**Jshell:**

1. Java 9 – jshell
2. Jshell
3. /exit
4. Literal – value don’t change

Puzzles

1. Precedence
2. BODMAS
3. PERDAS
4. Escape character “ \”jaya\” “
5. New Line \n
6. Tab \t
7. \\ - \
8. \\\\ - \\
9. Math.random() – 0 to 1
10. Formatter - %d, %s
11. Undeclare variable = 0
12. Java strongly typed values

* C:\Users\tharu>jshell
* | Welcome to JShell -- Version 22.0.1
* | For an introduction type: /help intro
* jshell> 5\*10
* $1 ==> 50
* jshell> /exit
* | Goodbye
* C:\Users\tharu>jshell
* | Welcome to JShell -- Version 22.0.1
* | For an introduction type: /help intro
* jshell> /help
* | Type a Java language expression, statement, or declaration.
* | Or type one of the following commands:
* | /list [<name or id>|-all|-start]
* | list the source you have typed
* | /edit <name or id>
* | edit a source entry
* | /drop <name or id>
* | delete a source entry
* | /save [-all|-history|-start] <file>
* | Save snippet source to a file
* | /open <file>
* | open a file as source input
* | /vars [<name or id>|-all|-start]
* | list the declared variables and their values
* | /methods [<name or id>|-all|-start]
* | list the declared methods and their signatures
* | /types [<name or id>|-all|-start]
* | list the type declarations
* | /imports
* | list the imported items
* | /exit [<integer-expression-snippet>]
* | exit the jshell tool
* | /env [-class-path <path>] [-module-path <path>] [-add-modules <modules>] ...
* | view or change the evaluation context
* | /reset [-class-path <path>] [-module-path <path>] [-add-modules <modules>]...
* | reset the jshell tool
* | /reload [-restore] [-quiet] [-class-path <path>] [-module-path <path>]...
* | reset and replay relevant history -- current or previous (-restore)
* | /history [-all]
* | history of what you have typed
* | /help [<command>|<subject>]
* | get information about using the jshell tool
* | /set editor|start|feedback|mode|prompt|truncation|format ...
* | set configuration information
* | /? [<command>|<subject>]
* | get information about using the jshell tool
* | /!
* | rerun last snippet -- see /help rerun
* | /<id>
* | rerun snippets by ID or ID range -- see /help rerun
* | /-<n>
* | rerun n-th previous snippet -- see /help rerun
* |
* | For more information type '/help' followed by the name of a
* | command or a subject.
* | For example '/help /list' or '/help intro'.
* |
* | Subjects:
* |
* | intro
* | an introduction to the jshell tool
* | keys
* | a description of readline-like input editing
* | id
* | a description of snippet IDs and how use them
* | shortcuts
* | a description of keystrokes for snippet and command completion,
* | information access, and automatic code generation
* | context
* | a description of the evaluation context options for /env /reload and /reset
* | rerun
* | a description of ways to re-evaluate previously entered snippets
* jshell> 5\*%
* ...> 5
* | Error:
* | illegal start of expression
* | 5\*%

| ^

jshell> 5\*%

...> 5

| Error:

| illegal start of expression

| 5\*%

| ^

jshell> 5\*5

$1 ==> 25

jshell> 60\*24

$2 ==> 1440

jshell> 24\*60\*60

$3 ==> 86400

jshell> 5/2

$4 ==> 2

jshell> 5//2

$5 ==> 5

jshell> 5/2.0

$6 ==> 2.5

jshell> 5.0/2

$7 ==> 2.5

jshell> System.out.println(3\*4);

12

jshell> System.out.println("3\*4=",3\*4);

| Error:

| no suitable method found for println(java.lang.String,int)

| method java.io.PrintStream.println() is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(boolean) is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(char) is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(int) is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(long) is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(float) is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(double) is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(char[]) is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(java.lang.String) is not applicable

| (actual and formal argument lists differ in length)

| method java.io.PrintStream.println(java.lang.Object) is not applicable

| (actual and formal argument lists differ in length)

| System.out.println("3\*4=",3\*4);

| ^----------------^

jshell> System.out.println("3\*4=12");

3\*4=12

jshell> System.out.println("3\*4="+3\*4);

3\*4=12

jshell> Math.random();

$11 ==> 0.4525647525448374

jshell> Math.random();

$12 ==> 0.9715172194277434

jshell> Math.random();

$13 ==> 0.4617582703863149

jshell> Math.min(45,22,189);

| Error:

| no suitable method found for min(int,int,int)

| method java.lang.Math.min(int,int) is not applicable

| (actual and formal argument lists differ in length)

| method java.lang.Math.min(long,long) is not applicable

| (actual and formal argument lists differ in length)

| method java.lang.Math.min(float,float) is not applicable

| (actual and formal argument lists differ in length)

| method java.lang.Math.min(double,double) is not applicable

| (actual and formal argument lists differ in length)

| Math.min(45,22,189);

| ^------^

jshell> Math.min(45,22);

$14 ==> 22

jshell> Math.min(45,22,18);

| Error:

| no suitable method found for min(int,int,int)

| method java.lang.Math.min(int,int) is not applicable

| (actual and formal argument lists differ in length)

| method java.lang.Math.min(long,long) is not applicable

| (actual and formal argument lists differ in length)

| method java.lang.Math.min(float,float) is not applicable

| (actual and formal argument lists differ in length)

| method java.lang.Math.min(double,double) is not applicable

| (actual and formal argument lists differ in length)

| Math.min(45,22,18);

| ^------^

jshell> i=10;j=20;

| Error:

| cannot find symbol

| symbol: variable i

| i=10;

| ^

jshell> int i=10;int j=20;

i ==> 10

j ==> 20

jshell> System.out.printf("3\*4=12");

3\*4=12$17 ==> java.io.PrintStream@5abca1e0

jshell> System.out.printf("3\*4=12").println();

3\*4=12

jshell> System.out.printf("3\*4=%d", 3\*4).println();

3\*4=12

jshell> System.out.printf("%d\*%d=%d", 3,4,3\*4).println();

3\*4=12

jshell> System.out.printf("%d", "jaya").println();

| Exception java.util.IllegalFormatConversionException: d != java.lang.String

| at Formatter$FormatSpecifier.failConversion (Formatter.java:4534)

| at Formatter$FormatSpecifier.printInteger (Formatter.java:3072)

| at Formatter$FormatSpecifier.print (Formatter.java:3027)

| at Formatter.format (Formatter.java:2797)

| at PrintStream.implFormat (PrintStream.java:1367)

| at PrintStream.format (PrintStream.java:1346)

| at PrintStream.printf (PrintStream.java:1245)

| at (#21:1)

jshell> System.out.printf("%s", "jaya").println();

jaya

jshell> x=5;

| Error:

| cannot find symbol

| symbol: variable x

| x=5;

| ^

jshell> int x=5;

x ==> 5

jshell> int number = 10;

number ==> 10

jshell> int result=0;

result ==> 0

jshell> for(int i=0;i<=number;i++)

...> {

...> System.out.println(x+"X"+i+"="+x\*i)

...> }

| Error:

| ';' expected

| System.out.println(x+"X"+i+"="+x\*i)

| ^

jshell> for(int i=0;i<=number;i++)

...> {

...> System.out.println(x+"X"+i+"="+x\*i);

...> }

5X0=0

5X1=5

5X2=10

5X3=15

5X4=20

5X5=25

5X6=30

5X7=35

5X8=40

5X9=45

5X10=50

1. Variable - Name, value, memory location
2. G
3. G
4. G

OOPS

1. Object, Class, Data (state), Actions (behaviors)
2. Class – Template
3. Object – Instance of a Class
4. **Instance variable refer by instance objects**
5. Object are independent
6. But have copy of the variable

package com.jb.OOPS;

public class Planet {

    int speed;

    public void print()

    {

        System.out.println(speed);

    }

    public static void main(String[] args) {

        Planet earth=new Planet();

        Planet venus=new Planet();

        earth.print();

        venus.print();

        earth.speed=100;

        earth.print();

        venus.print();

        earth.speed=120;

        venus.speed=20;

        earth.print();

        venus.print();

    }

}

// 0

// 0

// 100

// 0

// 120

1. // 20
2. Don’t directly access other class data by variable
3. Encapsulation – access data by methods of object
4. Binding of data
5. Hiding data
6. Make variable Private, then use method (ip/op/name)

package com.jb.OOPS;

public class Planet1

{

    private int speed;

    void setSpeed(int speed)

    {

        this.speed = speed;

    }

    int getSpeed()

    {

        return speed;

    }

    public void print()

    {

        System.out.println("Rotationg");

    }

    public static void main(String[] args)

{

        Planet1 earth=new Planet1();

        Planet1 venus=new Planet1();

        earth.print();

        venus.print();

        System.out.println(earth.getSpeed());

        System.out.println(venus.getSpeed());

        earth.setSpeed(100);

        venus.setSpeed(20);

        System.out.println(earth.getSpeed());

        System.out.println(venus.getSpeed());

}

}

// Rotationg

// Rotationg

// 0

// 0

// 100

// 20

1. Before this method call default values of Variable

package com.jb.OOPS;

public class Planet2

{

    private int speed;

    void setSpeed(int speed)

    {

        System.out.println(speed);

        System.out.println(this.speed);

        this.speed = speed;

    }

    int getSpeed()

    {

        return speed;

    }

    public void print()

    {

        System.out.println("Rotationg");

    }

    public static void main(String[] args)

{

        Planet2 earth=new Planet2();

        Planet2 venus=new Planet2();

        earth.print(); //ro

        venus.print(); //r0

        System.out.println(earth.getSpeed()); //0

        System.out.println(venus.getSpeed()); //0

        earth.setSpeed(100); //100, 0(default Speed 0) - call method

        venus.setSpeed(20);  //20 , 0

        System.out.println(earth.getSpeed()); //100

        System.out.println(venus.getSpeed()); //20

}

}

// Rotationg

// Rotationg

// 0

// 0

// 100

// 0

// 20

// 0

// 100

// 20

1. Use Method for Logic
2. Prevent bad data and prevent bad code on other classes
3. Validation and operation on variables by method

package com.jb.OOPS;

public class Planet3

{

    private int speed;

    void setSpeed(int speed)

    {

        if(speed>0) //only print grater than 0

        {

            this.speed=speed;

        }

    }

    int getSpeed()

    {

        return speed;

    }

    public void print()

    {

        System.out.println("Rotationg");

    }

    private void increaseSpeed(int howMuch)

    {

        this.speed=this.speed+howMuch;

    }

    public static void main(String[] args)

{

        Planet3 earth=new Planet3();

        Planet3 venus=new Planet3();

        earth.print(); //ro

        venus.print(); //r0

        System.out.println(earth.getSpeed()); //0

        System.out.println(venus.getSpeed()); //0

        earth.setSpeed(10);

        venus.setSpeed(-4);

        System.out.println(earth.getSpeed()); //10

        System.out.println(venus.getSpeed()); //0

        earth.increaseSpeed(10);

        venus.increaseSpeed(10);

        System.out.println(earth.getSpeed()); //10+10=20

        System.out.println(venus.getSpeed()); //0+10=10

}

}

// Rotationg

// Rotationg

// 0

// 0

// 10

// 0

// 20

// 10

1. Recursive Method for increment and decrement using setters

package com.jb.OOPS;

public class Planet3

{

    private int speed;

    void setSpeed(int speed)

    {

        if(speed>0) //only print grater than 0

        {

            this.speed=speed;

        }

    }

    int getSpeed()

    {

        return speed;

    }

    public void print()

    {

        System.out.println("Rotationg");

    }

    private void increaseSpeed(int howMuch)

    {

        // this.speed=this.speed+howMuch;

        setSpeed(this.speed+howMuch);

    }

    private void decreaseSpeed(int howMuch)

    {

        // this.speed=this.speed-howMuch;

        setSpeed(this.speed-howMuch);

    }

    public static void main(String[] args)

{

        Planet3 earth=new Planet3();

        Planet3 venus=new Planet3();

        earth.print(); //ro

        venus.print(); //r0

        System.out.println(earth.getSpeed()); //0

        System.out.println(venus.getSpeed()); //0

        earth.setSpeed(10);

        venus.setSpeed(-4);

        System.out.println(earth.getSpeed()); //10

        System.out.println(venus.getSpeed()); //0

        earth.increaseSpeed(10);

        venus.increaseSpeed(10);

        venus.decreaseSpeed(5);

        System.out.println(earth.getSpeed()); //10+10=20

        System.out.println(venus.getSpeed()); //0+10-5=5

}

}

// Rotationg

// Rotationg

// 0

// 0

// 10

// 0

// 20

// 5

1. Abtraction
2. Hiding of Logic / abtracting
3. Constructor
4. Special method
5. Name of method same as name of the class
6. Use constructor to set initial values to variables
7. Java Provides you a default Constructor
8. This – keyword is used to call another constructor by parameters

//abstraction & Constructor

package com.jb.T05OOPS;

public class Planet4

{

    private int speed;

    Planet4(int speed)

    {

        this.speed=speed;

    }

    void setSpeed(int speed)

    {

        if(speed>0) //only print grater than 0

        {

            this.speed=speed;

        }

    }

    int getSpeed()

    {

        return speed;

    }

    public void print()

    {

        System.out.println("Rotationg");

    }

    private void increaseSpeed(int howMuch)

    {

        // this.speed=this.speed+howMuch;

        setSpeed(this.speed+howMuch);

    }

    private void decreaseSpeed(int howMuch)

    {

        // this.speed=this.speed-howMuch;

        setSpeed(this.speed-howMuch);

    }

    public static void main(String[] args)

{

        Planet4 earth=new Planet4(100);

        Planet4 venus=new Planet4(900);

        earth.print(); //ro

        venus.print(); //r0

        System.out.println(earth.getSpeed()); //100

        System.out.println(venus.getSpeed()); //900

        earth.setSpeed(10);

        venus.setSpeed(-4);

        System.out.println(earth.getSpeed()); //10

        System.out.println(venus.getSpeed()); //900

        earth.increaseSpeed(10);

        venus.increaseSpeed(10);

        venus.decreaseSpeed(5);

        System.out.println(earth.getSpeed()); //10+10=20

        System.out.println(venus.getSpeed()); //900+10-5=905

}

}

// Rotationg

// Rotationg

// 100

// 900

// 10

// 900

// 20

// 905

Primitive Datatypes

Integer

1. Byte - 127 to -128, 8
2. Short – 2
3. Integer – 4
4. Long – 8

package com.jb.T06DataTypes;

public class Integer

{

    int i = 10000;

    long l = 50000000000l;

    int j =(int) l;

}

Octal, Hexadecimal

1. Int eight = 010;
2. Int sixteen =0x10;
3. Int fifteen =0XF;
4. Byte b= 1010;
5. Increment and decrement
6. Post a++, b--
7. Pre –b ,c++

package com.jb.T06DataTypes;

public class Integer

{

    private int a;

    private int b;

    Integer(int a, int b)

    {

        this.a=a;

        this.b=b;

    }

    public int add()

