Program 1 :

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.List;

**public** **class** StreamCollection {

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

// **TODO** Auto-generated method stub

List<Employee> listOfEmp = **new** ArrayList<>();

listOfEmp.add(**new** Employee(1,"Raj",12000));

listOfEmp.add(**new** Employee(2,"Ram",15000));

listOfEmp.add(**new** Employee(3,"Rahul",15000));

listOfEmp.add(**new** Employee(4,"Rahim",11000));

listOfEmp.add(**new** Employee(5,"Rajesh",16000));

**float** sum = 0;

Iterator<Employee> li = listOfEmp.iterator();

**while**(li.hasNext()) {

Employee emp = li.next();

sum = sum + emp.getSalary();

}

System.***out***.println("Sum Of Salary : "+sum);

// Using Stream

listOfEmp.stream().forEach((emp)->System.***out***.println(emp.getSalary()));

listOfEmp.stream().filter(emp->emp.getSalary()>15000).forEach((emp)->System.***out***.println(emp));

Double sumOfSalary = listOfEmp.stream().filter(emp->emp.getSalary()>0).mapToDouble(emp->emp.getSalary()).sum();

System.***out***.println(sumOfSalary);

}

}

O/P:

Sum Of Salary : 69000.0

12000.0

15000.0

15000.0

11000.0

16000.0

Employee [id=5, name=Rajesh, salary=16000.0]

69000.0

Program 2 :

**interface** Abc {

**void** dis1();

}

**class** Demo **implements** Abc {

**public** **void** dis1() {

System.***out***.println("Method Implementation");

}

}

**class** Demo1 {

**public** **void** ownMethod() {

System.***out***.println("Demo1 Non static Method Implementation");

}

**public** **static** **void** anotherownMethod() {

System.***out***.println("Demo1 static Method Implementation");

}

}

**public** **class** MethodRefDemo {

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

// **TODO** Auto-generated method stub

Abc obj = **new** Demo();

obj.dis1();

System.***out***.println("Before Lambda Expression");

Abc obj1 = ()->System.***out***.println("Method Override in Lambda Expression");

System.***out***.println("Method Reference ");

Abc a = Demo1 :: *anotherownMethod*;

a.dis1();

Demo1 ref = **new** Demo1();

Abc obj2 = ref :: ownMethod;

obj2.dis1();

}

O/P :

Method Implementation

Before Lambda Expression

Method Reference

Demo1 static Method Implementation

Demo1 Non static Method Implementation

Program 3:

**import** java.util.Optional;

**public** **class** OptionalClassDemo {

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

// **TODO** Auto-generated method stub

Employee emp[] = **new** Employee[10];

emp[0] = **new** Employee();

emp[3] = **new** Employee();

//System.out.println(emp[1].getId()); // Throws Null Pointer Exception

Optional<Employee> obj = Optional.*ofNullable*(emp[1]);

**if**(obj.isPresent()) {

System.***out***.println(emp[1].getId());

} **else** {

System.***out***.println("No Memory Allocated");

}

}

}

O/P:

No Memory Allocated

Program 4:

**import** java.time.LocalDate;

**import** java.time.LocalDateTime;

**import** java.time.LocalTime;

**public** **class** DateTimeDemo {

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

// **TODO** Auto-generated method stub

LocalDate dd1 = LocalDate.*now*();

System.***out***.println("LocalDate "+dd1);

LocalTime dd2 = LocalTime.*now*();

System.***out***.println("LocalTime "+dd2);

LocalDateTime dd3 = LocalDateTime.*now*();

System.***out***.println("LocalDateTime "+dd3);

Set<String> ss = ZoneRulesProvider.*getAvailableZoneIds*();

ss.forEach(val->System.***out***.println(val)); // Displays all available zones

LocalDateTime dd4 = LocalDateTime.*now*(ZoneId.*of*("America/Marigot"));

System.***out***.println("LocalDateTime America/Marigot "+dd4);

LocalDate dd5 = LocalDate.*of*(2018,Month.***JANUARY***,2);

System.***out***.println("LocalDateOf Method : "+dd5);

DateTimeFormatter dtf = DateTimeFormatter.*ofLocalizedDateTime*(FormatStyle.***MEDIUM***);

DateTimeFormatter dtf1 = DateTimeFormatter.*ofPattern*("dd-MM-yyyy");

System.***out***.println(dd3.format(dtf));

System.***out***.println(dd3.format(dtf1));

}

}

O/P :

LocalDate 2018-03-09

LocalTime 11:45:54.603

LocalDateTime 2018-03-09T11:45:54.603

LocalDateTime America/Marigot 2018-03-09T02:22:28.100

LocalDateOf Method : 2018-01-02

9 Mar, 2018 12:24:04 PM

09-03-2018

Factory Design Pattern

**public** **class** FactoryMethodTest {

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

// **TODO** Auto-generated method stub

Employee obj = Employee.*getInstance*();

obj.display();

}

}

**public** **class** Employee {

**private** Employee() {

}

**public** **void** display() {

System.***out***.println("Business Method");

}

**public** **static** Employee getInstance() {

**return** **new** Employee();

}

}

O/P:

Business Method

Singleton Pattern

**public** **class** FactoryMethodTest {

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

// **TODO** Auto-generated method stub

Employee obj = Employee.*getInstance*(1);

obj.a = 10;

obj.display();

Employee obj1 = Employee.*getInstance*(1);

obj1.display();

}

}

**public** **class** Employee {

**int** a;

**private** Employee() {

}

**public** **void** display() {

System.***out***.println("Business Method"+a);

}

**private** **static** Employee *emp* = **new** Employee();

**public** **static** Employee getInstance(**int** code) {

**if**(code == 1 ) {

**return** *emp*;

} **else** {

**return** **null**;

}

}

}

O/P:

Business Method10

Business Method10

Factory pattern

**class** Color {

**public** **void** paint() {

System.***out***.println("Normal Painting");

}

}

**class** Green **extends** Color {

**public** **void** paint() {

System.***out***.println("Green Painting");

}

}

**class** Red **extends** Color {

**public** **void** paint() {

System.***out***.println("Red Painting");

}

}

**class** Blue **extends** Color {

**public** **void** paint() {

System.***out***.println("Blue Painting");

}

}

**class** ColorFactory {

**public** Color getInstance(String sColorName) {

**if**(sColorName.equals("Red")) {

**return** **new** Red();

} **else** **if**(sColorName.equals("Blue")) {

**return** **new** Blue();

} **else** **if**(sColorName.equals("Green")) {

**return** **new** Green();

} **else** {

**return** **null**;

}

}

}

**public** **class** FactoryPatternTest {

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

ColorFactory cf = **new** ColorFactory();

Color cc = cf.getInstance("Blue");

cc.paint();

Color cc1 = cf.getInstance("Red");

cc1.paint();

Color cc2 = cf.getInstance("Green");

cc2.paint();

}

}

O/P:

Blue Painting

Red Painting

Green Painting