Declarations and acess controls

**package** package1;

**public** **class** Example1 {

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

//legal identifiers

**int** $abc;

**int** \_\_\_\_\_\_567;

**int** ab6789;

String foo; //identifiers are case sensitive. foo and Foo are two diff variables

String Foo;

//illeagl identifiers

/\*int #a;

int 6gahjaak;\*/

}

}

**package** package1;

**public** **class** Example2 {

//getters and setters naming convention

**private** **int** count;

**private** **boolean** status;

**public** **int** getCount() {

**return** count;

}

**public** **void** setCount(**int** count) {

**this**.count = count;

}

**public** **boolean** isStatus() {

**return** status;

}

**public** **void** setStatus(**boolean** status) {

**this**.status = status;

}

}

**package** package1;

**public** **class** Example3 {

//default or package level methods

**void** package1DefaultMethod() {

System.*out*.println("Package 1 default method");

}

}

**package** package1;

**public** **class** Example4 {

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

Example3 ex3 = **new** Example3();

ex3.package1DefaultMethod(); // default method can be acessesd in same package

}

}

**package** package1;

//class leave acess modifiers

//only default and public is possible

//public class

**public** **class** Example5 {

}

//default class

**class** Defaultclass {

}

//private class is not possible

/\*private class Privateclass {

}\*/

//protected class is not possible

/\*protected class Protectedclass {

}\*/

//two public class in same file is not possible, only one public class per source file

/\*public class SecondPublicClass {

}\*/

**package** package1;

//non acess modifiers of the class

//final class

**final** **class** FinalClass {

}

//abstarct class

**abstract** **class** AbstractClass {

**abstract** **void** display(); // abstract method will not have body

//abstract methods cannot be static, as they can be acessed through instance object

/\*abstract static void test();\*/

//abstract class can have non abstract methods

**void** add() {

System.*out*.println("Add method");

}

}

//if class has atleast one abstract method then class must be abstract

/\*class SecondAbsract {

abstract void calucalate();

void subtract() {

System.out.println("Subtract method");

}

}\*/

//final class cannot be inherited

/\*class FinalChildClass extends FinalClass {

}\*/

**class** AbstractChildClass **extends** AbstractClass {

//subclass has to override(implement) all abstract methods of Abstract class

@Override

**void** display() {

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

}

}

**class** Test {

//objects creation not possible for abstract class

//abstract class cannot be instantiated

/\*AbstractClass ab = new AbstractClass();\*/

}

//abstract class canot be final

//abstract and final are opposite, never together

//abstarcat = > has to sub class

//final = > sub class is not possible

/\*abstract final class ThirdAbstractClass {

}\*/

**abstract** **class** FourthAbstractClass {

//abstract methods cannot be private

//since abstract methods has to over ride in subclass

//but private willnot allow. So they are dead opposote

/\*abstract private multiply();\*/

}

**package** package1;

**interface** Runnable {

**int** *COUNT* = 1; //only final static allowed, by default they are final ,public ,

//static

**void** add();

**void** div();

//all methods are abstract in interface

//abstract needs objects to be created to acess, not with class name. So static is not allowed

//all methods has to override, so private or protected is not allowed

/\*static div1();

private div2();

protected div3();\*/

}

//interfaces are 100 percent abstract classes

//methods are default = > abstract, public

//variable are default => public, static , final

//above interface is equalent to

/\*interface Runnable {

public static final int COUNT = 1;

public abstract void add();

public abstract void div();

}\*/

// subclass which implements interface has to implement all methods

**public** **class** Example7 **implements** Runnable {

@Override

**public** **void** add() {

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

}

@Override

**public** **void** div() {

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

}

}

// class cannot extend interface

/\*class sample extends Runnable {

}\*/

**interface** wrappable {

}

//interface can extend one or more interface

**interface** Serializable **extends** Runnable,wrappable {

}

//interface cannot implement another interface

/\*interface testable implements Runnable {

}\*/

**package** package1;

//acess modifiers for instance variables and methods

**public** **class** Example8 {

**private** **int** a;

**protected** **static** **int** *b*;

**public** **int** c;

**int** d;

**private** **void** method1() {

System.*out*.println("private method");

}

**protected** **void** method2() {

System.*out*.println("protected method");

}

**public** **void** method3() {

System.*out*.println("public method");

}

**void** method4() {

System.*out*.println("default method");

}

}

**class** sample {

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

Example8 ex8 = **new** Example8();

