**public** **class** Simple{

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

**int** a=10;

**float** f=a;

System.out.println(a);

System.out.println(f);

}}

**class** Simple{

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

**byte** a=10;

**byte** b=10;

//byte c=a+b;//Compile Time Error: because a+b=20 will be int

**byte** c=(**byte**)(a+b);

System.out.println(c);

}}

**public** **class** IfExample {

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

    //defining an 'age' variable

**int** age=20;

    //checking the age

**if**(age>18){

        System.out.print("Age is greater than 18");

    }

}

}

//A Java Program to demonstrate the use of if-else statement.

//It is a program of odd and even number.

**public** **class** IfElseExample {

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

    //defining a variable

**int** number=13;

    //Check if the number is divisible by 2 or not

**if**(number%2==0){

        System.out.println("even number");

    }**else**{

        System.out.println("odd number");

    }

}

}

**24: Sept 2021 : Switch, for, while, do – while, increment operators, arrays**

class Main {

public static void main(String[] args) {

// create an array

int[] numbers = {3, 9, 5, -5};

// for each loop

for (int number: numbers) {

System.out.println(number);

}

}

}

// Java program to find the sum of positive numbers

import java.util.Scanner;

class Main {

public static void main(String[] args) {

int sum = 0;

// create an object of Scanner class

Scanner input = new Scanner(System.in);

// take integer input from the user

System.out.println("Enter a number");

int number = input.nextInt();

// while loop continues

// until entered number is positive

while (number >= 0) {

// add only positive numbers

sum += number;

System.out.println("Enter a number");

number = input.nextInt();

}

System.out.println("Sum = " + sum);

input.close();

}

}

class Main {

public static void main(String[] args) {

int sum = 0;

int number = 0;

// create an object of Scanner class

Scanner input = new Scanner(System.in);

// do...while loop continues

// until entered number is positive

do {

// add only positive numbers

sum += number;

System.out.println("Enter a number");

number = input.nextInt();

} while(number >= 0);

System.out.println("Sum = " + sum);

input.close();

}

}

class LabeledBreak {

public static void main(String[] args) {

// the for loop is labeled as first

first:

for( int i = 1; i < 5; i++) {

// the for loop is labeled as second

second:

for(int j = 1; j < 3; j ++ ) {

System.out.println("i = " + i + "; j = " +j);

// the break statement breaks the first for loop

if ( i == 2)

break first;

}

}

}

}

class Main {

public static void main(String[] args) {

// for loop

for (int i = 1; i <= 10; ++i) {

// if value of i is between 4 and 9

// continue is executed

if (i > 4 && i < 9) {

continue;

}

System.out.println(i);

}

}

}

class Main {

public static void main(String[] args) {

// outer loop is labeled as first

first:

for (int i = 1; i < 6; ++i) {

// inner loop

for (int j = 1; j < 5; ++j) {

if (i == 3 || j == 2)

// skips the current iteration of outer loop

continue first;

System.out.println("i = " + i + "; j = " + j);

}

}

}

}

**24-Sept-2021: Arrays, Increment, decrement, Constructors**

// Short form

// (can only be used as initializer in a declaration)

int[] arr1 = { 1, 2, 77 };

// Initialize with 10 zeroes

int[] arr2 = new int[10];

// General form

int[] arr3 = new int[] { 1, 2, 77 };

class Main {

public static void main(String[] args) {

// create an array

int[] age = {12, 4, 5};

// loop through the array

// using for loop

System.out.println("Using for Loop:");

for(int i = 0; i < age.length; i++) {

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

}

}

}

class Main {

public static void main(String[] args) {

// create an array

int[] age = {12, 4, 5};

// loop through the array

// using for loop

System.out.println("Using for-each Loop:");

for(int a : age) {

System.out.println(a);

}

}

}

class Main {

public static void main(String[] args) {

int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};

int sum = 0;

Double average;

// access all elements using for each loop

// add each element in sum

for (int number: numbers) {

sum += number;

}

// get the total number of elements

int arrayLength = numbers.length;

// calculate the average

// convert the average from int to double

average = ((double)sum / (double)arrayLength);

System.out.println("Sum = " + sum);

System.out.println("Average = " + average);

}

}

class PostDecrOptDemo

{

  public static void main(String args[])

  {

int i = 5, j = 5, sum = 0;

System.out.println("Value of i is " + i + ", j is " + j " and sum is " + sum);

sum = i + j--;

System.out.println("Value of i is " + i + ", j is " + j " and sum is " + sum);

  }

}

class Operator {

public static void main(String[] args) {

int a = 6, b = 6;

// a is displayed

// Then, a is increased to 7.