    {

        return a+b;

    }

    public int multi()

    {

        return a\*b;

    }

    public static void main(String[] args) {

    Integer calc= new Integer(8,7);

    System.out.println(calc.add()); //15

    System.out.println(calc.multi()); //56

    int i = 10000;

    long l = 50000000000l;

    i =(int) l;

    System.out.println(i); //-1539607552

    int a = 10;

    int b = a++; //10 //post - increment

    System.out.println(b); //11

    int c = 10;

    int d= ++c;  //11// pre - increment

    System.out.println(d); //11

    }

}

1. Floating Point value
2. Double and Float
3. Floating point literal – Double
4. F = 34.6f
5. D=34.3457
6. Explicit cast
7. F=(float)D
8. Why don’t use float and double ever? – not accurate; Financial Calculation
9. BigDecimal class – immutable; eg: 60.57836930 – import math bigdecimal
10. Number.{tab button}
11. Use add function to add
12. String – accurate
13. Double – not accurate
14. So use String Constructor
15. New BigDecimal(i)
16. Simple Interest

package com.jb.T06DataTypes;

import java.math.BigDecimal;

public class SimpleInterest //si=(p\*n\*r)/100 Total = P + SI

{

     BigDecimal principle;

     BigDecimal interest;

            public SimpleInterest(String principle, String interest)

            {

                this.principle=new BigDecimal(principle);

                this.interest=new BigDecimal(interest).divide(new BigDecimal(100));

            }

            public BigDecimal calcTotalyear(int noOfYears)

            {

            BigDecimal noOfYearsBig= new BigDecimal(noOfYears);

            return principle.add(principle.multiply(interest).multiply(noOfYearsBig)); //P+(p\*(r/100)\*n)

            }

        public static void main(String[] args)

        {

         SimpleInterest calc= new SimpleInterest("4500.00","7.5"); // by calculation divide by 100

         BigDecimal totalyear = calc.calcTotalyear(5);

         System.out.println(totalyear);   //6187.50000

        }

}

Boolean

1. Case Sensitive
2. Used in conditions
3. True, false

Logical

1. 2 or more conditions
2. &&, ||, !
3. ^ - XOR
4. TT -false
5. FF – false
6. FT - true
7. Truth table

Short Circuit (&&, ||)

1. If 1st condition Fails then it didn’t go to second condition

Character

1. Char ch =’j’;
2. Unicode values – each character assigned to byte
3. \u00A2 -C
4. 2 bytes
5. Char ch = 65 -> A
6. (int)ch -> 65
7. Ch ->\n (we can also store new line and tabs)

package com.jb.T06DataTypes;

public class Character

{

    private char ch;

        Character(char ch)

        {

            this.ch=ch;

        }

        public Boolean isVowel()

        {

            if(ch =='a' || ch =='e' || ch =='i' || ch =='o' || ch =='U' || ch =='A' || ch =='E' || ch =='I' || ch =='O' || ch =='U' )

            {

                return true;

            }

            return false;

        }

        public Boolean isConsonent()

        {

            if(isAlphabet() && !isDigit())

            {

                if(isVowel()) // not vowel and alpabet

                {

                    return false;

                }

                else

                {

                   return true;

                }

            }

            else return false;

        }

        public boolean isDigit()

        {

            if(ch >= 40 && ch <= 57)//0 to 9

            {

                return true;

            }

            else

            return false;

        }

        public boolean isLowerCase()

        {

            if(ch >= 97 && ch <=122)

            {

                return true;

            }

            else

            return false;

        }

        public boolean isUpperCase()

        {

            if(ch>=65 && ch<=90)

            {

                return true;

            }

            else

            return false;

        }

        public boolean isAlphabet()

        {

            if(isLowerCase() || isUpperCase())//a t0 z and A to Z

            {

                return true;

            }

            else

            {

                return false;

            }

        }

    public static void main(String[] args)

    {

        Character ch = new Character('J');

        System.out.println("Character is Vowel"+" "+ch.isVowel()); //Character is Vowel false

        System.out.println("Character is Consonent"+" "+ch.isConsonent()); //Character is Consonent true

        System.out.println("Character is Digit"+" "+ch.isDigit()); //Character is Digit false

        System.out.println("Character is Uppercase"+" "+ch.isUpperCase()); //Character is Uppercase true

        System.out.println("Character is Lowercase"+" "+ch.isLowerCase()); //Character is Lowercase false

        System.out.println("Character is Alphabet"+" "+ch.isAlphabet()); //Character is Alphabet true

    }

}

1. **Conditionals**
2. if, else if, else, switch, default
3. multiple if
4. nested block of if else
5. no integer and assignment are used in condition
6. not using { } – it consider only one statement

Get user inputs

1. Scanner and BufferedReader
2. Scanner

package com.jb.T07Conditionals;

import java.util.Scanner;

public class IfandSwitch

{

    public static void main(String[] args)

    {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter any number:");

        int number = scanner.nextInt(); //append the number

        System.out.println("Entered Number:"+" "+number);

        scanner.close();

    }

}

// Enter any number:

// 5

// Entered Number: 5

1. Arithmetic Calculator
2. Codingbt.com

package com.jb.T07Conditionals;

import java.util.Scanner;

public class IfandSwitch

{

    public static void main(String[] args)

    {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter 1st number: ");

        int number1 = scanner.nextInt(); //append the number

        System.out.print("Enter 2st number: ");

        int number2 = scanner.nextInt(); //append the number

        System.out.println("1st Number: "+number1);

        System.out.println("2nd Number: "+number2);

        System.out.println("Enter Choices: \n 1. Add \n 3. Sub \n 3. Mul \n 4. Div");

        System.out.print("Enter Choice Number: ");

        int choices = scanner.nextInt(); //append the number

        switch (choices) {

            case 1:

            System.out.println("Addition of "+number1+" and "+number2);

            System.out.println(number1 + number2); break;

            case 2:

            System.out.println("Subtraction of "+number1+" and "+number2);

            System.out.println(number1 - number2); break;

            case 3:

            System.out.println("Multiply of "+number1+" and "+number2);

            System.out.println(number1 \* number2); break;

            case 4:

            System.out.println("Division of "+number1+" and "+number2);

            System.out.println(number1 / number2); break;

            default:

            if(choices<1 || choices>4)

            {

                System.out.println("Enter valid number!");

            }

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

            break;

        }

        scanner.close();

    }

}

// Enter 1st number: 5

// Enter 2st number: 3

// 1st Number: 5

// 2nd Number: 3

// Enter Choices:

//  1. Add

//  3. Sub

//  3. Mul

//  4. Div

// Enter Choice Number: 3

// Multiply of 5 and 3

// 15

Switch

1. Case2
2. Case3
3. Both 2 and 3 have same condition – fall / flow through
4. Also, we can use default first, but no break
5. Use on specific type
6. Inside case no condition

Ternary operator

1. Condition ? true:false

Week calendar

package com.jb.T07Conditionals;

import java.util.Scanner;

public class WeekCalender

{

    public static boolean isWeekday(int day)

    {

        if(day>=2 && day <=6)

        {

            return true;

        }

        else return false;

    }

    public static String nameOfDay(int day)

    {

        if(day>=1 && day <=7)

        {

            switch (day)

            {

                default: return day+" is";

                case 1: return "Sunday";

                case 2: return "monday";

                case 3: return "tuesday";

                case 4: return "wednesday";

                case 5: return "thursday";

                case 6: return "friday";

                case 7: return "Saterday";

            }

        }

        else return "Enter valid day number (1-7)!";

    }

    public static String nameOfMonth(int month)

    {

        if(month>=1 && month <=12)

        {

            switch (month)

            {

                default: return month+" is";

                case 1: return "jan";

                case 2: return "feb";

                case 3: return "mar";

                case 4: return "apr";

                case 5: return "may";

                case 6: return "june";

                case 7: return "july";

                case 8: return "aug";

                case 9: return "sep";

                case 10: return "oct";

                case 11: return "nov";

                case 12: return "dec";

            }

        }

        else return "Enter valid month number (1-12)!";

    }

    public static void main(String[] args)

    {

        Scanner scanner=new Scanner(System.in);

        System.out.println("Enter day (1-7): ");

        int day = scanner.nextInt();

        System.out.println("Enter Month (1-12)");

        int month = scanner.nextInt();

        if(isWeekday(day))

        {

            System.out.println(day+"th day is week day");

        }

        else System.out.println(day+"th day is not week day");

        System.out.println(nameOfDay(day));

        System.out.println(nameOfMonth(month));

        scanner.close();

    }

}

// Enter day (1-7):

// 4

// Enter Month (1-12)

// 10

// 4th day is week day

// wednesday

// oct

1. **Loops**
2. Run same code again and again
3. For, while, do while

For

1. Initialization; Condition; Operation
2. ; must
3. Condition only mandatory

Prime Number (n), sum of numbers of n, sum of divisor of n, number triangle

Prime number

1. Number divisible from 2 to all numbers are prime

package com.jb.T08Loops;

import java.util.Scanner;

public class PrimeNumber

{

    private static boolean isPrime(int number)

    {

        //2,3,5,7,11,13,....

        for(int i=2;i<=number;i++) //check number from to 2 to n number

        {

            if(number%2==0) //check number divisble by any numbet from 2

            {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args)

    {

        Scanner scanner=new Scanner(System.in);

        System.out.print("Enter Number: ");

        int number = scanner.nextInt();

        System.out.println(number+" is prime number? \n"+isPrime(number));

        scanner.close();

    }

}

// Enter Number: 2

// 2 is prime number?

// false

Sum of n numbers

1. For sum always use as temp variable

package com.jb.T08Loops;

import java.util.Scanner;

public class SumOfN\_Numbers

{

    private static int sumofN\_Numbers(int number)

    {

        int sum = 0;

        for(int i=1;i<=number;i++) //check number from to 2 to n number

        {

            sum = sum + i;

        }

        return sum;

    }

    public static void main(String[] args)

    {

        Scanner scanner=new Scanner(System.in);

        System.out.print("Enter Number: ");

        int number = scanner.nextInt();

        System.out.println("Sum upto "+number+": "+sumofN\_Numbers(number));

        scanner.close();

    }

}

// Enter Number: 6

// Sum upto 6: 21

Sum of Divisors

1. Use modulus – if number remainder is zero then take that number and add(sum – temp) and return

package com.jb.T08Loops;

import java.util.Scanner;

public class SumOfDivisorOfN

{

    private static int sumofN\_Numbers(int number)

    {

        int sum = 0;

        for(int i=1;i<=number;i++) // include 1(first) and last divisor --- i=2;i<number;i++ - for divisor between first and last

         {

            if(number%i==0)

            {

                sum = sum+i;

            }

        }

        return sum;

    }

    public static void main(String[] args)

    {

        Scanner scanner=new Scanner(System.in);

        System.out.print("Enter Number: ");

        int number = scanner.nextInt();

        System.out.println("Sum of Divisor of "+number+": "+sumofN\_Numbers(number));

        scanner.close();

    }

}

// Enter Number: 6  (1x2,2x3,3x2,6x1)

// Sum of Divisor of 6: 12 (1+2+3+6)

Number Triangle (Right andgle triangle)

1. Nested For Loop

package com.jb.T08Loops;

import java.util.Scanner;

public class N\_NumberTriangle

{

    public static void numberTriangle(int number)

        {

            for(int i=1;i<=number;i++)

            {

                for(int j=1;j<=i;j++)

                {

                    System.out.print(j);

                }

                System.out.println();

            }

        }

        public static void main(String[] args)

        {

            Scanner scanner=new Scanner(System.in);

            System.out.print("Enter Number: ");

            int number = scanner.nextInt();

            numberTriangle(number);

        scanner.close();

    }

}

// 1

// 12

// 123

// 1234

// 12345

// 123456

While Loop

1. Outside implementation
2. 1st condition
3. 2nd operation
4. Execution stops only when the condition become true

Squares of numbers till N

I =0; while (i\*i<=number) { sysout(i\*i); i++}

package com.jb.T08Loops;

import java.util.Scanner;

public class SquaresOfNumbersTillN

{

    public static void numberSquares(int number)

    {

        int i=1;

        while(i\*i<=number)

        {

            System.out.println(i\*i);

            i++;

        }

    }

        public static void main(String[] args)

        {

            Scanner scanner=new Scanner(System.in);

            System.out.print("Enter Number: ");

            int number = scanner.nextInt();

            numberSquares(number);

        scanner.close();

    }

}

// Enter Number: 25

// 1

// 4

// 9

// 16

// 25

Cubes Of Number Till N

I =0; while (i\*i\*i<=number) { sysout(i\*i\*i); i++}

package com.jb.T08Loops;

import java.util.Scanner;

public class CubesOfNumbersTillN

{

    public static void numberCubes(int number)

    {

        int i=1;

        while(i\*i\*i<=number)

        {

            System.out.println(i\*i\*i);

            i++;

        }

    }

        public static void main(String[] args)

        {

            Scanner scanner=new Scanner(System.in);

            System.out.print("Enter Number: ");

            int number = scanner.nextInt();

            numberCubes(number);

        scanner.close();

    }

}

// Enter Number: 25

// 1

// 8

Do While Loop

1. Do
2. { operation)
3. While(Condtion)
4. Code will executes at least once
5. Do while this condition satisfies
6. Condition at last – do while
7. Condition at first – while

package com.jb.T08Loops;

import java.util.Scanner;

public class FindCubeNumber

{

        public static void main(String[] args)

        {

            Scanner scanner=new Scanner(System.in);

            int number=0;

            do

            {

                System.out.print("Enter Number: ");

                number = scanner.nextInt();

                System.out.println(number\*number\*number);

            }

            while(number>=0);

            scanner.close();

        }

}

// Enter Number: 3

// 27

// Enter Number: 2

// 8

// Enter Number: 5

// 125

// Enter Number: 0

// 0

// Enter Number: 2

// 8

// Enter Number: -1

// -1

Break and Continue

1. Break – break out of the loop
2. Continue – skip that iteration and continues till loops end

For vs While vs Do While

1. Know how many times – for
2. Know when loop ends – while
3. Loop at least once – do while
4. Readability matters
5. **Reference Type**
6. Any class we create is reference type
7. Instances of class – reference variable
8. Vary from primitive variable
9. 2 types memory – stack and heap
10. Object stored in heap
11. Heap contains location
12. Stack – variable name, reference value of object, location
13. Stack – variable and reference of the object. Eg:1A
14. Null -> dog – copy the reference values
15. 1c -> null
16. Null =dog
17. 1c -> dog
18. Each method has a separate stack
19. Heap – global shared
20. Null == dog -> true (same reference value, not depend on same value)