//private variable cannot be acessed outside class

/\*System.out.println(ex8.a);\*/

System.*out*.println(ex8.*b*);

System.*out*.println(ex8.c);

System.*out*.println(ex8.d);

//private variable cannot be acessed outside class

/\*ex8.method1();\*/

ex8.method2();

ex8.method3();

ex8.method4();

}

}

**package** package1;

**public** **class** Example9 **extends** Example8 {

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

Example8 ex8 = **new** Example8();

//private variable cannot be acessed outside class

/\*System.out.println(ex8.a);\*/

System.*out*.println(ex8.*b*);

System.*out*.println(ex8.c);

System.*out*.println(ex8.d);

//private variable cannot be acessed outside class

/\*ex8.method1();\*/

ex8.method2();

ex8.method3();

ex8.method4();

}

}

**package** package1;

//this

**public** **class** Example10 {

**private** **int** x;

**private** **int** y;

**public** **void** setValues(**int** x, **int** y) {

//shadowing

**this**.x=x;

**this**.y = y;

}

**public** **void** callThisMethod() {

**this**.setValues(8,9); // same class method can be invoked by this

}

}

**package** package1;

**public** **class** Example11 {

//local variables and acess modifiers

//only final

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

**final** **int** COUNT=1;

//other not

/\*private int a =1;

protected int b =1;

public int a =1;\*/

//static, abstract, synchronized, native can be applied only to the methods

/\*static int f = 5;

abstract int g=6;

synchronized int h=7;

native int i = 10;

\*/

}

}

**package** package1;

//final class cannot be subclassed

//final method cannot be overridden

//final variable value cannot be changed

**public** **class** Example12 {

}

**final** **class** FinalClass1 {

}

/\*class ChildFinalClass extends FinalClass {

}\*/

/\*interface Sample5 {

final void method() {

}

}

\*/

**class** Sample6 {

**final** **int** A =5;

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

Sample6 s6 = **new** Sample6();

/\*s6.A = 6;\*/

}

}

**package** package1;

//var-arg

//exactly one and it should be at end

**public** **class** Example13 {

//legal

**public** **void** args1(**int**... values) {

}

**public** **void** args2(**int** x, **int** y, **int**... values) {

}

//illegal

/\*public void args3(int values...) {

}

public void arg4(int...is values, int x) {

}

public void arg5(int... val1, int... val2) {

}

\*/

}

**package** package1;

//array declarations

**public** **class** Example14 {

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

//leagal

**int**[] arra1;

**int** arra2[];

//illeagl

/\*[]int arr3;

int[5] arr4;

\*/

}

}

**package** package1;

**enum** List {*SMALL*, *AVERAGE*, *BIG*}

**enum** List2 {*SMALL*, *AVERAGE*, *BIG*}; //semicolon is optional

**class** Retrival {

List size;

**public** **void** setSize() {

**this**.size = List.*AVERAGE*;

}

}

**enum** Rates {

*LESS*(5),*AVERAGE*(10),*HIGH*(20);

**private** **int** rate;

Rates(**int** rate) {

**this**.rate = rate;

}

**public** **int** getRate() {

**return** **this**.rate;

}

}

**public** **class** Example15 {

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

**for** (Rates rate : Rates.*values*()) {

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

}

}

}

Object orientation

**package** package1;

//polymorphism

//instance variable ha to be private

//methods has to be public

**public** **class** Example1 {

**private** **int** a;

**private** **boolean** flag;

**public** **int** getA() {

**return** a;

}

**public** **void** setA(**int** a) {

**this**.a = a;

}

**public** **boolean** isFlag() {

**return** flag;

}

**public** **void** setFlag(**boolean** flag) {

**this**.flag = flag;

}

}

**package** package1;

//instanceof

**public** **class** Example2 {

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

Example2 ex2 = **new** Example2();

**if**(ex2 **instanceof** Example2) {

System.*out*.println("ex2 is instance of Example2");

}

}

}

**package** package1;

//equals

**public** **class** Example3 {

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

Example3 ex2 = **new** Example3();

Example3 ex3 = **new** Example3();

**if**(ex2.equals(ex3)) {

System.*out*.println("ex2 equals ex3 using equals");

}

**else** **if**(ex2 == ex3) {

System.*out*.println("ex2 equals ex3 using ==");

}

**else** {

System.*out*.println("ex2 and ex3 are not equal");

}

Example3 ex4 = ex2;