System.out.println(a++);

// b is increased to 6

// Then, b is displayed

System.out.println(++b);

}

}

class Main {

int a;

boolean b;

public static void main(String[] args) {

// A default constructor is called

Main obj = new Main();

System.out.println("Default Value:");

System.out.println("a = " + obj.a);

System.out.println("b = " + obj.b);

}

}

class Main {

int a;

boolean b;

// a private constructor

private Main() {

a = 0;

b = false;

}

public static void main(String[] args) {

// call the constructor

Main obj = new Main();

System.out.println("Default Value:");

System.out.println("a = " + obj.a);

System.out.println("b = " + obj.b);

}

}

class Company {

String name;

// public constructor

public Company() {

name = "ABC";

}

}

class Main {

public static void main(String[] args) {

// object is created in another class

Company obj = new Company();

System.out.println("Company name = " + obj.name);

}

}

Constructor overloading:

class Main {

String language;

// constructor with no parameter

Main() {

this.language = "Java";

}

// constructor with a single parameter

Main(String language) {

this.language = language;

}

public void getName() {

System.out.println("Programming Langauage: " + this.language);

}

public static void main(String[] args) {

// call constructor with no parameter

Main obj1 = new Main();

// call constructor with a single parameter

Main obj2 = new Main("Python");

obj1.getName();

obj2.getName();

}

}

**Static, this, super….**

class VariableDemo

{

static int count=0;

public void increment()

{

count++;

}

public static void main(String args[])

{

VariableDemo obj1=new VariableDemo();

VariableDemo obj2=new VariableDemo();

obj1.increment();

obj2.increment();

System.out.println("Obj1: count is="+obj1.count);

System.out.println("Obj2: count is="+obj2.count);

}

}

class JavaExample{

static int age;

static String name;

//This is a Static Method

static void disp(){

System.out.println("Age is: "+age);

System.out.println("Name is: "+name);

}

// This is also a static method

public static void main(String args[])

{

age = 30;

name = "Steve";

disp();

}

}

class Test {

    // static variable

    static int a = m1();

    // static block

    static

    {

        System.out.println("Inside static block");

    }

    // static method

    static int m1()

    {

        System.out.println("from m1");

        return 20;

    }

    // static method(main !!)

    public static void main(String[] args)

    {

        System.out.println("Value of a : " + a);

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

    }

}

class Student

{

int rollNo;

String name;

static String collegeName;//static variable

}

class StaticVar

{

public static void main(String[] args)

{

//create 3 object which will share collegeName value

Student s1= new Student();

Student s2= new Student();

Student s3= new Student();

//assign value to static variable collegeName

Student.collegeName="Topper ";

//assign values to instance variables

s1.rollNo=1;

s1.name="stud1";

s2.rollNo=2;

s2.name="stud2";

s3.rollNo=3;

s3.name="stud3";

//Print the values of the objects

System.out.println("S1 Roll No.= "+s1.rollNo+" S1 Name= "+s1.name+" S1 College Name= "+s1.collegeName );

System.out.println("S2 Roll No.= "+s2.rollNo+" S2 Name= "+s2.name+" S2 College Name= "+s2.collegeName );

System.out.println("S3 Roll No.= "+s3.rollNo+" S3 Name= "+s3.name+" S3 College Name= "+s3.collegeName );

//if one object change the value of static variable then it will reflect into all objects

s2.collegeName="my college";

s2.name="ABC";

//Print the values of the objects after change

System.out.println("S1 Roll No.= "+s1.rollNo+" S1 Name= "+s1.name+" S1 College Name= "+s1.collegeName );

System.out.println("S2 Roll No.= "+s2.rollNo+" S2 Name= "+s2.name+" S2 College Name= "+s2.collegeName );

