CORE -JAVA

**Core Java Demo class 1:**

-3- Topics

1. **class-keyword**: it is a factory which generates objects.

public class Ex1 {

}

1. **new-Keyword**: it will tell class to create to an object and once the object is created, new keyword gets the address of the object and will be stored in reference variable.

public class Ex2 {

public static void main (String[] args){

Ex2 ex= new Ex2(); // object is created here.

}

}

1. **garbage-collection**: - it helps us to manage the memory in an efficient manner, i.e., it regularly removes the unused objects from the memory, hence avoids memory overflow.

public class Ex3 {

//refer notes

}

**Core Java Demo class 2:**

-4- Topics

1. **Static Variables and Non Static Variables:**

public class Ex1 {

int x = 20;// non static stores only in object without object we can't asses it.

static int y = 50;// static member stores only in class common memory and we call it by using class name. variable name.

public static void main(String[] args) {

Ex1 ex = new Ex1();// object creation

System.out.println(Ex1.y);//reading static member

System.out.println(ex.x);//reading non-static member

}

}

1. **Example:**

public class Ex2 {

int x = 10;// non static main copy.

public static void main(String[] args) {

Ex2 ex = new Ex2();// object 1 1st x copy and change value but it will not effect the main copy.

ex.x = 30;// changing copy of x value.

Ex2 ex1 = new Ex2();// object 2 2nd x copy it will be same as 10 it wont be 30.

System.out.println(ex.x);// printing value of 1st obj value which we changed.

System.out.println(ex1.x);// printing value of 2nd obj copy.

}

}

1. **Example:**

public class Ex2 {

int x = 10;// non static main copy.

public static void main(String[] args) {

Ex2 ex = new Ex2();// object 1 1st x copy and change value but it will not effect the main copy.

ex.x = 30;// changing copy of x value.

Ex2 ex1 = new Ex2();// object 2 2nd x copy it will be same as 10 it wont be 30.

System.out.println(ex.x);// printing value of 1st obj value which we changed.

System.out.println(ex1.x);// printing value of 2nd obj copy.

}

}

1. **Heap & Stack:**

* Heap: In java every Objects which gets created is stored in heap.
* Stack: It helps us to maintain the programs execution flow.

**Core Java Demo class 3:**

-4- Programs

1. **Example:**

//Practice with stack and heap concept

//Methods(basics):

public class Ex1 {

public static void main(String[] args) {

Ex1 ex = new Ex1();

ex.test();

}

public void test() {//non static methods

System.out.println(100);

}

}

//Stack: program execution flow is maintained in stack.

//Heap: all the objects are created in heap memory.

1. **Example:**

// write stack and heap model to understand "LIFO"

public class Ex2 {

public static void main(String[] args) {// step 1

System.out.println(10);// step 2

Ex2 ex = new Ex2();// step 3

ex.test();// step 4//step 9

}// step 10

public void test() {// step 5// non static methods

//it will always be stored in object.

Ex2 ex1 = new Ex2();// step 6

System.out.println(ex1);// step 7

}// step 8

}

1. **Example:**

//write stack and heap model to understand "LIFO"

public class Ex3 {

public static void main(String[] args) {//step 1

Ex3.test();//calling by class name so it means it is static method

//step 2//step 6

}//step 7 stop

public static void test() {//static method//step 3

System.out.println(100);//step 4

}//step 5

}

1. **Example:**

//write stack and heap model to understand "LIFO"

public class Ex4 {

public static void main(String[] args) {// step 1

Ex4 ex = new Ex4();// step 2

ex.test1();// step 3//step 11

}// step 12 stop

public void test1() {// step 4//non static method

Ex4.test2();// step 5//step 9

}// step 10

public static void test2() {// step 6//static method as methoed is

// called with class name

System.out.println(100);// step 7

}// step 8

}

**Core Java Demo class 4:**

-1- Installation

1. Eclipse installation and shortcuts.

**Core Java class 5:**

-3- Topics, 11- Programs

1. **LOCAL VARIABLES:**
   1. Example:

//\*Local Variable: it should be created inside a method and can be used only within created method , outside created method these variables are not accessible hence below program throws an error.

public class Ex1 {

public static void main(String[] args) {

int age =30;

System.out.println(age);

Ex1 ex = new Ex1();

ex.test1();

}

public void test1() {

//System.out.println(age);//error

}

}

* 1. Example:

//correct way of writing program

public class Ex2 {

public static void main(String[] args) {

int age =30;

System.out.println(age);//only can be access within the method itself.

}

}

* 1. Example:

// how we can access local variables.

public class Ex3 {

public static void main(String[] args) {

Ex3 ex3 = new Ex3();

ex3.test1();

// System.out.println(age); we cannot access here since the variable is created in test1()//method.//error

}

public void test1() {

int age =30;

System.out.println(age);//we can access here because age is created inside test1 method.

}

}

* 1. Example:

//Without inistialising local variable , if used then it iwll give an error

public class Ex4 {

public static void main(String[] args) {

@SuppressWarnings("unused")

int weight;

//System.out.println(weight);//error//we have to initialize local variable.// even we cannot store null value in it or assign null value to it, it will give an error.

}

}

* 1. Example:

// while writing big numbers if it is difficult to read such as example (5000000) we can't put "," here so instead we can put "\_" here to not give error as shown in bellow example.

public class Ex5 {

public static void main(String[] args) {

// int salary=50,00,000;// we can't put commas.

int salary = 50\_00\_000;// we can write it this way to read it easily.

System.out.println(salary);

}

}

1. **Static Variables:**
   1. Example:

/\*Static Variables:\*These variables are created inside a class but outside method.

\*We use static keyword to create these variable.

\*It can be access anywhere in the program.

\*It is similar to Global Variable;

\*It is not Mandatory to initialize static variable, then it prints output Based on data type of static variable it will stored,for example: int-0, float-0.0, string-null. as shown in "Ex9.java"

\*\*/

public class Ex6 {

static int age = 30;// static variable so that it can be access anywhere in the program. It is

// similar to Global Variable.

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

System.out.println(Ex6.age);

Ex6.test();// calling static method.

}

public static void test() {// static method

System.out.println(Ex6.age);

Ex6 ex = new Ex6();

ex.test1();// calling non static method.

}

public void test1() {// non static method.

System.out.println(Ex6.age);// static members can be access anywhere.

}

}

* 1. Example:

// There are three ways we can accept static variables, as shown in below program.

public class Ex7 {

static int weight = 40;

@SuppressWarnings("static-access")

public static void main(String[] args) {

System.out.println(Ex7.weight);// accepted to access static variables.

System.out.println(weight);// even this is accepted to access static variables.

//never do this//

Ex7 ex = new Ex7();

System.out.println(ex.weight);// it will give warning but no error , compiler will convert this ex.weight ->

// Ex7.weight// this is a very bad practice to access static variables.

}

}

* 1. Example:

/\*

\* static variable name and local variable names can be same but when we should know the imp thing which is shown below. \*/

public class Ex8 {

static int age = 30;// static variable

public static void main(String[] args) {

int age = 50; // even local variable name can be same as static variable.

System.out.println(age);// IMPORTANAT only local variable is given preference if we access like this

// in case if we need to access the static variable in particular we have to do

// the below thing shown

System.out.println(Ex8.age);//accessing static variable in particular, when local variable has same name as that of the static variable.

}

}

* 1. Example:

// refer Ex6 for theory.

public class Ex9 {

static int age;// Integer DT ->0

static float weight;// float DT ->0.0

static String name;// String Data Type -> null

static char ch;// Character DT -> empty space

static boolean bool; // Boolean DT -> false

public static void main(String[] args) {

System.out.println(Ex9.age);

System.out.println(Ex9.weight);

System.out.println(Ex9.name);

System.out.println(Ex9.ch);

System.out.println(Ex9.bool);

}

}

* 1. Example:

/\*Conventions:

\* 1) Blue color variable-> non static (non-italic)

\* 2) Blue color variable-> static ( italic)

\* 3) Brown color variable-> local Variable.

\*

\* 4) Class -> always write the starting letter with upper-case letter followed lower-case, when any second word it should again start with upper-case without any spaces followed by lower-case [Bank, BankAccount].

\*

\* 5) variable -> always starts with lower-case and continue same, if two words 2nd word should be written in upper-case.[age,yourAge,thisIsHerName]--> special character is allowed is only [$,number,\_] -->we can write "$ or \_" anywhere beginning middle or at end or we can also just name it $ and \_ it will not give error,IMPORTANT thing is variable name can never begin with number but it can be in middle or at the end.

\*

\* 6) Method -> same like variables with "()" as suffix.

\*

\* 7) Keywords -> all the keywords starts with lower-case

\*/

public class Ex10 {

}

1. **Non-Static Variables:**

/\*non-static variable:1)non static variables are created outside methods and inside class without static keyword.

\* 2) These variables can never be accessed without creating object

\* 3) It is not mandatory to initialize non static variable , because depending on data-type default value gets stored in it, They are also called as instance variable.

\*/

public class Ex11 {

int age1;// Integer DT ->0

float weight1;// float DT ->0.0

String name1;// String Data Type -> null

char ch1;// Character DT -> empty space

boolean bool1; // Boolean DT -> false

public static void main(String[] args) {

Ex11 ex = new Ex11();

System.out.println(ex.weight1);

System.out.println(ex.age1);

System.out.println(ex.name1);

System.out.println(ex.ch1);

System.out.println(ex.bool1);

}

}

**Core Java class 6:**

-4- Topics

1. **Data Types:**

//DataTypes-->memory size --> Default Value

public class Ex1 {

static byte b; //-> 1 Byte -> 0

static short s; //-> 2 Byte -> 0

static int i; //-> 4 Byte -> 0

static float f; //-> 4 Byte -> 0.0

static long l; //-> 8 Byte -> 0

static double d; //-> 8 Byte -> 0.0

static char c; //-> 2 Byte -> empty space

static boolean bool; //-> N/A -> false

static String str; //-> N/A -> null

public static void main(String[] args) {

System.out.println(b);

System.out.println(s);

System.out.println(i);

System.out.println(l);

System.out.println(f);

System.out.println(d);

System.out.println(c);

System.out.println(bool);

System.out.println(str);

}

}

1. Example:

//long: when we exceed the integer range we terminate with l or else l as suffix is not needed

//float: terminated with f

//char: should always be stored in ''.

public class Ex2 {

public static void main(String[] args) {

String s= "hello";

char c='a';

int a= 20;

long l = 6361317502l;// if it exceeds the int range then only we have to end it with l.

long l2 =10;

float f=20.54f;

double d=287.98;

boolean b = true;

System.out.println(s);

System.out.println(c);

System.out.println(a);

System.out.println(l);

System.out.println(l2);

System.out.println(f);

System.out.println(d);

System.out.println(b);

}

}

1. **Var -Type:**

// var-type:1) this was introduced in version 10 of java,

/\* 2)var-type can store any kind of value in it as shown in below example

\*

\* 3) var-type cannot be static and non static variable, it can be only local variable.

