindia.demo4;

import java.util.Arrays;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo4 {

public static void main(String[] args) {
```

```
int myarr1[] = { 20, 40, 30, 50, 10 };

//1.Lambda Style

Hello hello1 = (arr) -> {

        for (int i = 0; i < arr.length -1; i++) {
                for (int j = i + 1; j < arr.length; j++) {
                        if (arr[i] > arr[j]) {
                        int temp = arr[i];
                        arr[i] = arr[j];
                        arr[j] = temp;
                        }
                }
        }
};


hello1.test(myarr1);

for (int x : myarr1) {
System.out.println(x);
}
System.out.println("------------------");

int myarr2[] = { 99, 88, 20, 40, 30, 50, 10 };

//2.Method Refernce Style

Hello hello2 = Arrays::sort;

hello2.test(myarr2);

for (int x : myarr2) {
System.out.println(x);
}

}
}
```

Demo5.java

```
package com.jlcindia.demo5;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Stream;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
class Hello {
public static void show(int x) {
System.out.println(x);
}
}
public class Demo4 {
public static void main(String[] args) {

List<Integer> mylist = new ArrayList<>();
mylist.add(30);
mylist.add(20);
mylist.add(50);
mylist.add(10);
mylist.add(40);

Stream<Integer> mystream= mylist.stream();
mystream.forEach(Hello::show);

System.out.println("--------------------");

mylist.stream().forEach(Hello::show);  //Static Method Ref Style
System.out.println("--------------------");

mylist.stream().forEach(System.out::println);  //Instance Method Ref Style
System.out.println("--------------------");

mylist.stream().forEach( (x) -> System.out.println(x));  //Eambda Style
}
}
```

Demo6.java

```
package com.jlcindia.demo6;

import java.util.ArrayList;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
class MyNumber {
public static boolean isEven(int number) {
if (number % 2 == 0)
return true;
else
return false;
}
public static boolean isOdd(int number) {
if (number % 2 != 0)
return true;
else
return false;
}
}

public class Demo5 {
public static void main(String[] args) {
List<Integer> mylist = new ArrayList<>();
mylist.add(3); mylist.add(2); mylist.add(5); mylist.add(1); mylist.add(4);

mylist.stream()
.filter(MyNumber::isEven)
.forEach(System.out::println);

System.out.println("--------------------");

mylist.stream()
.filter(MyNumber::isOdd)
.forEach(System.out::println);
System.out.println("--------------------");

mylist.stream()
.filter(a -> a % 2 == 0)
.forEach(a -> System.out.println(a));
System.out.println("--------------------");
```

```
mylist.stream()
.filter(a -> a % 2 != 0)
.forEach(a -> System.out.println(a));
System.out.println("--------------------");
}
}
```

Demo7.java

```
package com.jlcindia.demo7;

import java.util.*;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo6 {
public static void main(String[] args) {

List<Integer> mylist = new ArrayList<>();
mylist.add(3);
mylist.add(2);
mylist.add(5);
mylist.add(1);
mylist.add(4);
mylist.add(6);
mylist.add(7);
mylist.add(8);

mylist.stream()
.filter(a -> a % 2 == 0)
.map(a ->  a* a)
.forEach(a -> System.out.println(a));

System.out.println("--------------------");

mylist.stream()
.filter(a -> a % 2 != 0)
.map(a ->  a* a)
.forEach(a -> System.out.println(a));
}
}
```

5. Functional Interfaces

- Interface which contains Only Single Abstract Method (SAM) is called Functional Interface.
- You can mark the Functional Interface optionally with **@FunctionalInterface**
- Functional Interface can have the following
    - One Abstract Method
    - Multiple default methods
    - Multiple static methods
    - Multiple private methods **(from Java9)**
    - Multiple private static methods **(from Java9)**

- You need a Functional Interface to write the Lambda Expressions.
- You can define the Functional Interface in your own or you can make use of Built-In Functional Interface
- Following are the List of most important of Built-In Functional Interfaces
    - Predicate
    - BiPredicate
    - Function
    - UnaryOperator
    - BiFunction
    - BinaryOperator
    - Consumer
    - BiConsumer
    - Suppiler
    - Primitive Functional Interfaces

Demo1: Files Required:

1. Hello.java	2. Demo1.java

1)Hello.java

```
package com.jlcindia.java8.demos;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hello {

        void test(int a,int b) throws ArithmeticException;

}
```

2)Demo1.java

```
package com.jlcindia.java8.demos;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo1 {
public static void main(String[] args) {

Hello hello=  (a,b) -> {
System.out.println("Lambda Code Starts");

try {
int result = a/b;
System.out.println("Result is "+ result);
}catch(Exception ex) {
ex.printStackTrace();
}

System.out.println("Lambda Code Ends");
};

hello.test(50, 0);
}
}
```

Demo2: Files Required:

1. Hello.java	2. Demo2.java

1)Hello.java

```
package com.jlcindia.java8.demos;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
@FunctionalInterface
public interface Hello {
        void test(int a,int b) throws ArithmeticException;
}
```

2)Demo2.java

```
package com.jlcindia.java8.demos;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo2 {
public static void main(String[] args) {
System.out.println("main Begin");

Hello hello=  (a,b) -> {
System.out.println("Lambda Begin");
int result = a/b;
System.out.println("Result is "+ result);
System.out.println("Lambda End");
};

//hello.test(50, 0);

try{
hello.test(50, 0);
}catch(Exception ex) {
ex.printStackTrace();
}
System.out.println("main End");
}
}
```

5.1. Predicate Functional Interfaces

- **Predicate** Functional Interface
  - Takes One Input Parameter
  - Returns Boolean after processing.
- **Predicate** Functional Interface has the following methods.