System.out.println("S3 Roll No.= "+s3.rollNo+" S3 Name= "+s3.name+" S3 College Name= "+s3.collegeName );

}

}

**This…**

this can be used to:

* Invoke current class constructor
* Invoke current class method
* Return the current class object
* Pass an argument in the method call
* Pass an argument in the constructor call

class this\_Test

{

     int val1;

     int val2;

    // Parameterized constructor

    this\_Test(int val1, int val2)

    {

        this.val1 = val1 + val1;

        this.val2 = val2 + val2;

    }

   void display()

    {

          System.out.println("Value val1 = " + val1 + " Value val2 = " + val2);

    }

}

class Main{

       public static void main(String[] args)

       {

            this\_Test object = new this\_Test(5,10);

            object.display();

       }

}

class Test\_method

{

    int val1;

    int val2;

    Test\_method()

    {

        val1 = 10;

        val2 = 20;

    }

    void printVal(Test\_method obj)

    {

             System.out.println("val1 = " + obj.val1 + "  val2 = " + obj.val2);

    }

    void get()

    {

           printVal(this);

    }

}

class Main{

    public static void main(String[] args)

    {

        Test\_method object = new Test\_method();

        object.get();

    }

}

class Test\_this {

    void print()

    {

        // calling fuctionshow()

       this.show();

       System.out.println("Test\_this:: print");

    }

    void show() {

        System.out.println("Test\_this::show");

    }

}

class Main{

    public static void main(String args[]) {

        Test\_this t1 = new Test\_this();

        t1.print();

    }

}

class Test\_this

{

   int val\_a;

   int val\_b;

    //Default constructor

   Test\_this()

    {

        val\_a = 10;

        val\_b = 20;

    }

   Test\_this get()

    {

        return this;

    }

   void display()

    {

         System.out.println("val\_a = " + val\_a + "  val\_b = " + val\_b);

    }

}

class Main{

    public static void main(String[] args)

    {

        Test\_this object = new Test\_this();

        object.get().display();

    }

}

class This\_construct

{

     int val1;

     int val2;

    //Default constructor

    This\_construct()

    {

        this(10, 20);

        System.out.println("Default constructor \n");

    }

    //Parameterized constructor

   This\_construct(int val1, int val2)

    {

        this.val1 = val1;

        this.val2 = val2;

        System.out.println("Parameterized constructor");

    }

}

class Main{

    public static void main(String[] args)

    {

         This\_construct object = new This\_construct();

    }

}

Super….

class Animal {

// default or no-arg constructor of class Animal

Animal() {

System.out.println("I am an animal");

}

}

class Dog extends Animal {

// default or no-arg constructor of class Dog

Dog() {

// calling default constructor of the superclass

super();

System.out.println("I am a dog");

}

}

class Main {

public static void main(String[] args) {

Dog dog1 = new Dog();

}

}

class Parentclass

{

//Overridden method

void display(){

System.out.println("Parent class method");

}

}

class Subclass extends Parentclass

{

//Overriding method

void display(){

System.out.println("Child class method");

}

void printMsg(){

//This would call Overriding method

display();

//This would call Overridden method

super.display();

}

public static void main(String args[]){

Subclass obj= new Subclass();

obj.printMsg();

}

}

**Method overloading, Method overriding, Inheritance:**

public class MethodOverLoadEx{

//Method 1

int sum(int x,int y){

return x+y;

}

//Method 2

int sum(int x,int y,int z){

return x+y+z;

}

public static void main(String[] args){

MethodOverLoadEx obj = new MethodOverLoadEx();

System.out.println(obj.sum(20,30));

System.out.println(obj.sum(20,30,40));

}}

public class MethodOverLoadEx{

//Method 1

double sum(int x,double y){

return x+y;

}

//Method 2

double sum(double x,int y){

return x+y;

}

public static void main(String[] args){

MethodOverLoadEx obj = new MethodOverLoadEx();

System.out.println(obj.sum(30,30.2));

System.out.println(obj.sum(50.2,20));

}}

public class MethodOverLoadEx{

//Method 1

long sum(int x,long y){

return x+y;