\* 4) var-type cannot be method argument,hence the below program throws error.

\*

\* 5) var can have a variable name as a var itslf as it is atype not an data type.

\*

\* 6) "-------"-------"

\*

\* 7) even when we write any keyword and its name is String and if we print output also it will not give error.

\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*2)

\* public class Ex3{

\* psvm(){

\* var x1 = "pankaj";

\* var x3 =10;

\* var x7 =30.98;

\* var x8 = 'a';

\* var a1= new Ex3();

\* sop(x1);

\* sop(x3);

\* sop(x7);

\* sop(x8);

\* sop(a1);

\* }

\* }

\* \*/

/\*3)

\* public class Ex3{

\* static var x2 =10;

\* var x3 =20;

\* psvm(){

\* var x1="pankaj"

\* }

\* }

\* \*/

/\*4)

\* public class Ex3{

\* psvm(){

\* Ex3 ex = new Ex3();

\* ex.test(10);

\* }

\* p v test(var x){// error

\* sop(x)

\* }

\* \*/

/\*5)

\* class Ex3 {

\* psvm(){

\* var var = 10;//it wont give error where as if it is a keyword it wont accept this syntax so we consider var as not a keyword.

//int int =20;//error

\* sop(var);

\* }

\* }

\* \*/

/\*6)

\* class Ex3 {

\* psvm(){

\* var String = 10;

\* sop(String)

\* }

\* }

\* \*/

/\*7)

\* class Ex3{

\* psvm(){

\* int String = 30;

\* sop(String); //no error it will print special case

\* }

\* }

\* \*/

public class Ex3 {

}

1. **Reference Variable:**

/\*reference variable : 1) can store objects address or null value

\* 2) if a reference variable is created in a method then it is a local variable.

\* 3) if you make reference variable static it has global access.

\* 4) Data type of reference variable is class name.

\* \*/

public class Ex4 {

static Ex4 ex;

public static void main(String[] args) {

Ex4 ex = new Ex4();//

System.out.println(ex);

ex.test();

}

public void test() {

System.out.println(ex);

}

}

**Core Java class 7:**

-2- Topics, 11- Programs

1. Methods:

//more on methods

public class Ex1 {

public static void main(String[] args) {

Ex1 ex = new Ex1();

ex.test(10, 10.3f, "test", true, 'a');

}

public void test(int i, float f, String string, boolean b, char c) {

System.out.println(i);

System.out.println(f);

System.out.println(string);

System.out.println(b);

System.out.println(c);

}

}

1. Example:

//special way to pass arguments into method.

public class Ex2 {

public static void main(String[] args) {

Ex2 ex = new Ex2();

ex.test(10, 20, 30, 40);

}

private void test(int... i) {//Array concept

System.out.println(i[0]);

System.out.println(i[1]);

System.out.println(i[2]);

System.out.println(i[3]);

}

}

1. Example:

/\* void :1)If a method is void then it means it cannot return any value

\* hence the below program throws error.

\* 2) we cannot anything after return keyword.

\* 3) we cannot write anything side to return keyword in void method ,

\* we can just put return keyword terminated with semicolon.

\* 4) we cannot just write return keyword without any value next to it in non void method.

\* \*/

public class Ex3 {

public static void main(String[] args) {

Ex3 ex = new Ex3();

ex.test();

int test2 = ex.test2();// assigning non void member's to local variable.

System.out.println(test2);// printing non void values

}

private int test2() {

return 100;// it will be thrown out and we should assign the call statement to variable.

// System.out.println(20);//error we can't write anything after return

}

private void test() {

// return 100;//error

System.out.println(100);// we can write anything before return keyword but not after.

return;// accepted

// System.out.println(20);//error we can't write anything after return

}

}

1. Example:

public class Ex4 {

public static void main(String[] args) {

Ex4 ex = new Ex4();

String var = ex.test();

System.out.println(var);

}

private String test() {// non void method

return "pankaj";

}

}

1. Example:

// if you write anything after return keyword then that will not run and it will give us unreachable code error, as shown in the below example.

public class Ex5 {

public static void main(String[] args) {

Ex5 ex = new Ex5();

String var = ex.test();// getting value from non void method and stores in local variable for printing.

System.out.println(var);// printing Stored value.

}

private String test() {// non void method

System.out.println(100);// it will run it will not be an unreachable code as it is before return

// keyword.

return "pankaj";// we have to write value after return as it's an non void method or else error.

// System.out.println(100);// error unreachable code

}

}

1. Example:

/\* A)return :

\* 1) return keyword can be used only in void method.

\* 2) its optional.

\* 3) It returns control to method calling statement.

\* 4) We can write return keyword in main method too.

\* 5) We can write return in static method too.

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

\* B) return value:

\* 1) cannot be used in void methods.

\* 2) returns control and value to the calling statement.

\* 3) if not a void method its mandatory to use [return value]

\* \*/

public class Ex6 {// A

public static void main(String[] args) {

Ex6 ex = new Ex6();

ex.test();

return;// it wont give any error

}

private void test() {

return;

}

}

/\*

\* public class Ex6 {//B public static void main(String[] args) { Ex6 ex = new

\* Ex6(); int a=ex.test(); System.out.println(a) return;// it wont give any

\* error }

\* private int test() { return 100; } }

\*/

1. Example:

// Continuation of 6 just some examples for methods.

public class Ex7 {

public static void main(String[] args) {

Ex7 ex = new Ex7();

ex.test();

return;// it wont give any error

}

private void test() {

System.out.println(100);

return;

// System.out.println(100);// error

}

}

1. Constructors:

/\*CONSTRUCTOR:

\* 1) It should have same name as that of Class

\* 2) Every time we create an object a constructor is called.

\*

\*

\* \*/

public class Ex8 {

// constructor start

Ex8() {// constructor should have the same name as that of the class its mandatory.

System.out.println(100);

}

// constructor end

// main method start

public static void main(String[] args) {

new Ex8();// 1st time, every time we create an object constructor is called.

new Ex8();// 2nd time object created so 2 times constructor is called.

}

// main method end

}

1. Example:

//constructor with arguments.

public class Ex9 {

Ex9(int x) {// constructor with argument

System.out.println(x);

}

public static void main(String[] args) {

new Ex9(10);// object created.

}

}

1. Example :

//fake constructor / void fakeism.

public class Ex10 {

void Ex10() {// this is not a constructor its an nonstatic method.

System.out.println(100);

}

public static void main(String[] args) {

new Ex10();// no output as it is not an constructor.

}

}

1. Example:

// continuation of Ex10

// Note: method name and class name can be same and wont give error until it has same arguments if so it will give error, as shown below.

public class Ex11 {

void Ex11() {// method not constructor.

System.out.println(100);

}

public static void main(String[] args) {

Ex11 ex = new Ex11();// object created but it wont call constructor as the above thing is void method

// so never call the constructor with void as prefix.

ex.Ex11();// we write this to access the non-static method.

}

}

**Core Java class 8:**

-2- Topics, 4- Programs

1. Difference Between JRE and JDK:

/\*

\* Difference between jre and jdk:

\*

\* A)jre:java runtime environment.

\* 1)it is used to run the java projects or files.

\* 2)if you want to run only .class files then install jre, customers use this to run programs

\*

\* B)jdk: java development kit

\* 1)it is used to compile and run the program.

\* 2)Developers use this to develop projects.

\*

\* \*/

/\*ASCII only one language but to write in multilanguage we have UNICODE.

\* Note: bin -> all the .class files are present here.

\* Note: src -> all your java files are present.

\* \*/

//uni-code example.

public class Ex1 {

public static void main(String[] args) {

int x = 'a';// English language

System.out.println(x);

int y = 'ए';// Hindi language.

System.out.println(y);

}

}

1. Constructor continuation:

//more on constructor

//

// new keyword will request to class to create an object.

// new keyword will call constructor that is mandatory.

// new keyword will gets object address and stores that in reference variable.

// if we create an object without an argument and explicitly constructor is not created by the programmer then during compilation in .class file empty body constructor gets added as shown in the below example.

public class Ex2 {

/\*Ex2(){

\*\*empty body constructor is created by compiler in .class file automatically\*\*

\* }

\* \*/

public static void main(String[] args) {

new Ex2();// object created an empty body constructor is created by new keyword automatically.

}

}

1. Example:

// if we create an object with a value, if constructor is not created it will give error, creating constructor here is mandatory.

public class Ex3 {

Ex3(int i) {// mandatory creation of constructor with value or else it will give error.

System.out.println(i);

}

public static void main(String[] args) {

new Ex3(100);// object with value// if constructor is not created then it will give error.

}

}

1. Example:

// creating constructor with value is mandatory if we pass value in object or else it will give error.

public class Ex4 {

public static void main(String[] args) {

//Ex4 ex = new Ex4(20);// error

}

}

**Core Java class 9:**

-2- Topics, 8- Programs

1. Constructor overloading

* Example1:

package p1;

/\*Constructor overloading:

\* 1) here we create more than one constructor in same class.

\* 2) create constructors with different number of arguments or different type of argument

\*

\* \*/

public class Ex1 {

Ex1() {// no of arguments=0.

System.out.println(300);

}

Ex1(int i) {// no of arguments = 1

System.out.println(i);

}

Ex1(int i, int y) {// no of arguments = 2

System.out.println(i);

System.out.println(y);

}

@SuppressWarnings("unused")

public static void main(String[] args) {

Ex1 ex = new Ex1();// object creation 1

Ex1 ex1 = new Ex1(10);// object creation 2

Ex1 ex2 = new Ex1(100, 200);// object creation 3

}

}

* Example 2:

package p1;

/\*Constructor overloading.

\* 1)different type of argument

\* \*/

public class Ex2 {

Ex2(int i) {// no of arguments = 1 type = int

System.out.println(i);

}

Ex2(char i) {// no of arguments = 1 type = char

System.out.println(i);

}

Ex2(String i) {// no of arguments = 1 type = string

System.out.println(i);

}

public static void main(String[] args) {

new Ex2(10);// object creation 1

new Ex2('a');// object creation 2

new Ex2("pankaj");// object creation 3

}

}

* Example 3:

package p1;

/\*Constructor overloading

\* 3) if those 2 different arguments belong to same data type like int and byte etc.