```
abstract boolean test(T);                    //SAM
static <T> Predicate<T> isEqual(Object); //Static
Predicate<T> negate();                       //Default
Predicate<T> and(Predicate<? super T>);//Default
Predicate<T> or(Predicate<? super T>);//Default
```

Demo3: Files Required:

1. Demo3.java	

Demo3.java

```
package com.jlcindia.java8.demos;

import java.util.function.Predicate;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo3 {
public static void main(String[] args) {

Predicate<Integer> predicate1 = (num) -> {
System.out.println(num);
return num % 2 == 0;
};

boolean mybool = predicate1.test(19);
System.out.println(mybool);

mybool = predicate1.test(28);
System.out.println(mybool);

Predicate<Integer> predicate2 = (num) -> num % 2 != 0;
```

```
mybool = predicate2.test(19);
System.out.println(mybool);
mybool = predicate2.test(28);
System.out.println(mybool);
}
}
```

Demo4: Files Required:

1. Demo4.java	

Demo4.java

```
package com.jlcindia.java8.demos;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo4 {
public static void main(String[] args) {

Predicate<Integer> predicate1 = (num) -> {
System.out.println(num);
return num % 2 == 0;
};

Predicate<Integer> predicate2 = (num) -> num % 2 != 0;

List<Integer> mylist1 = new ArrayList<>();
mylist1.add(20);      mylist1.add(21);      mylist1.add(22);   mylist1.add(23);
mylist1.add(24);      mylist1.add(25);      mylist1.add(26);

System.out.println(mylist1);
mylist1.removeIf(predicate1);
System.out.println(mylist1);

System.out.println("-------------------------");
List<Integer> mylist2 = new ArrayList<>();
mylist1.add(20);      mylist1.add(21);      mylist1.add(22);   mylist1.add(23);
mylist1.add(24);      mylist1.add(25);      mylist1.add(26);
```

```
System.out.println(mylist2);
mylist2.removeIf(predicate2);
System.out.println(mylist2);
}
}
```

Demo5: Files Required:

1. Demo5.java	

Demo5.java

```
package com.jlcindia.java8.demos;

import java.util.ArrayList;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo5 {
public static void main(String[] args) {

List<Integer> mylist1 = new ArrayList<>();
mylist1.add(20);      mylist1.add(21);      mylist1.add(22);   mylist1.add(23);
mylist1.add(24);      mylist1.add(25);      mylist1.add(26);

System.out.println(mylist1);
mylist1.removeIf((num) -> num %2 ==0); //IMP
System.out.println(mylist1);

System.out.println("-------------------------");
List<Integer> mylist2 = new ArrayList<>();
mylist1.add(20);      mylist1.add(21);      mylist1.add(22);   mylist1.add(23);
mylist1.add(24);      mylist1.add(25);      mylist1.add(26);

System.out.println(mylist2);
mylist2.removeIf((num) -> num %2 !=0);  //IMP
System.out.println(mylist2);

}
}
```

Demo6: Files Required:

1. Demo6.java	

Demo6.java

```
package com.jlcindia.java8.demos;

import java.util.function.Predicate;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo6 {
public static void main(String[] args) {

Predicate<String>  predicate1 = Predicate.isEqual("Hello Guys");

boolean mybool = predicate1.test("Hello Guys");
System.out.println(mybool);
mybool = predicate1.test("Hai Guys");
System.out.println(mybool);

Predicate<Integer>  predicate2 = Predicate.isEqual(99);

mybool = predicate2.test(99);
System.out.println(mybool);
mybool = predicate2.test(88);
System.out.println(mybool);
System.out.println("----------");

Predicate<Integer> predicate3 = (num) -> num % 2 == 0;
Predicate<Integer> predicate4 = (num) -> num % 2 != 0;

mybool = predicate3.test(28);
System.out.println(mybool); //T
mybool = predicate3.negate().test(28);
System.out.println(mybool); //F

mybool = predicate4.test(19);
System.out.println(mybool); //T
mybool = predicate4.negate().test(19);
System.out.println(mybool); //F
}
}
```

Demo7: Files Required:

1. Demo7.java	

Demo7.java

```
package com.jlcindia.java8.demos;

import java.util.function.Predicate;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo7 {
public static void main(String[] args) {

Predicate<Integer> predicate1 =  (num) ->{
System.out.println("Predicate 1");
return num % 2 == 0;
};

Predicate<Integer> predicate2 = (num) -> {
System.out.println("Predicate 2");
return num % 2 != 0;
};

Predicate<Integer> predicate3 = (num) -> {
System.out.println("Predicate 3");
return  num >= 25 && num <= 50;
};

// Check whether Number is Even and between 25 and 50
boolean mybool = predicate1.and(predicate3).test(28);
System.out.println(mybool);

mybool = predicate1.and(predicate3).test(19);
System.out.println(mybool);

// Check whether Number is Odd  and between 25 and 50
mybool = predicate2.and(predicate3).test(29);
System.out.println(mybool);

mybool = predicate2.and(predicate3).test(19);
System.out.println(mybool);
```

```
// Check whether Number is Even  or  between 25 and 50

mybool = predicate1.or(predicate3).test(29);
System.out.println(mybool);

mybool = predicate1.or(predicate3).test(28);
System.out.println(mybool);

mybool = predicate1.and(predicate3).test(18);
System.out.println(mybool);

// Check whether Number is Odd  or  between 25 and 50

mybool = predicate2.or(predicate3).test(29);
System.out.println(mybool);

mybool = predicate2.or(predicate3).test(28);
System.out.println(mybool);

mybool = predicate2.and(predicate3).test(18);
System.out.println(mybool);

}

}
```

Interview Questions:

Q1) What is a Function Interface?

Ans:

Q2) What is the annottaion to mark on the Function Interface?

Ans:

Q3) Is it mandatory to mark @FunctionalInterface?

Ans:

Q4) Can I write two abstract methods in One Function Interface?

Ans:

Q5) Can I have default methods in Function Interface?

Ans:

Q6) Can I have static methods in Function Interface?

Ans:

Q7) Can I have private methods in Function Interface? (Java 9)

Ans:

Q8) Can I have private static methods in Function Interface? (Java 9)

Ans:

Q9) Can I write Lambda exp without Function Interface?

Ans:

Q10) Can we change the return type or parameters of SAM during implementation?

Ans:

Q11) How can I throws exception at method level using Functional Interface?

Ans:

Q12) What are the built-IN functional Interfaces?

Ans:

Q13) Which of the following **Hello** are Valid Function Interfaces?

1)	interface **Hello** {
	
	}	

2)	interface **Hello** {
	void show();
	}	

3)	interface Hai {
	void show();
	}
	
	Interface **Hello** extends Hai{
	
	}	

4)	interface **Hello** {
	void show();
	default void display(){
	System.out.println("OK");
	}	

5)	interface **Hello** {
	void m1();
	void m2();
	}	

5.2. BiPredicate Functional Interfaces

- **BiPredicate** Functional Interface
  - Takes Two Input Parameters
  - Returns Boolean after processing.
- **BiPredicate** Functional Interface has the following methods.

abstract boolean **test**(T, U);

BiPredicate<T, U> **and**(BiPredicate<? super T, ? super U>);

BiPredicate<T, U> **negate**();

BiPredicate<T, U> **or**(BiPredicate<? super T, ? super U>);

Demo8: Files Required:

1. Demo8.java	

Demo8.java