}

//Method 2

int sum(int x,int y,int z){

return x+y;

}

public static void main(String[] args){

MethodOverLoadEx obj = new MethodOverLoadEx();

System.out.println(obj.sum(30,10)); //Now, here second parameter is int but it will be type

//promoted to long

System.out.println(obj.sum(50,20,30));

}}

class Human{

//Overridden method

public void eat()

{

System.out.println("Human is eating");

}

}

class Boy extends Human{

//Overriding method

public void eat(){

System.out.println("Boy is eating");

}

public static void main( String args[]) {

Boy obj = new Boy();

//This will call the child class version of eat()

obj.eat();

}

}

class ABC{

//Overridden method

public void disp()

{

System.out.println("disp() method of parent class");

}

}

class Demo extends ABC{

//Overriding method

public void disp(){

System.out.println("disp() method of Child class");

}

public void newMethod(){

System.out.println("new method of child class");

}

public static void main( String args[]) {

/\* When Parent class reference refers to the parent class object

\* then in this case overridden method (the method of parent class)

\* is called.

\*/

ABC obj = new ABC();

obj.disp();

/\* When parent class reference refers to the child class object

\* then the overriding method (method of child class) is called.

\* This is called dynamic method dispatch and runtime polymorphism

\*/

ABC obj2 = new Demo();

obj2.disp();

}

}

class Teacher {

void teach() {

System.out.println("Teaching subjects");

}

}

class Students extends Teacher {

void listen() {

System.out.println("Listening to teacher");

}

}

class CheckForInheritance {

public static void main(String args[]) {

Students s1 = new Students();

s1.teach();

s1.listen();

}

}

class Teacher {

void teach() {

System.out.println("Teaching subject");

}

}

class Student extends Teacher {

void listen() {

System.out.println("Listening");

}

}

class homeTution extends Student {

void explains() {

System.out.println("Does homework");

}

}

class CheckForInheritance {

public static void main(String argu[]) {

homeTution h = new himeTution();

h.explains();

d.teach();

d.listen();

}

}

There are four access modifiers keywords in Java and they are:

|  |  |
| --- | --- |
| Modifier | Description |
| Default | declarations are visible only within the package (package private) |
| Private | declarations are visible within the class only |
| Protected | declarations are visible within the package or all subclasses |
| Public | declarations are visible everywhere |

class Data {

// private variable

private String name;

}

public class Main {

public static void main(String[] main){

// create an object of Data

Data d = new Data();

// access private variable and field from another class

d.name = "abc";

}

}

class Data {

private String name;

// getter method

public String getName() {

return this.name;

}

// setter method

public void setName(String name) {

this.name= name;

}

}

public class Main {

public static void main(String[] main){

Data d = new Data();

// access the private variable using the getter and setter

d.setName("abc");

System.out.println(d.getName());

}

}

class Animal {

// protected method

protected void display() {

System.out.println("I am an animal");

}

}

class Dog extends Animal {

public static void main(String[] args) {

// create an object of Dog class

Dog dog = new Dog();

// access protected method

dog.display();

}

}

// Animal.java file

// public class

public class Animal {

// public variable

public int legCount;

// public method

public void display() {

System.out.println("I am an animal.");

System.out.println("I have " + legCount + " legs.");

}

}

// Main.java

public class Main {

public static void main( String[] args ) {

// accessing the public class

Animal animal = new Animal();

// accessing the public variable

animal.legCount = 4;

// accessing the public method

animal.display();

}

}

Abstraction is the process of separating ideas from specific instances of those ideas at work.

abstraction is the process of separating ideas from specific instances of those ideas at work

public abstract class Employee {

private String name;

private int paymentPerHour;

public Employee(String name, int paymentPerHour) {

this.name = name;

this.paymentPerHour = paymentPerHour;

}

public abstract int calculateSalary();

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getPaymentPerHour() {

return paymentPerHour;

}

public void setPaymentPerHour(int paymentPerHour) {

this.paymentPerHour = paymentPerHour;

}

}

abstract class Shape {

String color;

// these are abstract methods

abstract double area();

public abstract String toString();

