**Steps to download Java:**

**Google javadownload**

[**https://java.com/en/download/**](https://java.com/en/download/)

**click java download**

**Agree and start free download**

**After that download jdk and jre from**

[**http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html**](http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html)

**then download them and install in the computer.**

**Once the installation is done,open cmd in your system them type java and click enter you will get some notification.**

**Steps To Download Eclipse:**

1. **Download the eclipse installer from** [**http://www.eclipse.org/downloads**](http://www.eclipse.org/downloads)
2. **Start the Eclipse installer executable as RUN.**
3. **Select the package to install click Eclipse IDE for Java Developers.**
4. **Select your installation folder and click install.**
5. **Launch Eclipse.**

**How to create a workspace in eclipse?**

**Click on the file which is at the top left corner**

**Select switch workspace and click on the other**

**Name it with your own C:\Users\sande\Javasamples then click OK.**

**Eclipse will automatically closes and when you open it again it shows you a new workspace without any older ones.**

**Steps to create a project.?**

**Click file**

**Click new javaproject**

**Enter project name JavaSamples**

**Click finish**

**Double click on JavaSampkes->src->rightclick->new class.**

**Java :**

**What is Data Type?**

**A data type indicates what sort of value or the type of data the variable can represents, such as integer, floating-point numbers, character, boolean or an alphanumeric string.**

**There are other data types as well like short, long and float but in Selenium programming you may not face any situation where you have to use these data types. These data types are used when each byte of memory is important for better performance of the system.**

**Types:**

**Primitive**

**Reference data types**

**Primitive:**

**Byte -128 to 127**

**Short -32,768 to 32,767**

**Int - 2,147,483,648 to 2,147,483,647**

**Long -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807**

**Float**

**Double**

**Boolean**

**Char  '\u0000'  to**  **'\uffff'**

**What is a variable?**

**As it names suggest, variable is a value that can change. A simple computer program uses a set of instructions and data. Data can be constantsor fixed values that never change and it can be variablethat can change during the execution.**

**Create method with void?**

**The void keyword allows us to create methods which do not return a value. Here, in the following example we're considering a void method methodRankPoints. This method is a void method, which does not return any value. Call to a void method must be a statement i.e. methodRankPoints*(255.7);*. It is a Java statement which ends with a semicolon as shown in the following example.**

**Example**

**public class ExampleVoid {**

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

**methodRankPoints(255.7);**

**}**

**public static void methodRankPoints(double points) {**

**if (points >= 202.5) {**

**System.out.println("Rank:A1");**

**}else if (points >= 122.4) {**

**System.out.println("Rank:A2");**

**}else {**

**System.out.println("Rank:A3");**

**}**

**}**

**}**

**create .java file/class**

Right click on the package and select new class and give the name of the class in camel cases starting with capital letter. Make sure that the class starts with letter or any special character it should not start with number.

**what is main method will do?**

The main method is the method that's defined to be called at the start of an application. Without it, there is no place to start running.

You can compile any Java class without a main method, but a standalone application can't run without a main() method \*.

**creating variable, we can create variables inside method**

Syntax for creating variable

**Syntax: datatype variable;**

The variables can be created inside the methods but it is restricted to that method only.

**Creating method with return data type, int/string/double/float/date etc**

The method with return datatype with int/string/double/float/date etc would return that specific datatype to it’s super class.

**Method that will return hard coded value**

The method will return the hard-coded value if the actual value is forced into the source code.

**Creating method with return data type and parameter**

Java program that calls method in return statement

public class Program {

static int cube(int value) {

// Return number to the power of 3.

return (int) Math.pow(value, 3);

}

static intgetVolume(int size) {

// Return cubed number.

return cube(size);

}

public static void main(String[] args) {

// Assign to the return value of getVolume.

int volume = getVolume(2);

System.out.println(volume);

}

}

**Output** 8

**Creating static property:**

static variable (“static” Keyword = **Class Variables**) In Java Variables can be declared with the “ static ” keyword. Example: static int y = 0; When a variable is declared with the keyword static , its called a **class variable** .