```
package com.jlcindia.demos;

import java.util.function.BiPredicate;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo8 {
public static void main(String[] args) {

BiPredicate<Integer,Integer> predicate1 = (num1,num2) -> num1>num2;

boolean mybool = predicate1.test(10,20);
System.out.println(mybool);

BiPredicate<Integer,Integer> predicate2 = (num1,num2) -> num1<num2;
mybool = predicate2.test(10,20);
System.out.println(mybool);

}
}
```

5.3. Function Functional Interfaces

- **Function** Functional Interface
  - Takes One Input Parameter
  - Returns Output after processing.
- **Function** Functional Interface has the following methods.

```
abstract R apply(T); //SAM
static <T> Function<T, T> identity();
<V> Function<V, R> compose(Function<? super V, ? extends T>);
<V> Function<T, V> andThen(Function<? super R, ? extends V>);
```

Demo9: Files Required:

1. Demo9.java	

Demo9.java

```
package com.jlcindia.demos;

import java.util.function.Function;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo9 {
public static void main(String[] args) {

Function<String, String> fun1 = (input) -> input.toUpperCase();

String output1 = fun1.apply("Srinivas Dande");
System.out.println(output1);

Function<String, Integer> fun2 = (input) -> Integer.parseInt(input);

Integer output2 = fun2.apply("99");
System.out.println(output2);

Function<Integer, String> fun3 = (input) ->  String.valueOf(input);

String output3 = fun3.apply(99);
System.out.println(output3);
```

```
Function<String, String> fun4 = input -> input;

String output4= fun4.apply("Hello Guys");
System.out.println(output4);

Function<String, String> fun5 = Function.identity();

String output5= fun5.apply("Hello Guys");
System.out.println(output5);
}
}
```

Demo10: Files Required:

1. Demo10.java	

Demo10.java

```
package com.jlcindia.demos;

import java.util.function.Function;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo10 {
public static void main(String[] args) {

Function<Integer, Integer> fun1 = (num) -> {
System.out.println("Multiply by 2");
return num * 2;
};

Function<Integer, Integer> fun2 = (num) -> {
System.out.println("Multiply by 3");
return num * 3;
};

System.out.println(fun1.apply(5));
System.out.println(fun2.apply(5));

int result2= fun1.andThen(fun2).apply(10);
//fun1.apply(10) => 10 * 2 => 20
//fun2.apply(20) => 3 * 20 => 60
System.out.println(result2); //60
```

```
int result1= fun1.compose(fun2).apply(10);
//fun2.apply(10) => 10 * 3 => 30
//fun1.apply(30) =>30 * 2 =>  60
System.out.println(result1); //60
}
}
```

Demo11: Files Required:

1. Demo11.java	

Demo11.java

```
package com.jlcindia.demo1;

import java.util.function.Function;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo11 {
public static void main(String[] args) {

Function<String, Integer> fun1 = (input) -> {
System.out.println("Converting String to Integer");
return Integer.parseInt(input);
};

Function<Integer, Integer> fun2 = (num) -> {
System.out.println("add 10 to the number");
return num + 10;
};

int result2= fun1.andThen(fun2).apply("10");
System.out.println(result2); //20

int result1= fun2.compose(fun1).apply("10");
System.out.println(result1); //20
}
}
```

5.4. UnaryOperator Functional Interfaces

- **UnaryOperator** Functional Interface is sub type of Function which allows you specify only One Type for both parameter and Return Value.
- **UnaryOperator** Functional Interface
  - Takes One Input Parameter
  - Returns Output after processing.
- **UnaryOperator** Functional Interface has the following methods.

```
abstract R apply(T); //SAM
public static <T> UnaryOperator<T> identity();
<V> Function<V, R> compose(Function<? super V, ? extends T>);
<V> Function<T, V> andThen(Function<? super R, ? extends V>);
```

Demo12.java

```
package com.jlcindia.demos;

import java.util.function.Function;
import java.util.function.UnaryOperator;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo12 {
public static void main(String[] args) {

Function<String,String> fun1 = (input) -> input.toUpperCase();
System.out.println(fun1.apply("Hello"));

UnaryOperator<String> unary1 = (input) -> input.toUpperCase();
System.out.println(unary1.apply("Hello"));

Function<Integer, Integer> fun2 = (num) -> num * 2;
System.out.println(fun2.apply(50));

UnaryOperator<Integer> unary2 = (num) -> num * 2;
System.out.println(unary2.apply(50));
}
}
```

5.5. BiFunction Functional Interfaces

- **BiFunction** Functional Interface
  - Takes Two Input Parameters
  - Returns Output after processing.
- **BiFunction** Functional Interface has the following methods.

```
abstract R apply(T, U); //SAM
<V> BiFunction<T, U, V> andThen(Function<? super R, ? extends V>);
```

Demo13.java

```
package com.jlcindia.demos;

import java.util.function.BiFunction;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo13 {
public static void main(String[] args) {

BiFunction<String,String,String> fun1 = (input1,input2) -> input1 + input2;
String output = fun1.apply("Hello"," Guys");
System.out.println(output);

BiFunction<Integer, Integer,Integer> fun2 = (num1,num2) -> num1 * num2;
System.out.println(fun2.apply(5,25));

BiFunction<Integer, Integer,String> fun3 = (num1,num2) -> {
int result= num1 * num2;
String str = "Result is "+result;
return str;
};
System.out.println(fun3.apply(5,25));
}
}
```

5.6. BinaryOperator Functional Interfaces

- **BinaryOperator** Functional Interface is sub type of BiFunction which allows you specify only One Type for both parameter and Return Value.
- **BinaryOperator** Functional Interface
  - Takes Two Input Parameters
  - Returns Output after processing.
- **BinaryOperator** Functional Interface has the following methods.