// abstract class can have constructor

public Shape(String color) {

System.out.println("Shape constructor called");

this.color = color;

}

// this is a concrete method

public String getColor() {

return color;

}

}

class Circle extends Shape {

double radius;

public Circle(String color, double radius) {

// calling Shape constructor

super(color);

System.out.println("Circle constructor called");

this.radius = radius;

}

@Override

double area() {

return Math.PI \* Math.pow(radius, 2);

}

@Override

public String toString() {

return "Circle color is " + super.color + "and area is : " + area();

}

}

class Rectangle extends Shape {

double length;

double width;

public Rectangle(String color, double length, double width) {

// calling Shape constructor

super(color);

System.out.println("Rectangle constructor called");

this.length = length;

this.width = width;

}

@Override

double area() {

return length \* width;

}

@Override

public String toString() {

return "Rectangle color is " + super.color + "and area is : " + area();

}

}

public class AbstractionTest {

public static void main(String[] args) {

Shape s1 = new Circle("Red", 2.2);

Shape s2 = new Rectangle("Yellow", 2, 4);

System.out.println(s1.toString());

System.out.println(s2.toString());

}

}

abstract class Base {

    abstract void fun();

}

class Derived extends Base {

    void fun()

    {

        System.out.println("Derived fun() called");

    }

}

class Main {

    public static void main(String args[])

    {

        // Uncommenting the following line will cause

        // compiler error as the line tries to create an

        // instance of abstract class. Base b = new Base();

        // We can have references of Base type.

        Base b = new Derived();

        b.fun();

    }

}

abstract class Base {

    final void fun()

    {

        System.out.println("Derived fun() called");

    }

}

class Derived extends Base {

}

class Main {

    public static void main(String args[])

    {

        Base b = new Derived();

        b.fun();

    }

}

interface Polygon {

void getArea(int length, int breadth);

}

// implement the Polygon interface

class Rectangle implements Polygon {

// implementation of abstract method

public void getArea(int length, int breadth) {

System.out.println("The area of the rectangle is " + (length \* breadth));

}

}

class Main {

public static void main(String[] args) {

Rectangle r1 = new Rectangle();

r1.getArea(5, 6);

}

}

TRY THIS>>>write remaining code on your own…

interface A {

...

}

interface B {

...

}

interface C extends A, B {

...

}

interface In1

{

    final int a = 10;

    default void display()

    {

        System.out.println("hello");

    }

}

// A class that implements the interface.

class TestClass implements In1

{

    // Driver Code

    public static void main (String[] args)

    {

        TestClass t = new TestClass();

        t.display();

    }

}

Exceptions:

class Main {

public static void main(String[] args) {

try {

// code that generates exception

int divideByZero = 5 / 0;

}

catch (ArithmeticException e) {

System.out.println("ArithmeticException => " + e.getMessage());

}

finally {

System.out.println("This is the finally block");

}

}

}

import java.io.\*;

class Main {

// declareing the type of exception

public static void findFile() throws IOException {

// code that may generate IOException

File newFile = new File("test.txt");

FileInputStream stream = new FileInputStream(newFile);

}

public static void main(String[] args) {

try {

findFile();

}

catch (IOException e) {

System.out.println(e);

}

}

}

class StringIndexOutOfBound\_Demo

{

    public static void main(String args[])

    {

        try {

            String a = "This is like chipping "; // length is 22

            char c = a.charAt(24); // accessing 25th element

            System.out.println(c);

        }

        catch(StringIndexOutOfBoundsException e) {

            System.out.println("StringIndexOutOfBoundsException");

        }

    }

}

class  NumberFormat\_Demo

{

    public static void main(String args[])

    {

        try {

            // "akki" is not a number

            int num = Integer.parseInt ("aka") ;

            System.out.println(num);

        } catch(NumberFormatException e) {

            System.out.println("Number format exception");

        }

    }

}

class MyException extends Exception

{

    //store account information

    private static int accno[] = {1001, 1002, 1003, 1004};

    private static String name[] =

                 {"Nish", "Shubh", "Sush", "Abhi", "Akash"};

    private static double bal[] =

         {10000.00, 12000.00, 5600.0, 999.00, 1100.55};