**class** Counter2{

**static** **int** count=0;//will get memory only once and retain its value

Counter2(){

count++;

System.out.println(count);

}

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

Counter2 c1=**new** Counter2();

Counter2 c2=**new** Counter2();

Counter2 c3=**new** Counter2();

 }

}

**Output**:1

2

3

**Creating static method**

If you apply static keyword with any method, it is known as static method.

A static method belongs to the class rather than object of a class.

A static method can be invoked without the need for creating an instance of a class.

static method can access static data member and can change the value of it.

//Program of changing the common property of all objects(static field).

**class** Student9{

**int** rollno;

     String name;

**static** String college = "ITS";

**static** **void** change(){

     college = "BBDIT";

     }

     Student9(**int** r, String n){

     rollno = r;

     name = n;

     }

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

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

    Student9.change();

    Student9 s1 = **new** Student9 (111,"Karan");

    Student9 s2 = **new** Student9 (222,"Aryan");

    Student9 s3 = **new** Student9 (333,"Sonoo");

    s1.display();

    s2.display();

    s3.display();

    }

}

**Output**:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

**Create static block**

Is used to initialize the static data member.It is executed before main method at the time of classloading.

**class** A2{

**static**{System.out.println("static block is invoked");}

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

   System.out.println("Hello main");

  }

}

Output:static block is invoked

Hello main

**Creating object:**

An entity that has state and behavior is known as an object. An Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

**Calling method with void**

**class** Student{

**int** rollno;

 String name;

**void** insertRecord(**int** r, String n){

  rollno=r;

  name=n;

 }

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

}

**class** TestStudent4{

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

  Student s1=**new** Student();

  Student s2=**new** Student();

  s1.insertRecord(111,"Karan");

  s2.insertRecord(222,"Aryan");

  s1.displayInformation();

  s2.displayInformation();

 }

}

Output: 111 Karan

222 Aryan

**Calling method with no return and parameter**

public class ExampleVoid {

public static void main(String[] args) {

methodRankPoints(255.7);

}

public static void methodRankPoints(double points) {

if (points >= 202.5) {

System.out.println("Rank:A1");

}else if (points >= 122.4) {

System.out.println("Rank:A2");

}else {

System.out.println("Rank:A3");

}

}

}

**Output**

Rank:A1

**Calling method with return and no parameter**

**publicclass**TestMethod {

**staticint***x*=19;

**publicstaticvoid** main(String[] args) {

**for**(**int**i=0; i<=3; i++)

{

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

}

}

**staticint** fill(){

*x*=*x*+1;

**return***x*;

}

}

Output:

20

21

22

23

**Calling method with return and parameter**

**publicclass**ReturnParameter {

**publicstaticvoid** main(String[] args) {

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

System.***out***.print(*add*(1, 3));

}

**publicstaticint** add(**int**value1, **int**value2) {

**return**value1 + value2;

}

}

**Output 4**

**Calling method with return and storing the return data**

**package**my\_training;

**publicclass**StoreReturnValue {

**staticint***value*;

**publicstaticvoid** main(String[] args) {

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

*value* = *add*(1, 3);

System.***out***.print(*value*);

}

**publicstaticint** add(**int**value1, **int**value2) {

**return**value1 + value2;

}

}

**Output 4**

**Calling static method**

**publicclass**StaticMethodCall {

**publicstaticvoid** method(){

System.***out***.println("Called");

}

**publicstaticvoid** main(String[] args){

StaticMethodCalls = **null**;

s.*method*();

}

}

**Output:** Called

**Create classes under multiple packages**

package package1;

public class Package1Class {

}

package package2;

public class Package2Class {

}

**Calling classes under different packages**

package package1;

public class Package1Class {

}

package package2;

import package1.Package1Class;

public class Package2Class {

private Package1Class x;

public Package2Class (Package1Class x) {

this.x = x

}

}

**write code to handle exceptions with try/catch/finally**

**package** training;

**publicclass** Exception {

**publicstaticvoid** main(String[] args)

{

System.***out***.println(" The Program Starts" );