```
static <T> BinaryOperator<T> minBy(Comparator<? super T>);
static <T> BinaryOperator<T> maxBy(Comparator<? super T>);
```

Demo14.java

```
package com.jlcindia.demos;

import java.util.function.BiFunction;
import java.util.function.BinaryOperator;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo14 {

public static void main(String[] args) {

BiFunction<String,String,String> fun1 = (input1,input2) -> input1 + input2;
String output = fun1.apply("Hello"," Guys");
System.out.println(output);

BinaryOperator<String> binary1 = (input1,input2) -> input1 + input2;
String output1 = binary1.apply("Hello"," Guys");
System.out.println(output1);

BiFunction<Integer, Integer,Integer> fun2 = (num1,num2) -> num1 * num2;
System.out.println(fun2.apply(5,25));

BinaryOperator<Integer> binary2 = (num1,num2) -> num1 * num2;
System.out.println(binary2.apply(5,25));
}
}
```

5.7. Consumer and BiConsumer Interfaces

- **Consumer** Functional Interface
  - Takes One Input Parameter
  - No Return Value.
- **Consumer** Functional Interface has the following methods.

  abstract void accept(T); //SAM

  Consumer<T> andThen(Consumer<? super T>);

- **BiConsumer** Functional Interface
  - Takes Two Input Parameters
  - No Return Value.
- **BiConsumer** Functional Interface has the following methods.

  abstract void accept(T, U);

  BiConsumer<T, U> andThen(BiConsumer<? super T, ? super U>);

Demo15.java

```
package com.jlcindia.demos;

import java.util.function.BiConsumer;
import java.util.function.Consumer;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo15 {
public static void main(String[] args) {

Consumer<String> consumer1 = (input) -> System.out.println(input.toUpperCase());

consumer1.accept("Hello");
consumer1.accept("Srinivas");
consumer1.accept("Hai");

BiConsumer<String,String> consumer2 = (input1,input2) ->
System.out.println(input1+input2);
consumer2.accept("Hello"," Guys");
}
}
```

5.8. Supplier Functional Interfaces

- **Supplier** Functional Interface
  - Takes No Input Parameters
  - Returns Any Type .
- **Supplier** Functional Interface has the following methods.

  abstract T get();

Demo16.java

```
package com.jlcindia.demos;

import java.time.DayOfWeek;
import java.time.LocalDate;
import java.util.function.Supplier;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo16 {
public static void main(String[] args) {

Supplier<String> supplier1 = () -> "Hello Guys, How are you?";

String str = supplier1.get();
System.out.println(str);

Supplier<Integer> supplier2 = () ->  LocalDate.now().getDayOfMonth();
System.out.println(supplier2.get());

Supplier<DayOfWeek> supplier3 = () ->  LocalDate.now().getDayOfWeek();
DayOfWeek dow = supplier3.get();

System.out.println(dow);
System.out.println(dow.getValue());

}
}
```

6. Optional Class

- **Optional is a container object** which may or may not contain a non-null value.
- **Purpose of Optional class** is to provide a type-level solution for representing optional values instead of null references.
- **Optional class** has the following methods.

```
static <T> Optional<T> empty();
static <T> Optional<T> of(T);
static <T> Optional<T> ofNullable(T);

T get();
boolean isPresent();
T orElse(T);

void ifPresent(Consumer<? super T>);
Optional<T> filter(Predicate<? super T>);

Optional<U> map(Function<? super T, ? extends U>);
Optional<U> flatMap(Function<? super T, Optional<U>>);

T orElseGet(Supplier<? extends T>);
T orElseThrow(Supplier<? extends X>) throws X;
```

Demo1.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo1 {
public static void main(String[] args) {

// Optional<String> myopts = new Optional<String>();

//Optional.empty() method
Optional<String> myopts= Optional.empty();

System.out.println("1. "+myopts);
System.out.println("2. "+myopts.orElse("Hello Guys"));
System.out.println("3. "+myopts);
System.out.println("4. "+myopts.isPresent());

//System.out.println("5. "+myopts.get());

if(myopts.isPresent()) {
System.out.println("5. "+myopts.get());
}else {
System.out.println("6. No value Found");
}

}
}
```

Demo2.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
```

```
public class Demo2 {
public static void main(String[] args) {

//Optional.of() method
String str ="Srinivas";
Optional<String> myopts= Optional.of(str);

System.out.println("1. "+myopts);
System.out.println("2. "+myopts.orElse("Hello Guys"));
System.out.println("3. "+myopts);
System.out.println("4. "+myopts.isPresent());
//System.out.println("5. "+myopts.get());

if(myopts.isPresent()) {
System.out.println("5. "+myopts.get());
}else {
System.out.println("6. No value Found");
}

}
}
```

Demo3.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo3 {

public static void main(String[] args) {

//Optional.of()  with null
String str =null;
Optional<String> myopts= Optional.of(str);
System.out.println(myopts);

}
}
```

Demo4.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo4 {
public static void main(String[] args) {

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

System.out.println("1. "+myopts);
System.out.println("2. "+myopts.orElse("Hello Guys"));
System.out.println("3. "+myopts);
System.out.println("4. "+myopts.isPresent());

if(myopts.isPresent()) {
System.out.println("5. "+myopts.get());
}else {
System.out.println("6. No value Found");
}

}
}
```

Demo5.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo5 {
public static void main(String[] args) {
```

```
//Optional. ofNullable() method
String str ="Srinivas";
Optional<String> myopts= Optional.ofNullable(str);

System.out.println("1. "+myopts);
System.out.println("2. "+myopts.orElse("Hello Guys"));
System.out.println("3. "+myopts);
System.out.println("4. "+myopts.isPresent());

if(myopts.isPresent()) {
System.out.println("5. "+myopts.get());
}else {
System.out.println("6. No value Found");
}

}
}
```

Demo6.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo6 {
public static void main(String[] args) {

//isPresent() Vs ifPresent()
String str =null;
//String str ="Srinivas";
Optional<String> myopts= Optional.ofNullable(str);

if(myopts.isPresent()) {
System.out.println(myopts.get());
}

myopts.ifPresent(input -> System.out.println(input));

if(myopts.isPresent()) {
System.out.println(myopts.get().toUpperCase());
}
```

```
myopts.ifPresent(input -> System.out.println(input.toUpperCase()));

System.out.println("--------------------");