**if**(ex2.equals(ex4)) {

System.*out*.println("ex2 equals ex4 using equals");

}

**if**(ex2 == ex4) {

System.*out*.println("ex2 equals ex4 using ==");

}

**else** {

System.*out*.println("ex2 and ex4 are not equal");

}

}

}

**package** package1;

//inheritence = > extends and implements

**public** **class** Example4 {

}

**class** Sample2 {

}

**class** Sample3 **extends** Sample2 {

}

**class** Sample4 {

}

//multiple inheritance is not possible

/\*class Sample5 extends Sample2, Sample4 {

}\*/

//class cannot implement other class

/\*class Sample5 implements Sample4 {

}\*/

**interface** Runnable {

}

**interface** Serilizable {

}

//calss cannot extend interface, it has to implement

/\*class Sample6 extends Runnable {

}\*/

**class** Sample7 **implements** Serilizable, Runnable {

}

//interface cannot implement other interface

//it can only extend one or more interfaces

/\*interface Derivable implemets Runnable {

}\*/

**interface** Testable **extends** Runnable, Serilizable {

}

**package** package1;

**import** java.io.IOException;

//overridden = > achieved thru extends

**public** **class** Example5 {

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

/\*Sample11 s11 = new Sample11();

s11.add();

s11.add(7, 8);

s11.add("a", "b"); // string arguments method not exist in Sample11

s11.add('c','d'); // char are implicitly casted to int. So legal

\*/

/\*Sample12 s12 = new Sample12();

s12.add();

s12.add(6,8); // Sample12 not contains method with int arguments, but parent class has. So legal

s12.add("a","b");

s12.add('c', 'd');

s12.add(12.6, 13.6); //float not possible

\*/

//Supreclass reference can hold sub class instance object

//But Subclass reference cannot hold superclass intance object

/\*Sample12 s13 = new Sample11();\*/

Sample11 s14 = **new** Sample12();

//overridden method calling is always based on instance object(RHS)

//overloaded method calling is always based on reference object(LHS)

//refernce(s14) = instance object(new Sample12();)

s14.add(); //Sample12 objects add method is called

//Even though Sample12 has the method with 2 float arguments,

//During runtime it calls Sample12 method

//But during compile time reference object(Sample11) must contains method wih 2 float arguments)

/\*s14.add("a","b");\*/

}

}

**class** Sample11 {

**public** **void** add() {

System.*out*.println("Sample11 add method");

}

**public** **void** add(**int** a, **int** b) {

System.*out*.println("Sample11 add method with 2 int arguments");

}

**public** **void** div() {

System.*out*.println("Sample11 div method");

}

**final** **public** **void** sub() {

System.*out*.println("Sample11 sub method");

}

**static** **public** **void** mul() {

System.*out*.println("Sample11 mul method");

}

**public** **void** exception() **throws** ArithmeticException {

System.*out*.println("Sample11 art exception");

}

}

**class** Sample12 **extends** Sample11 {

**public** **void** add() {

System.*out*.println("Sample12 add method");

}

//its not overridden methods, since arguments are diff

//Sample12 own method(overloaded method)

**public** **void** add(String a, String b) {

System.*out*.println("Sample12 add method with 2 string arguments");

}

//in overridden method, acess specifiers should not be more restrictive

/\*private void div() {

System.out.println("Sample12 div method");

}\*/

//final method cannot be overridden

/\*public void sub() {

System.out.println("Sample12 sub method");

}\*/

//Static method cannot be overridden

/\*public void mul() {

System.out.println("Sample12 sub method");

}\*/

//overridden is possible with related expression

**public** **void** exception() **throws** ArrayIndexOutOfBoundsException {

System.*out*.println("Sample11 art exception");

}

//Other releated exception are not possible

/\*public void exception() throws IOException {

System.out.println("Sample12 IO exception");

}\*/

}

**package** package1;

//this and super

**public** **class** Example6 {

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

Sample23 s23 = **new** Sample23();

Sample23 s23\_1 = **new** Sample23(1);

Sample24 s24 = **new** Sample24();

}

}

**class** Sample22 {

Sample22() {

System.*out*.println("Sample22 default constructor");

}

**public** Sample22(**int** a) {

System.*out*.println("Sample22 constructor with one argument");

}

}

**class** Sample23 **extends** Sample22 {

Sample23() {

**super**(); //cals parent class default constructor

System.*out*.println("Sample23 default constructor");