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

// Declaring An Array of Size 2

**int** [] arr1 = **newint** [2];

arr1[0] = 25;

arr1[1] = 55;

System.***out***.println("First Element is "+arr1[0] );

System.***out***.println("Second Element is "+arr1[1] );

**try**

{

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

System.***out***.println("Entering Try Block" );

System.***out***.println("First Element is "+arr1[2] );

}

**catch**(ArrayIndexOutOfBoundsExceptione1)

{

System.***out***.println("Array Index Out of bound Exception Caught in Catch block" );

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

}

**catch**(ArithmeticExceptione2)

{

System.***out***.println("Arithmetic Exception of dvision by Zero Caught in Catch block" );

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

}

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

System.***out***.println(" The Program Ends" );

}

}

**Output:**

**The Program Starts**

First Element is 25

Second Element is 55

Entering Try Block

Array Index Out of bound Exception Caught in Catch block

The Program Ends

**what is final keyword, create final class, final method, final property**

the **final keyword** is used in several different contexts to define an entity that can only be assigned once. Once a **final** variable has been assigned, it always contains the same value.

**write code for creating abstract class**

A method that is declared as abstract and does not have implementation is known as abstract method.

**abstractclass**AbstractClassBike {

**abstractvoid**run();

}

**class** Honda4 **extends**AbstractClassBike{

**void** run(){

System.***out***.println("running safely..");

}

**publicstaticvoid** main(String args[]){

AbstractClassBikeobj = **new** Honda4();

obj.run();

}

}

**Output:**

running safely..

**implement method overloading**

**package** training;

**abstractclass** Gmail

{

**abstractvoid** Display();

}

**class** Inbox **extends** Gmail

{

// Method OverRiding

**void** Display()

{

System.***out***.println("Displaying Inbox Details");

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

}

}

**class** Trash **extends** Gmail

{

// Method overloading

**void** Display(**int**a)

{

System.***out***.println("Method Overloaded with an Integer Argument");

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

}

**void** Display()

{

System.***out***.println("Displaying Trash Details");

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

}

}

**class** Spam **extends** Gmail

{

// Method OverRiding

**void** Display()

{

System.***out***.println("Displaying Spam Details");

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

}

}

**class**MyClass

{

**staticvoid** Test(Gmail g1)

{

g1.Display();

}

**staticvoid** Test(**int**a)

{

System.***out***.println("Method of Test Overloaded with an Integer Argument");

}

}

**publicclass**MethodOverload {

**publicstaticvoid** main(String[] args) {

System.***out***.println("The Program Starts");

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

MyClass.*Test*(**new** Inbox());

MyClass.*Test*(**new** Trash());

MyClass.*Test*(**new** Spam());

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

System.***out***.println("The Program Ends");

}

}

**Output**: The Program Starts

Displaying Inbox Details

Displaying Trash Details

Displaying Spam Details

The Program Ends

**implement method overriding**

**package** training;

// Super Class

**class** X

{

**void** test1()

{

System.***out***.println("Running the Test1 Method of Class X ");

//System.out.println();

}

}

// Y is a Subclass

**class** Y **extends** X

{

// Test1 Method of Super class (Class X) is Overloaded in Sub Class y

**void** test1()

{

// Method Overriding because The test1() method s Signature is same..