System.out.println("Done!!!");
}
}
```

Demo7.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo7 {
public static void main(String[] args) {

//filter() method
String str1 =null;
Optional<String> myopts1 = Optional.ofNullable(str1);

Optional<String> myopts4= myopts1.filter(input -> input.contains("Sri"));
System.out.println("1 . "+myopts4);

String str2 ="Srinivas";
Optional<String> myopts2 = Optional.ofNullable(str2);

Optional<String> myopts5= myopts2.filter(input -> input.contains("Sri"));
System.out.println("2 . "+myopts5);

String str3 ="Hello Guys";
Optional<String> myopts3 = Optional.ofNullable(str3);

Optional<String> myopts6= myopts3.filter(input -> input.contains("Sri"));
System.out.println("3 . "+myopts6);

System.out.println("Done!!!");

}
}
```

Demo8.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo8 {
public static void main(String[] args) {

//map() method
String str1 ="Srinivas";
Optional<String> myopts1 = Optional.ofNullable(str1);
System.out.println("1. "+myopts1);

Optional<String> myopts2 = myopts1.map(input -> input);
System.out.println("2. "+myopts2);

Optional<String> myopts3 = myopts1.map(input -> input.toUpperCase());
System.out.println("3. "+myopts3);

Optional<String> myopts4 = myopts3.map(input -> new
StringBuilder(input).reverse().toString());

System.out.println("4. "+myopts4);

String mystr = null;
Optional<String> myopts = Optional.ofNullable(mystr);
System.out.println("5. "+myopts);

Optional<String> myopts5 = myopts.map(input -> input.toUpperCase());
System.out.println("6. "+myopts5);

System.out.println("Done!!!");
}
}
```

Demo9.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo9 {
public static void main(String[] args) {

String str1 ="Srinivas";
Optional<Optional<String>> myopts1 = Optional.of(Optional.ofNullable(str1));

System.out.println("1. "+myopts1);
System.out.println("2. "+myopts1.map(input -> input));
System.out.println("3. "+myopts1.flatMap(input -> input));

System.out.println("------------");
Optional<Optional<String>> x =          myopts1.map(
input1 -> input1.map(input2 -> input2.toUpperCase())
);
System.out.println("x = "+x);

Optional<String> y =    myopts1.flatMap(
input1 -> input1.map(input2 -> input2.toUpperCase())
);

System.out.println("y = "+y);
System.out.println("------------");

String str2 ="jlc";
Optional<Optional<Optional<String>>> myopts2 =
Optional.of(Optional.of(Optional.ofNullable(str2)));

Optional<Optional<Optional<String>>>  aa = myopts2.map(input1-> input1);
System.out.println(aa);

System.out.println("Done!!!");
}
}
```

Demo10.java

```
package com.jlcindia.demos;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo10 {
public static void main(String[] args) {

String str ="jlc";
Optional<Optional<Optional<String>>> myopts =
Optional.of(Optional.of(Optional.ofNullable(str)));

System.out.println("1. "+myopts);

Optional<Optional<Optional<String>>>  aa =
myopts.map(input1-> input1.map(input2->input2.map(input3-> input3.toUpperCase())));
System.out.println("2. "+aa);

Optional<Optional<String>>  bb =
myopts.flatMap(input1-> input1.map(input2->input2.map(input3-> input3.toUpperCase())));
System.out.println("3. "+bb);

Optional<String>  cc =
myopts.flatMap(input1-> input1.flatMap(input2->input2.map(input3->
input3.toUpperCase())));
System.out.println("4. "+cc);

System.out.println("Done!!!");

}
}
```

Demo11.java

```
package com.jlcindia.demos;

import java.util.NoSuchElementException;
import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo11 {

public static void main(String[] args) {

String str1 ="Srinivas";
Optional<String> myopts= Optional.ofNullable(str1);

System.out.println("1. "+myopts);
System.out.println("2. "+myopts.orElse("Hello Guys"));
System.out.println("3. "+myopts.orElseGet( () -> "Ok Guys"));
System.out.println("4. "+myopts.orElseThrow(NoSuchElementException::new));

String str2 =null;
myopts= Optional.ofNullable(str2);

System.out.println("1. "+myopts);
System.out.println("2. "+myopts.orElse("Hello Guys"));
System.out.println("3. "+myopts.orElseGet( () -> "Ok Guys"));
System.out.println("4. "+myopts.orElseThrow(NoSuchElementException::new));

}
}
```

Data Model without Optional Class

Files Required:

1. Trainer.java	2. Course.java
3. Student.java	4. Demo1.java
5. Demo2.java	

1) Trainer.java

```
package com.jlcindia.demos1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Trainer {
private String trainerName;
private String trainerEmail;
private String trainerPhone;

public Trainer() {}

public Trainer(String trainerName, String trainerEmail, String trainerPhone) {
super();
this.trainerName = trainerName;
this.trainerEmail = trainerEmail;
this.trainerPhone = trainerPhone;
}

//Setters and Getters

}
```

2) Course.java

```
package com.jlcindia.demos1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Course {
private String courseName;
private double coursePrice;
private Trainer trainer;
```

```
public Course() {}

public Course(String courseName, double coursePrice, Trainer trainer) {
super();
this.courseName = courseName;
this.coursePrice = coursePrice;
this.trainer = trainer;
}

//Setters and Getters

}
```

3) Student.java

```
package com.jlcindia.demos1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Student {
private String studentName;
private String studentEmail;
private long studentPhone;
private Course course;

public Student() {
}

public Student(String studentName, String studentEmail, long studentPhone, Course course) {
super();
this.studentName = studentName;
this.studentEmail = studentEmail;
this.studentPhone = studentPhone;
this.course = course;
}

//Setters and Getters

}
```