String Literal – Instance of class String

1. .length()
2. .charAt(0)
3. .substring(5,13) – from 5 index to 12 index – includes gaps
4. .indexOf(‘jaya’) – returns location of j – 1
5. .indexOf(‘jaya’) - 1
6. In multiple chat - .lastIndexOf(‘a’) – 3
7. .contains(‘ay’) – true
8. .startsWith(‘ja’) – true
9. .endsWith(‘ay’) – false
10. .isEmpty() – false
11. ‘jaya’.equals(‘bala’) – false
12. ‘jaYa’.equalsIgnoreCase(‘Jaya’) - true
13. String is immutable – not change
14. Str =”aya”
15. ‘j’.concat(str) – jaya
16. Str – aya (not changed)
17. Create new string to change
18. Old string remains unaffected
19. .toUpperCase()
20. .toLowerCase
21. .trim() – remove spaces from start and end
22. String Concatenation
23. 1+2=3
24. “1”+”2”=”12”
25. 1+2+”3”=”33”
26. Integer + String = integer(op)string
27. String + Integer = string(full)
28. “33” +1 +3 = ”3313” (left to right conversion)
29. Use brackets for priority
30. “33+(1+3) = “334”
31. String.join(“+”, “2”,”3”,”4”) -> “2+3+4” – index one for representing the char to append / join
32. .replace(“jaya”,”JAYA”) – jaya -> JAYA
33. Str. -> jshell
34. Java doc -> google

StringBuffer and StringBuilder (mutable)

1. “j”+”a”+”y”+”a” ->”jaya” (j,a,y,a ->ja,y,a -> jay,a -> jaya
2. 1 (1)
3. 1+1=2 (3)
4. (3)+1+1+1 – 6
5. 7
6. Instance created -> 7 for 4 concat

StringBuffer

1. Like object creation – new StringBuffer
2. Mutable (used for changeable string)
3. .append(‘a’)
4. .setCharAt(1,’a’)

StringBuilder

1. Synchronized class – ready for multithreading

Wrapper Classes

1. Primitive types – wrapper around values
2. Offers additional options
3. Use to store primitive values to collections -> then it became objects
4. Integer integer = new Integer(5); / Integer.valuesOf(5);
5. Wrapper class – immutable
6. valueOf use same reference value/address
7. but new integer creates new reference value for same elements

Auto boxing

1. Interger int1= 7 -> Integer integer = Integer.valueOf(7);
2. .MAX\_VALUE, .MIN\_VALUE, .SIZE, .BYTE

Date API

1. JODA Time Framework
2. LocalDate
3. LocalDateTime
4. LocalTime
5. Package – java.time
6. .now – yyyy-mm-dd
7. 2025-01-15T18:12:39.586066600
8. 18:12:59.818063500
9. Today.getYear()
10. Today.getDayOfMonth(), Week,Year
11. Today.getMonth()
12. Today.getMonthValue()
13. Today.getYear()
14. .lendthOfYear,Month()
15. today.plusDays(100) -> retuen date
16. today – immutable
17. LocalDate.of(2018,01,31) – set date
18. .withMonth() compare and return value
19. .isBefore(yesterday) -check and return true or false
20. **Array and ArrayList**

Array – collection of data with same datatype

package com.jb.T10ArrayArrayList;

import java.util.Arrays;

public class Array

{

    public static void main(String[] args)

    {

        int [] arr1 = new int[5];

        int [] arr2 = {1,2,3,4,5,6};

        arr1[0]=1;

        arr1[1]=19;

        arr1[2]=13;

        arr1[3]=12;

        arr1[4]=14;

        System.out.println(arr1);

        for(int element:arr1)

        {

            System.out.println(element);

        }

        for(int i=arr2.length;i>=arr2[0];i--)

        {

            System.out.println(i);

        }

        System.out.println(Arrays.toString(arr2)); //[1, 2, 3, 4, 5, 6]

    }

}

// 1

// 19

// 13

// 12

// 14

// 6

// 5

// 4

// 3

// 2

// 1

1. //Enhanced for Loop
2. Arrays – Static Methods
3. Arrays.fill(arr,100)
4. Arrays.equal(arr1,arr2) – length and elements match

package com.jb.T10ArrayArrayList;

import java.math.BigDecimal;

public class Student

{

    private int[] marks;

    public Student(String name, int[] marks)

    {

        this.marks=marks;

    }

    public int getNumberOfMarks()

    {

        return marks.length;

    }

    public int getSumOfMarks()

    {

        int sum=0;

        for(int mark:marks)

        {

            sum=sum+mark;

        }

        return sum;

    }

    public int getMaxMarks()

    {

        int max=0;

        for(int mark:marks)

        {

            if(mark>max)

            {

                max=mark;

            }

        }

        return max;

    }

    // public int getMinMarks()

    // {

    //     int max=getMaxMarks();

    //     for(int mark:marks)

    //     {

    //         if(mark<max) // > max;  opp < min mark

    //         {

    //             max=mark;

    //         }

    //     }

    //     return max;

    // }

    public int getMinMarks()

    {

        int max=Integer.MAX\_VALUE;

        int min = max;

        for(int mark:marks)

        {

            if(mark<min) // > max;  opp < min mark

            {

                min=mark;

            }

        }

        return min;

    }

    public BigDecimal getAverageOfMarks()

    {

        int total=getSumOfMarks();

        int totalMarks=getNumberOfMarks();

        // return new BigDecimal(total/totalMarks); // no decimal

        return new BigDecimal(total).divide(new BigDecimal(totalMarks)); //accurate

    }

        public static void main(String[] args)

        {

            int [] marks ={98,99,100,97,98};

            Student student = new Student("jaya",marks);

            System.out.println("Number of Marks: "+student.getNumberOfMarks());

            System.out.println("sum of Marks: "+student.getSumOfMarks());

            System.out.println("Maximum Mark: "+student.getMaxMarks());

            System.out.println("Minimum Mark: "+student.getMinMarks());

            System.out.println("Average Mark: "+student.getAverageOfMarks());

      }

}

// Number of Marks: 5

// sum of Marks: 492

// Maximum Mark: 100

// Minimum Mark: 97

// Average Mark: 98.4

1. Variable Argument – int… (3 dots) – always want to be a last argument
2. Object array

package com.jb.T10ArrayArrayList;

public class WeekDays

{

    public static void main(String[] args)

    {

        String[] days = {"sunday","monday","tuesday","wednesday","thursday","friday","saturday"};

        String dayWithMostChar="";

        for(String day:days)

        {

            if(day.length()>dayWithMostChar.length())

            {

                dayWithMostChar=day;

            }

        }

        System.out.println("Day with most number of Characters: "+dayWithMostChar);

        for(int i=days.length-1;i>=0;i--)

        {

            System.out.println(days[i]);

        }

    }

}

// Day with most number of Characters: wednesday

// saturday

// friday

// thursday

// wednesday

// tuesday

// monday

// sunday

ArrayList

1. Arrays sizes are immutable
2. ArrayList – adding and removing elements in array
3. .add(\_\_)
4. .remove(\_\_)
5. Add any datatypes
6. <String> - generics – define datatypes – only add/remove elements on same datatypes
7. Array -> .length()
8. ArrayList -> .size()

package com.jb.T10ArrayArrayList;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListStudent

{

    private ArrayList<Integer> marks = new ArrayList<Integer>();

        @SuppressWarnings("unused")

        private String name;

        public ArrayListStudent(String name, int... marks)

        {

            this.name=name;

        for(int mark:marks)

        {

            this.marks.add(mark);

        }

    }

    public int getNumberOfMarks()

    {

        return marks.size();

    }

    public int getSumOfMarks()

    {

        int sum=0;

        for(int mark:marks)

        {

            sum=sum+mark;

        }

        return sum;

    }

    public int getMaxMarks()

    {

        return Collections.max(marks);

    }

    public int getMinMarks()

    {

        return Collections.min(marks);

    }

    public BigDecimal getAverageOfMarks()

    {

        int total=getSumOfMarks();

        int totalMarks=getNumberOfMarks();

        // return new BigDecimal(total/totalMarks); // no decimal

        return new BigDecimal(total).divide(new BigDecimal(totalMarks)); //accurate

    }

    @Override

    public String toString()

    {

        return name+" "+marks;

    }

        public static void main(String[] args)

        {

            int [] marks ={98,99,100,97,98};

            ArrayListStudent student = new ArrayListStudent("jaya",marks);

            System.out.println("Number of Marks: "+student.getNumberOfMarks());

            System.out.println("sum of Marks: "+student.getSumOfMarks());

            System.out.println("Maximum Mark: "+student.getMaxMarks());

            System.out.println("Minimum Mark: "+student.getMinMarks());

            System.out.println("Average Mark: "+student.getAverageOfMarks());

            System.out.println("Name and Marks: "+student.toString());

      }

}

// Number of Marks: 5

// sum of Marks: 492

// Maximum Mark: 100

// Minimum Mark: 97

// Average Mark: 98.4

1. Adding and Delete in Student Marks

 public void addNewMark(int mark)

    {

        marks.add(mark);

    }

    public void removeMarkAtIndex(int index)

    {

        marks.remove(index);

    }

1. **OOPS Again**
2. Object – state
3. Methods – behaviors of object
4. Constructors

Designing the Class

1. Fan
2. State – make, radius, color, isOn, speed
3. Constructor – (default – initial statee)
4. Behaviors – switchOn(), switchOff(), setSpeed(),
5. toString() – to print object behavior

package com.jb.T11OOPSAgain;

public class Fan {

    //state

    //basic

    private String make;

    private double radius;

    private String color;

    //other

    private boolean isOn;

    private byte speed; //1 -5

    //constructor -

    public Fan(String make, double radius, String color)

    {

        this.make=make; this.radius=radius; this.color=color;

    }

    @Override

    public String toString() {

        //return "Fan [make=" + make + ", radius=" + radius + ", color=" + color + ", isOn=" + isOn + ", speed="+ speed + "]";

        return String.format("make- %s , radius- %f ,color- %s ,isOn- %b ,speed- %d", make,radius,color,isOn,speed);

    }

    //States

    //isOn

    public void switchOn()

    {

        this.isOn=true;

        setSpeed((byte)5);

    }

    public void switchOff()

    {

        this.isOn=false;

        setSpeed((byte)0);

    }

    public void setSpeed(byte speed)

    {

        this.speed=speed;

    }

        public static void main(String[] args) {

            Fan fan=new Fan("Manufact 1",7.537,"brown");

            System.out.println(fan);

            fan.switchOn();

            System.out.println(fan);

            fan.setSpeed((byte)4); //integer -> byte - type cast

            System.out.println(fan);

            fan.switchOff();

            System.out.println(fan);

    }

}

// make- Manufact 1 , radius- 7.537000 ,color- brown ,isOn- false ,speed- 0

// make- Manufact 1 , radius- 7.537000 ,color- brown ,isOn- true ,speed- 5

// make- Manufact 1 , radius- 7.537000 ,color- brown ,isOn- true ,speed- 4

// make- Manufact 1 , radius- 7.537000 ,color- brown ,isOn- false ,speed- 0

1. Rectangle Class

package com.jb.T11OOPSAgain;

public class Rectangle {

    //state

    private int length;

    private int width;

    //behavior

    public Rectangle(int length, int width) {

            this.length=length;

            this.width=width;

        }

    public int getLength() {

        return length;

    }

    public void setLength(int length) {

        this.length = length;

    }

    public int getWidth() {

        return width;

    }

    public void setWidth(int width) {

        this.width = width;

    }

    @Override

    public String toString() {

        return "Rectangle [length=" + length + ", width=" + width + ", area()=" + area() + ", perimeter()="

                + perimeter() + "]";

    }

    //action

    public int area()

    {

        return length\*width;

    }

    public int perimeter()

    {

        return 2\*(length+width);

    }

        public static void main(String[] args) {

            Rectangle rectangle=new Rectangle(12,13);

            System.out.println(rectangle);

    }

}

// Rectangle [length=12, width=13, area()=156, perimeter()=50]

1. Object Composition – Object inside Object
2. Customer Address

package com.jb.T11OOPSAgain;

public class Customer

{

    //state

    private String name;

    private Address homeAddress; // address is another class

    private Address workAddress; // address is another class

    //constructor - behavior

    public Customer(String name, Address homeAddress)

    {

        this.name=name;

        this.homeAddress=homeAddress;

    }

    //action

public Address getHomeAddress() {

    return homeAddress;

}

public void setHomeAddress(Address homeAddress) {

    this.homeAddress = homeAddress;

}

public Address getWorkAddress() {

    return workAddress;

}

public void setWorkAddress(Address workAddress) {

    this.workAddress = workAddress;

}

    @Override

public String toString() {

    return "Customer [name=" + name + ", homeAddress=" + homeAddress + ", workAddress=" + workAddress + "]";

}

    public static void main(String[] args) {

        Address homeAddress=new Address("plqt", "cpt", "603002");

        Customer customer=new Customer("jaya",homeAddress);

        Address workAddress=new Address("plqt", "erd", "603020");

        customer.setWorkAddress(workAddress);

        System.out.println(customer);

    }

}

// Customer [name=jaya, homeAddress=Address [line1=plqt, city=cpt, zip=603002], workAddress=Address [line1=plqt, city=erd, zip=603020]]

package com.jb.T11OOPSAgain;

public class Address {

    //state

    private String line1;

    private String city;

    private String zip;

    //behavior

    public Address(String line1, String city, String zip) {

        this.line1 = line1;

        this.city = city;

        this.zip = zip;

    }

    @Override

    public String toString() {

        return "Address [line1=" + line1 + ", city=" + city + ", zip=" + zip + "]";

    }

}

1. ep 07: Programming Exercise PE-OOP-02
2. Exercises
3. Write a program that manages Books and their Reviews:
4. Book:
5. Id
6. Name
7. Author
8. Review:
9. Id
10. Description
11. Rating
12. Book book = new Book(123, "Object Oriented Programming With Java", "Ranga");
13. book.addReview(new Review(10, "Great Book", 4));
14. book.addReview(new Review(101, "Awesome", 5));
15. System.out.println(book);
16. Solution To PE-OOP-02
17. BookReviewRunner.java
18. package ;
19. public class BookReviewRunner {
20. public static void main(String[] args) {
21. Book book = new Book(123, "Object Oriented Programming With Java", "Ranga");
22. book.addReview(new Review(10, "Great Book", 4));
23. book.addReview(new Review(101, "Awesome", 5));
24. System.out.println(book);
25. }
26. }
27. Review.java
28. package ;
29. public class Review {
30. private int id;
31. private String description;
32. private byte rating;
33. public Review(int id, String description, byte rating) {
34. this.id = id;
35. this.description = description;
36. this.rating = rating;
37. }
38. public String toString() {
39. return "(Review-" + id + ", " + description + ", " + rating + ")";
40. }
41. }
42. Book.java
43. package ;
44. public class Book {
45. private int id;
46. private String title;
47. private String author;
48. private ArrayList<Review> reviewList = new ArrayList<Review>();
49. public Book(int id, String title, String author) {
50. this.id = id;
51. this.title = title;
52. this.author = author;
53. }
54. public void addReview(Review review) {
55. reviewList.add(review);
56. }
57. public String toString() {
58. return "Book-" + id + ", " + title + ", " + author + ", " + reviews);
59. }
60. }
61. com.in28minutes.oops.level2
62. com.in28minutes.oops.level2
63. com.in28minutes.oops.level2
64. Console Output
65. Book-123, Object Oriented Programming With Java, Ranga, [(Review-10, Great Book", 4), (Review-101, Awesome,
66. 5)]