System.***out***.println("OverRiding The Test1 Method of Class X in Y");

System.***out***.println("Changing the Implementation of Test1 Method of Class X in Y");

}

}

**publicclass** Override {

**publicstaticvoid** main(String[] args)

{

System.***out***.println("The program starts");

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

X x1 = **new** X();

Y y1 = **new** Y();

y1.test1();

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

x1.test1();

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

System.***out***.println("The program starts");

}

}

Output:

The program starts

OverRiding The Test1 Method of Class X in Y

Changing the Implementation of Test1 Method of Class X in Y

Running the Test1 Method of Class X

The program starts

**implementing polymorphism**

**package** training;

**abstractclass** Gmail

{

**abstractvoid** Display();

}

**class** Inbox **extends** Gmail

{

// Method OverRiding

**void** Display()

{

System.***out***.println("Displaying Inbox Details");

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

}

}

**class** Trash **extends** Gmail

{

// Method overloading

**void** Display(**int**a)

{

System.***out***.println("Method Overloaded with an Integer Argument");

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

}

**void** Display()

{

System.***out***.println("Displaying Trash Details");

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

}

}

**class** Spam **extends** Gmail

{

// Method OverRiding

**void** Display()

{

System.***out***.println("Displaying Spam Details");

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

}

}

**class**MyClass

{

**staticvoid** Test(Gmail g1)

{

g1.Display();

}

**staticvoid** Test(**int**a)

{

System.***out***.println("Method of Test Overloaded with an Integer Argument");

}

}

**publicclass**MethodOverload {

**publicstaticvoid** main(String[] args) {

System.***out***.println("The Program Starts");

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

MyClass.*Test*(**new** Inbox());

MyClass.*Test*(**new** Trash());

MyClass.*Test*(**new** Spam());

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

System.***out***.println("The Program Ends");

}

}

**Output:**

The Program Starts

Displaying Inbox Details

Displaying Trash Details

Displaying Spam Details

The Program Ends

**implementing interface**

public class RectanglePlus

implements Relatable {

public int width = 0;

public int height = 0;

public Point origin;

// four constructors

public RectanglePlus() {

origin = new Point(0, 0);

}

public RectanglePlus(Point p) {

origin = p;

}

public RectanglePlus(int w, int h) {

origin = new Point(0, 0);

width = w;

height = h;

}

public RectanglePlus(Point p, int w, int h) {

origin = p;

width = w;

height = h;

}

// a method for moving the rectangle

public void move(int x, int y) {

origin.x = x;

origin.y = y;

}

// a method for computing

// the area of the rectangle

public intgetArea() {

return width \* height;

}

// a method required to implement

// the Relatable interface

public intisLargerThan(Relatable other) {

**RectanglePlusotherRect**

**= (RectanglePlus)other;**

if (this.getArea() <otherRect.getArea())

return -1;

else if (this.getArea() >otherRect.getArea())

return 1;

else

return 0;

}

}

**How to do inheritance in java (using extend keyword)**

public class AccessDemo1

{

// Public Access Level

public void testn1()

{

System.out.println("Invoking the public Testn1 Method");

System.out.println();

}

// Private Access Level

private void testn2()

{

System.out.println("Invoking the Private Testn2 Method");

System.out.println();

}

// Protected Access Level

protected void testn3()

{

System.out.println("Invoking the Protected Testn3 Method");

System.out.println();

}

// Default Access Level

void testn4()

{

System.out.println("Invoking the Default Testn4 Method");

System.out.println();

}

}

2. This Program in the Different Package

package com.JavaTraining2;

// importing the Class of the Other Package into the Current Package

import com.javaTraining.AccessDemo1;

// Achieving Inheritance for the Class present in other Package

public class AccessDemo3 extends AccessDemo1

{

public static void main(String[] args)

{

System.out.println("Program Starts");

System.out.println();

AccessDemo3 Ad1 = new AccessDemo3();

// Accessing the Method of Public access level present in other package

Ad1.testn1();

// Accessing the Method of Protected access level present in other package

Ad1.testn3();

System.out.println();

System.out.println("Program Ends");

}

}

**write code to add items to integer, string array**

**publicclass**AddIntIndex {

**publicstaticvoid** main(String[] args) {

**int**[] series = **newint**[0];

**int**x = 5;

series = *addInt*(series, x);

//print out the array with commas as delimiters

System.***out***.print("New series: ");