\* Note: in the below program, it will call int because all numbers by default is treated as int.

\* \*/

public class Ex3 {

Ex3(int i) { // no of arguments = 1 type = int

System.out.println(i);

System.out.println("int");

}

Ex3(byte i) {// no of arguments = 1 type = byte

System.out.println(i);

System.out.println("byte");

}

public static void main(String[] args) {

new Ex3(10);// object creation 1

// \*\*\* to pass byte value we write as new //Ex3((byte)10);

// \*\*\* to pass long value we write as new //Ex3(10L);

// in java if we write any number it will always be treated as int, if we

// explicitly want to pass byte value then we do down casting.

}

}

* Example 4:

package p1;

/\* Note: class name constructor name method name and variable name can be same as shown in the below example.

\* \*/

public class main {

main() {

System.out.println(1000);

}

public static void main(String[] args) {

int main = 10;

main m = new main();

System.out.println(main);

}

}

1. Packages:

* Example 5:

package p1;

/\* Package:

\* 1) packages are nothing but folder, that is to store all data organized way.

\* 2) package names cannot be keywords or java or upper-case.

\* 3) we cannot write anything above package it should be the first word in program

\* 4) packages are folders created in java to store programs in organised manner.

\* \*/

public class Ex5 {

}

* Example 6:

package p1;

import p2.Ex1p2; // we have to write it mandatorily.

import p3.p4.p5.A; // we have to write it mandatorily

/\*CONTINUATION OF Ex5

\*continuation from package p2->Ex1p2-> import p2.Ex1p2;.

\* continuation from package p3.p4.p5->A-> import p3.p4.p5.A;.

\* \*/

public class Ex6 {

@SuppressWarnings("unused")

public static void main(String[] args) {

Ex5 ex = new Ex5(); // object of Ex5 from same package p1, accessing in different class.

Ex1p2 ex1 = new Ex1p2(); // object of Ex1p2 from different package p2, accessing in different class of

// different package.

A a = new A(); // object of A from different package p3.p4.p5, accessing in different class of

// different package.

}

}

* Example 7:

package p2;

public class Ex1p2 {

}

* Example 8:

package p3.p4.p5;

public class A {

}

**Core Java class 10:**

-2- Topics

1. Packages continuation:

package p1;

public class A {

}

package p2;

/\* Packages:

\* 1) helps us to resolve naming convention problem\* \*/

public class B {

public static void main(String[] args) {

p1.A a1 = new p1.A();// 2nd method to import other packages. this is use-full when 2 packages have class of same name we use this type of accessing (packagename.classname)

p3.A a2= new p3.A();

}

}

package p3;

public class A {

}

1. OOPS CONCEPT- inheritance:

* Example 1:

package inheritance;

/\* Inheritance:

\* 1) here we inherit the members of parent class to child class with an intention of reusability.

\*

\* \*/

public class Ex1 {// parent class

int x = 10;

}

package inheritance;

// NOTE: only one class should have public when they are created in same file, and which is public should be name of class, interview question.

/\*Extends: is a keyword which helps us to inherit\*/

public class Ex1\_continuation extends Ex1 {// child class //replicating which are present in parent class to child name.

// it is not moved they are copied.

public static void main(String[] args) {

Ex1\_continuation ex = new Ex1\_continuation();// object stores all the non-static variables from both parent and

// class name

System.out.println(ex.x);

}

}

* Example2:

package inheritance;

public class Ex2 {

int x= 10;

public void test() {

System.out.println(100);

}

}

package inheritance;

public class Ex2\_continuation extends Ex2 {

public static void main(String[] args) {

Ex2\_continuation ex = new Ex2\_continuation(); //here method and variables gets stored from parent class

ex.test();

System.err.println(ex.x);

}

}

* Example3:

package inheritance;

// in the below example A and B are non-subclasses and there is no inheritance happening between them

public class Ex3 {

int x = 10;

}

package inheritance;

// no inheritance

public class Ex3\_c {

public static void main(String[] args) {

Ex3 ex = new Ex3();

System.out.println(ex.x);

}

}

* Example4:

package inheritance;

// we cannot do multiple inheritance in java.

public class Ex4 {

public void test1() {

System.out.println(100);

}

}

package inheritance;

// continuation of Ex4

public class Ex4\_1 extends Ex4{

public void test2() { // test1() inherited

System.out.println(200);

}

}

package inheritance;

//object-oriented programming is applicable only in non-static.

public class Ex4\_2 extends Ex4\_1 {

public void test3() {// test1() test2() inherited

System.out.println(300);

}

public static void main(String[] args) {

Ex4\_2 ex = new Ex4\_2();

ex.test1();

ex.test2();

ex.test3();

}

}

* Example5:

package inheritance;

//multiple inheritance in java is not possible/supported in class level, as it is shown below.

public class Ex5 {

}

package inheritance;

public class Ex5\_1 {

}

package inheritance;

public class Ex5\_2 /\* extends Ex5 , EX5\_1\*/{

// we cannot inherit multiple classes.

}

* Example6:

/\* we cannot inheritance without importing package where other program is present in other class and other package. \*/

package inheritance;

public class Ex6 {

int x = 10;

}

package inheritance\_1;

import inheritance.Ex6;//importing package is mandatory.

public class Ex6\_1 extends Ex6{

public static void main(String[] args) {

Ex6\_1 ex = new Ex6\_1();

System.out.println(ex.x);

}

}

1. OOPS CONCEPT- polymorphism:

* Example7:

package polymorphism;

/\* polymorphism

\* 1) it can only be applied on methods and not variables.

\* 2) here we develop a feature such that it can take more than one form.

\* \*/

/\*OVERRIDING

\* 1) inheritance is mandatory to do this overriding.

\* 2) here we inherit a method from parent class and then we modify the logic of inherited method in child class by once again creating a method with a same signature in the child class

\* \*/

public class Ex1 {

public void test() {

System.out.println(100);

}

}

package polymorphism;

//overriding

public class Ex1\_1 extends Ex1 {

public void test() { // here overriding is happening. here the copy is not taking place, here test method is taking 2 forms that is called as polymorphism

System.out.println(200);

}

public static void main(String[] args) {

Ex1\_1 ex = new Ex1\_1();

ex.test();

}

}

* Example8:

package polymorphism;

/\* overriding second example.\*/

public class Ex2 {

public void test() {

System.out.println(100);

}

}

package polymorphism;

/\* overriding happens here the same method from class Ex2 is getting logic modified. \*/

public class Ex2\_2 extends Ex2 {

public void test() {// overriding happens here the same method from class Ex2 is getting logic

// modified.

System.out.println(500);

}

public void test1() {

System.out.println(100);

}

public static void main(String[] args) {

Ex2\_2 ex = new Ex2\_2();

ex.test();

ex.test1();

}

}

**Core Java class 11:**

-3- Topics,11- Programs.

1. Overriding continuation:

* Example1:

package p1;

public class Ex1 {

public void test1(){

System.out.println(100);

}

}

package p1;

//@ override annotation helps us to check weather overriding is happening or not. if overriding is not happening then we would get an error as shown in the below example.

// overriding

public class Ex1\_1 extends Ex1 {

//@Override - if we dont write this it will give an error.

public void teset1(){// if it matches it won’t give error, here it is not matching so it is giving error.

System.out.println(100);

}

public static void main(String[] args) {

Ex1\_1 ex = new Ex1\_1();

ex.test1();

}

}

* Example 2:

package p1;

// it should have same return type too like in example given below.

public class Ex2 {

public int test1() {

return 100;

}

}

package p1;

public class Ex2\_1 extends Ex2 {

// public void test1(){

// System.out.println(100);

// }

@Override

public int test1() {

return 200;

}

public static void main(String[] args) {

Ex2\_1 ex = new Ex2\_1();

System.out.println(ex.test1());

}

}

* Example3:

package p1;

// Practical Example of overriding.

public class GoldAccount {

public void onlineBanking() {

System.out.println("yes");// this method remains as it is because its common bw 2 accounts so we dont have

// to change the logic.

}

public void chqBooks() {

System.out.println("2/year");// this method doesnot remains same values inside or in general logic will be

// changed by overriding.

}

public void rateOfIntrest() {

System.out.println("nill");// this method doesnot remains same values inside or in general logic will be

// changed by overriding.

}

}

package p1;

public class PlatinumAccount extends GoldAccount {

@Override // checks weather method is going to override,

// so that typo is avoided

public void chqBooks() {// overriden method

System.out.println("unlimitted"); // overrideen and logic is been alterd.

}

@Override // checks weather method is going to override,

// so that typo is avoided

public void rateOfIntrest() { // overriden method

System.out.println("7% pa");// overrideen and logic is been alterd.

}

public static void main(String[] args) {

PlatinumAccount p = new PlatinumAccount();// object of child class

p.chqBooks(); // no inheritance

p.onlineBanking(); // inheritance is happening

p.rateOfIntrest();// no inheritance

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

GoldAccount g = new GoldAccount(); // object of parent class

g.chqBooks();// no inheritance

g.onlineBanking();// inheritance is happening

g.rateOfIntrest();// no inheritance

}

}

1. Overloading:

* Example 4:

package p1;

/\*overloading:

\* 1) creating more than one method with the same name in same class provided they have diff number of argument or diff type of argument is called as method overloading

\* \*/

public class Ex4 {

public void test() {

System.out.println(100);

}

public void test(int i) {

System.out.println(i);

}

public void test(int i, int y) {

System.out.println(i);

System.out.println(y);

}

public void test(byte b) {

System.out.println(b);

}

}

* Example 5:

package p1;

// example email to understand overloading.

public class Ex5 {

public void sendEmail(String tcid) {

System.out.println("transactional emailer");

}

public void sendEmail() {

System.out.println("promotional");

}

public static void main(String[] args) {

Ex5 ex = new Ex5();

ex.sendEmail();// will call promotional mail.

}

}

1. This Keyword:

* Example 6:

package p1;

/\* this keyword:

\* 1) it’s a special reference variable created automatically to store objects address.

\*

\* \*/

public class Ex6 {

public static void main(String[] args) {

Ex6 ex = new Ex6();// object created one adress gets stored in rv by us compiler assigns obj adress

// to this keyword.

System.out.println(ex);

ex.test();

}

public void test() {

System.out.println(this);// it refers to same obj which we created in main method.

}

}

* Example 7:

package p1;

// using this keyword, we can access non static members of the class.

public class Ex7 {

int x = 10;

public static void main(String[] args) {

Ex7 ex = new Ex7();// object created one adress gets stored in rv by us compiler assigns obj adress

// to this keyword.

