* 1. Generic Class

Write a Java Program to demonstrate a Generic Class.

Code:

**package** practical1;

**public** **class** test <t>

{

t obj;

test(t obj)

{**this**.obj=obj;}

**public** t getobject()

{**return** **this**.obj;}

}

**class** Generic\_class

{

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

{

test<Integer>iobj=**new** test<Integer>(46);

System.***out***.println("the value of Integer is:"+iobj.getobject());

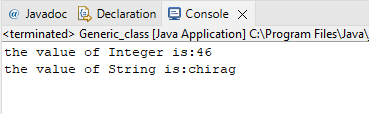
test<String>sobj=**new** test<String>("chirag");

System.***out***.println("the value of String is:"+sobj.getobject());

}

}

Output:



1.2) Generic method

Write a Java Program to demonstrate Generic Methods.

Code:

**package** practical;

**public** **class** generic\_method {

**void** display()

{

System.***out***.println(" generic method example");

}

<t> **void** gdisplay (t e)

{

System.***out***.println(e.getClass().getName()+"="+e);

}

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

generic\_method g1=**new** generic\_method();

g1.display();

g1.gdisplay(46);

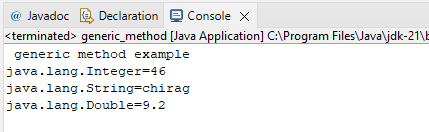
g1.gdisplay("chirag");

g1.gdisplay(9.2);

}

}

Output:



1.3) Wild Card

Write a Java Program to demonstrate Wildcards in Java Generics.

Code: **package** practical1;

**import** java.util.\*;

**public** **class** wild\_card {

//upper bounded

**private** **static** **double** sum(List<? **extends** Number>list)

{

**double** sum=0.0;

**for**(Number i:list)

{sum=sum + i.doubleValue();}

**return** sum;

}

//lower bounded

**private** **static** **void** show(List<? **super** Integer>list)

{

list.forEach((x)->{

System.***out***.print(x+" ");

});

}

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

System.***out***.println("Upper Bounded:");

List <Integer>l\_1=Arrays.*asList* (5,3,7,4,2,6);

System.***out***.println("List 1 Sum:"+*sum*(l\_1));

List <Double>l\_2=Arrays.*asList* (5.2,3.4,7.3,4.4,2.5,6.2);

System.***out***.println("List 2 Sum:"+*sum*(l\_2));

System.***out***.println("\n lower Bounded:");

List <Integer>l\_3=Arrays.*asList* (5,3,7,4,2,6);

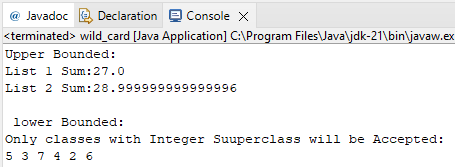
System.***out***.println("Only classes with Integer Suuperclass will be Accepted:");

*show*(l\_3);

}

}

Output:



2.1) List

Write a Java program to create a List containing a list of items of type String and use for each loop to print the items of the list.

Code: **package** practical2;

**import** java.util.ArrayList;

**public** **class** array1 {

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

ArrayList<String>list=**new** ArrayList<String>();

list.add("MATHS");

list.add("ADBMS");

list.add("JAVA");

list.add("PYTHON");

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

System.***out***.println("Traversing listt Through for each loop");

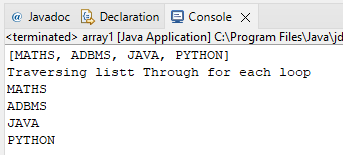
**for**(String subject:list)

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

}

}

Output:



2.2) ListIterator

Write a Java program to create List containing list of items and use List Iterator

interface to print items present in the list. Also print the list in reverse/ backward Direction.