Inheritance

1. Reuse code, instead of creating new class
2. Using keywords “extends”
3. Child (sub) class have behaviors of parent (super) class
4. Is A (stu is a Per)
5. Every class extends Object – Default (root of the class hierarchy)

package com.jb.T11OOPSAgain;

public class Person extends Object

{

    private String name;

    private String mail;

    private String phoneNumber;

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getMail() {

        return mail;

    }

    public void setMail(String mail) {

        this.mail = mail;

    }

    public String getPhoneNumber() {

        return phoneNumber;

    }

    public void setPhoneNumber(String phoneNumber) {

        this.phoneNumber = phoneNumber;

    }

    public static void main(String[] args) {

    }

}

package com.jb.T11OOPSAgain;

public class Student extends Person

{

    private String college;

    private int year;

    public String getCollege() {

        return college;

    }

    public void setCollege(String college) {

        this.college = college;

    }

    public int getYear() {

        return year;

    }

    public void setYear(int year) {

        this.year = year;

    }

    public static void main(String[] args) {

        Student student=new Student();

        student.setName("jaya");

        student.setMail("jaya04@gmail.com");

        System.out.println(student.getName());//jaya

        Person person=new Person();

        String value=person.toString();

        System.out.println(person); //com.jb.T11OOPSAgain.Person@5b275dab

        System.out.println(value); //com.jb.T11OOPSAgain.Person@5b275dab

    }

}

Method Overriding

1. Same method, same parameter, but different behaviour
2. Override the implementation by super class inside your sum class

package com.jb.T11OOPSAgain;

public class Person extends Object

{

    private String name;

    private String mail;

    private String phoneNumber;

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getMail() {

        return mail;

    }

    public void setMail(String mail) {

        this.mail = mail;

    }

    public String getPhoneNumber() {

        return phoneNumber;

    }

    public String toString() {

        return getClass().getName() + "#" + Integer.toHexString(hashCode()); //overriding

    }

    public void setPhoneNumber(String phoneNumber) {

        this.phoneNumber = phoneNumber;

    }

    public static void main(String[] args) {

    }

}

package com.jb.T11OOPSAgain;

public class Student extends Person

{

    private String college;

    private int year;

    public String getCollege() {

        return college;

    }

    public void setCollege(String college) {

        this.college = college;

    }

    public int getYear() {

        return year;

    }

    public void setYear(int year) {

        this.year = year;

    }

    public static void main(String[] args) {

        Student student=new Student();

        student.setName("jaya");

        student.setMail("jaya04@gmail.com");

        System.out.println(student.getName());//jaya

        Person person=new Person();

        String value=person.toString();

        System.out.println(person); //com.jb.T11OOPSAgain.Person#5b275dab

        System.out.println(value); //com.jb.T11OOPSAgain.Person#5b275dab

    }

}

Super class object, method:

1. Using super keyword to append the parent class objects

package com.jb.T11OOPSAgain;

import java.math.BigDecimal;

public class Employee extends Person

{

    private String title;

    private String employeeName;

    private char employeeGrade;

    private BigDecimal salary;

    public String getTitle() {

        return title;

    }

    public void setTitle(String title) {

        this.title = title;

    }

    public String getEmployeeName() {

        return employeeName;

    }

    public void setEmployeeName(String employeeName) {

        this.employeeName = employeeName;

    }

    public char getEmployeeGrade() {

        return employeeGrade;

    }

    public void setEmployeeGrade(char employeeGrade) {

        this.employeeGrade = employeeGrade;

    }

    public BigDecimal getSalary() {

        return salary;

    }

    public void setSalary(BigDecimal salary) {

        this.salary = salary;

    }

    @Override

    public String toString() {

        //return "Employee [title=" + title + ", employeeName=" + employeeName + ", employeeGrade=" + employeeGrade + "Email: " +super.getMail() + "name: "+ super.getName() + ",]";

        return super.toString() + " Employee [title=" + title + ", employeeName=" + employeeName + ", employeeGrade=" + employeeGrade +",]";

    }

}

package com.jb.T11OOPSAgain;

public class Student extends Employee

{

    private String college;

    private int year;

    public String getCollege() {

        return college;

    }

    public void setCollege(String college) {

        this.college = college;

    }

    public int getYear() {

        return year;

    }

    public void setYear(int year) {

        this.year = year;

    }

    public static void main(String[] args) {

        Employee employee=new Employee();

        employee.setName("jaya");

        employee.setMail("jaya04@gmail.com");

        employee.setEmployeeName("bala");

        employee.setPhoneNumber("8072");

        employee.setEmployeeGrade('A');

        employee.setTitle("ASE");

        System.out.println(employee.toString());

        // Person [name=jaya, mail=jaya04@gmail.com, phoneNumber=8072] Employee [title=ASE, employeeName=bala, employeeGrade=A,]

        // Employee [title=ASE, employeeName=bala, employeeGrade=AEmail: jaya04@gmail.comname: jaya,]

        // Student student=new Student();

        // student.setName("jaya");

        // student.setMail("jaya04@gmail.com");

        // System.out.println(student.getName());//jaya

        // Person person=new Person();

        // String value=person.toString();

        // System.out.println(person); //com.jb.T11OOPSAgain.Person#5b275dab

        // System.out.println(value); //com.jb.T11OOPSAgain.Person#5b275dab

    }

}

1. Super() class constructor
2. Preson<- Employee<- Student

package com.jb.T11OOPSAgain;

public class Person extends Object

{

    private String name;

    private String mail;

    private String phoneNumber;

    public Person(String name) {

        super();

        System.out.println("Person Constructor");

        this.name = name;

    }

    public String getName() {

        return name;

    }

    // public void setName(String name) {

    //     this.name = name;

    // }

    public String getMail() {

        return mail;

    }

    public void setMail(String mail) {

        this.mail = mail;

    }

    public String getPhoneNumber() {

        return phoneNumber;

    }

    @Override

    public String toString() {

        return "Person [name=" + name + ", mail=" + mail + ", phoneNumber=" + phoneNumber + "]";

    }

    public void setPhoneNumber(String phoneNumber) {

        this.phoneNumber = phoneNumber;

    }

    public static void main(String[] args) {

    }

}

package com.jb.T11OOPSAgain;

import java.math.BigDecimal;

public class Employee extends Person

{

    private String title;

    private String employeeName;

    private char employeeGrade;

    private BigDecimal salary;

    public Employee(String name, String title) {

        super(name);

        this.title=title;

        System.out.println("Employee Constructor");

    }

    public String getTitle() {

        return title;

    }

    public void setTitle(String title) {

        this.title = title;

    }

    public String getEmployeeName() {

        return employeeName;

    }

    public void setEmployeeName(String employeeName) {

        this.employeeName = employeeName;

    }

    public char getEmployeeGrade() {

        return employeeGrade;

    }

    public void setEmployeeGrade(char employeeGrade) {

        this.employeeGrade = employeeGrade;

    }

    public BigDecimal getSalary() {

        return salary;

    }

    public void setSalary(BigDecimal salary) {

        this.salary = salary;

    }

    @Override

    public String toString() {

        //return "Employee [title=" + title + ", employeeName=" + employeeName + ", employeeGrade=" + employeeGrade + "Email: " +super.getMail() + "name: "+ super.getName() + ",]";

        return super.toString() + " Employee [title=" + title + ", employeeName=" + employeeName + ", employeeGrade=" + employeeGrade +",]";

    }

}

package com.jb.T11OOPSAgain;

public class Student extends Employee

{

    private String college;

    private int year;

    public Student(String name, String college) {

        super(name, college);

        this.college=college;

        System.out.println("Student Constructor");

    }

    public String getCollege() {

        return college;

    }

    public void setCollege(String college) {

        this.college = college;

    }

    public int getYear() {

        return year;

    }

    public void setYear(int year) {

        this.year = year;

    }

    public static void main(String[] args) {

        Employee employee=new Employee("jaya","ASE");

        //employee.setName("jaya");

        employee.setMail("jaya04@gmail.com");

        employee.setEmployeeName("bala");

        employee.setPhoneNumber("8072");

        employee.setEmployeeGrade('A');

        employee.setTitle("ASE");

        System.out.println(employee);

        // Person [name=jaya, mail=jaya04@gmail.com, phoneNumber=8072] Employee [title=ASE, employeeName=bala, employeeGrade=A,]

        // Employee [title=ASE, employeeName=bala, employeeGrade=AEmail: jaya04@gmail.comname: jaya,]

        // Student student=new Student();

        // student.setName("jaya");

        // student.setMail("jaya04@gmail.com");

        // System.out.println(student.getName());//jaya

        // Person person=new Person();

        // String value=person.toString();

        // System.out.println(person); //com.jb.T11OOPSAgain.Person#5b275dab

        // System.out.println(value); //com.jb.T11OOPSAgain.Person#5b275dab

    }

}

//Person Constructor

//Employee Constructor

//Person [name=jaya, mail=jaya04@gmail.com, phoneNumber=8072] Employee [title=ASE, employeeName=bala, employeeGrade=A,]

1. Multiple inheritance not supports in java
2. But allow chain of classes
3. Animal <- Pet <- Dog
4. Dog extends pet
5. Pet extends Animal
6. Animal extend Object(root)
7. In super class we can hold sub class variable
8. Pet instanceof Dog -> true

Abstract Class

1. Use keyword abstract
2. When don’t need to provide implementation, I only provide alogorithm, I don’t want to provide how to done, sub class is should take care of this.
3. To provide the blueprint of other classes

package com.jb.T11OOPSAgain;

public abstract class AbstractRecipe {

    public void execute()

    {

        getReady();

        doTheDish();

        cleanUp();

    }

    abstract void getReady();

    abstract void doTheDish();

    abstract void cleanUp();

}

package com.jb.T11OOPSAgain;

public class Recipe1 extends AbstractRecipe{

    @Override

    void getReady() {

        System.out.println("Get the raw marterials");

    }

    @Override

    void doTheDish() {

        System.out.println("do the dish");

    }

    @Override

    void cleanUp() {

        System.out.println("Clean UP the utensils");

    }

    public static void main(String[] args) {

        Recipe1 recipe1=new Recipe1();

        recipe1.execute();

    }

}

// Get the raw marterials

// do the dish

// Clean UP the utensils

package com.jb.T11OOPSAgain;

public class RecipeWithMicroWave extends AbstractRecipe{

    @Override

    void getReady() {

        System.out.println("Get the raw marterials");

        System.out.println("switch on the microwave");

    }

    @Override

    void doTheDish() {

        System.out.println("do the dish");

    }

    @Override

    void cleanUp() {

        System.out.println("Clean UP the utensils");

        System.out.println("switch off the microwave");

    }

    public static void main(String[] args) {

        RecipeWithMicroWave recipeWithMicroWave=new RecipeWithMicroWave();

        recipeWithMicroWave.execute();

    }

}

// Get the raw marterials

// switch on the microwave

// do the dish

// Clean UP the utensils

// switch off the microwave

1. We can create abstract class without any method present in that class
2. New class extend abstract class must be abstract or have abstract method
3. Abstract class can extend another abstract class
4. Abstract class allow non abstract method

Interface

1. Game <- remote button(interface)
2. Same code different implementation – polymorphism
3. Extends – inheritance – behavior between the classes
4. Implements – interface – action between the classes
5. Support multiple inheritance in interface
6. Java doesn’t support multiple inheritance due to ambiguity

package com.jb.T11OOPSAgain;

public interface GamingConsole

{

    public void up();

    public void down();

    public void left();

    public void right();

}

package com.jb.T11OOPSAgain;

public class MarioGame implements GamingConsole

{

    @Override

    public void up() {

    System.out.println("jump");

    }

    @Override

    public void down() {

        System.out.println("goes into hole");

    }

    @Override

    public void left() {

        System.out.println("go back");

    }

    @Override

    public void right() {

        System.out.println("go front");

      }

}

package com.jb.T11OOPSAgain;

public class ChessGame implements GamingConsole

{

    @Override

    public void up() {

    System.out.println("Move Piece UP");

    }

    @Override

    public void down() {

        System.out.println("Move Piece DDown");

    }

    @Override

    public void left() {

        System.out.println("Move Piece Left");

    }

    @Override

    public void right() {

        System.out.println("Move Piece Right");

      }

}

package com.jb.T11OOPSAgain;

public class GameRunner {

    public static void main(String[] args) {

       //GamingConsole game=new MarioGame();

        GamingConsole game= new ChessGame();

        game.up();;

        game.down();

        game.left();

        game.right();

    }

}

//chess

// Move Piece UP

// Move Piece DDown

// Move Piece Left

// Move Piece Right

//mario

// jump

// goes into hole

// go back

// go front

1. Interface is used to communicate between the projects
2. Interface is a contract

package com.jb.T11OOPSAgain;

public interface ComplexAlgorithm {

    int ComplexAlgorithm(int number1, int number2);

}

package com.jb.T11OOPSAgain;

public class DummyAlgorithm implements ComplexAlgorithm{

    @Override

    public int ComplexAlgorithm(int number1, int number2) {

    return number1+number2;

    }

}

package com.jb.T11OOPSAgain;

public class RealAlgorithm implements ComplexAlgorithm{

    @Override

    public int ComplexAlgorithm(int number1, int number2) {

    return number1\*number2;

    }

}

package com.jb.T11OOPSAgain;

public class Project {

    public static void main(String[] args) {

        ComplexAlgorithm algorithm=new RealAlgorithm();

        System.out.println(algorithm.ComplexAlgorithm(10, 20));//200

    }

}

1. Interface can extends another interface
2. Class implements Interface, also want to implement the all methods
3. Abstract class implements abstract interface, can implements the parts / methods of the class
4. Use interface keyword to create interface
5. Interface we can create constant, not variable
6. Not int var; but int var=5;(constant not changeable)
7. Interface allow default method implementation

Abstract class vs interface

1. Communication – relation -> one party implementation and another party interface
2. Abstract – inheritance – is A – share blue print with other class
3. Abstract can have all kind of variables, but interface on constant
4. Interface – multiple inheritance
5. Interface – Common method for 2 different stuffs

package com.jb.T11OOPSAgain;

interface Flyable

{

    void fly();

}

class Bird implements Flyable

{

    public void fly()

    {

        System.out.println("with wings");

    }

}

class Aeroplane implements Flyable

{

    public void fly()

    {

        System.out.println("with fuel");

    }

}

public class FlyableRunner {

 public static void main(String[] args) {

    Flyable[] flyableObjects = { new Bird(), new Aeroplane()};

    for(Flyable object: flyableObjects)

    {

        object.fly();

    }

 }

};

// with wings

// with fuel

1. 97 Things Every Programmer should know <https://github.com/97-things/97-things-every-programmer-should-know/tree/master/en>

Polymorphism

1. Same code from 2 different classes
2. Interface – Common / same reference for 2 different stuffs
3. Interclass – multiple classes don’t support inside another class

package com.jb.T11OOPSAgain;

abstract class Animal

{

    abstract void bark();

}

class Dog extends Animal{

    public void bark()

    {

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

    }

}

class Cat extends Animal{

    public void bark()

    {

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

    }

}

public class AnimalRunner {

    public static void main(String[] args) {

        Animal[] animals={new Cat(), new Dog()};

        for(Animal animal: animals)

        {

            animal.bark();

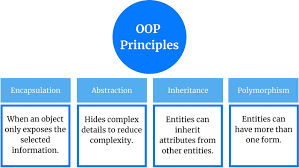
        }

    }

}

// meow meow

// bow bow

1. 