System.out.println(ex.x);

ex.test();

}

public void test() {

System.out.println(this.x);// it refers to same obj which we created in main method.

}

}

* Example 8:

package p1;

public class Ex8 {

public static void main(String[] args) {

Ex8 ex = new Ex8();// object created one adress gets stored in rv by us compiler assigns obj adress// to this keyword.

ex.test();

}

public void test() {

this.test1();// we can use this keyword to call the method too

}

public void test1() {

System.out.println(100);

}

}

* Example 9:

package p1;

/\*limitations:

1) this keyword cannot be used inside static method, hence the below program throws error.

\*/

public class Ex9 {

public static void main(String[] args) {

// System.out.println(this);// error.

Ex9.test();

}

public static void test() {

// System.out.println(this); //error.

}

}

* Example 10:

package p1;

/\*2) using this keyword we can access static variable, as shown in the example below., it gives warning but not error. its bad practise.

\* \*/

public class Ex10 {

static int x = 10;

@SuppressWarnings("static-access")

public static void main(String[] args) {

Ex10 ex = new Ex10();// object created one adress gets stored in rv by us compiler assigns obj adress

// to this keyword.

System.out.println(ex.x);

ex.test();

}

public void test() {

System.out.println(this.x);// it refers to same obj which we created in main method.

}

}

**Core Java Assignment:**

-3- Topics.

1. IIB:

* Example 1:

package iib;

/\*IIB:

\* 1) IIB are executed when objects are created, number of times

\* we create object same number of times IIB will be called.

\* 2)IIB are used to initialize all instance variables in one place &

\* that gives us a better readability of the code.

\* \*/

public class Ex1 {

{// IIB

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

}

public static void main(String[] args) {

// no output because no object is created.

}

}

* Example 2:

package iib;

//Note: we have to create Obj mandatorily to call the IIB's.

public class Ex2 {

{// IIB

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

}

public static void main(String[] args) {

new Ex2();// object should be created

// mandatorily to call the IIB

}

}

* Example 3:

package iib;

//since 2 obj are created twice output will print twice.

public class Ex3 {

{// IIB

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

}

public static void main(String[] args) {

new Ex3();// object should be created

// mandatorily to call the IIB

new Ex3();// since 2 obj are created twice output will print twice.

}

}

* Example 4:

package iib;

// main method> IIB> Constructor.

public class Ex4 {

{// IIB // runs second

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

}

Ex4(){ //runs third

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

}

public static void main(String[] args) { // runs first

new Ex4();// object should be created mandatorily to call the IIB

}

}

* Example 5:

package iib;

//main method> IIB> Constructor- (special case).

public class Ex5 {

{// IIB // runs second //step3

System.out.println("from IIB");//step4

}//step5

Ex5() { // runs third //step6

System.out.println("from constructor"); //step7

}//step8

public static void main(String[] args) { // runs first//step1

new Ex5();// object should be created mandatorily to call the IIB

//step2 //step9

System.out.println("from main");//step10

}//step11 stop.

}

* Example 6:

package iib;

//main method> IIB> Constructor- (special case).

public class Ex6 {

{// IIB // runs second //step4

System.out.println("from IIB");//step5

}//step6

Ex6() { // runs third //step7

System.out.println("from constructor"); //step8

}//step9

public static void main(String[] args) { // runs first//step1

System.out.println("main start");// step2

new Ex6();// object should be created mandatorily to call the IIB

//step3 //step10

System.out.println("from main");//step11

}//step12 stop.

}

* Example 7:

package iib;

/\*Multiple IIB's

\* 1) which is present at first prints first ie by order wise

\* 2) MM>IIB1>IIB(n)>Constructor.

\* \*/

public class Ex7 {

{// IIB 1 // runs second

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

}

{// IIB 2 // runs third

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

}

Ex7() { // runs fourth

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

}

public static void main(String[] args) { // runs first

new Ex7();// object should be created mandatorily to call the IIB

}

}

* Example 8:

package iib;

//Example represntation.

public class Ex8 {

int i; // non static variable.

{

i = 30; // initialization of var inside IIB

System.out.println(i);

}

public static void main(String[] args) {

new Ex8();// object creation to call IIB

}

}

* Example 9:

package iib;

/\*Note: We can initialize both static and

\* non-static variables in IIB

\* \*/

public class Ex9 {

static int i; // static variable declaration.

{

i = 30;// initialization of var inside IIB

System.out.println(i);

}

public static void main(String[] args) {

new Ex9(); // object creation to call IIB

}

}

1. SIB:

* Example 10:

package sib;

/\*SIB:

\* 1) runs before main method and it doesn't require any invoking statement.

\* 2) no Object creation is required or mandatory to call SIB

\* \*/

public class Ex1 {

static { // syntax of SIB, starts with a prefix of static.

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

}

public static void main(String[] args) {

//no object creation is required to call SIB as it belongs to

// class common memory, this program will give an output.

}

}

* Example 11:

package sib;

// SIB>Main method.

public class Ex2 {

static { // syntax of SIB, starts with a prefix of static.//step1

System.out.println("from SIB"); //step 2

}//step3

public static void main(String[] args) { //step4

System.out.println("from main");//step 5

}//step 6 // stop program.

}

* Example 12:

package sib;

/\* SIB can only store static variables if non static

\* variables are initialized in SIB then it will give an error\*/

public class Ex3 {

static int i;

int x ;

static { // syntax of SIB, starts with a prefix of static.

i =20;

System.out.println(i);

//x=30;// we cannot initialize non-static variables inside

//SIB it will give error.

}

public static void main(String[] args) {

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

}

}

* Example 13:

package sib;

/\*SIB-1>SIB-2>MM>IIB>Constructor>Method\*/

public class Ex4 {

static { // 1st SIB

System.out.println("first SIB");

}

static {// 2nd SIB

System.out.println("second SIB");

}

public static void main(String[] args) {

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

}

}

* Example 14:

package sib;

/\*combination of all types

\* SIB>MM>IIB>Constructor>Method\*/

public class Ex5 {

{//IIB

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

}

static { //SIB

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

}

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

new Ex5();// object

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

Ex5.test();

}

public static void test() {

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

}

}

* Example 15:

package sib;

/\* 1) Object creation in SIB is accepted.

\* 2) Object creation in IIB can be written , it wont give any error

\* but it will give exception.

\* \*/

public class Ex6 {

{//IIB

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

}

static { //SIB

new Ex6();// object creation inside SIB

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

}

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

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

Ex6.test();

}

public static void test() {

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

}

}

1. Unary Operators:

* Example 16:

package unary\_operator;

/\*1) pre-increment/pre decrement: (++i/--i) -here we first increase

\* the value by 1 and stored there itself and it will remain

\* same when we see next i.

\*2) post increment/post decrement:(i++/i--) -here we first don't

\* increase the value by 1 here we store same value but when we see

\* next i value will be increased.

\* \*/

public class Ex1 {

public static void main(String[] args) {

int i =10;

int j = ++i + i++;// 11, 11++;

System.out.println(i);// 11++ -> 12 output.

System.out.println(j); // 11+11 -> 22 output.

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

int k =10;

int l = --k + k--;//9,9--

System.out.println(k);//9-- -> 8 output.

System.out.println(l);// 9+9 -> 18 output.

}

}

**Core Java class 12:**

-4- Topics,31- Programs.

1. This keyword continuation:

* Example 1:

package p1;

/\*when there are several objects created in same class then this keyword will

\* select objects based on which is executing at that point of time, and when

\* there are no objects for this keyword to point then it will go back to

\* first object \*/

/\*1) this keyword points to current object running in the program.\*/

public class Ex1 {

public static void main(String[] args) {

Ex1 ex1 = new Ex1();

System.out.println(ex1);

ex1.test();

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

Ex1 ex12 = new Ex1();

System.out.println(ex12);

ex12.test();

}

public void test() {

System.out.println(this);

}

}

* Example 2:

package p1;

/\*2) in non static method we can access non static members even without

\* this keyword as shown in the below example\*/

public class Ex2 {

int x = 10;

public static void main(String[] args) {

Ex2 ex2 = new Ex2();

ex2.test();

}

public void test() {

System.out.println(/\*this.\*/x);// this.x

}

}

1. Access modifiers method/variables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Private | Default | Protected | Public |
| Same class | Yes | Yes | Yes | Yes |
| Sub class same package | No | Yes | Yes | Yes |
| Non-Sub class same package | No | Yes | Yes | Yes |
| Sub class different package | No | No | Yes | Yes |
| Non-Sub class different package | No | No | No | Yes |

1. Private: if you make variable /method private it can be accessed in same class.

* Example1:

package p2;

/\*Main point: if you make variable /method private

\* it can be accessed in same class\*/

/\*

\* Data-hiding: here we make variable private so that it cannot be

\* accessed outside class.

\*

\* Hiding-implementation of Data/details:

\* Encapsulation concept.

\* \*/

/\* Private: if we make any variable or method private we can

access in same class.\*/

public class Private\_1 {

private int x = 20; // non static private variable.

private void test() { // private method.

System.out.println("accessible");

}

public static void main(String[] args) {

Private\_1 a = new Private\_1(); // object creation

System.out.println(a.x);// accessing private variable // no error

a.test();// accessing private method // no error

}

}

* Example2:

package p2;

/\* private(sub class same package): if we make any variable or

\* method private we cannot access in sub class same package.\*/

public class Private\_2 extends Private\_1 {

@SuppressWarnings("unused")

public static void main(String[] args) {

Private\_2 a = new Private\_2(); // object creation

// System.out.println(a.x);//cannot access private variable //error

// a.test();// cannot access private method //error

}

}

* Example3:

package p2;

/\* private( non sub class same package): if we make any variable or

\* method private we cannot access in non sub class same package.\*/

public class Private\_3 { // no inheritance

@SuppressWarnings("unused")

public static void main(String[] args) {

Private\_1 a = new Private\_1();

/\*

\* object creation for non subclass that is we create parent

\* class object in child class without inheritance.

\*/

// System.out.println(a.x);//cannot access private variable //error

// a.test();// cannot access private method //error

}

}

* Example4:

package p3;

import p2.Private\_1;

/\* private(sub class different package): if we make any variable or

\* method private we cannot access in sub class different package.\*/

public class Private\_4 extends Private\_1 {

@SuppressWarnings("unused")

public static void main(String[] args) {

Private\_4 a = new Private\_4(); // object creation

// System.out.println(a.x);//cannot access private variable //error

// a.test();// cannot access private method //error

}

}

* Example5:

package p3;

import p2.Private\_1;

/\* private(non sub class different package): if we make any variable or

\* method private we cannot access in non sub class different package.\*/

public class Private\_5 { // no inheritance

@SuppressWarnings("unused")

public static void main(String[] args) {

Private\_1 a = new Private\_1(); // object creation for non sub class.