Code: **package** practical2;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.ListIterator;

**public** **class** test {

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

List<String>Product=**new** ArrayList<>();

Product.add("Product 1");

Product.add("Product 2");

Product.add("Product 3");

Product.add("Product 4");

Product.add("Product 5");

System.***out***.println("Printing porduct in forward direction using List iterator:");

ListIterator<String>forwarditerator = Product.listIterator();

**while**(forwarditerator.hasNext())

{

System.***out***.println(forwarditerator.next());

}

System.***out***.println("printing product in reverse direction using List iterator:");

ListIterator<String>backwarditerator = Product.listIterator(Product.size());

**while**(backwarditerator.hasPrevious())

{

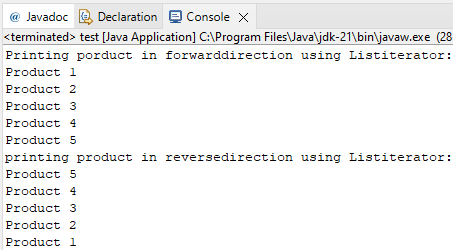
System.***out***.println(backwarditerator.previous());

}

}

}

Output:



3.1) Set

Write a Java program to create a Set containing list of items of type String and print

the items in the list using Iterator interface. Also print the list in reverse/ backward direction.

Code: **package** practical3;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.ListIterator;

**public** **class** set\_1 {

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

{

List<String>name=**new** LinkedList<>();

name.add("chirag");

name.add("GNIMS");

name.add("MCA");

ListIterator<String>listIterator=name.listIterator();

System.***out***.println("ForwardDirection Iterator:");

**while**(listIterator.hasNext()){

System.***out***.println(listIterator.next());

}

System.***out***.println("Backward Direction Iteration:");

**while**(listIterator.hasPrevious()){

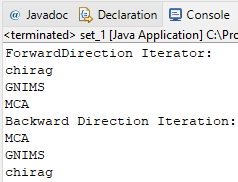
System.***out***.println(listIterator.previous());

}

}

}

Output:



3.2) Sets Operations

Write a Java program using Set interface containing list of items and perform the

following operations:

a. Add items in the set.

b. Insert items of one set into another set.

c. Remove items from the set

d. Search the specified item in the set

Code: **package** practical3;

**import** java.util.HashSet;

**import** java.util.Set;

**public** **class** set {

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

Set<String>set1=**new** HashSet<>();

set1.add("sujit");

set1.add("arun");

set1.add("chirag");

set1.add("rohit");

set1.add("sourabh");

System.***out***.println("set1:"+set1);

Set<String>set2=**new** HashSet<>();

set2.add("sumit");

set2.add("ritik");

set2.add("raj");

set1.addAll(set2);

System.***out***.println("set1 after adding item fromset2:"+set1);

set1.remove("ritik");

System.***out***.println("set1 after removing ritik:"+set1);

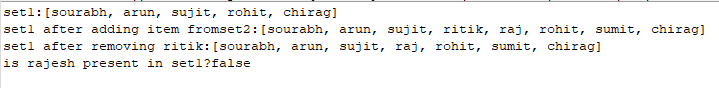
**boolean** isPresent=set1.contains("rajesh");

System.***out***.println("is rajesh present in set1?"+isPresent);

}

}

Output:



4) Map

Write a Java program using Map interface containing list of items having keys and

associated values and perform the following operations:

a. Add items in the map.

b. Remove items from the map

c. Search specific key from the map

d. Get value of the specified key

e. Insert map elements of one map into another map.

f. Print all keys and values of the map.

Code: **package** pratical4;

**import** java.util.Map;

**import** java.util.HashMap;

**public** **class** test {

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

Map<Integer,String>map1 =**new** HashMap<>();

map1.put(1,"sujit");

map1.put(2,"sumit");

map1.put(3,"rohit");

map1.put(4,"vijay");

map1.put(5,"chirag");

System.***out***.println("Map1:"+map1);

Map<Integer,String>map2 =**new** HashMap<>();

map2.put(6,"suresh");

map2.put(7,"ajay");

map2.put(8,"niraj");

map1.putAll(map2);

System.***out***.println("Map1 after adding item formap2:" +map1);

map1.remove(3);

System.***out***.println("map1 after removing key 3:"+map1);

**boolean** isPresent=map1.containsKey(4);

System.***out***.println("Is key 4 present in map1?"+isPresent);

String value=map1.get(2);

System.***out***.println("the value of key 2 is:"+ value);

System.***out***.println("\nprinting All keys and values Of map1:");