4 OOPS Principles

Thoughtworks Radar <https://www.thoughtworks.com/radar>

**Collections**

1. List
2. Set
3. Queue
4. Map

List [ ]

1. Allow duplicate elements
2. Cares about which position
3. Specifying position to added
4. No specification, it will be added at the end
5. Use List.of
6. List<String> word = List.of(“apple”,”bat”,”cat”);
7. Collection - > size()
8. .isEmpty()
9. words.get(0) -> apple
10. words.contains(“cat”) -> false
11. words.indexOf(“cat”) -> 2
12. .of -> immutable (not changeable)
13. Mutable -> ArrayList, LinkedList, Vector
14. List<String> wordArrayList = new ArrayList<String>(words)
15. .add(“Dog”) – app,bat,cat,dog
16. ArrayList vs LinkedList
17. LinkedList – Reference from one element to another element
18. LinkedList – Access: slow, Delete: Fast
19. ArrayList – Access: Fast, Insertion and Deletion: Slow
20. DoublyLinkedList – Link Forword and Backward
21. Vector (1.0) vs ArrayList (1.2)
22. Vector: Synchronized
23. ArrayList: Not Synchronized
24. Synchronized – 25 synchronized method, instance shared between multiple threads, only one threads is executed, thread safe(no change of behavior in thread) – each Method executed by thread
25. ArrayList – No Thread Safety
26. Vector – Thread Safety
27. Interface – Methods
28. ArrayList=[“ja”,”ya”]
29. .add(“j”) ->[“ja”,”ya”,”j”]
30. .add(2,”a”) -> [“ja”,”ya”,”a”,”j”]
31. .add(“j”) -> [“ja”,”ya”,”a”,”j”,”j”] – allow duplication
32. newArrayList=[“ba”,”la”]
33. ArrayList.addAll(newList) -> true
34. ArrayList -> [“ja”,”ya”,”a”,”j”,”j”,” ba”,”la”]
35. ArrayList.set(6,”jee”) -> replace
36. [“ja”,”ya”,”a”,”j”,”j”,” ba”,”jee”]
37. ArrayList.remove(2) -> delete
38. [“ja”,”ya”,”j”,”j”,” ba”,”jee”]
39. ArrayList.remove(“ba”)
40. [“ja”,”ya”,”j”,”j”,”jee”]
41. Iteration in List and ArrayList
42. For(i=0;i<words.size();i++) {
43. System.out.println(words.get(i));
44. }
45. For(String word:words)
46. {
47. Sysout(word);
48. }
49. List -> Iterator
50. Use while
51. Iterator wordsIterator=words.iterator();
52. While(wordsIterator.hasNext())
53. {
54. Sysout(wordsIterator.next());
55. }
56. Words=List.of(“aa”,”bbaa”,”ccab”);
57. List<String> wordsAL=new ArrayList<>(words);
58. For(String word:words) {
59. If(word.endWith(“at”)) {
60. Words.remove(word)
61. Sysout(“done”)
62. }
63. }
64. Error: we cant remove elements using for loop
65. So we can use iterator
66. wordsAL ==> [Apple, Cat]
67. jshell> Iterator wordsIterator = wordsAL.iterator(); wordsIterator ==> [java.util.AbstractList$Itr@3712b94](mailto:java.util.AbstractList$Itr@3712b94)
68. jshell> while(wordsIterator.hasNext()) {
69. ...> if(wordsIterator.next().endsWith("at")){
70. ...> wordsIterator.remove();
71. ...> }
72. ...> }
73. jshell> wordsAL
74. wordsAL ==> [Apple]
75. In List we can’t store primitive
76. Autobox
77. Wrapper Class -> string, Character, Integer Double
78. <String> - Only string
79. <Integer> - Only Integer Values
80. numberAl.remove(101); -> Intec
81. numberAl.remove(Integer.valueOf(101)); -> values

Sort List Of Numbers

Collections package - sort

package com.jb.T12Collections;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class SortingList {

    public static void main(String[] args) {

        List<Integer> numbers = List.of(21,3,78,9);

        List<Integer> numbersAL= new ArrayList<>(numbers);

        System.out.println(numbersAL); //[21, 3, 78, 9]

        Collections.sort(numbersAL);

        System.out.println(numbersAL); //[3, 9, 21, 78]

    }

}

ArrayList

Use collections to sort with implementing Comparable interface

package com.jb.T12Collections;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class Student implements Comparable<Student>

{

    private int id;

    private String name;

    Student(int id, String name)

    {

        this.id=id;

        this.name=name;

    }

    @Override

    public String toString() {

        return "Student [id=" + id + ", name=" + name + "]";

    }

    public int getId() {

        return id;

    }

    public void setId(int id) {

        this.id = id;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    @Override

    public int compareTo(Student that) {

       return Integer.compare(this.id,that.id);

    }

    public static void main(String[] args) {

        List <Student> students=List.of(new Student(1,"jaya"),

                                        new Student(4,"Bala"),

                                        new Student(2,"JB"),

                                        new Student(5,"Tharun"),

                                        new Student(3,"Eva"));

        System.out.println(students);

        // [Student [id=1, name=jaya], Student [id=2, name=Bala], Student [id=3, name=JB], Student [id=4, name=Tharun], Student [id=5, name=Eva]]

        ArrayList<Student> studentsAL=new ArrayList<>(students);

        System.out.println(studentsAL);

        // ArrayList before sorting: [Student [id=1, name=jaya], Student [id=4, name=Bala], Student [id=2, name=JB], Student [id=5, name=Tharun], Student [id=3, name=Eva]]

        Collections.sort(studentsAL);

        System.out.println(studentsAL);

        //Sorted ArrayList: [Student [id=1, name=jaya], Student [id=2, name=JB], Student [id=3, name=Eva], Student [id=4, name=Bala], Student [id=5, name=Tharun]]

    }

}

Ascending and Descending Sorting

package com.jb.T12Collections;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class Student implements Comparable<Student>

{

    private int id;

    private String name;

    Student(int id, String name)

    {

        this.id = id;

        this.name = name;

    }

    @Override

    public String toString() {

        return "Student [id=" + id + ", name=" + name + "]";

    }

    public int getId() {

        return id;

    }

    public void setId(int id) {

        this.id = id;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    @Override

    public int compareTo(Student that) {

        return Integer.compare(this.id, that.id);

    }

    // Static nested comparator class for descending order

    static class DescendingStudentComparator implements Comparator<Student>

    {

        @Override

        public int compare(Student student1, Student student2) {

            return Integer.compare(student2.getId(), student1.getId());  // Descending order

        }

    }

    public static void main(String[] args) {

        List<Student> students = List.of(

            new Student(1, "jaya"),

            new Student(4, "Bala"),

            new Student(2, "JB"),

            new Student(5, "Tharun"),

            new Student(3, "Eva")

        );

        System.out.println("Original List: " + students);

        ArrayList<Student> studentsAL = new ArrayList<>(students);

        System.out.println("ArrayList before sorting: " + studentsAL);

        // Ascending sort (default Comparable sorting by id)

        Collections.sort(studentsAL);

        System.out.println("ASC: " + studentsAL);

        // Descending sort using comparator

        studentsAL.sort(new DescendingStudentComparator());

        System.out.println("DESC: " + studentsAL);

    }

}

// Original List: [Student [id=1, name=jaya], Student [id=4, name=Bala], Student [id=2, name=JB], Student [id=5, name=Tharun], Student [id=3, name=Eva]]

// ArrayList before sorting: [Student [id=1, name=jaya], Student [id=4, name=Bala], Student [id=2, name=JB], Student [id=5, name=Tharun], Student [id=3, name=Eva]]

// ASC: [Student [id=1, name=jaya], Student [id=2, name=JB], Student [id=3, name=Eva], Student [id=4, name=Bala], Student [id=5, name=Tharun]]

// DESC: [Student [id=5, name=Tharun], Student [id=4, name=Bala], Student [id=3, name=Eva], Student [id=2, name=JB], Student [id=1, name=jaya]]

Set

1. Extends collection interface
2. Only unique things
3. Sets don’t care about position

package com.jb.T12Collections;

import java.util.HashSet;

import java.util.Set;

public class SetDemo {

    public static void main(String[] args) {

    Set <String> set=Set.of("apple","banana","cat");

    System.out.println(set); //[apple, cat, banana]

    Set<String> hashSet = new HashSet<>(set);

    System.out.println(hashSet); //[banana, apple, cat]

    }

}

Data Structure – Array, LinkedList, HashTable

1. HashList
2. Store elements in buckets by hashing function
3. Elements:
4. 13, 15, 45, 58, 6 ,34, 10, 23, 12
5. 13%13=0
6. 13 in 0th bucket
7. We using hashing function as 13
8. Insert, search, delete
9. hashCode – java – hashing table

Tree

1. Store elements in Sorted order
2. LHS<Root<RHS
3. Making sure that making element in sorted order
4. Insert, Delete, Search

package com.jb.T12Collections;

import java.util.HashSet;

import java.util.LinkedHashSet;

import java.util.Set;

import java.util.TreeSet;

public class SetDemo {

    public static void main(String[] args) {

    Set <String> set=Set.of("apple","banana","cat");

    System.out.println(set); //[apple, cat, banana] m

    Set<String> hashSet = new HashSet<>(set);

    System.out.println(hashSet); //[banana, apple, cat] m

    Set<Integer> numbers = new HashSet<>();

    numbers.add(765432);

    numbers.add(76543);

    numbers.add(7654);

    numbers.add(765);

    numbers.add(76);

    System.out.println(numbers); //[765432, 7654, 76, 765, 76543] m

    System.out.println(hashSet); //[banana, apple, cat] m

    Set<Integer> numbers1 = new LinkedHashSet<>();

    numbers1.add(765432);

    numbers1.add(76543);

    numbers1.add(7654);

    numbers1.add(765);

    numbers1.add(76);

    System.out.println(numbers1); //[765432, 76543, 7654, 765, 76] im

    Set<Integer> numbers2 = new TreeSet<>();

    numbers2.add(765432);

    numbers2.add(76543);

    numbers2.add(7654);

    numbers2.add(765);

    numbers2.add(76);

    System.out.println(numbers2); //[76, 765, 7654, 76543, 765432] sorted and no duplicates

    }

}

1. Find Unique Characters
2. treeSet – but 1st preference to Capital letters
3. package com.jb.T12Collections;

import java.util.HashSet;

import java.util.LinkedHashSet;

import java.util.List;

import java.util.Set;

import java.util.TreeSet;

public class FindUniqueChar {

    public static void main(String[] args) {

        List<Character> characters=List.of('j','A','Y','a','B','a','L','A');

        Set<Character> treeSet= new TreeSet<>(characters);

        System.out.println(treeSet); //[A, B, L, Y, a, j] //unique and sorted

        Set<Character> hashSet= new HashSet<>(characters);

        System.out.println(hashSet); //A, a, B, Y, j, L] //unique but not sorted (randomly sort)

        Set<Character> linkedHashSet= new LinkedHashSet<>(characters);

        System.out.println(linkedHashSet); //A, a, B, Y, j, L] //unique but same order (insertion order)

    }

}

1. TreeSet – NavigableSet

package com.jb.T12Collections;

import java.util.Set;

import java.util.TreeSet;

public class NavigableSet {

    public static void main(String[] args) {

        TreeSet<Integer> number= new TreeSet<>(Set.of(65,54,34,12,99));

        System.out.println(number.floor(34)); //number<=34 -- 34

        System.out.println(number.lower(34)); //number<34 -- 12

        System.out.println(number.ceiling(34)); //number>=34 -- 34

        System.out.println(number.higher(34)); //number>34 -- 54

        System.out.println(number.subSet(12, 65)); //start>= , end before<(not included) -- [12, 34, 54]

        System.out.println(number.subSet(12,true, 65,true)); //start>= , end before<=(included) -- [12, 34, 54, 65]

        System.out.println(number.headSet(54));//before that element, but that elements not included -- [12, 34]

        System.out.println(number.tailSet(54));//after that element, that element also included -- [54, 65, 99]

    }

}

Queue

1. FIFO
2. ToDo List
3. Support all collection
4. Methods – add, offer, remove, poll, peek
5. Priority Queue – Sorted Order
6. Preference to caps as always(First Letter)

package com.jb.T12Collections;

import java.util.List;

import java.util.PriorityQueue;

import java.util.Queue;

public class QueueDemo {

    public static void main(String[] args) {

        Queue<String> queue=new PriorityQueue<>();

        System.out.println(queue.poll()); //getout queue / remove --null

        queue.offer("jaya"); //add

        System.out.println(queue); //[jaya]

        queue.addAll(List.of("bala","jb","tharun"));

        System.out.println(queue); //[bala, jaya, jb, tharun] -- sorted due to priorityqueue

        System.out.println(queue.poll()); //bala

        System.out.println(queue.poll()); //jaya

        System.out.println(queue.poll()); //jb

        System.out.println(queue.poll()); //tharun

        System.out.println(queue.poll()); //null

        System.out.println(queue); //[]

    }

}

1. Custom Priority Queue

package com.jb.T12Collections;

import java.util.Comparator;

import java.util.List;

import java.util.PriorityQueue;

import java.util.Queue;

public class QueueDemo {

    class StringLengthComparator implements Comparator<String> {

        @Override

        public int compare(String value1, String value2) {

            return Integer.compare(value2.length(), value1.length());

        }

    }

    public static void main(String[] args) {

Queue<String> queue1=new PriorityQueue<>(new QueueDemo().new StringLengthComparator());

        queue1.addAll(List.of("bala","jb","tharun","Jaya","jB"));

        System.out.println(queue1); //[tharun, Jaya, bala, jb, jB] -- sorted desc based on length and caps

    }

}

Map

1. Part of collection framework, but not a collection interface
2. Key:value – pairs
3. Position of values are not fixed
4. HashMap, HashTable, LinkedHashMap, TreeMap
5. HashMap – unsorted and unordered, Key with Null value
6. HashTable - unsorted and unordered, Synchronized
7. LinkedHashMap – insertion order is maintained, slower insertion and deletion, faster iteration
8. TreeMap – Sorted order, navigable map;

package com.jb.T12Collections;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.TreeMap;

public class MapDemo {

    public static void main(String[] args) {

        Map<String, Integer> map=Map.of("J",2,"A",4,"E",2);