}

**public** Sample23(**int** a) {

**super**(5); //calls parent class contructor with int argument

System.*out*.println("Sample23 constructor with one argument");

}

}

**class** Sample24 **extends** Sample22 {

**private** **static** **int** *X* = 5;

Sample24() {

**this**(5); //calls same class constructor with one argument

System.*out*.println("Sample24 default constructor");

}

**public** Sample24(**int** a) {

//only static variable can be used

**super**(Sample24.*X*);

System.*out*.println("Sample24 constructor with one argument");

}

}

**package** package1;

**public** **class** Example7 {

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

Sample31 sp31 = **new** Sample31();

sp31.add();

sp31.add(5, 6);

Sample32 sp32 = **new** Sample32();

sp32.add();

sp32.add(5, 6);

sp32.add("a","b");

//overload

//referece type method will get called

Sample31 sp = **new** Sample32();

sp.add();

sp.add(5, 6);

}

}

**class** Sample31 {

**public** **void** add() {

System.*out*.println("Sample31 add method");

}

**public** **void** add(**int** a, **int** b) {

System.*out*.println("Sample31 add int method");

}

}

**class** Sample32 **extends** Sample31 {

**public** **void** add(String a, String b) {

System.*out*.println("Sample32 add string method");

}

}

**package** package1;

//casting

//...Downcasting => superclass object to subclass(only explicit

//...Upcasting => subclass object to superclass(implicit/Explicit)

// base class object = super class object (downcasting)

//super class object = base class object(upcasting)

**public** **class** Example8 {

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

Sample41 [] sp41 = {**new** Sample41(), **new** Sample42() };

**for** (Sample41 s41 : sp41) {

**if**(s41 **instanceof** Sample42) {

//error, casting needed

// s41.div();

//downcasting

Sample42 ss42 = (Sample42)s41;

ss42.div();

}

}

Sample42 sp42 = **new** Sample42();

Sample41 sp41\_1 = (Sample41) sp42; //upcasting

sp41\_1.add();

}

}

**class** Sample41 {

**public** **void** add() {

System.*out*.println("Sample41 add method");

}

}

**class** Sample42 **extends** Sample41 {

**public** **void** add() {

System.*out*.println("Sample42 add method");

}

**public** **void** div() {

System.*out*.println("Sample42 div method");

}

}

**package** package1;

//interface and abstract class

**public** **class** Example9 {

}

**interface** Calcalatable {

**public** **void** add();

**public** **void** div();

}

//abstract class decision to implement or not of interface

**abstract** **class** Sample51 **implements** Calcalatable {

}

//if calss extends abstract class(which implements interface)

//then class has to implement all interface methods which was not implemented by abstract class

**class** Sample52 **extends** Sample51 {

@Override

**public** **void** add() {

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

}

@Override

**public** **void** div() {

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

}

}

**package** package1;

//return types

**public** **class** Example10 {

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

Example10 ex10 = **new** Example10();

**int** a = ex10.getIntValue();

System.*out*.println(a);

ex10.display();

}

**private** **void** display() {

**return**; //for void just return or without return is allowed

}

**private** **int** getIntValue() {

**return** 'c'; //implicitly converted to int

}

}

**package** package1;

//constructors

//constructor can be overloaded

//constructor cannot be overridden

**public** **class** Example11 {

Example11() {

System.*out*.println("Example11 constructor");

}

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

//if there are parameter constructor, explicitly we have to write default constructor

/\*Sample61 s61 = new Sample61();\*/

Sample61 s61 = **new** Sample61(6,7);

Sample62 s62 = **new** Sample62();

Sample62 s62\_1 = **new** Sample62(7);

//private constructor, not allowed outside class

// Sample63 sp6 = new Sample63();

Sample63 sp6\_1 = **new** Sample63(5);

Sample63 sp6\_2 = **new** Sample63(5,6);

Sample63 sp6\_3 = **new** Sample63(5,6,7);

Example11 ex11 = **new** Example11();

}

}

**class** Sample61 {

Sample61(**int** a, **int** b) {

System.*out*.println("Sample61 argument constructor");

}

}

**class** Sample62 {

Sample62() {

System.*out*.println("Sample62 default constructor");

}

Sample62(**int** a) {

System.*out*.println("Sample62 argument constructor");

}

}

**class** Sample63 {

**private** Sample63() {

System.*out*.println("Sample63 private constructor");

}

**protected** Sample63(**int** a) {

System.*out*.println("Sample63 protected constructor");