**for**(Map.Entry<Integer,String>entry:map1.entrySet())

{

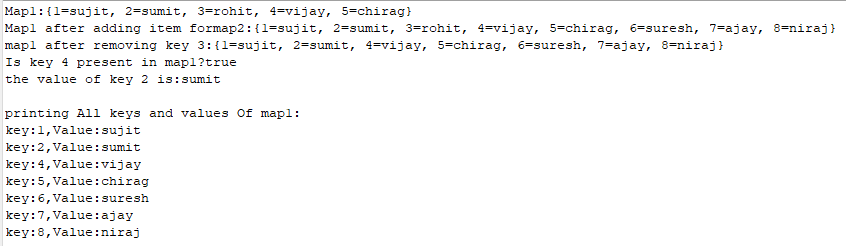
System.***out***.println("key:"+entry.getKey()+",Value:"+entry.getValue());

}

}

}

Output:



5)Lambda Expression

5.1) Lambda Expression to print “Hello World”

Aim: Write a Java program using Lambda Expression to print “Hello World”.

Code: **package** prac5;

**public** **class** lambdaex {

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

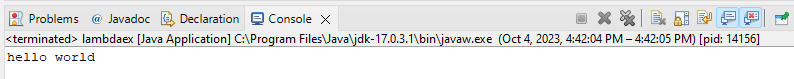
Runnable helloworld=()->System.***out***.println("hello world");

helloworld.run();

}

}

Output:



5.2) Lambda Expression with single parameters

Aim: Write a Java program using Lambda Expression with single parameters.

Code: **package** prac5;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** demo2 {

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

List<String> names =Arrays.*asList*(" sujit" ,"chirag", "rohit", "arun");

names.forEach(name-> System.***out***.println("hello " + name + "!"));

}

}

Output:

A screenshot of a computer

Description automatically generated

5.3) Lambda Expression with multiple parameters to add two numbers.

Aim: Write a Java program using Lambda Expression with multiple parameters to add two numbers.

Code: **package** prac5;

**interface** B

{

**int** add (**int** i, **int** j);

}

**public** **class** demo3 {

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

B obj =(i,j)-> i+j;

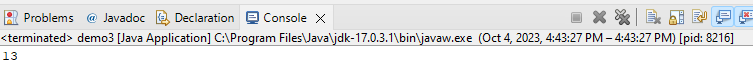
**int** result = obj.add(6, 7);

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

}

}

Output:



5.4) Lambda Expression for Conversion

Aim: Write a Java program using Lambda Expression to calculate the following:

a. Convert Fahrenheit to Celsius.

b. Convert Kilometres to Miles.

Code: **package** prac5;

**interface** converter{

**double** convert (**double** input);

}

**public** **class** demo4 {

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

converter a = f ->(f-32)\*5/9;

**double** celsius = a.convert(98);

System.***out***.println("98 degrees Fahrenheit is " + celsius +

"degree celsius");

converter kilometersToMiles = km -> km/1.609344;

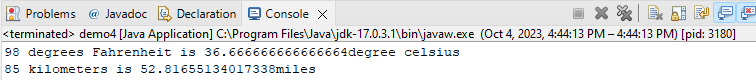
**double** miles = kilometersToMiles.convert(85);

System.***out***.println("85 kilometers is " + miles + "miles");

}

}

Output:



5.5) Lambda Expression with or without return keyword

Aim: Write a Java program using Lambda Expression with or without return keyword.

Code: **package** prac5;

**interface** calculator{

**int** calculate(**int** x, **int** y);

}

**public** **class** demo5 {

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

calculator add=(x,y) -> x+y;

calculator subtract=(x,y)-> {**return** x-y ;};

**int** sum = add.calculate(125678, 56345);

**int** difference =subtract.calculate(190, 56);

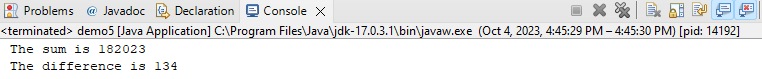
System.***out***.println(" The sum is " +sum);

System.***out***.println(" The difference is " +difference);

}

}

Output:



5.6) Lambda Expression to concatenate two strings.

Aim: Write a Java program using Lambda Expression to concatenate two strings.