        System.out.println(map); //{A=4, J=2, E=2} m

        System.out.println(map.get("A")); //4

        System.out.println(map.size()); //3

        System.out.println(map.isEmpty()); //false

        System.out.println(map.containsKey("j")); //false

        System.out.println(map.containsValue(4)); //true

        System.out.println(map.keySet()); //[E, A, J] m

        System.out.println(map.values()); //[2, 2, 4] m

        Map<String, Integer> hashMap=new HashMap<>(map);

        System.out.println(hashMap.put("v", 8)); //null - return previous value

        System.out.println(hashMap); //{A=4, J=2, E=2, v=8} - sorted the values im

        Map<String, Integer> linkedHashMap=new LinkedHashMap<>(hashMap);

        System.out.println(linkedHashMap.put("v", 9)); //8 - return previous value

        System.out.println(linkedHashMap.put("A", 12)); //4 - return previous value

        System.out.println(linkedHashMap.put("B", 19)); //null - return previous value

        System.out.println(linkedHashMap); //{A=12, J=2, E=2, v=9, B=19} - insertion order im

        Map<String, Integer> treeMap=new TreeMap<>(linkedHashMap);

        System.out.println(treeMap.put("z", 18)); //null - return previous value

        System.out.println(treeMap); //{A=12, B=19, E=2, J=2, v=9, z=18} - sorted the values im

    }

}

1. Count Characters and String

package com.jb.T12Collections;

import java.util.HashMap;

import java.util.Map;

import java.util.Arrays;

public class CountChar {

    public static void main(String[] args) {

        String str = "power rangers mystic force" + "SPD Emergency ";

        // Counting character occurrences

        Map<Character, Integer> occurance = new HashMap<>();

        char[] characters = str.toCharArray();

        for (char character : characters) {

            Integer integer = occurance.get(character);

            if (integer == null) {

                occurance.put(character, 1);

            } else {

                occurance.put(character, integer + 1);

            }

        }

        System.out.println(occurance);

        // Expected output: { =5, a=1, c=3, D=1, e=5, E=1, f=1, g=2, i=1, m=2, n=2, o=2, p=1, P=1, r=5, s=2, S=1, t=1, w=1, y=2}

        // Counting word occurrences

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

        String[] words = str.split(" ");

        for (String word : words) {

            Integer integer = strOccurance.get(word);

            if (integer == null) {

                strOccurance.put(word, 1);

            } else {

                strOccurance.put(word, integer + 1);

            }

        }

        System.out.println(Arrays.toString(words)); // Displaying words properly

        System.out.println(strOccurance); // Displaying word count

//         [power, rangers, mystic, forceSPD, Emergency]

//         {rangers=1, mystic=1, forceSPD=1, power=1, Emergency=1}

    }

}

package com.jb.T12Collections;

import java.util.HashMap;

import java.util.Map;

import java.util.TreeMap;

import java.util.Arrays;

public class CountChar {

    public static void main(String[] args) {

        String str = "power rangers mystic force" + " "+"SPD Emergency ";

        // Counting character occurrences

        Map<Character, Integer> occurance = new HashMap<>();

        char[] characters = str.toCharArray();

        for (char character : characters) {

            Integer integer = occurance.get(character);

            if (integer == null) {

                occurance.put(character, 1);

            } else {

                occurance.put(character, integer + 1);

            }

        }

        System.out.println(occurance);

        // Expected output: { =5, a=1, c=3, D=1, e=5, E=1, f=1, g=2, i=1, m=2, n=2, o=2, p=1, P=1, r=5, s=2, S=1, t=1, w=1, y=2}

        // Counting word occurrences

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

        String[] words = str.split(" ");

        for (String word : words) {

            Integer integer = strOccurance.get(word);

            if (integer == null) {

                strOccurance.put(word, 1);

            } else {

                strOccurance.put(word, integer + 1);

            }

        }

        System.out.println(Arrays.toString(words)); // Displaying words properly

        System.out.println(strOccurance); // Displaying word count

//         [power, rangers, mystic, forceSPD, Emergency]

//         {rangers=1, mystic=1, forceSPD=1, power=1, Emergency=1}

//power rangers mystic force" + " "+"SPD Emergency "

TreeMap<String, Integer> treeMap = new TreeMap<>(strOccurance);

System.out.println(treeMap);

//{Emergency=1, SPD=1, force=1, mystic=1, power=1, rangers=1}

//.ceilingKey(""SPD") >=

//.higherKey(" ") >

//.lowerKey(" ") <

//.floorKey(" ") <=

//.firstEntry()

//.lastEntry()

//.subMap("c","y")

//

//

//

//

    }

}

1. List – Duplicates
2. Set – No Duplicates
3. Queue – order, only once value
4. Map – key – value pairs
5. Hash – unordered and unsorted
6. Link – Link with each, order is maintained
7. Tree – sorted, navigable (like a tree)

**Generics**

1. Any Datatype in list – generics
2. Use T – change list respect to the given type
3. We can use any Letter (caps) – T – type

package com.jb.T13Generics;

public class GenericsRunner {

    public static void main(String[] args) {

        MyCustomList<String> list=new MyCustomList<>();

        list.addElement("Element-1");

        list.addElement("Element-2");

        String text = list.get(0);

        System.out.println(text);

        MyCustomList<Integer> list2 = new MyCustomList<>();

        list2.addElement(Integer.valueOf(5));

        list2.addElement(Integer.valueOf(9));

        Integer num = list2.get(1);

        System.out.println(num);

    }

}

Snippet-3 : Generic Type Restrictions

MyCustomList.java

package ;

public class MyCustomList<T extends Number> {

ArrayList<T> list = new ArrayList<>();

public void addElement(T element) {

list.add(element);

}

public void removeElement(T element) {

list.remove(element);

}

public String toString() {

return list.toString();

}

public T get(int index) {

return list.get(index);

}

}

GenericsRunner.java

package ;

import MyCustomList;

public class GenericsRunner {

public static void main(String[] args) {

//MyCustomList<String> list = new MyCustomList<>();

MyCustomList<Long> list1 = new MyCustomList<>();

list1.addElement(5l);

list1.addElement(7l);

Long long = list1.get(0);

System.out.println(long);

MyCustomList<Integer> list2 = new MyCustomList<>();

list2.addElement(Integer.valueOf(5));

list2.addElement(Integer.valueOf(9));

Integer num = list2.get(1);

System.out.println(num);

}

}

Console Output

5

9

When we specify T extends Number as the type, we can use all the methods in the API of class Number are

available for use.

com.in28minutes.generics

com.in28minutes.generics

com.in28minutes.generics.

Generic Methods

We can create generic methods as well. Let's look at a few examples:

Snippet-4 : Generic Method

GenericsRunner.java

package ;

import MyCustomList;

public class GenericsRunner {

static <X> X doSomething(X value) {

return value;

}

static <X extends List> void duplicate(X list) {

list.add(list);

}

public static void main(String[] args) {

String text = doSomething("Hello");

Integer value = doSomething(Integer.valueOf(7));

ArrayList<String> list = doSomething(new ArrayList<String>(List.of("A", "B", C")));

duplicate(list);

System.out.println(list);

LinkedList<Integer> list2 = doSomething(new LinkedList<String>(List.of(1, 2, 3)));

duplicate(list2);

System.out.println(list2);

}

}

Console Output

[A, B, C, A, B, C]

[1, 2, 3, 1, 2, 3]

Generics And Wild-Cards

You can use wild card with generics too - ? extends Number

Snippet-5

GenericsRunner.java

package ;

import MyCustomList;

public class GenericsRunner {

static double sumOfNumberList(List<? extends Number> numbers) {

double sum = 0.0;

for(Number number:numbers) {

sum += number;

}

return sum;

}

public static void main(String[] args) {

System.out.println(sumOfNumberList(List.of(1, 2, 3, 4, 5)));

System.out.println(sumOfNumberList(List.of(1.1, 2.1, 3.1, 4.1, 5.1)));

System.out.println(sumOfNumberList(List.of(1l, 2l, 3l, 4l, 5l)));

com.in28minutes.generics

com.in28minutes.generics.

com.in28minutes.generics

com.in28minutes.generics.

}

}

Console Output

15.0

15.5

15.0

Snippet-5 Explained

The symbol ? in the definition of the method static double sumOfNumberList(List<? extends Number>

numbers) is the wild-card symbol. It denotes the fact that in order to be a valid argument to sumOfNumberList ,

numbers can be a List of any elements, so long as all of them are of type sub-classed from Number .

This includes Integer , Long , Short , Byte , Float and Double .

It also includes their primitive type counterparts, since they can be converted implicitly to their Wrapper class

counterparts.

Of course, all these elements of List numbers need to be of a homogeneous type.

Restricted Heterogeneous Lists

The generic wildcard we saw in the previous section is referred to as a Upper-Bounded Wild-Card. It can be used

to specify homogeneous types with a restriction. There is another category of wild-cards called Lower-Bounded

Wild-Card, which can be used with create Heterogeneous types of elements , within the restriction. Here is an

example.

Snippet-6 : More wild-cards

GenericsRunner.java

package ;

import MyCustomList;

public class GenericsRunner {

static void addAFewNumbers(List<? super Number> numbers) {

numbers.add(1);

numbers.add(1l);

numbers.add(1.0);

numbers.add(1.0l);

}

public static void main(String[] args) {

List<Number> numberList = new ArrayList<>();

addAFewNumbers(numberList);

System.out.println(numberList);

}

}

Console Output

[1, 1, 1.0. 1.0]

**Functional Programming**

1. Function – write code once, run anywhere , when we call function it execute that block of code

package com.jb.T14FunctionalProgramming;

import java.util.List;

public class FunctionalProRunner {

    public static void main(String[] args) {

        List<String> list = List.of("Apple","banana","cat","dog");

        //printBasic(list);

        printWithFP(list);

    }

    // private static void printBasic(List<String> list) {

    //     for(String string:list)

    //     {

    //         System.out.println(string);

    //     }

    // }

    // instead of loop, we streaming through each element

    private static void printWithFP(List<String> list) {

        list.stream().forEach(

            element -> System.out.println(element)

        );

    }

}

// Apple

// banana

// cat

// dog

package com.jb.T14FunctionalProgramming;

import java.util.List;

public class FPDemoListOfNo {

    public static void main(String[] args) {

        List<Integer> list = List.of(1,4,7,9);

        list.stream().forEach(

            element -> System.out.println(element)

        );

    }

}

// 1

// 4

// 7

// 9

1. Streaming into list with logic
2. Stream().forEach()
3. Filter -> stream().fliter().forEach()
4. Odd and Even

package com.jb.T14FunctionalProgramming;

import java.util.List;

public class EvenOddFPDemoListOfNo {

    public static void main(String[] args) {

        List<Integer> list = List.of(1,4,7,9);

        list.stream().forEach(

            element -> System.out.println(element)

        );

        filterEven(list);

        filterOdd(list);

    }

    private static void filterOdd(List<Integer>list)

    {

        list.stream()

        .filter(number -> number%2==1)

        .forEach(number -> System.out.println("Odd:"+number)

        );

    }

    private static void filterEven(List<Integer>list)

    {

        list.stream()

        .filter(number -> number%2==0)

        .forEach(number -> System.out.println("Even:"+number)

        );

    }

}

// 1

// 4

// 7

// 9

// Even:4

// Odd:1

// Odd:7

// Odd:9

End with

package com.jb.T14FunctionalProgramming;

import java.util.List;

public class FunctionalProRunner {

    public static void main(String[] args) {

        List<String> list = List.of("ordered","bat","cat","red");

        //printBasic(list);

        printWithFP(list);

        printWithFiltering(list);

        printWithFilteringWithFP(list);

    }

    // private static void printBasic(List<String> list) {

    //     for(String string:list)

    //     {

    //         System.out.println(string);

    //     }

    // }

    // instead of loop, we streaming through each element

    private static void printWithFP(List<String> list) {

        list.stream().forEach(

            element -> System.out.println(element)

        );

    }

    private static void printWithFiltering(List<String> list) {

        for(String string: list)

        {

            if(string.endsWith("at"))

            {

                System.out.println("elemernt end with at:"+string);

            }

        }

    }

    private static void printWithFilteringWithFP(List<String> list) {

        list.stream()

        .filter(elemnt -> elemnt.endsWith("red"))

        .forEach(element -> System.out.println("element end with red:" +element)

        );

    }

}

// ordered

// bat

// cat

// red

// elemernt end with at:bat

// elemernt end with at:cat

// element end with red:ordered

// element end with red:red

Sum of List of Numbers

1. 1,2 3,4,5
2. Reduce the 5 numbers into 1 number
3. .reduce(0,(num1,num2) -> num1+num2); - intial ,n1 and n2 -> stream
4. Use brackets – when we have 2 parameters in method
5. 1,2,3,4,5 -> 3,3,4,5 -> 6,4,5,->10,5->15

package com.jb.T14FunctionalProgramming;

import java.util.List;

public class SumOfListOfNumbers {

    public static void main(String[] args) {

        List<Integer> numbers = List.of(1,4,7,9,5,8);

        numbers.stream().forEach(element -> System.out.println(element));

        getSumNormal(numbers);

        getSumFP(numbers);

    }

    private static void getSumFP(List<Integer>list)

    {

       int sum = list.stream()

        .reduce(0,(number1, number2) -> number1+number2);

        System.out.println("FP sum: "+sum);

    }

    private static void getSumNormal(List<Integer> numbers) {

        int sum=0;

        for(int number:numbers)

        {

            sum=sum+number;

        }

        System.out.println("Total: "+sum);

    }

}

// 1

// 4

// 7

// 9

// 5

// 8

// Total: 34

// FP sum: 34

Value changing - mutation

Sum Of Even and Odd numbers

1. Function programming – avoid mutation

package com.jb.T14FunctionalProgramming;

import java.util.List;

public class SumOfOddandEven {

    public static void main(String[] args) {

        List<Integer> numbers = List.of(1,4,7,9,5,8);

        numbers.stream().forEach(element -> System.out.println(element));

        getOddSumFP(numbers);

        getEvenSumFP(numbers);

    }

    private static void getOddSumFP(List<Integer>list)

    {

       int sum = list.stream()

       .filter(number -> number%2==1)

       .reduce(0,(number1, number2) -> number1+number2);

        System.out.println("FP Odd sum: "+sum);

    }

    private static void getEvenSumFP(List<Integer>list)

    {

       int sum = list.stream()

       .filter(number -> number%2==0)

       .reduce(0,(number1, number2) -> number1+number2);

        System.out.println("FP even sum: "+sum);

    }

}

// 1

// 4

// 7

// 9

// 5

// 8

// FP Odd sum: 22

// FP even sum: 12

Anonymous Function in Java

In Java, anonymous functions, also known as lambda expressions. It introduced in Java 8 as a way to provide more concise and readable code. They allow us to define a function in a single line of code without having to explicitly define a class or interface.

What is an Anonymous Function?

An anonymous function is a function that has no name and is not bound to an identifier. It is typically defined inline and can be passed as an argument to another function or returned as a value from a function. In Java, anonymous functions are implemented as lambda expressions, which are a shorthand way of defining an anonymous function.