}

**public** Sample63(**int** a, **int** b) {

System.*out*.println("Sample63 public constructor");

}

Sample63(**int** a, **int** b, **int** c) {

System.*out*.println("Sample63 default constructor");

}

**public** **void** add() {

//calling constructor inside method is illegal

/\*Sample63();\*/

System.*out*.println("add method");

}

}

Assignments

**package** package1;

//objects and its instance variable resides in heap

//methods and local variable resides in stack

**public** **class** Example1 {

**private** String a; //heap

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

Example1 ex1 = **new** Example1(); //heap

**int** a = 5; //stack

}

}

**package** package1;

//literals

**public** **class** Example2 {

**private** String title;

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

**int** a = 500;

**int** z = 07; //octal

**int** y = 0Xabc8;

**int** x = 0xABCDF;

**long** b = 1000L;

**long** c = 3000l;

**long** d = (**long**) 1000;

**float** f = 3.14f;

**float** g = 3.14F;

**float** h = (**float**)3.14;

**boolean** i = **true**;

**boolean** j = **false**;

/\*boolean k = 0; //illeaga\*/

/\*boolean l = TRUE; //illeagal \*/

**char** m = 'd';

**byte** n = 20;

**short** o = 10;

//assignments

//implicit casting

**byte** bb = 0;

**short** cc = 0;

**int** aa;

**float** dd = 0.0f;

**double** ff;

aa = bb;

aa = cc;

cc = bb;

ff = dd;

//explicit casting

bb = (**byte**)aa;

dd= (**float**) ff;

**byte** abc = 5;

**byte** def = 6;

**int** gh = abc+def; //two smaller are operated to int

**byte** ghj = (**byte**) (abc+def); // explicit casting

//shorthand operators performs implicit cast

**byte** hij = 6;

**byte** jkl = 8;

jkl += hij;

//primitive value references, change in one will not affects other

**int** num1 = 5;

System.*out*.println("before "+num1);

**int** num2 = num1;

num2 = num2+5;

System.*out*.println("after "+num1);

//object reference, change in one will affect other also.

Example2 ex1 = **new** Example2();

ex1.setTitle("hello");

System.*out*.println("Before "+ex1.getTitle());

Example2 ex2 = **new** Example2();

ex2 = ex1;

ex2.setTitle("hi");

System.*out*.println("After "+ex1.getTitle());

//string objects are exceptional, changes will not affect

String str1 = **new** String("hi");

System.*out*.println("before "+str1);

String str2 = str1;

str2 = "hello";

System.*out*.println("after "+str1);

}

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String string) {

**this**.title = string;

}

}

**package** package1;

//scope

//static init blocks loads when class is loaded and exist as long as class exist in jvm

//called only once when class is loaded first time

//instance init blocks loads whenever new instance is created

//local variables exists as long as method

**public** **class** Example3 {

**static** {System.*out*.println("Fist static init blocks") ;}

Example3() {

System.*out*.println("default constructor");

}

{ System.*out*.println("Fist instnce init block"); }

Example3(String test) {

System.*out*.println("argument constructor");

}

**static** {System.*out*.println("Second static init blocks") ;}

{ System.*out*.println("Second instnce init block"); }

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

**new** Example3();

**new** Example3("hi");

}

}

**package** package1;

//compiler provides default values for instance variables

//local variables has to explicitly initialize before they use

//arrays , the contents get default value depends on primitive type

**class** Sample {

}

**public** **class** Example4 **extends** Sample {

**private** **int** a; //dafault value 0

**private** String c; //dafault value null

**private** **float** d; //dafault value 0.0

**private** Sample sc; //dafault value null

**private** **boolean** flag; //default value false

**private** **char** df; //default value \u00000 => blank it prints

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

Example4 ex4 = **new** Example4();

System.*out*.println(ex4.a);

System.*out*.println(ex4.c);

System.*out*.println(ex4.d);

System.*out*.println(ex4.sc);

System.*out*.println(ex4.flag);

System.*out*.println(ex4.df);

**int** a;

//local variables has to intialize before they use

/\*a = a+5;\*/

**int** abc[] = **new** **int**[5];

**for** (**int** i : abc) {

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

}

}

}

**package** package1;

//shadowing

**public** **class** Example5 {

**static** **int** *size* = 7;

**public** **static** **void** setSize(**int** size) {

size = size+20;

System.*out*.println(size);