4) Demo1.java

```
package com.jlcindia.demos1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo1 {
public static void main(String[] args) {

Trainer trainer1 = new Trainer("Srinivas","sri@jlc","12345");
Course course1=new Course("DevOps",15000,trainer1);
Student student1=new Student("Hello", "hello@gmail", 111, course1);

// I have access to Only Student Object  and  want trainer Email and Phone
String trainerName = student1.getCourse().getTrainer().getTrainerName();
String trainerEmail = student1.getCourse().getTrainer().getTrainerEmail();
String trainerPhone = student1.getCourse().getTrainer().getTrainerPhone();

System.out.println(trainerName);
System.out.println(trainerEmail);
System.out.println(trainerPhone);
System.out.println("------------------------------");

Course course2=new Course("Java8 New",500,null);
Student student2=new Student("Hai", "hai@gmail",222, course2);

// I have access to Only Student Object  and  want trainer Email and Phone
trainerName = student2.getCourse().getTrainer().getTrainerName();
trainerEmail = student2.getCourse().getTrainer().getTrainerEmail();
trainerPhone = student2.getCourse().getTrainer().getTrainerPhone();

System.out.println(trainerName);
System.out.println(trainerEmail);
System.out.println(trainerPhone);

Student student3=new Student("Sri", "sri@gmail",333, null);

// I have access to Only Student Object  and  want trainer Email and Phone
trainerName = student3.getCourse().getTrainer().getTrainerName();
trainerEmail = student3.getCourse().getTrainer().getTrainerEmail();
trainerPhone = student3.getCourse().getTrainer().getTrainerPhone();
```

```
System.out.println(trainerName);
System.out.println(trainerEmail);
System.out.println(trainerPhone);
}
}
```

5) Demo2.java

```
package com.jlcindia.demos1;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo2 {

public static void main(String[] args) {

//        Trainer trainer = new Trainer("Srinivas", "sri@jlc", "12345");
//        Course course = new Course("DevOps", 15000, null);
Student mystudent = new Student("Hello", "hello@gmail", 111, null);

// I have access to Only Student Object
// I want trainer Email and Phone
String trainerName = "No Name available";
String trainerEmail = "No Email Provided";
String trainerPhone = "No Phone Provided";

if (mystudent != null) {
Course mycourse = mystudent.getCourse();
if (mycourse != null) {
Trainer mytrainer = mycourse.getTrainer();
if (mytrainer != null) {
trainerName = mytrainer.getTrainerName();
trainerEmail = mytrainer.getTrainerEmail();
trainerPhone = mytrainer.getTrainerPhone();
}
}
}
System.out.println(trainerName);
System.out.println(trainerEmail);
System.out.println(trainerPhone);
}
}
```

Data Model with Optional Class

Files Required:

1. Trainer.java	2. Course.java
3. Student.java	4. Demo1.java
5. Demo2.java	6. Demo3.java
7. Demo4.java	

1) Trainer.java

```
package com.jlcindia.demos3;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Trainer {
private String trainerName;
private String trainerEmail;
private String trainerPhone;
public Trainer() {}
public Trainer(String trainerName, String trainerEmail, String trainerPhone) {
this.trainerName = trainerName;
this.trainerEmail = trainerEmail;
this.trainerPhone = trainerPhone;
}
//Setters and Getters
}
```

2) Course.java

```
package com.jlcindia.demos3;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Course {
        private String courseName;
        private double coursePrice;
        private Optional<Trainer> trainer;

        public Course() {}
```

```
public Course(String courseName, double coursePrice, Optional<Trainer> trainer) {
super();
this.courseName = courseName;
this.coursePrice = coursePrice;
this.trainer = trainer;
}

//Setters and Getters

}
```

3) Student.java

```
package com.jlcindia.demos3;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Student {
private String studentName;
private String studentEmail;
private long studentPhone;
private Optional<Course> course;

public Student() {
}

public Student(String studentName, String studentEmail, long studentPhone,
Optional<Course> course) {
super();
this.studentName = studentName;
this.studentEmail = studentEmail;
this.studentPhone = studentPhone;
this.course = course;
}

//Setters and Getters

}
```

4) Demo1.java

```
package com.jlcindia.demos3;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo1 {

public static void main(String[] args) {

Trainer trainer = new Trainer("Srinivas","sri@jlc","12345");
Course course=new Course("DevOps",15000,Optional.ofNullable(trainer));
Student student=new Student("Hello", "hello@gmail", 111, Optional.ofNullable(course));

Optional<Student> mystudent = Optional.ofNullable(student);

Optional<Course> mycurse =mystudent.flatMap(mystu -> mystu.getCourse());
Optional<Trainer> mytrainer = mycurse.flatMap(mycou -> mycou.getTrainer());

Optional<String> emailOtps= mytrainer.map(mytra -> mytra.getTrainerEmail());
String email = emailOtps.orElse("No Email is given");

Optional<String> nameOpts= mytrainer.map(mytra -> mytra.getTrainerName());
String name = nameOpts.orElse("No Name is given");

Optional<String> phoneOpts= mytrainer.map(mytra -> mytra.getTrainerPhone());
String phone = phoneOpts.orElse("No Phone is given");

System.out.println(name);
System.out.println(email);
System.out.println(phone);

}
}
```

5) Demo2.java

```
package com.jlcindia.demos3;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo2 {

public static void main(String[] args) {

Trainer trainer = new Trainer("Srinivas","sri@jlc","12345");
Course course=new Course("DevOps",15000,Optional.ofNullable(trainer));
Student student=new Student("Hello", "hello@gmail", 111, Optional.ofNullable(course));

Optional<Student> mystudent = Optional.ofNullable(student);

String name= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerName())))
.orElse("NO Name is Provided");

String email= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerEmail())))
.orElse("NO Email is Provided");

String phone= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerPhone())))
.orElse("NO Phone is Provided");

System.out.println(name);
System.out.println(email);
System.out.println(phone);
}
}
```

6) Demo3.java

```
package com.jlcindia.demos3;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo3 {
public static void main(String[] args) {

Trainer trainer =null;
Course course=new Course("DevOps",15000,Optional.ofNullable(trainer));
Student student=new Student("Hello", "hello@gmail", 111, Optional.ofNullable(course));

Optional<Student> mystudent = Optional.ofNullable(student);

String name= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerName()))).orElse("NO Name is Provided");

String email= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerEmail()))).orElse("NO Email is Provided");

String phone= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerPhone()))).orElse("NO Phone is Provided");

System.out.println(name);
System.out.println(email);
System.out.println(phone);

System.out.println("--------------------");

String cname= mystudent.flatMap(
mystu -> mystu.getCourse().map(
mycou-> mycou.getCourseName())).orElse("NO Course is Provided");
```

```
double price= mystudent.flatMap(
mystu -> mystu.getCourse().map(
mycou-> mycou.getCoursePrice())).orElse(25000.0);

System.out.println(cname);
System.out.println(price);

}
}
```