// System.out.println(a.x);//cannot access private variable //error

// a.test();// cannot access private method //error

}

}

1. Default: If you make the member default, then it can be accessed anywhere in same package and same class.

* Example1:

package p2;

/\*

\* Main definition: If you make the member default,Then it can be accessed

\* anywhere in same package and same class.

\* \*/

/\*Default(same class):if you make variable and method default,

\* then you can access it in same class.

\* \*/

public class Default\_1 {

int x = 10;// default variable.

void test() { // default method

System.out.println("accesible");

}

public static void main(String[] args) {

Default\_1 a = new Default\_1(); // object creation

System.out.println(a.x); // accessing default variable.//no error.

a.test();// accessing default method.//no error.

}

}

* Example2:

package p2;

/\*Default(sub class same package):if you make variable and method default,

\* then you can access it in sub class same package.

\* \*/

public class Default\_2 extends Default\_1 { // inheritance is there

public static void main(String[] args) {

Default\_2 a = new Default\_2(); // object creation of child class

System.out.println(a.x); // accessing default variable.//no error.

a.test();// accessing default method.//no error.

}

}

* Example3:

package p2;

/\*Default(non sub class same package):if you make variable and method default,

\* then you can access it in non sub class same package.

\* \*/

public class Default\_3 {//no inheritance.

public static void main(String[] args) {

Default\_1 a = new Default\_1(); // object creation of parent class

System.out.println(a.x); // accessing default variable.//no error.

a.test();// accessing default method.//no error.

}

}

* Example4:

package p3;

import p2.Default\_1;

/\*Default( sub class different package):if you make variable and method default,

\* then you cannot access it in non sub class different package.

\* \*/

public class Default\_4 extends Default\_1 { // inheritance is there.

@SuppressWarnings("unused")

public static void main(String[] args) {

Default\_4 a = new Default\_4(); // object creation of child class.

// System.out.println(a.x); //cannot accessing default variable.//error.

// a.test();// cannot accessing default method.// error.

}

}

* Example5:

package p3;

import p2.Default\_1;

/\*Default(non sub class different package):if you make variable and method

\* default,then you cannot access it in non sub class same package.

\* \*/

public class Default\_5 {// no inheritance.

@SuppressWarnings("unused")

public static void main(String[] args) {

Default\_1 a = new Default\_1(); // object creation of parent class

// System.out.println(a.x); //cannot accessing default variable.//error.

// a.test();// cannot accessing default method.// error.

}

}

1. Protected: If you make the variable/method protected, then it can be accessed anywhere in same package and different package only through inheritance.

* Example1:

package p2;

/\* Main definition: If you make the variable/method protected,Then

\* it can be accessed anywhere in same package and different package

\* only through inheritance.

\* \*/

/\*protected(same class):if you make variable and method protected,

\* then you can access it in same class.

\* \*/

public class Protected\_1 {

protected int x = 10;// protected variable.

protected void test() { // protected method

System.out.println("accesible");

}

public static void main(String[] args) {

Protected\_1 a = new Protected\_1(); // object creation

System.out.println(a.x); // accessing protected variable.//no error.

a.test();// accessing protected method.//no error.

}

}

* Example2:

package p2;

/\*Default(sub class same package):if you make variable and method protected,

\* then you can access it in sub class same package.

\* \*/

public class Protected\_2 extends Protected\_1 { // inheritance is there

public static void main(String[] args) {

Protected\_2 a = new Protected\_2(); // object creation of child class

System.out.println(a.x); // accessing protected variable.//no error.

a.test();// accessing protected method.//no error.

}

}

* Example3:

package p2;

/\*Default(non sub class same package):if you make variable and method protected,

\* then you can access it in non sub class same package.

\* \*/

public class Protected\_3 {// no inheritance.

public static void main(String[] args) {

Protected\_1 a = new Protected\_1(); // object creation of parent class

System.out.println(a.x); // accessing protected variable.//no error.

a.test();// accessing protected method.//no error.

}

}

* Example4:

package p3;

import p2.Protected\_1;

/\*Default( sub class different package):if you make variable and

\* method Protected, then you can access it in non sub class different package.

\* \*/

public class Protected\_4 extends Protected\_1 { // inheritance is there.

public static void main(String[] args) {

Protected\_4 a = new Protected\_4(); // object creation of child class.

System.out.println(a.x); // can access Protected variable.//no error.

a.test();// can access Protected method.// no error.

}

}

* Example5:

package p3;

import p2.Protected\_1;

/\*Default(non sub class different package):if you make variable and method

\* Protected,then you cannot access it in non sub class same package.

\* \*/

public class Protected\_5 {// no inheritance.

@SuppressWarnings("unused")

public static void main(String[] args) {

Protected\_1 a = new Protected\_1(); // object creation of parent class

// System.out.println(a.x); //cannot accessing Protected variable.//error.

// a.test();// cannot accessing Protected method.// error.

}

}

1. Public: if we make variable/method public then it can be accessed anywhere in the program.

* Example1:

package p2;

/\* Main Definition: if we make variable/method public then it can be accessed

\* anywhere in the program.

\* \*/

public class Public\_1 {

}

1. Access modifies for class:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | private | Default | Protected | public |
| Sub class same package | N/A | Yes | N/A | yes |
| Sub class different package | N/A | No | N/A | yes |

1. Public class: if we make class public then it can be accessed anywhere.

* Example1:

package p2;

/\*Main definition: if we make class public then

it can be accessed anywhere\*/

/\* Note: class can only be public or default,

\* It can never be protected or private\*/

public class Public\_class\_1 {

}

* Example2:

package p2;

/\*if we make class public,

\* we can use in same package, it wont give error\*/

public class Public\_class\_2 extends Public\_class\_1 { // inheritance done.//no error

}

* Example3:

package p3;

import p2.Public\_class\_1;

/\*if we make class public,

\* we can use in different package, it will give no error\*/

public class Public\_class\_3 extends Public\_class\_1 { // no error

}

1. Default class: if we make class default then it can only be accessed in same package.

* Example1:

package p2;

/\*Main definition: if we make class default then

it can only be accessed in same package\*/

class Default\_class\_1 {// Default class

}

* Example2:

package p2;

/\*if we make class default,

\* we can use in same package, it wont give error\*/

class Default\_class\_2 extends Default\_class\_1 { // inheritance done.//no error

}

* Example3:

package p3;

//import p2.Default\_class\_1;// error

/\*if we make class default,

\* we cannot use in different package, it will give error\*/

class Default\_class\_3 /\* extends Default\_class\_1 \*/ { // error

}

1. Protected class: Not Applicable.
2. Private class: Not Applicable.
3. Constructor access specifiers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | private | Default | Protected | public |
| Same class | yes | Yes | Yes | Yes |
| Diff class same package. | yes | Yes | no | Yes |
| Diff class diff package. | No | No | No | Yes |

1. Public constructor: if you make constructor public, then its object can be created anywhere in the program.
   * Example1:

package p2;

/\*Main definition: if you make constructor public, then

\* its object can be created anywhere in the program. \*/

public class Public\_constructor\_1 {

public Public\_constructor\_1() {

System.out.println(100);

}

public static void main(String[] args) {

new Public\_constructor\_1();

}

}

1. Default constructor: if we make constructor default, then we can create object only in same class and in same package, cannot be used in different package it will give an error.
   * Example1:

package p2;

/\*Main definition: if we make constructor default, then we can create

\* object only in same class and in same package, cannot be used in

\* different package it will give an error

\* \*/

/\*Object creation to call constructor in same class is accepted.\*/

public class Default\_Constructor\_1 {

Default\_Constructor\_1() { // default constructor.

System.out.println(100);

}

public static void main(String[] args) {

new Default\_Constructor\_1(); // object creation in same class accepted.

}

}

* + Example2:

package p2;

/\* we can access default constructor by

\* creating object in different class but it should be same package\*/

public class Default\_Constructor\_2 {

public static void main(String[] args) {

new Default\_Constructor\_1();

// object creation in different class accepted in default constructor.

}

}

* + Example3:

package p3;

/\* we cannot access default constructor by

\* Creating object in different package\*/

public class Default\_Constructor\_3 {

public static void main(String[] args) {

// new Default\_Constructor\_1(); //error

// object creation in different package not accepted.

}

}

1. Private constructor: if we make constructor private, then we can create object only in same class, cannot be used outside class or if it is used in different class or package, it will give an error.
   * Example1:

package p2;

/\*Main definition: if we make constructor private, then we can create

\* object only in same class, cannot be used outside class / if it is used

\* in different class it will give an error

\* \*/

/\*Object creation to call constructor in same class is accepted.\*/

public class Private\_Constructor\_1 {

private Private\_Constructor\_1() { // private constructor.

System.out.println(100);

}

public static void main(String[] args) {

new Private\_Constructor\_1(); // object creation in same class accepted.

}

}

* + Example2:

package p2;

/\* we cannot access private constructor by

\* creating object in different class\*/

public class Private\_Constructor\_2 {

public static void main(String[] args) {

// new Private\_Constructor\_1(); //error

// object creation in different class not accepted.

}

}

* + Example3:

package p3;

/\* we cannot access private constructor by

\* creating object in different class package\*/

public class Private\_Constructor\_3 {

public static void main(String[] args) {

// new Private\_Constructor\_1(); //error

// object creation in different class package not accepted.

}

}

1. Protected constructor: if we make constructor protected, then we can create object only in same class and in same package, cannot be used in different package it will give an error.
   * Note: protected and Default works in same way when it comes to constructor.
   * Example1:

package p2;

/\*Main definition: if we make constructor protected, then we can create

\* object only in same class and in same package, cannot be used in

\* different package it will give an error

\* \*/

/\* Note: protected and Default works in same

\* way when it comes to constructor.\*/

/\*Object creation to call constructor in same class is accepted.\*/

public class Protected\_Constructor\_1 {

protected Protected\_Constructor\_1() { // protected constructor.

System.out.println(100);

}

public static void main(String[] args) {

new Protected\_Constructor\_1(); // object creation in same class accepted.

}

}

**Core Java class 13:**

-3- Topics,9- Programs.

1. Constructor chaining:

* Example 1:

package p1;

//constructor chaining.

//Note: we cannot call constructor with this keyword from method.

/\*this:

\* 1)this() -> used to call the constructor.