    // default constructor

    MyException() {    }

    // parameterized constructor

    MyException(String str) { super(str); }

    // write main()

    public static void main(String[] args)

    {

        try  {

            // display the heading for the table

            System.out.println("ACCNO" + "\t" + "CUSTOMER" +

                                           "\t" + "BALANCE");

            // display the actual account information

            for (int i = 0; i < 5 ; i++)

            {

                System.out.println(accno[i] + "\t" + name[i] +

                                               "\t" + bal[i]);

                // display own exception if balance < 1000

                if (bal[i] < 1000)

                {

                    MyException me =

                       new MyException("Balance is less than 1000");

                    throw me;

                }

            }

        } //end of try

        catch (MyException e) {

            e.printStackTrace();

        }

    }

}

public class Main {

static void checkAge(int age) {

if (age < 18) {

**throw** new ArithmeticException("Access denied - You must be at least 18 years old.");

}

else {

System.out.println("Access granted - You are old enough!");

}

}

public static void main(String[] args) {

checkAge(15); // Set age to 15 (which is below 18...)

}

}

FILE Handling: I/O:

// importing the File class

import java.io.File;

class Main {

public static void main(String[] args) {

// create a file object for the current location

File file = new File("newFile.txt");

try {

// trying to create a file based on the object

boolean value = file.createNewFile();

if (value) {

System.out.println("The new file is created.");

}

else {

System.out.println("The file already exists.");

}

}

catch(Exception e) {

e.getStackTrace();

}

}

}

// importing the FileReader class

import java.io.FileReader;

class Main {

public static void main(String[] args) {

char[] array = new char[100];

try {

// Creates a reader using the FileReader

FileReader input = new FileReader("input.txt");

// Reads characters

input.read(array);

System.out.println("Data in the file:");

System.out.println(array);

// Closes the reader

input.close();

}

catch(Exception e) {

e.getStackTrace();

}

}

}

// importing the FileWriter class

import java.io.FileWriter;

class Main {

public static void main(String args[]) {

String data = "This is the data in the output file";

try {

// Creates a Writer using FileWriter

FileWriter output = new FileWriter("output.txt");

// Writes string to the file

output.write(data);

System.out.println("Data is written to the file.");

// Closes the writer

output.close();

}

catch (Exception e) {

e.getStackTrace();

}

}

}

import java.io.File;

class Main {

public static void main(String[] args) {

// creates a file object

File file = new File("file.txt");

// deletes the file

boolean value = file.delete();

if(value) {

System.out.println("The File is deleted.");

}

else {

System.out.println("The File is not deleted.");

}

}

}

import java.io.File;

class Main {

public static void main(String[] args) {

// creates a file object with specified path

File file = new File("Java Example\\directory");

// tries to create a new directory

boolean value = file.mkdir();

if(value) {

System.out.println("The new directory is created.");

}

else {

System.out.println("The directory already exists.");

}

}

}

import java.io.InputStreamReader;

import java.io.FileInputStream;

class Main {

public static void main(String[] args) {

// Creates an array of character

char[] array = new char[100];

try {

// Creates a FileInputStream

FileInputStream file = new FileInputStream("input.txt");

// Creates an InputStreamReader

InputStreamReader input = new InputStreamReader(file);

// Reads characters from the file

input.read(array);

System.out.println("Data in the stream:");

System.out.println(array);

// Closes the reader

input.close();

}

catch(Exception e) {

e.getStackTrace();

}

}

import java.io.FileOutputStream;

import java.io.OutputStreamWriter;

public class Main {

public static void main(String args[]) {

String data = "This is a line of text inside the file.";

try {

// Creates a FileOutputStream

FileOutputStream file = new FileOutputStream("output.txt");

// Creates an OutputStreamWriter

OutputStreamWriter output = new OutputStreamWriter(file);

// Writes string to the file

output.write(data);

// Closes the writer

output.close();

}

catch (Exception e) {

e.getStackTrace();

}

}

}

import java.io.FileReader;

import java.io.BufferedReader;

class Main {

public static void main(String[] args) {

// Creates an array of character

char[] array = new char[100];

try {

// Creates a FileReader

FileReader file = new FileReader("input.txt");

// Creates a BufferedReader

BufferedReader input = new BufferedReader(file);

// Reads characters

input.read(array);

System.out.println("Data in the file: ");

System.out.println(array);

// Closes the reader

input.close();

}

catch(Exception e) {

e.getStackTrace();

}

}

}

import java.io.FileWriter;

import java.io.BufferedWriter;

public class Main {

public static void main(String args[]) {

String data = "This is the data in the output file";

try {

// Creates a FileWriter

FileWriter file = new FileWriter("output.txt");

// Creates a BufferedWriter

BufferedWriter output = new BufferedWriter(file);

// Writes the string to the file

output.write(data);

// Closes the writer

output.close();

}

catch (Exception e) {

e.getStackTrace();

}

}

}

Wrapper classes:

class Main {

public static void main(String[] args) {

// create primitive types

int a = 5;

double b = 5.65;

//converts into wrapper objects

Integer aObj = Integer.valueOf(a);

Double bObj = Double.valueOf(b);

if(aObj instanceof Integer) {

System.out.println("An object of Integer is created.");

}

if(bObj instanceof Double) {

System.out.println("An object of Double is created.");

}

}

}

class Main {

public static void main(String[] args) {

// creates objects of wrapper class

Integer aObj = Integer.valueOf(23);

Double bObj = Double.valueOf(5.55);

// converts into primitive types

int a = aObj.intValue();

double b = bObj.doubleValue();

System.out.println("The value of a: " + a);

System.out.println("The value of b: " + b);

}

}

public class Main {

public static void main(String[] args) {

Integer myInt = 5;

Double myDouble = 5.99;

Character myChar = 'A';

System.out.println(myInt);

System.out.println(myDouble);

System.out.println(myChar);

}

}

public class Main {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

for (String i : cars) {

System.out.println(i);

}

}

}

import java.util.ArrayList;

public class Main {

public static void main(String[] args) {

ArrayList<Integer> myNumbers = new ArrayList<Integer>();

myNumbers.add(10);

myNumbers.add(15);

myNumbers.add(20);

myNumbers.add(25);

for (int i : myNumbers) {

System.out.println(i);

}

}

}

// Import the HashMap class

import java.util.HashMap;

public class Main {

public static void main(String[] args) {

// Create a HashMap object called people

HashMap<String, Integer> people = new HashMap<String, Integer>();

// Add keys and values (Name, Age)

people.put("John", 32);

people.put("Steve", 30);

people.put("Angie", 33);

for (String i : people.keySet()) {

System.out.println("key: " + i + " value: " + people.get(i));

}

}

}

import java.util.HashSet;

public class Main {

public static void main(String[] args) {

// Create a HashSet object called numbers

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

// Add values to the set

numbers.add(4);

numbers.add(7);

numbers.add(8);

// Show which numbers between 1 and 10 are in the set

for(int i = 1; i <= 10; i++) {

if(numbers.contains(i)) {

System.out.println(i + " was found in the set.");

} else {

System.out.println(i + " was not found in the set.");

}

}

}

}

import java.util.ArrayList;

import java.util.Iterator;

public class Main {

public static void main(String[] args) {

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

numbers.add(12);

numbers.add(8);

numbers.add(2);

numbers.add(23);

Iterator<Integer> it = numbers.iterator();

while(it.hasNext()) {

Integer i = it.next();

if(i < 10) {

it.remove();

}

}

System.out.println(numbers);

}

}

Generics, Inner classes and Threads:

class Main {

public static void main(String[] args) {

// initialize generic class

// with Integer data

GenericsClass<Integer> intObj = new GenericsClass<>(5);

System.out.println("Generic Class returns: " + intObj.getData());

// initialize generic class

// with String data

GenericsClass<String> stringObj = new GenericsClass<>("Java Programming");

System.out.println("Generic Class returns: " + stringObj.getData());

}

}

// create a generics class

class GenericsClass<T> {

// variable of T type

private T data;

public GenericsClass(T data) {

this.data = data;

}

// method that return T type variable

public T getData() {

return this.data;

}

}

class Main {

public static void main(String[] args) {

// initialize the class with Integer data

DemoClass demo = new DemoClass();

// generics method working with String

demo.<String>genericsMethod("Java Programming");

// generics method working with integer

demo.<Integer>genericsMethod(25);

}

}

class DemoClass {

// creae a generics method

public <T> void genericsMethod(T data) {

System.out.println("Generics Method:");

System.out.println("Data Passed: " + data);

}

}

class Test<T, U>

{

    T obj1;  // An object of type T

    U obj2;  // An object of type U

    // constructor

    Test(T obj1, U obj2)

    {

        this.obj1 = obj1;

        this.obj2 = obj2;

    }

    // To print objects of T and U

    public void print()

    {

        System.out.println(obj1);

        System.out.println(obj2);

    }

}

// Driver class to test above

class Main

{

    public static void main (String[] args)

    {

        Test <String, Integer> obj =

            new Test<String, Integer>("GfG", 15);

        obj.print();

    }

}

package demotest;

public class thread\_example1 implements Runnable {

@Override

public void run() {

}

public static void main(String[] args) {

Thread guruthread1 = new Thread();

guruthread1.start();

try {

guruthread1.sleep(1000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

guruthread1.setPriority(1);

int gurupriority = guruthread1.getPriority();

System.out.println(gurupriority);

System.out.println("Thread Running");

}

}

class NewThread implements Runnable {

    String name;

    Thread thread;

    NewThread (String name){

        this.name = name;

        thread = new Thread(this, name);

        System.out.println( "A New thread: " + thread+ "is created\n" );

        thread.start();

    }

    public void run() {

    try {

        for(int j = 5; j > 0; j--) {

            System.out.println(name + ": " + j);

            Thread.sleep(1000);

        }

    }catch (InterruptedException e) {

        System.out.println(name + " thread Interrupted");

    }

     System.out.println(name + " thread exiting.");

    }

}

class ThreadExample2 {

    public static void main(String args[]) {

        new NewThread("1st");

        new NewThread("2nd");

        new NewThread("3rd");

        try {

            Thread.sleep(8000);

        } catch (InterruptedException excetion) {

            System.out.println("Inturruption occurs in Main Thread");

        }

        System.out.println("We are exiting from Main Thread");

    }

}

public class ThreadExample1 extends Thread {

     // run() method to perform action for thread.

     public void run()

     {

        int a= 10;

        int b=12;

        int result = a+b;

        System.out.println("Thread started running..");

        System.out.println("Sum of two numbers is: "+ result);

     }

     public static void main( String args[] )

     {

      // Creating instance of the class extend Thread class

        ThreadExample1 t1 = new  ThreadExample1();

        //calling start method to execute the run() method of the Thread class

        t1.start();

     }

}

class Outer\_Demo {

int num;

// inner class

private class Inner\_Demo {

public void print() {

System.out.println("This is an inner class");

}

}

// Accessing he inner class from the method within

void display\_Inner() {

Inner\_Demo inner = new Inner\_Demo();

inner.print();

}

}

public class My\_class {

public static void main(String args[]) {

// Instantiating the outer class

Outer\_Demo outer = new Outer\_Demo();

// Accessing the display\_Inner() method.

outer.display\_Inner();

}

}

public class Outerclass {

// instance method of the outer class

void my\_Method() {

int num = 23;

// method-local inner class

class MethodInner\_Demo {

public void print() {

System.out.println("This is method inner class "+num);

}

} // end of inner class

// Accessing the inner class

MethodInner\_Demo inner = new MethodInner\_Demo();

inner.print();

}

public static void main(String args[]) {

Outerclass outer = new Outerclass();

outer.my\_Method();

}

}

abstract class AnonymousInner {

public abstract void mymethod();

}

public class Outer\_class {

public static void main(String args[]) {

AnonymousInner inner = new AnonymousInner() {

public void mymethod() {

System.out.println("This is an example of anonymous inner class");

}

};

inner.mymethod();

}

}

public class Outer {

static class Nested\_Demo {

public void my\_method() {

System.out.println("This is my nested class");

}

}

public static void main(String args[]) {

Outer.Nested\_Demo nested = new Outer.Nested\_Demo();

nested.my\_method();

}

}