7) Demo4.java

```
package com.jlcindia.demos3;

import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo4 {
public static void main(String[] args) {

Course course=null;
Student student=new Student("Hello", "hello@gmail", 111, Optional.ofNullable(course));

Optional<Student> mystudent = Optional.ofNullable(student);

String name= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerName()))).orElse("NO Name is Provided");

String email= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerEmail()))).orElse("NO Email is Provided");

String phone= mystudent.flatMap(
mystu -> mystu.getCourse().flatMap(
mycou-> mycou.getTrainer().map(
mytra-> mytra.getTrainerPhone()))).orElse("NO Phone is Provided");
```

```
System.out.println(name);
System.out.println(email);
System.out.println(phone);

System.out.println("--------------------");

String cname= mystudent.flatMap(
mystu -> mystu.getCourse().map(
mycou-> mycou.getCourseName())).orElse("NO Course is Provided");

double price= mystudent.flatMap(
mystu -> mystu.getCourse().map(
mycou-> mycou.getCoursePrice())).orElse(25000.0);

System.out.println(cname);
System.out.println(price);

}
}
```

7. Working with Streams

Demo1.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo1 {
public static void main(String[] args) {

List<Integer> numsList = Arrays.asList(1,2,3,4,5,6,7,8,9);
System.out.println(numsList);
System.out.println("--------------------");

Stream<Integer> mystream = numsList.stream(); //1
Stream<Integer> oddStream = mystream.filter(num -> num % 2 !=0);//2
Stream<Integer> squareStream = oddStream.map(num -> num * num);//2
squareStream.forEach(System.out::println); //3

System.out.println("--------------------");

numsList.stream().filter(num -> num % 2 ==0)
.map(num -> num * num)
.forEach(System.out::println);

System.out.println("--------------------");
System.out.println(numsList);
}
}
```

Demo2.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;
/*
```

```
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo2 {
public static void main(String[] args) {

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

Stream<Integer> mystream1 = numsList.stream(); //1
mystream1.forEach(System.out::println); //3
//mystream1.forEach(System.out::println); //3 //Exception

System.out.println("----------");
Stream<Integer> mystream2 = numsList.stream(); //1
mystream2.forEach(System.out::println); //3

System.out.println("----------");
Stream<Integer> mystream3 = numsList.stream(); //1
mystream3.forEach(System.out::println); //3
}
}
```

Demo3.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo3 {
public static void main(String[] args) {

List<Integer> numsList = Arrays.asList(1,2,3,4,5,6,7,8,9);

numsList.stream().forEach(System.out::println);
System.out.println("----------");
numsList.parallelStream().forEach(System.out::println);
}
}
```

Demo4.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.Collection;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo4 {
public static void main(String[] args) {

List<Integer> list1 = Arrays.asList(1,2,3);
List<Integer> list2 = Arrays.asList(10,20,30);

List<List<Integer>> mylist = Arrays.asList(list1,list2);

mylist.stream().forEach(System.out::println);

System.out.println("----------------------");

mylist.stream()
.map(Collection::stream)
.forEach(System.out::println);

System.out.println("----------------------");

mylist.stream()
.map(Collection::stream)
.flatMap(input -> input)
.forEach(System.out::println);
}
}
```

Demo5.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
```

```
* */
public class Demo5 {
public static void main(String[] args) {

List<Integer> mylist = Arrays.asList(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);

mylist.stream().limit(10).forEach(System.out::println);

System.out.println("----------------------");
mylist.stream()
.limit(10)
.filter(num -> num%2!=0)
.map(num -> num * num)
.forEach(System.out::println); //3

}
}
```

Demo6.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo6 {
public static void main(String[] args) {

List<Integer> mylist = Arrays.asList(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);

mylist.stream().skip(4).limit(9).forEach(System.out::println);

System.out.println("----------------------");
mylist.stream()    //1
.skip(4)
.limit(9)
.filter(num -> num%2!=0)
.map(num -> num * num)
.forEach(System.out::println); //3
}
}
```

Demo7.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo7 {
public static void main(String[] args) {

List<Integer> mylist = Arrays.asList(1,2,3,4,5,6,7,8,9,10);

System.out.println("----------------------");
mylist.stream()    //1
.skip(3)
.limit(5)
.peek(System.out::println)
.filter(num -> num%2!=0)
.peek(System.out::println)
.map(num -> num * num)
.forEach(System.out::println); //3

}
}
```

Demo8.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo8 {
public static void main(String[] args) {

List<Integer> mylist = Arrays.asList(5,2,7,5,8,3,9,9,4,2,3,7,1,6,1);
```

```
mylist.stream()
.limit(15)
.filter(num -> num%2!=0)
.map(num -> num * num)
.sorted()
.forEach(System.out::println);

System.out.println("----------------------");
mylist.stream()
.limit(15)
.distinct()
.filter(num -> num%2!=0)
.map(num -> num * num)
.sorted()
.forEach(System.out::println);
}
}
```

Demo9.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo9 {
public static void main(String[] args) {

System.out.println("-----1------");
Stream<Integer> myStream1 =  Stream.empty();
myStream1.forEach(System.out::println);

System.out.println("------2------");
Stream<Integer> myStream2 = Stream.of(11,12,13);
myStream2.forEach(System.out::println);

List<Integer> numsList = Arrays.asList(11,12,13,14,15);

System.out.println("-----3------");
Stream<Integer> myStream3 = numsList.stream();
myStream3.forEach(System.out::println);
```

```
System.out.println("------4-------");
Stream<Integer> myStream4 =  numsList.parallelStream();
myStream4.forEach(System.out::println);
}
}
```

Demo10.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo10 {
public static void main(String[] args) {

Stream<Integer> myStream1 = Stream.of(11,12,13);

List<Integer> numsList = Arrays.asList(21,22,23);
Stream<Integer> myStream2 = numsList.stream();

Stream<Integer> myStream3 =Stream.concat(myStream2, myStream1);
myStream3.forEach(System.out::println);
}
}
```

Demo11.java

```
package com.jlcindia.demos;

import java.util.stream.Stream;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo11 {
public static void main(String[] args) {
```

```
Stream.iterate(11, n-> n+1)
.limit(10)
.forEach(System.out::println);

System.out.println("--------------------");
Stream.iterate(11, n-> n+1)
.limit(10)
.filter(num -> num %2==0)
.forEach(System.out::println);

System.out.println("--------------------");
Stream.iterate(11, n-> n+1)
.limit(10)
.filter(num -> num %2!=0)
.forEach(System.out::println);

}
}
```