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

**if** (i == series.length - 1){

System.***out***.println(series[i]);

}

**else**{

System.***out***.print(series[i] + ", ");

}

}

}

**publicstaticint**[] addInt(**int** [] series, **int**newInt){

//create a new array with extra index

**int**[] newSeries = **newint**[series.length + 1];

//copy the integers from series to newSeries

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

newSeries[i] = series[i];

}

//add the new integer to the last index

newSeries[newSeries.length - 1] = newInt;

**return**newSeries;

}

}

Output: 5

**package**my\_training;

**publicclass**StringArray {

**publicstaticvoid** main(String args[]){

/\*

\* Java String array can be created in below given ways.

\*/

/\*

\* Declare and initialize String array in single statement as given below.

\* This method is particularly useful when we are dealing with very small size array.

\*/

String[] myFirstStringArray = **new** String[]{"String 1", "String 2", "String 3"};

/\*

\* Declaration and assignment can be done separately as given below.

\*/

//first declare String array

String[] mySecondStringArray = **new** String[3];

//Observe that giving size is mandatory here. While there was no size given in the first method.

//Now Assign individual String array elements

mySecondStringArray[0] = "String 1";

mySecondStringArray[1] = "String 2";

mySecondStringArray[2] = "String 3";

//Note that, like every other arrays, String array starts with index 0 and not index 1.

/\*

\* Retrieve values from String Array: \*

\* String array elements can be retrieved by directly accessing using index. You may also iterate

\* String array using loop.

\*/

//this will retrieve second element of first String array

System.***out***.println(myFirstStringArray[1]);

//iterate the String array using loop

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

System.***out***.println(mySecondStringArray[i]);

}

}

}

Output: String 2

String 1

String 2

String 3

**write code to add items to ArrayList collection**

**package**my\_training;

**import**java.util.ArrayList;

**class**A

{

}

**publicclass**ArrayListAdd

{

**publicstaticvoid** main(String[] args)

{

System.***out***.println(" The Program Starts" );

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

// Creating an Instance of Array List

ArrayListList1 = **new**ArrayList();

// Creating an Instance of Class A

Aa1= **new**A();

// Using Add() Method to add the Elements to List

List1.add(10);

List1.add(500.5);

List1.add(**true**);

List1.add(**false**);

List1.add(**null**);

List1.add(10);

List1.add('A');

List1.add(a1);

List1.add(**new**A());

List1.add("Sunil");

List1.add("Aradhya");

System.***out***.println("The Size of the Given List List1 is "+List1.size());

System.***out***.println("Printing The reference variable of List");

System.***out***.println("-----------------------------------------------------");

System.***out***.println(List1);

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

System.***out***.println("Printing The Elements based on Index using Standard For Loop");

System.***out***.println("-----------------------------------------------------");

**for**(**int**i=0; i<List1.size();i++)

{

System.***out***.println(List1.get(i));

}

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

System.***out***.println(" The Program Ends" );

}

}

Output:

The Program Starts

The Size of the Given List List1 is 11

Printing The reference variable of List

-----------------------------------------------------

[10, 500.5, true, false, null, 10, A, my\_training.A@15db9742, my\_training.A@6d06d69c, Sunil, Aradhya]

Printing The Elements based on Index using Standard For Loop

-----------------------------------------------------

10

500.5

true

false

null

10

A

my\_training.A@15db9742

my\_training.A@6d06d69c

Sunil

Aradhya

The Program Ends

**write code to retrieve items from arraylist (using for each loop**

**package**my\_training;

**import**java.util.ArrayList;

**class**C

{

}

**publicclass**ArrayListRetrieve {

**publicstaticvoid** main(String[] args)

{

System.***out***.println(" The Program Starts" );

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

// Creating 3 Instances of Array List

ArrayListList1 = **new**ArrayList();

ArrayListList2 = **new**ArrayList();

ArrayListList3 = **new**ArrayList();