}

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

System.*out*.println("before "+Example5.*size*);

Example5.*setSize*(50);

System.*out*.println("after "+Example5.*size*);

}

}

**package** package1;

//diff ways of arrays declaration

**public** **class** Example6 {

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

**int** a[];

a = **new** **int**[2];

a[0] = 1;

a[1] = 2;

/\*a[2] = 3; //runtime outofbound exception

\*/

**int** b[] = **new** **int**[2];

b[0] = 1;

b[1] = 2;

**int** c[] = **new** **int**[] {1,2};

**int** e[] = {5, 6}; //without new

//illegal

/\*int d[8]; //size in declaration

int d[] = new int[2] {1,2}; //size int[2]

int f[] = new int[]; //without size

\*/

String [][]abc; //two dimensional array

String []cdf[]; //two dimensional array

Parent[] p = {**new** Parent(),**new** Child1(),**new** Child2()};

/\*Child2[] ch2 = {new Parent()}; //superclass to subclass is not possible

\*/

Child1[] ch1 = {**null**};

}

}

**class** Parent {

}

**class** Child1 **extends** Parent {

}

**class** Child2 **extends** Parent {

}

**package** package1;

//wrapper classes

**public** **class** Example7 {

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

Integer a = **new** Integer(6);

Integer b = **new** Integer("6");

Float c = **new** Float(3.14f);

Float d = **new** Float("3.14f");

Character e = **new** Character('c');

Boolean f = **new** Boolean(**true**);

Boolean g = **new** Boolean("true");

//wrapper methods

//valueof ==> to convert wrapper to primitive

Integer h = Integer.*valueOf*("101010", 2);

//xxxvalue ==> one wrapper to other wrapper

Byte i = a.byteValue();

//parseXXX => returns named primitive

Integer j = Integer.*parseInt*("10101",2);

//tostring

String s = Double.*toString*(3.143);

//toXXXstring

String s4 = Long.*toOctalString*(254);

}

}

**package** package1;

//overloading with

// widening

// boxing

// varargs

//widening of smallest primitive will be done to nearest primitive

//primitive and wrapper, priority is to primitive, calls primitive

//varargs and primitive/wrapper, priority is to primitive/wrapper

//priority = primitive -> wrapperP->var args

**public** **class** Example8 {

**public** **void** add(**int** a){

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

}

**public** **void** add(**double** a){

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

}

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

Example8 ex8 = **new** Example8();

**byte** a = 5;

**short** b = 7;

**float** d = 3.14f;

ex8.add(a);

ex8.add(b);

ex8.add(d);

Sample11 s11 = **new** Sample11();

s11.add1(5);

**byte** c = 5;

s11.add1(c);

Sample12 s12 = **new** Sample12();

s12.add2(5, 6);

}

}

**class** Sample11 {

**public** **void** add1(**int** a){

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

}

**public** **void** add1(Integer a){

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

}

**public** **void** add1(Byte a){

System.*out*.println("wrapper byte");

}

}

**class** Sample12 {

**public** **void** add2(**int** a, **int** b){

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

}

**public** **void** add2(**int**... a){

System.*out*.println("var int");

}

}

**package** package1;

**import** java.util.Date;

//garbage collection => only on heap = > only for objects and its instance variable

//finalize will execute before garbage collection

//so if we dnt want objects to delete, then we can put those objects in finalize

**public** **class** Example9 {

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

Sample22 s22 = **new** Sample22();

s22.setTitle("hi");

Sample22 s23 = **new** Sample22();

s23.setTitle("hello");

s22 = s23; //on heap object with hi will mark for garbage collection, as it lost the s22 reference

s23.modifyTiltle(s23.getTitle());

s23 = **null**; //now s23 is marked for garbage collection

Runtime rt = Runtime.*getRuntime*();

System.*out*.println("Total memory "+rt.totalMemory());

System.*out*.println("Available memory "+rt.freeMemory());

Date da;

**for**(**int** i=0;i<100;i++) {

da = **new** Date();

da = **null**;

}

System.*out*.println("Available memory after null "+rt.freeMemory());

System.*gc*();

System.*out*.println("Available memory after system gc "+rt.freeMemory());

}

}

**class** Sample22 {

**private** String title;

**public** String getTitle() {

**return** title;

}

**public** **void** modifyTiltle(String title2) {

StringBuffer sb = **new** StringBuffer("hhhhh"); // sb is marked has garbage collection after method is executed

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

}