Syntax

The syntax for an anonymous function in Java is as follows:

(parameters) -> { body }

1. Stream – source of object
2. Intermediate operation – any operation, result have another stream; filter, sort
3. Terminal operation – printing or result; min max

Operation – sort, distinct, map

package com.jb.T14FunctionalProgramming;

import java.util.List;

public class FPOperation {

     public static void main(String[] args) {

        List<Integer> numbers = List.of(3,55,7,8,24,31,4,47,9,54,18);

        numbers.stream().forEach(element -> System.out.println(element));

        numbers.stream().sorted().forEach(sortE -> System.out.println("sorted :"+sortE));

        numbers.stream().distinct().forEach(disE -> System.out.println("Distinct :"+disE));

        numbers.stream().distinct().sorted().forEach(disSortE -> System.out.println("Distinct sorted:"+disSortE));

        numbers.stream().distinct().sorted().forEach(disSortE -> System.out.println("Distinct sorted:"+disSortE));

        numbers.stream().distinct().sorted().map(sqE -> sqE\*sqE).forEach(sqE-> System.out.println(" Square Distinct sorted:"+sqE));

     }

}

// 3

// 55

// 7

// 8

// 24

// 31

// 4

// 47

// 9

// 54

// 18

// sorted :3

// sorted :4

// sorted :7

// sorted :8

// sorted :9

// sorted :18

// sorted :24

// sorted :31

// sorted :47

// sorted :54

// sorted :55

// Distinct :3

// Distinct :55

// Distinct :7

// Distinct :8

// Distinct :24

// Distinct :31

// Distinct :4

// Distinct :47

// Distinct :9

// Distinct :54

// Distinct :18

// Distinct sorted:3

// Distinct sorted:4

// Distinct sorted:7

// Distinct sorted:8

// Distinct sorted:9

// Distinct sorted:18

// Distinct sorted:24

// Distinct sorted:31

// Distinct sorted:47

// Distinct sorted:54

// Distinct sorted:55

// Distinct sorted:3

// Distinct sorted:4

// Distinct sorted:7

// Distinct sorted:8

// Distinct sorted:9

// Distinct sorted:18

// Distinct sorted:24

// Distinct sorted:31

// Distinct sorted:47

// Distinct sorted:54

// Distinct sorted:55

//  Square Distinct sorted:9

//  Square Distinct sorted:16

//  Square Distinct sorted:49

//  Square Distinct sorted:64

//  Square Distinct sorted:81

//  Square Distinct sorted:324

//  Square Distinct sorted:576

//  Square Distinct sorted:961

//  Square Distinct sorted:2209

//  Square Distinct sorted:2916

//  Square Distinct sorted:3025

Intermediate operation

package com.jb.T14FunctionalProgramming;

import java.util.List;

import java.util.stream.IntStream;

public class CubeLowerUpperOperation {

    public static void main(String[] args) {

        IntStream.range(1,11) //1 to 10

        .map(e->e\*e\*e).forEach(p->System.out.println(p)); //cude

        List<String> list = List.of("ordered","bat","cat","red");

        list.stream().map(s ->s.toUpperCase()).forEach(s -> System.out.println(s));

        list.stream().map(s ->s.length()).forEach(s -> System.out.println(s));

    }

}

// 1

// 8

// 27

// 64

// 125

// 216

// 343

// 512

// 729

// 1000

// ORDERED

// BAT

// CAT

// RED

// 7

// 3

// 3

// 3

IntStream – stream of values from range

Terminal Operation max, get

package com.jb.T14FunctionalProgramming;

import java.util.List;

public class MaxMin {

       public static void main(String[] args) {

       List <Integer> var1 = List.of(1, 4, 7, 9, 5, 8);

       int max=var1.stream().max((n1,n2) -> Integer.compare(n1, n2)).get();

       int min=var1.stream().min((n1,n2) -> Integer.compare(n1, n2)).get();

       System.out.println("Max: "+max);

       System.out.println("Min: "+min);

    }

}

// Max: 9

// Min: 1

package com.jb.T14FunctionalProgramming;

import java.util.List;

import java.util.stream.Collectors;

import java.util.stream.IntStream;

public class MaxMin {

       public static void main(String[] args) {

       List <Integer> var1 = List.of(1, 4, 7, 9, 5, 8);

       int max=var1.stream().max((n1,n2) -> Integer.compare(n1, n2)).get();

       int min=var1.stream().min((n1,n2) -> Integer.compare(n1, n2)).get();

       System.out.println("Max: "+max);

       System.out.println("Min: "+min);

       List<Integer> sq=IntStream.range(1,11).map(e -> e\*e).boxed().collect(Collectors.toList());

       System.out.println(sq); //[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

    }

}

// Max: 9

// Min: 1

.orElse(0) – present – return; or else zero

Classroom Exercise FP-CE-03

. Using method references, write java functional code to determine the maximum even number, in a given list of

integers.

Solution To FP-CE-03

MethodReferencesRunner.java

package ;

import List;

public class MethodReferencesRunner {

public static void print(Integer number) {

System.out.println(number);

}

public static void main(String[] args) {

List.of("Ant", "Bat", "Cat", "Dog", "Elephant").stream()

.map(s -> s.length())

.forEach(l -> MethodReferencesRunner.print

List.of("Ant", "Bat", "Cat", "Dog", "Elephant").stream()

.map(s -> s.length())

.forEach(MethodReferencesRunner::print);

}

}

package ;

import List;

public class MethodReferencesRunner {

public static void print(Integer number) {

System.out.println(number);

}

public static void main(String[] args) {

List.of("Ant", "Bat", "Cat", "Dog", "Elephant").stream()

.map(s -> s.length())

.forEach(MethodReferencesRunner::print);

List.of("Ant", "Bat", "Cat", "Dog", "Elephant").stream()

.map(String::length)

.forEach(MethodReferencesRunner::print);

}

}

com.in28minutes.functionalprogramming

java.util.

com.in28minutes.functionalprogramming

java.util.

package ;

import List;

public class MethodReferencesRunner {

public static void print(Integer number) {

System.out.println(number);

}

public static void main(String[] args) {

int max = List.of(23, 45, 67, 34).stream()

.filter(num -> num % 2 == 0)

.max( (n1, n2) -> Integer.compare(n1, n2) )

.orElse(0);

System.out.println(max);

int maximum = List.of(23, 45, 67, 34).stream()

.filter(MethodReferencesRunner::isEven)

.max(Integer::compare)

.orElse(0);

System.out.println(maximum);

}

public static booelan isEven(Integer number) {

return (number %2 == 0);

}

}

Console Output

34

34

Summary

In this step, we:

Understood what is a method reference

Learned that both built-in, and user defined class methods can be invoked using method references

Observed that method references work for static and non-static methods

Step 17: FP - Functions As First-Class Citizens

Are functions first class citizens in Java?

Here are few questions to think about?

Can you pass a function as an argument to a method?

Can you assign a function to a variable?

Can you obtain a function as a return value, from a method invocation?

Passing function as method argument

We looked at several examples of this earlier.

In the example below, num -> num % 2 == 0 is passed to filter method.

int max = List.of(23, 45, 67, 34).stream()

.filter(num -> num % 2 == 0)

com.in28minutes.functionalprogramming

java.util.

.max( (n1, n2) -> Integer.compare(n1, n2) )

.orElse(0);

Storing functions in reference variables

evenPredicate and oddPredicate represent functions.

package ;

import List;

import Predicate;

public class FPNumberRunner {

public static void main(String[] args) {

List<Integer> numbers = List.of(23, 12, 34, 45, 36, 48);

Predicate<? super Integer> evenPredicate = num -> num % 2 == 0;

Predicate<? super Integer> oddPredicate = num -> num % 2 == 1;

numbers.stream()

.filter(evenPredicate)

.map(n -> n\*n)

.forEach(e -> System.out.println(e));

}

}

Console Output

1156

1296

2304

Returning functions from methods

createEvenPredicate and createOddPredicate are examples of methods returning functions.

import Stream;

import Predicate;

public class FPNumberRunner {

public static Predicate<? super Integer> createEvenPredicate() {

return num -> num%2 == 0;

}

public static Predicate<? super Integer> createOddPredicate() {

return num -> num%2 == 1;

}

public static void main(String[] args) {

List<Integer> numbers = List.of(23, 12, 34, 45, 36, 48);

//Predicate<? super Integer> evenPredicate = num -> num % 2 == 0;

//Predicate<? super Integer> evenPredicate = createEvenPredicate();

numbers.stream()

//.filter(num -> num%2 == 0)

//.filter(evenPredicate)

.map(n -> n\*n)

.forEach(e -> System.out.println(e));

}

}

com.in28minutes.functionalprogramming

java.util.

java.util.function.

java.util.stream.

java.util.function.

Console Output

1156

1296

2304

Summary

In this step, we observed that the following is true for a function:

It can be passed as a method argument

It can be stored in a reference variable

It can be returned from a method

**Threads**

1. Run tasks in parallel
2. Parallelism – utilizing CPU
3. 2 ways
4. Extending class thread
5. Implementing interface runnable

package com.jb.T15ThreadsAndConcurrency;

public class ThreadDemo {

    public static void main(String[] args) {

        // Task1

        for (int i = 101; i <= 199; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\n Task1 Done");

        // Task2

        for (int i = 201; i <= 299; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\n Task2 Done");

        // Task3

        for (int i = 301; i <= 399; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\n Task3 Done");

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

    }

}

// 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199

//  Task1 Done

// 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299

//  Task2 Done

// 0 331 332 333 334 335 336 337 338 339 340 341 342 343 344 30 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399

//  Task3 Done

// Main Done

1. By Extending class Thread
2. Signature – run

package com.jb.T15ThreadsAndConcurrency;

class Task1 extends Thread {

    public void run() {

        System.out.println("Task1 Started ");

        for (int i = 101; i <= 199; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask1 Done");

    }

}

public class ThreadDemo {

    public static void main(String[] args) {

        System.out.println("T1 KO");

        // Task1

        Task1 task1 = new Task1();

        //task1.run(); //typical method execution

        task1.start(); // to run as thread

        System.out.println("\n Task1 Done");

        System.out.println("T2 KO");

        // Task2

        for (int i = 201; i <= 299; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\n Task2 Done");

        System.out.println("T3 KO");

        // Task3

        for (int i = 301; i <= 399; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\n Task3 Done");

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

    }

}

// T1 KO

//  Task1 Done

// T2 KO

// Task1 Started

// 201 202 203 204 205 206 207 208 101 102 103 104 105 106 107 209 210 211 212 108 109 213 110 214 111 112 113 114 115 116 215 216 217 117 118 119 120 218 219 121 122 220 123 221 124 222 223 224 125 225 126 226 227 127 128 129 130 131 228 132 133 134 229 135 230 136 137 138 139 140 141 231 142 143 232 144 145 233 146 234 235 236 237 238 239 240 241 242 243 244 245 246 147 148 149 150 151 247 152 248 249 250 251 252 253 254 255 153 154 155 156 157 158 256 257 159 258 259 260 261 160 161 162 163 164 165 262 263 264 166 167 168 265 169 170 266 267 268 269 270 271 272 273 171 172 173 174 175 176 274 275 177 276 277 278 279 280 281 178 179 180 181 282 283 182 183 184 185 186 187 188 284 285 286 189 287 288 289 190 191 192 193 194 290 195 196 291 197 198 199 292 293 294 295

// Task1 Done

// 296 297 298 299

//  Task2 Done

// T3 KO

// 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399

//  Task3 Done

// Main Done

Task 1 Code Running in Parallel with task 2 and 3

1. Implementing interface runnable
2. Implements, object, thread class

package com.jb.T15ThreadsAndConcurrency;

class Task1 extends Thread {

    public void run() {

        System.out.println("Task1 Started ");

        for (int i = 101; i <= 199; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask1 Done");

    }

}

class Task2 implements Runnable {

    @Override

    public void run() {

        System.out.println("Task2 Started ");

        for (int i = 201; i <= 299; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask2 Done");

    }

}

public class ThreadDemo {

    public static void main(String[] args) {

        System.out.println("T1 KO");

        // Task1

        Task1 task1 = new Task1();

        // task1.run(); //typical method execution

        task1.start(); // to run as thread

        System.out.println("\n Task1 Done");

        System.out.println("T2 KO");

        // Task2

        Task2 task2 = new Task2();

        Thread task2Thread = new Thread(task2);

        task2Thread.start();

        System.out.println("T3 KO");

        // Task3

        for (int i = 301; i <= 399; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\n Task3 Done");

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

    }

}

// T2 KO

// Task1 Started

// T3 KO

// Task2 Started

// 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 101 102 301 228 229 230 231 103 104 105 106 302 303 304 232 107 305 233 234 108 306 307 308 235 109 309 236 237 238 239 240 241 242 243 110 111 112 113 114 115 116 117 310 244 245 246 247 118 311 312 313 248 119 120 121 122 314 315 316 317 318 319 249 250 251 252 123 124 125 320 321 322 323 253 254 255 126 127 128 129 130 131 132 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 256 257 258 259 133 134 135 339 340 341 342 343 344 345 346 260 261 262 263 264 265 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 347 348 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 162 163 164 165 166 349 350 351 352 353 354 355 281 167 168 169 356 357 358 282 283 284 285 286 287 288 289 290 291 292 293 294 295 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 296 297 298 185 375 376 377 378 299 186 187 188 189 190 191 192 193 379 380 381 382

// Task2 Done

// 194 195 383 384 385 386 387 388 389 390 196 197 198 199 391 392 393 394 395 396 397 398 399

// Task1 Done

//  Task3 Done

// Main Done

Task 1 and 2 running parallel with task 3

1. Different states of threads – Thread Life Cycle
2. New
3. Runnable
4. Running
5. Blocked / Waiting
6. terminated
7. New – thread is ready, but not started
8. Runnable – Ready to run, waiting to run
9. Running – Thread Running
10. Blocked / Waiting – waiting for any process
11. Terminated / Dead – after completion, thread is terminated
12. Priority
13. setPriority() – request
14. only request, not order
15. min =1; norm=5; max=10

package com.jb.T15ThreadsAndConcurrency;

class Task1 extends Thread {

    public void run() {

        System.out.println("Task1 Started ");

        for (int i = 101; i <= 199; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask1 Done");

    }

}

class Task2 implements Runnable {

    @Override

    public void run() {

        System.out.println("Task2 Started ");

        for (int i = 201; i <= 299; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask2 Done");

    }

}

public class ThreadDemo {

    public static void main(String[] args) {

        System.out.println("T1 KO");

        // Task1

        Task1 task1 = new Task1();

        // task1.run(); //typical method execution

        task1.setPriority(1);

        task1.start(); // to run as thread

        System.out.println("\n Task1 Done");

        System.out.println("T2 KO");

        // Task2

        Task2 task2 = new Task2();

        Thread task2Thread = new Thread(task2);

        task2Thread.setPriority(10);

        task2Thread.start();

        System.out.println("T3 KO");

        // Task3

        for (int i = 301; i <= 399; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\n Task3 Done");