\* 2) using this keyword we can call the constructor, but this call should happen only from another constructor.

\* \*/

public class Ex1 {

Ex1(){

System.out.println(500);

}

Ex1(int x){

this();// this keyword calling constructor from line 10

}

public static void main(String[] args) {

new Ex1(100);//object calling constructor at line 14

}

}

* Example 2:

package p1;

//constructor chaining:

public class Ex2 {

Ex2(){

this(500); /\* calling constructor from line 9 and we can pass value

in this keyword while calling the constructor\*/

}

Ex2(int x){

System.out.println(x);

}

public static void main(String[] args) {

new Ex2();//object calling constructor at line 6

}

}

* Example 3:

package p1;

// while calling a constructor this keyword should always be first statement in another constructor.

public class Ex3 {

Ex3() {

System.out.println(500);

// this(500);//error /\* calling constructor from line 10 and we can pass value in this keyword while calling the constructor\*/

}

Ex3(int x) {

System.out.println(x);

}

public static void main(String[] args) {

new Ex3(100);// object calling constructor at line 5

}

}

* Example 4:

package p1;

//constructor chaining best example.

public class Ex4 {

Ex4() {

this(500);

/\*

\* calling constructor from line 13 and we can pass

\* value in this keyword while calling the constructor

\*/

}

Ex4(int x) {

this(30, 40);

/\*

\* calling constructor from line 22 and we can pass value

\* in this keyword while calling the constructor

\*/

System.out.println(x);

}

Ex4(int x, int y) {

System.out.println(x);

System.out.println(y);

}

public static void main(String[] args) {

new Ex4();// object calling constructor at line 5

}

}

1. Getters and Setters:

* Example1:

package p1;

/\*GETTERSS AND SETTERS\*/

public class Ex5 {

int age; // Shouldn't be initialize manually or by using iib,

// only by method, getters and setters.

public int getAge() { // getter

return age;

}

public void setAge(int age) {// setter

/\* non static var with this keyword \*/ this.age = /\* local var \*/age;

}

public static void main(String[] args) {

Ex5 ex5 = new Ex5();

ex5.setAge(20); // to set the value

System.out.println(ex5.getAge());// to print the getter value

}

}

1. Encapsulation:

* Example 1:

package p1;

/\*Encapsulation:

\* 1) Bundling of data with the method it operates on it restricting

\* direct access to that variable is called as encapsulation.

\* 2) to restrict direct access of the variable we make variable

\* private to operate on those variables, we create

\* Publicly defined getters and setters.

\* 3) data hiding is the part of encapsulation.

\* \*/

public class Ex6 {

private int age;

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

package p1;

//continuation of Ex6

public class Ex7 extends Ex6 {

public static void main(String[] args) {

Ex7 ex7 = new Ex7();

ex7.setAge(100);

int age = ex7.getAge();

System.out.println(age);

}

}

* Example2:

package p1;

//encapsulation example 2

public class Ex8 {

private int age;

private String name;

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

package p1;

//continuation of Ex8

public class Ex9 extends Ex8 {

public static void main(String[] args) {

Ex9 ex9 = new Ex9();

ex9.setAge(200);

ex9.setName("pankaj");

int age = ex9.getAge();

String name = ex9.getName();

System.out.println(age);

System.out.println(name);

}

}

**Core Java class 14:**

-3- Topics.

1. Type Casting:

* Example1:

package p1;

/\* Type casting:

\* converting particular data type into required

\* data type is called as type casting.

\*

\* 2 types:

\* 1) explicit down-casting

\* 2) auto up-casting

\* \*/

/\*Note: here this concept is applicable on static,

\* n/static and local variables.\*/

public class Ex1 {

}

* Example 2:

package p1;

/\* Auto up-casting:

\* 1)Here we convert smaller data type to bigger data type

\* 2)during auto up-casting data loss should not happen.

\* \*/

public class Ex2 {

public static void main(String[] args) {

int x=10;//4 bytes

long y = x; //up-casting //8 bytes

System.out.println(y);

}

}

* Example 3:

package p1;

//continuation of up-casting

public class Ex3 {

public static void main(String[] args) {

float x=10; //4 bytes

double y = x; //up-casting //8 bytes

System.out.println(y);

}

}

* Example 4:

package p1;

// in below example during conversion there is data loss of 0.3 hence

// up-casting cannot happen hence there is an error.

public class Ex4 {

@SuppressWarnings("unused")

public static void main(String[] args) {

float x = 10.3f; // 4 bytes

/\*

\* long y = x; //up-casting //8 bytes it can only store integer values not

\* decimal values so no up-casting

\*/

// System.out.println(y);

}

}

* Example 5:

package p1;

/\* Explicit down-casting:

\* 1) here we convert bigger data type to smaller data type.

\* 2) during down-casting data loss might happen.

\* \*/

public class Ex5 {

public static void main(String[] args) {

long x = 10; // 8 bytes

int y = (int) x; // down-casting //4 bytes // no data loss

System.out.println(y);

}

}

* Example 6:

package p1;

/\* Explicit down-casting:

\* 1) here we convert bigger data type to smaller data type.

\* 2) during down-casting data loss might happen.

\* \*/

public class Ex6 {

public static void main(String[] args) {

float x = 10.3f; // 4 bytes

long y = (long) x; // down-casting //8 bytes //data loss

System.out.println(y);

}

}

* Example 7:

package p1;

//continuation of down-casting.

public class Ex7 {

public static void main(String[] args) {

short x = 10; // 2 bytes

byte y = (byte) x; // down-casting //1 bytes // no data loss

System.out.println(y);

}

}

* Example 8:

package p1;

// special case

public class Ex8 {

public static void main(String[] args) {

int x = 97; // 4 bytes

char y = (char) x; // down-casting // conversion of data

System.out.println(y);

}

}

* Example 9:

package p1;

/\*class casting

\* 1) class up-casting: here we store child class address into

\* parent class reference variable.

\* 2) class down-casting: here we store parent

\* class object address into child class reference variable..

\* \*/

//Note: applicable only on sub classes , on non sub classes it wont work.

public class Ex9 {

public static void main(String[] args) {

}

}

1. Class casting:

* Example 1:

package p1;

public class Ex10 {

}

package p1;

// running method //class up-casting

public class Ex10\_1 extends Ex10 {

@SuppressWarnings("unused")

public static void main(String[] args) {

Ex10 ex = new Ex10\_1();

ex = new Ex10\_2();

System.out.println(ex);

}

}

package p1;

public class Ex10\_2 extends Ex10 {

}

* Example 2:

package p1;

public class Ex11 {

}

package p1;

// running method class down-casting

public class Ex11\_1 extends Ex11 {

@SuppressWarnings("unused")

public static void main(String[] args) {

Ex11\_1 ex = (Ex11\_1) new Ex11(); //class down-casting

Ex11\_2 ex1 = (Ex11\_2) new Ex11(); //class down-casting

}

}

package p1;

public class Ex11\_2 extends Ex11 {

}

1. Object:

* Example 1:

package p1;

//object is the super most class by default.

public class Ex12 extends Object { // it is not mandatory to write "extends object"

// garbage collection method belongs to object.

}

**Core Java class 15:**

-2- Topics.

1. Run time polymorphism:

* Example 1:

package p1;

// run-time polymorphism : if we do overriding and class

//up-casting as shown in the below example.

//overriding + class up-casting.

public class Ex1 {

public void test() {

System.out.println(20);

}

}

package p1;

//continuation of Ex1

public class Ex1\_1 extends Ex1 {

@Override

public void test() {

System.out.println(30);

}

public static void main(String[] args) {

Ex1 ex = new Ex1\_1(); //class up-casting

ex.test();

}

}

* Example 2:

package p1;

/\* Note : During overriding access specifier

\* scope can be increased but it cannot be decreased

\* as shown in below example.

\* \*/

public class Ex2 {

protected void test() {

System.out.println(20);

}

}

package p1;

public class Ex2\_1 extends Ex2 {

@Override

public void test() { //no error

System.out.println(30);

}

public static void main(String[] args) {

Ex1 ex = new Ex1\_1(); //class up-casting

ex.test();

}

}

* Example 3:

package p1;

/\* Note : During overriding access specifier

\* scope can be increased but it cannot be decreased

\* as shown in below example.

\* \*/

public class Ex3 {

protected void test() {

System.out.println(20);

}

}

package p1;

public class Ex3\_1 extends Ex3 {

// @Override

// void test() { // error //default method

// System.out.println(30);

// }

@SuppressWarnings("unused")

public static void main(String[] args) {

Ex3\_1 ex = new Ex3\_1(); // class up-casting

// ex.test();

}

}

* Example 4:

package p1;

/\* Note: static members are never inherited, so overriding

\* of that is not allowed.

\* \*/

public class Ex4 {

public static void test() {

System.out.println(500);

}

}

package p1;

public class Ex4\_1 extends Ex4 {

// @Override

// public static void test() { // static methods cannot be over-ridden

// System.out.println(300);

// }

}

* Example 5:

package p1;

//example to prove that static members don't get inherited.

// continue in Ex5\_1

public class Ex5 {

public static void test() {

System.out.println(500);

}

}

package p1;

//in the below example EX5\_1.test() is converted into Ex5.test()

//automatically by compiler.

public class Ex5\_1 extends Ex5 {

public static void main(String[] args) {

Ex5\_1.test(); // here no inheritance happening compiler is just

// changing EX5\_1.test() to Ex5.test() automatically

// to avoid exception

}

}

* Example 6:

package p1;

//example to prove that static members don't get inherited.

// continue in Ex6\_1

public class Ex6 {

public static void test() {

System.out.println(500);

}

}

package p1;

//in the below example ex.test() is converted into Ex6.test()

//automatically by compiler.

public class Ex6\_1 extends Ex6 {

static Ex6\_1 ex = null;

public static void main(String[] args) {

ex.test(); // here no inheritance happening compiler is just

// changing ex.test() to Ex6.test() automatically

// to avoid exception

}

}

1. Abstraction:

* Example 1:

package p1;

/\*Abstraction:

\* 1)Interface:

\* a) interface can consist of only incomplete methods in it.

\* b) abstract keyword helps us to define this method is incomplete, not

\* mandatory to write it, we can write it at beginning or after

\* public keyword.

\* c) it is the blueprint to project.

\*/

/\*Note : it is like constructing building , i.e. we should have a blue print

\* , same way we should develop a blueprint i.e. which contains all

\* the features which should be developed in project i.e. we

\* create incomplete methods which are nothing but

\* Features if we don't use that methods it will give

\* error because class never allows incomplete methods.

\* \*/

/\* Note:

\* interface file is also saved as .java only.