Code: **package** prac5;

**interface** Concatenator{

String concatenate( String s1,String s2);

}

**public** **class** demo6 {

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

String str1 ="hello";

String str2 ="chirag";

Concatenator concatenator =(s1,s2) -> s1+s2;

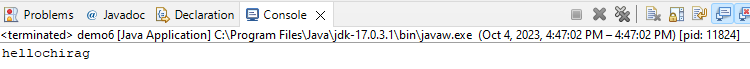
String result = concatenator.concatenate(str1,str2);

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

}

}

Output:



Practical-6

Practical-8

8.1

Aim: Write a program to demonstrate Spring AOP – before advice.

Code:

Application.java:

**package** com.example;

**import** com.example.service.YourService;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

**public** **class** YourApplication {

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

ConfigurableApplicationContext context = SpringApplication.*run*(YourApplication.**class**,

args);

YourService yourService = context.getBean(YourService.**class**);

yourService.yourmethod();

}

}

Aspect.java:

**package** com.example.aspect;

**import** org.aspectj.lang.annotation.Aspect;

**import** org.aspectj.lang.annotation.Before;

**import** org.springframework.stereotype.Component;

@Aspect

@Component

**public** **class** myaspect {

@Before("execution(\* com.example.service.YourService.yourmethod())")

**public** **void** beforeAdvice() {

System.***out***.println("Before advice executed.");

}

}

**YourService.java**

**package** com.example.service;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** YourService {

**public** **void** yourmethod() {

System.***out***.println("Your method executed.");

}

}

In pom.xml only change

<parent>

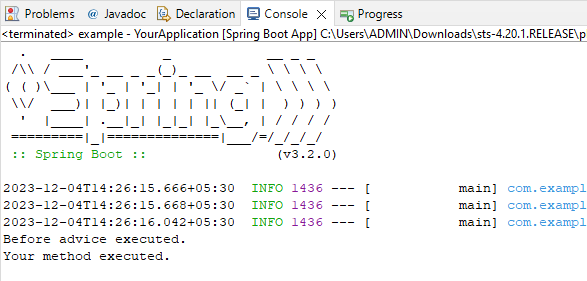
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.0</version>

</parent>

Output:



8.2

Aim: Write a program to demonstrate Spring AOP – after advice.

Code:

**package** com.example;

**import** com.example.service.YourService;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

**public** **class** YourApplication {

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

ConfigurableApplicationContext context = SpringApplication.*run*(YourApplication.**class**,

args);

YourService yourService = context.getBean(YourService.**class**);

yourService.yourmethod();

}

}

Aspect.java:

**package** com.example.aspect;

**import** org.aspectj.lang.annotation.After;

**import** org.aspectj.lang.annotation.Aspect;

**import** org.springframework.stereotype.Component;

@Aspect

@Component

**public** **class** myaspect {

@After("execution(\* com.example.service.YourService.yourmethod())")

**public** **void** beforeAdvice() {

System.***out***.println("After advice executed.");

}

}

**YourService.java**

**package** com.example.service;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** YourService {

**public** **void** yourmethod() {

System.***out***.println("Your method executed.");

}

}

In pom.xml only change

<parent>

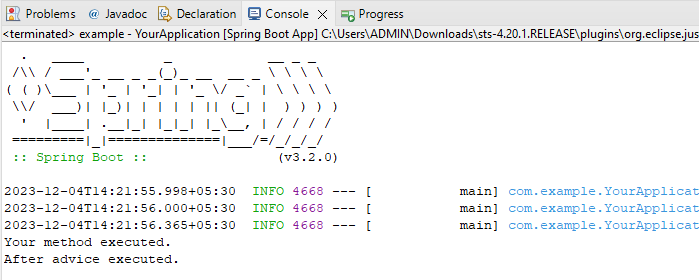
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.0</version>

</parent>

Output:



8.3

Aim: Write a program to demonstrate Spring AOP – Around advice.