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

    }

}

// T1 KO

//  Task1 Done

// T2 KO

// Task1 Started

// T3 KO

// Task2 Started

// 101 102 103 104 301 201 202 203 105 106 107 302 204 205 206 207 208 209 210 211 108 109 110 111 112 303 304 305 306 307 308 309 310 311 312 212 113 114 313 314 315 316 213 214 215 115 317 318 319 320 216 217 116 321 218 219 220 117 118 119 120 322 221 222 223 121 122 123 323 324 325 224 225 226 124 125 326 327 328 329 330 331 332 333 334 335 336 227 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 337 338 228 229 230 231 232 143 144 145 339 340 341 342 343 344 233 234 235 236 146 345 346 347 348 237 147 148 149 349 350 351 352 353 354 238 239 150 151 152 153 154 155 355 356 357 240 241 242 243 244 245 156 157 158 159 160 161 162 163 358 359 246 247 248 249 250 251 164 360 252 253 254 255 256 165 166 167 168 361 257 258 259 260 169 170 171 172 362 363 364 365 261 262 263 264 265 173 174 366 266 267 268 269 270 175 176 177 178 179 180 181 182 183 184 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 271 272 273 185 186 187 188 189 190 191 192 193 194 195 196 197 384 385 386 387 274 275 276 277 198 199 388 278 279 280 281 282 283

// Task1 Done

// 389 390 284 391 285 286 287 288 289 290 392 291 292 293 294 295 296 297 393 298 299 394

// Task2 Done

// 395 396 397 398 399

//  Task3 Done

1. Communication between threads
2. Task.join() – to interrupt that task to complete and other have to wait till complete
3. Throws exception – Interrupted Exception

package com.jb.T15ThreadsAndConcurrency;

class Task1 extends Thread {

    public void run() {

        System.out.println("Task1 Started ");

        for (int i = 101; i <= 199; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask1 Done");

    }

}

class Task2 implements Runnable {

    @Override

    public void run() {

        System.out.println("Task2 Started ");

        for (int i = 201; i <= 299; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask2 Done");

    }

}

public class ThreadDemo {

    public static void main(String[] args) throws InterruptedException {

        System.out.println("T1 KO");

        // Task1

        Task1 task1 = new Task1();

        // task1.run(); //typical method execution

        task1.setPriority(1);

        task1.start(); // to run as thread

        System.out.println("\n Task1 Done");

        System.out.println("T2 KO");

        // Task2

        Task2 task2 = new Task2();

        Thread task2Thread = new Thread(task2);

        task2Thread.setPriority(10);

        task2Thread.start();

        task1.join();

        task2Thread.join();

        System.out.println("T3 KO");

        // Task3

        for (int i = 301; i <= 399; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\n Task3 Done");

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

    }

}

// T1 KO

// Task1 Done

// T2 KO

// Task1 Started

// Task2 Started

// 201 202 203 204 205 206 101 207 208 209 102 103 104 105 106 210 211 212 213 214 107 215 108 109 110 111 112 113 114 115 216 217 218 219 116 117 220 221 222 223 224 225 226 227 118 119 228 120 229 230 231 232 233 121 122 123 124 125 234 126 127 128 235 236 237 129 130 131 132 238 133 134 239 135 136 137 138 240 241 242 243 244 245 246 247 248 249 139 140 141 250 142 143 251 144 145 146 147 148 149 150 151 152 153 154 155 252 253 254 156 157 158 255 256 159 257 258 160 161 162 163 259 164 260 261 262 165 166 167 168 169 170 263 264 265 266 171 172 173 174 175 176 177 178 179 180 181 182 267 268 269 270 271 272 273 274 275 276 277 183 278 279 280 281 282 283 284 285 286 184 185 287 288 289 186 187 188 189 190 290 291 191 292 293 294 295 192 193 194 195 296 297 298 196 299

// Task2 Done

// 197 198 199

// Task1 Done

// T3 KO

// 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399

// Task3 Done

// Main Done

1. Threadkeywords / method
2. Thread.yield() – after execution, TaskThread have enough CPU utilization , so other task may use
3. Thread.sleep(10000) – wait for 10 sec
4. Synchronized - 1 thread execute the 100 synchronized lines of code, others have to wait
5. Performance of the system might get affected – without using other threads – thread safety
6. Executor Service
7. Printing a result back using thread is not possible, so executer service somes to the picture

package com.jb.T15ThreadsAndConcurrency;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class Task1 extends Thread {

    public void run() {

        System.out.println("Task1 Started ");

        for (int i = 101; i <= 199; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask1 Done");

    }

}

class Task2 implements Runnable {

    @Override

    public void run() {

        System.out.println("Task2 Started ");

        for (int i = 201; i <= 299; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask2 Done");

    }

}

public class ExecutorServiceRunner {

    public static void main(String[] args) {

        ExecutorService executorService = Executors.newSingleThreadExecutor();

        executorService.execute(new Task1());

        executorService.execute(new Thread(new Task2()));

    }

}

// Task1 Started

// 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199

// Task1 Done

// Task2 Started

// 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299

// Task2 Done

1. Task 3 in main

package com.jb.T15ThreadsAndConcurrency;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class Task1 extends Thread {

    public void run() {

        System.out.println("Task1 Started ");

        for (int i = 101; i <= 199; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask1 Done");

    }

}

class Task2 implements Runnable {

    @Override

    public void run() {

        System.out.println("Task2 Started ");

        for (int i = 201; i <= 299; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask2 Done");

    }

}

public class ExecutorServiceRunner {

    public static void main(String[] args) {

        ExecutorService executorService = Executors.newSingleThreadExecutor();

        executorService.execute(new Task1());

        executorService.execute(new Thread(new Task2()));

        System.out.print("\nTask3 Kicked Off\n");

        for (int i = 301; i <= 399; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask3 Done");

        System.out.println("\nMain Done");

        executorService.shutdown();

    }

}

// Task1 Started

// Task3 Kicked Off

// 101 102 103 301 302 104 303 304 305 306 307 308 309 310 105 106 107 108 109 110 111 311 112 312 113 313 314 315 114 316 115 116 117 317 318 319 320 321 322 118 119 323 120 324 121 325 326 122 327 328 123 124 329 125 330 126 127 128 129 130 331 332 131 333 334 335 336 337 132 338 339 340 133 134 135 341 342 136 137 138 343 344 345 346 347 348 139 349 350 351 352 140 141 353 354 355 356 142 143 144 145 146 147 148 149 150 357 151 152 153 154 155 156 358 359 157 360 361 362 158 363 364 365 366 367 368 159 160 161 162 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 163 164 165 166 167 168 169 170 389 390 391 171 172 173 392 393 174 175 176 177 178 179 180 394 395 181 182 183 184 396 185 186 187 188 189 190 397 398 399 191

// Task3 Done

// Main Done

// 192 193 194 195 196 197 198 199

// Task1 Done

// Task2 Started

// 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299

// Task2 Done

1. Executor service execute one thread time
2. Also, we can run 2 thread task concurrently
3. 100 -T1
4. 200 – T2
5. So, use constructor
6. Use newFixedThreadPool

package com.jb.T15ThreadsAndConcurrency;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class Task extends Thread {

    private int number;

    public Task(int number) {

        this.number = number;

    }

    public void run() {

        System.out.println("Task " + number + " Started");

        for (int i = number \* 100; i <= number \* 100 + 99; i++) {

            System.out.print(i + " ");

        }

        System.out.println("\nTask " + number + " Done");

    }

}

public class ExecutorServiceRunner1 {

    public static void main(String[] args) {

        ExecutorService executorService = Executors.newFixedThreadPool(2);

        executorService.execute(new Task(1));

        executorService.execute(new Task(2));

        executorService.execute(new Task(3));

        executorService.shutdown();

    }

}

// Task 2 Started

// Task 1 Started

// 100 101 102 103 104 105 106 200 201 202 203 204 205 206 207 208 209 210 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 211 212 124 213 214 125 215 126 127 128 129 130 216 131 217 132 218 133 134 135 136 219 137 220 221 138 222 139 140 141 142 143 223 144 145 146 147 224 148 225 226 227 228 229 149 150 151 152 153 230 154 231 155 156 157 158 232 233 234 159 235 236 237 238 239 240 241 242 243 244 245 246 247 160 161 162 163 164 165 166 248 249 250 251 252 253 254 167 168 255 169 256 170 171 172 173 174 175 176 257 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 258 259 260 261 262 263 264 265 266 192 267 268 269 270 271 272 273 193 274 275 276 277 194 278 279 280 281 282 283 284 195 285 196 197 286 198 199 287 288 289 290 291 292 293 294 295

// Task 1 Done

// 296 297 298 299

// Task 2 Done

// Task 3 Started

// 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399

// Task 3 Done

1. Returning values from task, also many callable task

package com.jb.T15ThreadsAndConcurrency;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutionException;

class CallableTask implements Callable<String> {

    private String name;

    public CallableTask(String name) {

        this.name = name;

    }

    @Override

    public String call() throws Exception {

        Thread.sleep(1000);

        return "Hello " + name;

    }

}

public class CallableRunner {

    public static void main(String[] args) throws InterruptedException, ExecutionException {

        ExecutorService executorService = Executors.newFixedThreadPool(1);

        Future<String> welcomeFuture = executorService.submit(new CallableTask("Study with JB"));

        System.out.println("Callable Task Study with JB  Submitted");

        String welcomeMessage = welcomeFuture.get();

        System.out.println(welcomeMessage);

        executorService.shutdown();

    }

}

package com.jb.T15ThreadsAndConcurrency;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

import java.util.List;

import java.util.concurrent.ExecutionException;

public class MultipleCallableRunner {

    public static void main(String[] args) throws InterruptedException, ExecutionException {

        ExecutorService executorService = Executors.newFixedThreadPool(1);

        List<CallableTask> tasks = List.of(new CallableTask("Study With JB"),

                new CallableTask("Jaya"),

                new CallableTask("Bala"));

        List<Future<String>> welcomeAll = executorService.invokeAll(tasks);

        for (Future<String> welcomeFuture : welcomeAll) {

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

        }

        executorService.shutdown();

    }

}

// Hello Study With JB

// Hello Jaya

// Hello Bala

**Exception Handling**

1. **Exception** handling in Java is an effective mechanism for managing runtime errors to ensure the application's regular flow is maintained.
2. **Compile time error and Run time error**

Main -> Met 1 -> med 2 (have exception) -> when exception is met, then rest of the code is not printed/ executed

package com.jb.T16ExceptionHandling;

public class ExceptionHandlingDemo {

    public static void main(String[] args) {

        method1(); //Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "str" is null

    System.out.println("main ended");

    }

    static void method1() {

        method2();//exception

        System.out.println("med1 ended");

    }

    private static void method2() {

        String str = null;

        int len = str.length();

        System.out.println(len);

        System.out.println("med2 ended");

    }

}

1. Try – Catch to Handle Exception
2. Program won’t fail when it meet exception
3. Continue next line of code

package com.jb.T16ExceptionHandling;

public class ExceptionHandlingDemo {

    public static void main(String[] args) {

        method1(); // Exception in thread "main" java.lang.NullPointerException: Cannot invoke

                   // "String.length()" because "str" is null

        System.out.println("main ended");

    }

    static void method1() {

        method2();// exception

        System.out.println("med1 ended");

    }

    // try - catch

    private static void method2() {

        try {

            String str = null;

            int len = str.length();

            System.out.println(len);

            System.out.println("med2 ended");

        } catch (Exception e) {

            //System.out.println(e);

            e.printStackTrace();

        }

    }

}

// java.lang.NullPointerException: Cannot invoke "String.length()" because "str" is null

//         at javaTraining/com.jb.T16ExceptionHandling.ExceptionHandlingDemo.method2(ExceptionHandlingDemo.java:19)

//         at javaTraining/com.jb.T16ExceptionHandling.ExceptionHandlingDemo.method1(ExceptionHandlingDemo.java:11)

//         at javaTraining/com.jb.T16ExceptionHandling.ExceptionHandlingDemo.main(ExceptionHandlingDemo.java:5)

// med1 ended

// main ended

// java.lang.NullPointerException: Cannot invoke "String.length()" because "str"

// is null

// med1 ended

// main ended

1. Exception Hierarchy
2. Based on the exception class
3. We can multiple catch block with that particular exception
4. Specific exception would match that exception

package com.jb.T16ExceptionHandling;

public class ExceptionHandlingDemo {

    public static void main(String[] args) {

        method1(); // Exception in thread "main" java.lang.NullPointerException: Cannot invoke

                   // "String.length()" because "str" is null

        System.out.println("main ended");

    }

    static void method1() {

        method2();// exception

        System.out.println("med1 ended");

    }

    // try - catch

    private static void method2() {

        try {

            int[] i ={1,2};

            System.out.println(i[2]);

            String str = null;

            int len = str.length();

            System.out.println(len);

            System.out.println("med2 ended");

        }

        catch(ArrayIndexOutOfBoundsException e)

        {

            System.out.println(e);

        }

        catch (Exception e) {

            //System.out.println(e);

            e.printStackTrace();

        }

    }

}

// java.lang.NullPointerException: Cannot invoke "String.length()" because "str" is null

//         at javaTraining/com.jb.T16ExceptionHandling.ExceptionHandlingDemo.method2(ExceptionHandlingDemo.java:19)

//         at javaTraining/com.jb.T16ExceptionHandling.ExceptionHandlingDemo.method1(ExceptionHandlingDemo.java:11)

//         at javaTraining/com.jb.T16ExceptionHandling.ExceptionHandlingDemo.main(ExceptionHandlingDemo.java:5)

// med1 ended

// main ended

// java.lang.NullPointerException: Cannot invoke "String.length()" because "str"

// is null

// med1 ended

// main ended

1. Finally
2. Finally block always executed, even though there is no exception
3. System.exit(1) – crash the application
4. Finally, not executed if it comes under if condition
5. Try – finally -> code executes

package com.jb.T16ExceptionHandling;

import java.util.Scanner;

public class FinalluDemo {

    public static void main(String[] args) {

        Scanner scanner=new Scanner(System.in);

        try{

        int[] numbers={12,3,4,5};

        int number=numbers[2];

        System.out.println(number);

        //scanner.close(); //if not close - cause leakage

        }

        catch(Exception e)

        {

            System.out.println(e);

            System.out.println("Excetion Occurs");

        }

        finally

        {

            System.out.println("Before scanner close");

            scanner.close();

        }

        System.out.println("after scanner close");

    }

}

// 4

// Before scanner close

// after scanner close

1. Checked Exception and Unchecked Exception
2. Instead of handling the exception, we can throws the exception
3. Error & Exception -> throwable
4. Error – not handle
5. Exceptions – can handle
6. Runtime and sub classes of runtime Exception
7. Under exception which is not a sub classes run time exception – unchecked exception
8. Under exception and not under runtime exception – checked exception
9. Runtime exception - unchecked
10. F
11. F
12. F
13. F
14. Ff
15. F
16. F
17. F
18. F
19. F
20. F
21. F
22. Ff
23. F
24. f
25. f
26. r