// Adding Elements for First Array List

List2.add("Sunil");

List2.add("10");

List2.add("45");

// Adding Elements for First Array List

List1.add("Hello");

List1.add("30");

List1.add("100");

// Adding Elements for First Array List

List3.add("Hi");

List3.add("75");

List3.add("200");

System.***out***.println(List1);

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

System.***out***.println(List2);

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

System.***out***.println(List3);

List1.addAll(List2);

List2.addAll(List3);

List3.addAll(List1);

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

System.***out***.println(List1);

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

System.***out***.println(List2);

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

System.***out***.println(List3);

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

System.***out***.println(" The Program Ends" );

}

}

Output:

The Program Starts

[Hello, 30, 100]

[Sunil, 10, 45]

[Hi, 75, 200]

[Hello, 30, 100, Sunil, 10, 45]

[Sunil, 10, 45, Hi, 75, 200]

[Hi, 75, 200, Hello, 30, 100, Sunil, 10, 45]

The Program Ends

**write code to add items HashMap**

**package**my\_training;

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

**publicclass**HashMapDemo {

**publicstaticvoid** main(String args[]) {

// create hash map

HashMapnewmap = **new**HashMap();

// populate hash map

newmap.put(1, "tutorials");

newmap.put(2, "point");

newmap.put(3, "is best");

System.***out***.println("Map value before change: "+ newmap);

// put new values at key 3

String prevvalue=(String)newmap.put(3,"is great");

// check returned previous value

System.***out***.println("Returned previous value: "+ prevvalue);

System.***out***.println("Map value after change: "+ newmap);

}

}

Output:

Map value before change: {1=tutorials, 2=point, 3=is best}

Returned previous value: is best

Map value after change: {1=tutorials, 2=point, 3=is great}

**Write code to retrieve items to hashset**

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

**publicclass** \_HashSet {

**publicstaticvoid** main(String args[])

{

HashSet<String>set = **new**HashSet<String>();

//Adding values to the HashSet

set.add("test1");

set.add("test2");

set.add("test3");

System.***out***.println("Retrieving values from HashSet using Iterator");

*retrieveValuesFromListMethod1*(set);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

System.***out***.println("Retrieving values from HashSet using Enumeration");

*retrieveValuesFromListMethod2*(set);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

}

/\*This method retrieves values from HashSet using Iterator

\*/

**publicstaticvoid** retrieveValuesFromListMethod1(Setset)

{

Iteratoritr = set.iterator();

**while**(itr.hasNext())

{

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

}

}

/\*This method retrieves values from HashSet using Enumeration

\*/

**publicstaticvoid** retrieveValuesFromListMethod2(Setset)

{

Enumeratione = Collections.*enumeration*(set);

**while**(e.hasMoreElements())

{

System.***out***.println(e.nextElement());

}

}

}

**Output:**

Retrieving values from HashSet using Iterator

test2

test3

test1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Retrieving values from HashSet using Enumeration

test2

test3

test1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**what is super and this keywords**

The Class from where the members are inherited is known as Super Class and the class to which the

members are iinherited is called as Sub Class. Super Class is also called as Base Class and Sub Class is also called as Derived Class. Super statement is used to invoke the specific constructor of the Super class in subclass.

This keyword is used to point the instance of the current class.

**can we call parent method from child method?**

Yes, using the inheritance concept. We can also use super keyword.

**can we create object for abstract class?**

If **we will create** an **object** of the **abstract class** and calls the method having no body(as the method is pure virtual) it **will** give an error. That is why **we** cant **create object** of **abstract class**. In short, it is legal to have a public constructor on an **abstract class**.

**can we over ride static methods, final methods?**

The answer is ‘Yes’. We can have two ore more static methods with same name, but differences in input parameters. For example, consider the following Java program.

|  |
| --- |
| // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public static void foo(int a) {          System.out.println("Test.foo(int) called ");      }      public static void main(String args[])      {          Test.foo();          Test.foo(10);      }  } |