\* \*/

/\*Note:

\* 1)java is not complete object oriented because we can run programs

\* even without creating objects which is not mandatory.

\* \*/

public interface Ex7 { // interface

public abstract void test(); // incomplete method

}

package p1;

public class Ex7\_1 implements Ex7 {// implements is the keyword used to

// do inheritance between class and interface.

@Override

public void test() { // completing interface method is mandatory

// if not it will give error.

System.out.println(100);

}

public static void main(String[] args) {

Ex7\_1 ex7\_1 = new Ex7\_1();

ex7\_1.test();

}

}

* Example 2:

package p1;

// best example to demonstrate interface, to show why it is used.

public interface Ex8 {

abstract public void balance(); // incomplete method.

// if we don't override it in class where we

// inherit it , it will give an error.

public abstract void transferMoney();// incomplete method.

// if we don't override it in class where we

// inherit it , it will give an error.

public abstract void generateBankStatement();// incomplete method.

// if we don't override it in class where we

// inherit it , it will give an error.

}

package p1;

//continuation of Ex8

public class Ex8\_1 implements Ex8 { // implements is the keyword used to

// do inheritance between class and interface.

@Override

public void balance() { // if we don't override this method it will

//give error, so this is very benifitial

//so that we don't miss any features.

System.out.println("logic of balance");

}

@Override

public void transferMoney() {

System.out.println("logic of transfer money");

}

@Override

public void generateBankStatement() {

System.out.println("logic of getting bank statement ");

}

public static void main(String[] args) {

Ex8\_1 ex8\_1 = new Ex8\_1();

ex8\_1.balance();

ex8\_1.transferMoney();

ex8\_1.generateBankStatement();

}

}

**Core Java class 16:**

-3- Topics.

1. More on interfaces:

* Example 1:

package p1;

/\*

\* Note: inheritance

\* class -> class --> extends.

\* interface -> interface--> extends

\* interface -> class --> implements

\* class -> interface--> nothing not possible

\* \*/

/\*

\* example : interface1 extends interface2 -> implements interface2

\* \*/

/\*1)in an interface only abstract methods can be created.

\*

\* \*/

public interface Ex1 {

public void test1();

}

package p1;

public interface Ex1\_1 extends Ex1 {

public void test2();

}

package p1;

public class Ex1\_2 implements Ex1\_1 {

@Override

public void test1() {

System.out.println(1);

}

@Override

public void test2() {

System.out.println(2);

}

public static void main(String[] args) {

Ex1\_2 c = new Ex1\_2();

c.test1();

c.test2();

}

}

* Example 2:

package p1;

// interfaces in java supports multiple inheritance.

// interface 1 -> interface 2-> interface 3-> class Ex2\_3 extends interface 3

public interface Ex2 {

public void test1();

}

package p1;

public interface Ex2\_1 {

public void test2();

}

package p1;

public interface Ex2\_2 extends Ex2 ,Ex2\_1 {

public void test3();

}

package p1;

public class Ex2\_3 implements Ex2\_2 {

@Override

public void test1() {

System.out.println(1);

}

@Override

public void test2() {

System.out.println(2);

}

@Override

public void test3() {

System.out.println(3);

}

public static void main(String[] args) {

Ex2\_3 d = new Ex2\_3();

d.test1();

d.test2();

d.test3();

}

}

* Example 3:

package p1;

/\*

\* marker interface:

\* 1) an empty interface in java is called as marker

\* interface as shown below.

\* \*/

public interface Ex3 {

// marker interface

}

* Example 4:

package p1;

// on a particular class we can perform extends and

//implements both , but ensure extends is used first then implements

public interface Ex4 {

public void test1();

}

package p1;

public class Ex4\_1 {

public void test2() {

System.out.println(100);

}

}

package p1;

// child class > extends parent class> implements interface

public class Ex4\_2 extends Ex4\_1 implements Ex4 {

@Override

public void test1() {

System.out.println(50);

}

public void test3() {

System.out.println(200);

}

public static void main(String[] args) {

Ex4\_2 c = new Ex4\_2();

c.test1();

c.test2();

c.test3();

}

}

* Example 5:

package p1;

// multiple inheritance

public interface Ex5 {

public void test1();

}

package p1;

public interface Ex5\_1 {

public void test2();

}

package p1;

public class Ex5\_2 implements Ex5, Ex5\_1 {

@Override

public void test2() {

System.out.println(1);

}

@Override

public void test1() {

System.out.println(2);

}

public static void main(String[] args) {

Ex5\_2 c = new Ex5\_2();

c.test1();

c.test2();

}

}

1. Final keyword:

* Example 1:

package p1;

/\*

\* final keyword:

\* 1) if you make a variable final then we cannot change its value.

\* 2) if you make static/ non static variable as final and if not initialize

\* then it will give error.

\* 3) all final variable should be written in all upper cases.

\* 4) if you make class as final then inheritance is not allowed.

\* 5) if you make a method final then overriding is not possible.

\* \*/

public class Ex6 {

public static void main(String[] args) {

final int x = 10;

// x= 10; we cannot change with the same value or different value.

System.out.println(x);

}

}

* Example 2:

package p1;

//if you make static variable as final and if not initialize

//then it will give error.

public class Ex7 {

final static int BUILDING\_NUMBER = 6; // bold and italic and all final

// var should be inn all upper-cases

// if not initialized will give error.

public static void main(String[] args) {

}

}

* Example 3:

package p1;

//if you make non static variable as final and if not initialize

//then it will give error.

public class Ex8 {

final int BUILDING\_NUMBER = 6; // bold and italic and all final

// var should be inn all upper-cases.

// if not initialized will give error.

public static void main(String[] args) {

}

}

* Example 4:

package p1;

final public class Ex9 {

}

package p1;

public class Ex9\_1 /\* extends Ex9 \*/ {

// if you make pc as final then inheritance is not allowed.

public static void main(String[] args) {

new Ex9();

// but we can create object of parent class i.e. final class

}

}

* Example 5:

package p1;

public class Ex10 {

final public void test() {

System.out.println(100);

}

}

package p1;

public class Ex10\_1 extends Ex10{

// @Override

// public void test() {

// System.out.println(100);

// }

// i.e. we cannot override final methods

}

**Core Java Assignment -2 :**

-1- Topics.

1. Super Keyword:

* Example 1:

package p1;

/\*super keyword():

\* 1) using super keyword we can access the members of parent class.

\* 2) using super keyword we can access static and non static members both

\* 3) using super() keyword we can call constructor of parent class but then

\* we should super keyword in child class constructor and it should be the

\* very first statement.

\* 4) if we don't keep super keyword inside child class constructor

\* then compiler will automatically place the super keyword

\* such that it can call only no argument constructor of parent class.

\* 5) super() keyword will only call no args constructor.

\* \*/

/\*limitations:

\* 1) super keyword cannot be used inside static methods

\* 2) super() keyword should be the first statement in the child

\* class constructor.

\* 3) we cannot call parent class constructor from child class method

\* using super() keyword.

\* \*/

/\*super class is also called as parent class.

\* \*/

public class Ex1 {

int i = 10;

}

package p1;

public class Ex1\_1 extends Ex1/\*super class\*/{

public static void main(String[] args) {

Ex1\_1 b = new Ex1\_1();

b.test();

}

private void test() {

System.out.println(super.i); // super keyword accesses parent

//class var.

}

}

* Example 2:

package p1;

//second example.

public class Ex2 {

public void xyz() {

System.out.println("xyz");

}

}

package p1;

public class Ex2\_1 extends Ex2/\*super class\*/{

public static void main(String[] args) {

Ex2\_1 b = new Ex2\_1();

b.test();

}

private void test() {

super.xyz(); // super keyword accesses parent

//class method.

}

}

* Example 3:

package p1;

//second example.

public class Ex3 {

public void xyz() {

System.out.println("xyz");

}

}

package p1;

public class Ex3\_1 extends Ex2/\*super class\*/{

public static void main(String[] args) {

Ex3\_1 b = new Ex3\_1();

b.test();

}

private void test() {

super.xyz(); // super keyword accesses parent

//class method.

}

}

* Example 4:

package p1;

/\*super keyword():

\* 1) using super keyword we can access the members of parent class

\* 2) using super keyword we can access static and non static members both

\* \*/

/\*super class is also called as parent class.

\* \*/

public class Ex4 {

static int i = 10; // static variable.

}

package p1;

public class Ex4\_1 extends Ex1/\* super class \*/ {

public static void main(String[] args) {

Ex4\_1 b = new Ex4\_1();

b.test();

Ex4\_1.test1();

}

private static void test1() {

// System.out.println(super.i);//error.

}

private void test() {

System.out.println(super.i); // super keyword accesses parent

// class variable even if it is static variable.

}

}

* Example 5:

package p1;

public class Ex5 {

Ex5(){

System.out.println(100);

}

}

package p1;

public class Ex5\_1 extends Ex5 {

Ex5\_1(){

super(); // calls parent class constructor.

}

public static void main(String[] args) {

new Ex5\_1();// object to call constructor.

}

}

* Example 6:

package p1;

/\*if we don't keep super keyword inside child class constructor

\* then compiler will automatically place the super keyword

\* such that it can call only no argument constructor of parent class.\*/

public class Ex6 {

Ex6(){

System.out.println("parent class");

}

}

package p1;

public class Ex6\_1 extends Ex6{

Ex6\_1(){//super(); it is default written in every child class constructor.

}

public static void main(String[] args) {

new Ex6\_1();

}

}

* Example 7:

package p1;

public class Ex7 {

Ex7(){

System.out.println("parent class constructor");

}

}

package p1;

// if you dont create child class constructor without argument then

//compiler will automatically place no args constructor with super keyword

public class Ex7\_1 extends Ex7 {

/\*by default we know that whenever we write new keyword anywhere

a default constructor is created and we know that every constructor

in child class created will have an super keyword by default

so it will call parent class constructor even if we don't create

the constructor and call the parent constructor from

child class constructor\*/

public static void main(String[] args) {

new Ex7\_1();

}

}

* Example 8:

package p1;

public class Ex8 {

Ex8(){

System.out.println("parent class constructor");

}

}

package p1;

public class EX8\_1 extends Ex8 {

EX8\_1(){

//super(); will always be added by default buy compiler.

System.out.println("child class constructor.");

}

public static void main(String[] args) {

new EX8\_1();

}

}

* Example 9:

package p1;

public class Ex9 {

Ex9(){

System.out.println("parent class constructor");

}

}

package p1;

public class Ex9\_1 extends Ex9 {

Ex9\_1(int i){

//super(); will always be added by default buy compiler.