Demo12.java

```
package com.jlcindia.demos;

import java.util.stream.Stream;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo12 {

public static void main(String[] args) {

Stream.iterate(101, n-> n+1)
.skip(25)
.limit(10)
.forEach(System.out::println);

System.out.println("--------------------");
Stream.iterate(101, n-> n+1)
.skip(25)
.limit(10)
.filter(num -> num %2==0)
.forEach(System.out::println);
```

```
System.out.println("--------------------");
Stream.iterate(101, n-> n+1)
.skip(25)
.limit(10)
.filter(num -> num %2!=0)
.forEach(System.out::println);
}
}
```

Demo13.java

```
package com.jlcindia.demos;

import java.util.Random;
import java.util.stream.Stream;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo13 {
public static void main(String[] args) {

Stream.generate(() -> (new Random()).nextInt(100))
.limit(10)
.forEach(System.out::println);

System.out.println("--------------------");
Stream.generate(() -> (new Random()).nextInt(100))
.limit(10)
.filter(num -> num % 2 != 0)
.map(num -> num * num)
.forEach(System.out::println);

System.out.println("--------------------");
Stream.generate(() -> (new Random()).nextInt(100))
.limit(10)
.filter(num -> num % 2 == 0)
.map(num -> num * num)
.forEach(System.out::println);
System.out.println("Done!!!");
}
}
```

Demo14.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo14 {
public static void main(String[] args) {

List<Integer> numsList = Arrays.asList(1, 2, 3, 4, 5);
int sumResult1 = numsList.stream().reduce(0, (a, b) -> a + b);
System.out.println(sumResult1);
System.out.println("--------------------");

int sumResult2 = numsList.stream().reduce(0, Integer::sum);
System.out.println(sumResult2);
System.out.println("--------------------");

int mulResult = numsList.stream().reduce(1, (a, b) -> a * b);
System.out.println(mulResult);
System.out.println("--------------------");

Integer lastElement = numsList.stream()
.reduce((num1, num2) -> num2)
.orElse(-1);

System.out.println(lastElement);
}
}
```

Demo15.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
```

```
public class Demo15 {
public static void main(String[] args) {

List<Integer> mylist = Arrays.asList(2,3,6,4,5,9,7,8,5);


mylist.stream()
.filter(num -> num%2!=0)
.map(num -> num * num)
.forEach(System.out::println); //3
System.out.println("---------------------");

long count= mylist.stream()
.filter(num -> num%2!=0)
.map(num -> num * num)
.count();
System.out.println(count);
System.out.println("---------------------");

Optional<Integer> maxNum = mylist.stream()
.filter(num -> num%2!=0)
.map(num -> num * num)
.max((num1, num2) -> num1.compareTo(num2));
System.out.println(maxNum);
maxNum.ifPresent(System.out::println);

System.out.println("---------------------");
Optional<Integer> minNum = mylist.stream()
.filter(num -> num%2!=0)
.map(num -> num * num)
.min((num1, num2) -> num1.compareTo(num2));
System.out.println(minNum);
minNum.ifPresent(System.out::println);

System.out.println("---------------------");

System.out.println("Done!!!");
}
}
```

Demo16.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo16 {
public static void main(String[] args) {

List<Integer> numList = Arrays.asList(11,22,33,44,55,66,77,88,99);

Optional<Integer>  mycourse1 =numList.stream().findAny();
mycourse1.ifPresent(System.out::println);
System.out.println("-------------");

Optional<Integer>  mycourse2 =numList.stream().findFirst();
mycourse2.ifPresent(System.out::println);
System.out.println("-------------");

Optional<Integer>  mycourse3 =numList.stream().parallel().findAny();
mycourse3.ifPresent(System.out::println);
System.out.println("-------------");

Optional<Integer>  mycourse4 =numList.stream().parallel().findFirst();
mycourse4.ifPresent(System.out::println);

System.out.println("-------------");

System.out.println("Done!!!");
}
}
```

Demo17.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.stream.Collectors;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo17 {
public static void main(String[] args) {

List<Integer> numsList = Arrays.asList(5, 4, 8, 3, 6, 7, 2, 9);

List<Integer> mylist1 = numsList.stream()
.filter(num -> num % 2 == 0)
.map(num -> num * num)
.sorted()
.collect(Collectors.toList());

System.out.println(mylist1);

System.out.println("----------");
List<Integer> mylist2 = numsList.stream()
.filter(num -> num % 2 == 0)
.map(num -> num * num)
.sorted()
.collect(Collectors.toCollection(LinkedList::new));

System.out.println(mylist2);

System.out.println("----------");

System.out.println("Done!!!");
}
}
```

Demo18.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;
import java.util.stream.Collectors;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo18 {
public static void main(String[] args) {

List<Integer> numsList = Arrays.asList(5, 4, 8, 3, 6, 7, 2, 9);

System.out.println("-------------");
Set<Integer> myset1 = numsList.stream()
.filter(num -> num % 2 == 0)
.map(num -> num * num)
.sorted()
.collect(Collectors.toSet());

System.out.println(myset1);

System.out.println("-------------");
Set<Integer> myset2 = numsList.stream()
.filter(num -> num % 2 == 0)
.map(num -> num * num)
.sorted()
.collect(Collectors.toCollection(TreeSet::new));

System.out.println(myset2);

System.out.println("-------------");
System.out.println("Done!!!");
}
}
```

Demo19.java

```
package com.jlcindia.demos;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo19 {
public static void main(String[] args) {

List<Integer> numsList = Arrays.asList(1, 2, 3, 4, 5);

long count = numsList.stream()
.filter(num -> num % 2 != 0)
.collect(Collectors.counting());

System.out.println(count);

long sumResult = numsList.stream()
.filter(num -> num % 2 != 0)
.collect(Collectors.reducing(0, (num1,num2)-> num1+num2));

System.out.println(sumResult);

long mulResult = numsList.stream()
.filter(num -> num % 2 != 0)
.collect(Collectors.reducing(1, (num1,num2)-> num1*num2));

System.out.println(mulResult);

List<String> courseList = Arrays.asList("Java","SpringBoot","DevOps");

String result  = courseList.stream()
.collect(Collectors.joining(" --- "));

System.out.println(result);
}
}
```