Code:

Application.java:

**package** com.example;

**import** com.example.service.YourService;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

**public** **class** YourApplication {

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

ConfigurableApplicationContext context = SpringApplication.*run*(YourApplication.**class**,

args);

YourService yourService = context.getBean(YourService.**class**);

yourService.yourmethod();

}

}

Aspect.java:

**package** com.example.aspect;

**import** org.aspectj.lang.ProceedingJoinPoint;

**import** org.aspectj.lang.annotation.Around;

**import** org.aspectj.lang.annotation.Aspect;

**import** org.springframework.stereotype.Component;

@Aspect

@Component

**public** **class** myaspect {

@Around("execution(\* com.example.service.YourService.yourmethod())")

**public** **void** aspect(ProceedingJoinPoint joinPoint) {

**try** {

System.***out***.println("Before Advice");

joinPoint.proceed(); // Proceed with the original method

System.***out***.println("After Advice");

} **catch** (Throwable e) {

System.***err***.println("Exception in advice: " + e.getMessage());

}

}

}

**YourService.java**

**package** com.example.service;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** YourService {

**public** **void** yourmethod() {

System.***out***.println("Your method executed.");

}

}

In pom.xml only change

<parent>

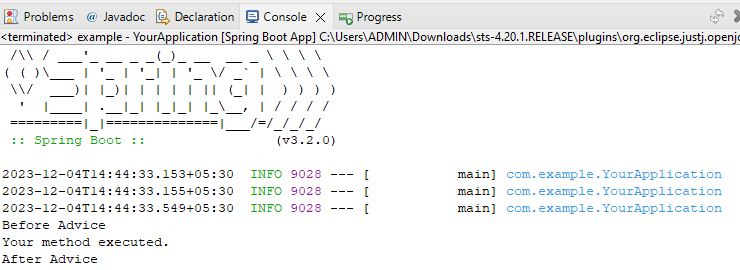
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.0</version>

</parent>

Output:



8.4

Aim: Write a program to demonstrate Spring AOP – After Returning advice.

Code:

Application.java:

**package** com.example;

**import** com.example.service.YourService;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

**public** **class** YourApplication {

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

ConfigurableApplicationContext context = SpringApplication.*run*(YourApplication.**class**,

args);

YourService yourService = context.getBean(YourService.**class**);

yourService.yourmethod();

}

}

Aspect.java:

**package** com.example.aspect;

**import** org.aspectj.lang.annotation.AfterReturning;

**import** org.aspectj.lang.annotation.Aspect;

**import** org.springframework.stereotype.Component;

@Aspect

@Component

**public** **class** myaspect {

@AfterReturning(

pointcut = "execution(\* com.example.service.YourService.yourmethod())",

returning = "result"

)

**public** **void** afterReturningAdvice() {

System.***out***.println("After Returning Advice: Aspect executed after service2()returns");

}

}

**YourService.java**

**package** com.example.service;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** YourService {

**public** **void** yourmethod() {

System.***out***.println("Your method executed.");

}

}

In pom.xml only change

<parent>

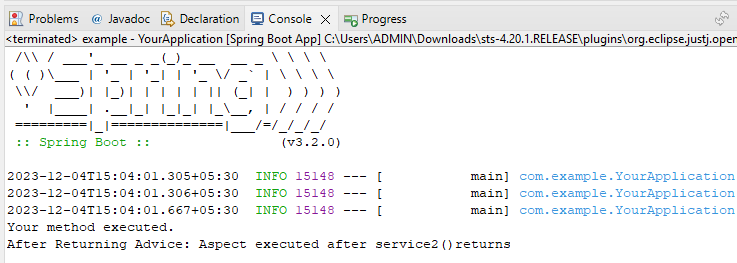
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.0</version>

</parent>

Output:



8.5

Aim: Write a program to demonstrate Spring AOP – After throwing advice.

Code:

Application.java:

**package** com.example;

**import** com.example.service.YourService;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

**public** **class** YourApplication {

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

ConfigurableApplicationContext context = SpringApplication.*run*(YourApplication.**class**,

args);

YourService yourService = context.getBean(YourService.**class**);

yourService.yourmethod();

}

}

Aspect.java:

**package** com.example.aspect;

**import** org.aspectj.lang.annotation.AfterThrowing;

**import** org.aspectj.lang.annotation.Aspect;

**import** org.springframework.stereotype.Component;

@Aspect

@Component

**public** **class** myaspect {

@AfterThrowing(

pointcut = "execution(\* com.example.service.YourService.yourmethod())",

throwing = "result"

)

**public** **void** afterReturningAdvice() {

System.***out***.println("After Returning Advice: Aspect executed after service2()returns");

}

}

**YourService.java**

**package** com.example.service;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** YourService {

**public** **void** yourmethod() {

System.***out***.println("Your method executed.");

}

}

In pom.xml only change

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.0</version>

</parent>

Output:

