Create default/paramterzied constructors?

public class Sample{

public static void main(String args[]){

One o=new One("25");

}

}

class One{

public One(String a){

System.out.println(a);

}

}

**creating static property in java**

**class** Student8{

**int** rollno;

String name;

**static** String college ="ITS";

Student8(**int** r,String n){

   rollno = r;

   name = n;

   }

**void** display (){System.out.println(rollno+" "+name+" "+college);}

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

Student8 s1 = **new** Student8(111,"Karan");

Student8 s2 = **new** Student8(222,"Aryan");

s1.display();

s2.display();

 }

}

**Static Class**

class Example2{

int num;

//Static class

static class X{

static String str="Inside Class X";

num=99;

}

public static void main(String args[])

{

Example2.X obj = new Example2.X();

System.out.println("Value of num="+obj.str);

}

}

**creating static method**

**class** Languages {

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

display();

}

**static** **void** display() {

System.out.println("Java is my favorite programming language.");

}

}

**create static block**

1………..

private static final HashMap<String, String> MAP = new HashMap<String, String>();

static {

MAP.put("banana", "honey");

MAP.put("peanut butter", "jelly");

MAP.put("rice", "beans");

}

**2………………**

class Example3{

static int num;

static String mystr;

static{

num = 97;

mystr = "Static keyword in Java";

}

public static void main(String args[])

{

System.out.println("Value of num="+num);

System.out.println("Value of mystr="+mystr);

}

}

Static block is mostly used for changing the default values of static variables.

**Creating objects in java**

public class Puppy {

public Puppy(String name) {

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

}

public static void main(String []args) {

// Following statement would create an object myPuppy

Puppy myPuppy = new Puppy( "tommy" );

}

}

calling method with no return and parameter

public class Program {

static void displayPassword(String password) {

// Write the password to the console.

System.out.println("Password: " + password);

// Return if our password is long enough.

if (password.length() >= 5) {

**return**;

}

System.out.println("Password too short!");

// An implicit return is here.

}

public static void main(String[] args) {

displayPassword("furball");

displayPassword("cat");

}

}

write code to handle exceptions with try/catch/finally

class TestExceptions {

static void myMethod(int testnum) throws Exception {

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

if (testnum == 12)

throw new Exception();

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

return;

}

public static void main(String args[]) {

int testnum = 12;

try {

System.out.println("try - first statement");

myMethod(testnum);

System.out.println("try - last statement");

}

catch ( Exception ex) {

System.out.println("An Exception");

}

finally {

System. out. println( "finally") ;

}

System.out.println("Out of try/catch/finally - statement");

}

}

what is checked exception/unchecked exception

**Checked:** are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throwskeyword.

**Unchecked** are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

**write code for creating abstract class**

abstract public class AbstractDemo{

public void myMethod(){

System.out.println("Hello");

}

abstract public void anotherMethod();

}

public class ConcreteDemo{

public void anotherMethod() {

System.out.print("Abstract method");

}

public static void main(String args[])

{

//Can't create object of abstract class - error!

AbstractDemo obj = new AbstractDemo();

obj.display();

}

}

**implement method overriding**

class Human{

public void eat()

{

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

}

}

class Boy extends Human{

public void eat(){

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

}

public static void main( String args[]) {

Boy obj = new Boy();

obj.eat();

}

}

Here Boy class extends Human class. Both the classes have a common method void eat(). Boy class is giving its own implementation to the eat() method or in other words it is overriding the method eat().

write code to add items to integer, string array

public static void main(String[] args) {

int[] series = {4,2};

series = addElement(series, 3);

series = addElement(series, 1);

}

static int[] addElement(int[] a, int e) {

a = Arrays.copyOf(a, a.length + 1);

a[a.length - 1] = e;

return a;

}

write code to retrieve items from integer, string array

import java.util.\*;

public class Temperature {

public static void main(String[] args) {

Scanner console = new Scanner(System.in);

// Input the number of days from the user.

System.out.print("How many days' temperatures? ");

int days = console.nextInt( );

// Declare and create an array, maybe should check if days is positive

int[ ] temps = new int[days];

// Input and store the temperatures in the array

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

System.out.print("Day " + i + "'s high temp: ");

temps[i] = console.nextInt( );

}

// Calculate and print the average

int sum = 0;

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

sum += temps[i];

}

// need a cast to avoid integer division

double average = (double) sum / temps.length;

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

// Count the number of values that were above average

int count = 0;

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

if (temps[i] > average) {

count++;

}

}

System.out.println(count + " days were above average");

}

}

write code to add items to ArrayList collection

import java.util.ArrayList;

public class Program {

public static void main(String[] args) {

// Create new ArrayList.

**ArrayList**<Integer> elements = new ArrayList<>();

// Add three elements.

elements.add(10);

elements.add(15);

elements.add(20);

// Get size and display.

int count = elements.size();

System.out.println("Count: " + count);

// Loop through elements.

for (int i = 0; i < elements.size(); i++) {

int value = elements.get(i);

System.out.println("Element: " + value);

}

}

}

write code to add items HashMap

import java.util.HashMap;

public class Program {

public static void main(String[] args) {

// Create new HashMap.

// ... Uses diamond inference on right side.

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

// Put three keys with values.

hash.put("dog", 1);

hash.put("cat", 2);

hash.put("rabbit", 3);

// Look up some known values.

int a = hash.get("dog");

int b = hash.get("cat");

// Display results.

System.out.println(a);

System.out.println(b);

}

}

Write code to add items to hashset

import java.util.HashSet;

public class HashSetExample {

public static void main(String args[]) {

// HashSet declaration

HashSet<String> hset =

new HashSet<String>();

// Adding elements to the HashSet

hset.add("Apple");

hset.add("Mango");

hset.add("Grapes");

hset.add("Orange");

hset.add("Fig");

//Addition of duplicate elements

hset.add("Apple");

hset.add("Mango");

//Addition of null values

hset.add(null);

hset.add(null);

//Displaying HashSet elements

System.out.println(hset);

}

}

**Method overloading:**

**package** session10AM;

**public** **class** Example

{

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

{

**int** a=10;

**int** count=args.length;

**int** result=a/count;

System.***out***.println("Result is:"+result);

}

}

create Employee class

public class Employee {

String lastName = null;

String firstName = null;

double ID;

public Employee(String lastName, String firstName, double ID){

this.lastName = lastName;

this.firstName = firstName;

this.ID = ID;

}

public String empStat(){

return "Last Name: " + lastName + "First Name: " + firstName + "ID" + ID;

}

}

**Java program for interface**

**package** session10AM;

**public** **interface** Animals {

/\* File name : MammalInt.java \*/

**public** **class** MammalInt **implements** Animals {

**public** **void** eat() {

System.***out***.println("Mammal eats");

}

**public** **void** travel() {

System.***out***.println("Mammal travels");

}

**public** **int** noOfLegs() {

**return** 0;

}

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

MammalInt m = **new** MammalInt();

m.eat();

m.travel();

}

}

}

**Write a program for single inheritance**

**package** beginnersbook.com;

**public** **class** Inherit\_Single {

**protected** String str;

Inherit\_Single() {

str = "Java ";

}

}

**class** SubClass **extends** Inherit\_Single {

SubClass() {

str = str.concat("World !!!");

}

**void** display()

{

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

}

}

**class** MainClass {

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

SubClass obj = **new** SubClass();

obj.display();

}

}

Difference between string, string buffer, string builder with example

**String**  
  
Stringis *immutable*  ( once created can not be changed )object  . The object created as a String is stored in the  **Constant String Pool**  .   
Every immutable object in Java is thread safe ,that implies String is also thread safe . String can not be used by two threads simultaneously.  
String  once assigned can not be changed.

**StringBuffer**  
  
StringBufferis mutable means one can change the value of the object . The object created through StringBuffer is stored in the heap . StringBuffer  has the same methods as the StringBuilder , but **each method in StringBuffer is synchronized**that is **StringBuffer is thread safe** .

**public** **class** BufferTest{

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

 StringBuffer buffer=**new** StringBuffer("hello");

buffer.append("java");

System.out.println(buffer);

  }

}

**StringBuilder**  
  
StringBuilder  is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that**StringBuilder is also not thread safe.**  
StringBuilder is fast as it is not thread safe .

**public** **class** BuilderTest{

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

StringBuilder builder=**new** StringBuilder("hello");

 builder.append("java");

 System.out.println(builder);

    }

}

**write a code to save data into excel file and read from excel file (POI and jexcelAPI)**

import java.io.File;

import java.io.IOException;

import jxl.\*;

import jxl.read.biff.BiffException;

public class ReadExcel {

    public static void main(String[] args) throws IOException, BiffException {

        Workbook workbook = Workbook.getWorkbook(new File("resources/sample.xls"));

        Sheet sheet = workbook.getSheet(0);

        Cell A = sheet.getCell(0,0);

        Cell B = sheet.getCell(1,0);

        Cell C = sheet.getCell(2,0);

        String stringA = A.getContents();

        String stringB = B.getContents();

        String stringC = C.getContents();

        System.out.println(stringA);

        System.out.println(stringB);

        System.out.println(stringC);

    }

}

how to update the data into XML file and read data from XML file

protected void setNodeValue(String tagName, String value, NodeList nodes) {

    Node node = getNode(tagName, nodes);

    if ( node == null )

        return;

    // Locate the child text node and change its value

    NodeList childNodes = node.getChildNodes();

    for (int y = 0; y < childNodes.getLength(); y++ ) {

        Node data = childNodes.item(y);

        if ( data.getNodeType() == Node.TEXT\_NODE ) {

            data.setNodeValue(value);

            return;

        }

    }

}