System.out.println("child class constructor.");

}

public static void main(String[] args) {

new Ex9\_1(10);

}

}

* Example 10:

package p1;

// if in the parent class there is only constructors with arguments

//then as a programmer we should explicitly write super() with arguments

// which matches in child class constructor.

public class Ex10 {

Ex10(int i){

System.out.println("parent class constructor");

System.out.println(i);

}

}

package p1;

public class Ex10\_1 extends Ex10 {

Ex10\_1(){

super(10); //we should explicitly write with arguments.

System.out.println("child class constructor.");

}

public static void main(String[] args) {

new Ex10\_1();

}

}

* Example 11:

package p1;

// super key is not auto kept when there are only constructors with

//arguments in parent class, where as super keyword will be placed //automatically when in parent class there is constructor with no arguments.

public class Ex11 {

Ex11(int i){

System.out.println("parent class constructor");

System.out.println(i);

}

Ex11(){

System.out.println("from no args constructor.");

}

}

package p1;

public class Ex11\_1 extends Ex11 {

Ex11\_1(){

super(); //we should explicitly write with arguments.

System.out.println("child class constructor.");

}

public static void main(String[] args) {

new Ex11\_1();

}

}

* Example 12:

package p1;

/\* Example \*/

public class Ex12 {

Ex12(){

System.out.println("without no args constructor");

}

Ex12(int i){

System.out.println("with args constructor");

System.out.println(i);

}

}

package p1;

public class Ex12\_1 extends Ex12{

Ex12\_1(){

//super keyword kept by default

System.out.println("from parent constructor.");

}

Ex12\_1(int i){

//super keyword kept by default

System.out.println("from parent args constructor.");

}

public static void main(String[] args) {

new Ex12\_1();

new Ex12\_1(10);

}

}

* Example 13:

package p1;

// we cannot write both super and this in the same constructor as

//either of statement becomes second statement so it will give error.

public class Ex13 {

Ex13(){

System.out.println("from parent class constructor");

}

}

package p1;

// if your child class constructor consist of this keyword then

// that constructor then super keyword will not be automatically placed.

public class Ex13\_1 extends Ex13{

Ex13\_1(){

System.out.println("from child class constructor no arrgs");

}

Ex13\_1(int i){

// this(); super(); //error

this();// if there is this keyword super() keyword cannot

//be automatically placed.

System.out.println(i);

}

public static void main(String[] args) {

}

}

**Core Java class 17 :**

-2- Topics.

1. More on interface:

* Example 1:

package p1;

/\*Note:

\* static abstract methods cannot be created in

\* interface hence below program throws error.

\* \*/

public interface Ex1 {

//public static void test(); //error.

}

* Example 2:

package p1;

public class Ex2\_1 {

public static void main(String[] args) {

System.out.println(Ex2.age);

// non static members don't undergo inheritance

System.out.println(Ex2.carNumber);

}

}

package p1;

/\*Note:

\* every variable created in an interface by default is final and static.

\* \*/

public interface Ex2 {

int age = 100; // by default final static

final static int carNumber =1000;

}

* Example 3:

package p1;

/\*

\* reference variable of an interface can be

\* created however an object of interface cannot be created

\* \*/

public interface Ex3 {

}

package p1;

public class Ex3\_1 implements Ex3 {

public static void main(String[] args) {

// Ex3 ex = new Ex3() ; //error

Ex3 ex = new Ex3\_1(); // possible to create reference var of interface

}

}

1. JAVA 8 new features:

* Example 1:

package p1;

/\*Java 8 new features

\* 1) default keyword: this keyword helps us to create complete methods

\* inside interface, this features was introduced in version 8 of java.

\* \*/

public interface Ex4 {

default void test() { // we can use to create complete methods.

System.out.println(100);

}

}

package p1;

public class Ex4\_1 implements Ex4{

public static void main(String[] args) {

Ex4\_1 ex = new Ex4\_1();

ex.test();

}

}

* Example 2:

package p1;

/\* default keyword example 2\*/

public interface Ex5 {

default void test() { // we can use to create complete methods.

System.out.println(100);

}

}

package p1;

public class Ex5\_1 implements Ex5 {

public static void main(String[] args) {

Ex5\_1 a = new Ex5\_1();

a.test();

}

}

* Example 3:

package p1;

/\* 2) functional interface: in this interface

\* only one incomplete method should be present

\* \*/

@FunctionalInterface

public interface Ex6 {

public void test() ;

}

package p1;

public class Ex6\_1 implements Ex6 {

public void test() {

System.out.println(100);

}

}

* Example 4:

package p1;

/\*\*\*/

@FunctionalInterface

public interface Ex7 {

public void test(); // only one incomplete method , zero also error

//more than 1 also error, but complete methods can be any number.

default void test2() {

System.out.println("compltete method 1");

}

default void test3() {

System.out.println("compltete method 2");

}

}

package p1;

public class Ex7\_1 implements Ex7 {

@Override

public void test() {

System.out.println("100");

}

}

* Example 5:

package p1;

/\*\*

\* Lambdas: 1) it helps us to develop code with less lines, that is why java is

\* functional oriented program as well.

\* Note: limitations: code readability is

\* difficult.

\* Note: lambdas expression is applicable only on functional interfaces.

\*/

public interface Ex8 {

public void test1();

}

package p1;

public class Ex8\_1 /\*no implements we have to write\*/{

public static void main(String[] args) {

Ex8 ex = (/\* no argument in method \*/) -> { //lambdas syntax

System.out.println(100);

System.out.println(200);

};

ex.test1();

}

}

* Example 6:

package p1;

// lambdas example 2 with argument

public interface Ex9 {

public void test1(int x);

}

package p1;

public class Ex9\_1 {

public static void main(String[] args) {

Ex9 ex = (int x) -> { //lambdas syntax

System.out.println(x);

};

ex.test1(100); // calls lambdas

}

}

* Example 7:

package p1;

//lambdas example 3 with argument

public interface Ex10 {

public void test1(int x);

}

package p1;

public class Ex10\_1 {

public static void main(String[] args) {

Ex10 ex = (int x) -> System.out.println(x);

// single line lambdas syntax when only one

//sentence is present no need to use {}

ex.test1(100); // calls lambdas

}

}

* Example 8:

package p1;

// Example 4 where method is non void

public interface Ex11 {

public int test1();

}

package p1;

public class Ex11\_1 {

public static void main(String[] args) {

Ex11 ex = () -> {return 100;} ;

// single line lambdas syntax when only one

//sentence is present no need to use {}

int val = ex.test1();

System.out.println(val);

}

}

* Example 9:

package p1;

// Stream API> will be studied in collectisons.

public interface Ex12 {

public int test1();

}

**Core Java class 18:**

-2- Topics.

1. Abstract class:

* Example 1:

package p1;

/\*Abstract classes in java:

\* 1) an abstract class can consist of complete and incomplete methods.

\* 2) to create incomplete method it is mandatory to use abstract keyword.

\* 3) an abstract class can be 0-100% incomplete.

\* 4) an abstract class can consist of main method.

\* 5) reference variable of abstract class can be created but an object

\* of an abstract class cannot be created.

\* \*/

abstract public class Ex1 {

static int y = 20;

int x = 30;// only by inheritance we can access it

public abstract void test();

public void test2() { // only by inheritance we can access it

System.out.println(200);

}

public static void main(String[] args) {

// A a1 /\*can create reference variable\*/ /\* =new A();\*/ //error

System.out.println(Ex1.y);

}

}

* Example 2:

package p1;

/\*2nd example for abstraction of non static members\*/

abstract public class Ex2 {

int x= 20;

public void test2() {

System.out.println(2000);

}

}

package p1;

public class Ex2\_1 extends Ex2 {

public static void main(String[] args) {

Ex2\_1 a = new Ex2\_1();

System.out.println(a.x);

a.test2();

}

}

* Example 3:

package p1;

/\*

\* Example of both complete and incomplete methods.

\* \*/

abstract public class Ex3 {

int x= 20;

public void test2() {

System.out.println(2000);

}

public abstract void test1();

}

package p1;

public class Ex3\_1 extends Ex3 {

public static void main(String[] args) {

Ex3\_1 a = new Ex3\_1();

System.out.println(a.x);

a.test2();

a.test1();

}

@Override

public void test1() {

System.out.println(100);

}

}

* Example 4:

package p1;

// interface to abstract class inheritance.

public interface Ex4 {

public void test();

}

package p1;

abstract public class Ex4\_1 implements Ex4 {

// as abstract class can contain both complete and incomplete class

// we don't have to override mandatory.

}

* Example 5:

package p1;

/\*abstract classes does not support multiple inheritance \*/

public abstract class Ex5 {

public static void main(String[] args) {

}

}

package p1;

public abstract class Ex5\_1 {

}

package p1;

public class Ex5\_2 extends Ex5 /\*, Ex5\_1\*/ {

// no multiple inheritance is supported.

}

* Example 6:

package p1;

/\*difference between interface and abstract class

\* interface:

\* 1) they are 100% incomplete.

\* 2) they support multiple inheritance.

\* abstract class:

\* 1) they can be 0-100 % incomplete.

\* 2) they do-not support multiple inheritance.

\* \*/

/\*

\* what is abstraction?

\* Answer - hiding of implementation details is called as abstraction, the

\* way we achieve that in java is using interfaces and abstract

\* class concept.

\*

\* Note: data hiding: making variable private.

\* \*/

public class Ex6 {

}

1. Exception:

* Example 1:

package p1;

/\* Exception:

\* 1)whenever we a bad user input is given, program

\* execution will halts abruptly.

\* 2) to handle exception we use try catch block, if any line of code

\* in try block causes exception then try block will create in Exception

\* block, and its object address try block will give it to catch block,

\* catch block will now suppress the exception and hence the further

\* code will continue execution.

\* \*/

public class Ex7 {

public static void main(String[] args) {

try {

int x = 20;

int y = 0;

int z = x / y;// after exception no code will run

System.out.println(z);// will not run

System.out.println("hello"); // will not run

} catch (Exception e) {

System.out.println(e);

}

System.out.println(100);// will run

}

}

* Example 2:

package p1;

//e.printStackTrace() will give exact line number where exception happens.

public class Ex8 {

public static void main(String[] args) {

try {

int x = 20;

int y = 0;

int z = x / y;// after exception no code will run

System.out.println(z);// will not run

System.out.println("hello"); // will not run

} catch (Exception e) {

e.printStackTrace();

// will give exact line number where exception happens

}

System.out.println(100);// will run

}

}

**Core Java class 19:**

-2- Topics.