**NON STATIC MEMBERS AND THEIR CONTROL FLOW**

**What members are called non – static members?**

Class level members which do not have static keyword in their creation statement are called non static members.

**Types of non – static members**

1. Non Static Variables
2. Non Static methods
3. Non Static Block
4. Constructors

**When do all these members get memory location and why?**

All the above members gets memory location only if object is created with new keyword and constructor of that class.

**Non Static Variables**

The class level variables that does not have static keyword in its definition is called non static variable.

**When, where, how and by whom non static variables get memory location?**

Non Static variables get memory location in heap area in continuous memory locations by JVM when object is created.

**Program defining a class with non – static members, creating object**

class Example

{

int a = 10;

int b = 20;

public static void main(String[] args)

{

Example e = new Example();

}

}

**Draw JVM architecture**

Suppose 1010 is the object reference. It is also called object’s base address.

**Note**

Object reference is returned by **new** keyword, not by constructor.

**What is the output printed if we print referenced variable / object?**

If we print null reference variable, print() or println() methods prints null.

If we print object referenced variable, print() or println() methods prints

**classname@hashCode (in hexadecimal String format)**

**What is hash code?**

Hash code is a 32 – bit unsigned integer value.

Hash Code of a java object is simply a number that allows an object to be managed by hash based data structure.

Hash Code is mapped with address of the object.

We can’t find address of object in java.

**Program to get hashCode**

class Example

{

public static void main(String args[])

{

Example e = new Example();

System.out.println(e);

System.out.println(e.hashCode());

System.out.println(Integer.toHexString(e.hashCode()));

}

}

Output

Example@1db9742

31168322

1db9742

**When we print object, how it’s hash code is printed in its hexa string format?**

It is the internal implementation of **toString()** method.

**A small introduction to toString() method**

It is a pre – defined method available in **java.lang.Object** class to return object information in string format. It is called internally by **print()** or **println()** to print object.

**Program to demonstrate the above concept**

class Example

{

public String toString()

{

return getClass().getName() + "@" + Integer.toHexString(hashCode());

}

public static void main(String args[])

{

Example e = new Example();

System.out.println(e);

}

}

**How can we print object state when we print object?**

Overriding toString() method

class Example

{

int eid = 101;

String name = "Ashu Akash Singh";

String company = "Set Academy";

public String toString()

{

return "eid: " + eid + "\n" +

"name: " + name + "\n" +

"company: " + company;

}

public static void main(String args[])

{

Example e = new Example();

System.out.println(e);

}

}

**Can we access non static variable or non – static method with null referenced variable?**

No, it leads to RE: **java.lang.NullPointerException**.

**Can we access static variable or static method with null referenced variable?**

Yes, there is no CE or RE, program is executed successfully, because static variable or static methods gets memory directly with respect to the class.

**Static variables can be accessed using**

1. Its name directly
2. Class name
3. Null referenced variable
4. Object referenced variable means object

**Non – static variables can be accessed using**

1. Object referenced variable means object only

**Note**

1. If non – static variables are accessed directly by their names or by using class name, then it leads to CE: non – static variable cannot be referenced from a static context.
2. If non – static variables are accessed by null referenced variable, then it leads to RE: java.lang.NullPointerException

**Find out CE and RE in this below program**

class Example

{

static int a = 10;

static int b = 20;

int x = 30;

int y = 40;

public static void main(String args[])

{

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

System.out.println("b: " + b);

System.out.println("Example.a: " + Example.a);

System.out.println("Example.b: " + Example.b);

Example e = new Example();

System.out.println("e.x: " + e.x);

System.out.println("e.y: " + e.y);

System.out.println("e.a: " + e.a);

System.out.println("e.b: " + e.b);

// System.out.println("Example.x: " + Example.x); // CE

// System.out.println("Example.y: " + Example.y); // CE

Example e1 = null;

System.out.println("e1.a: " + e1.a);

System.out.println("e1.b: " + e1.b);

// System.out.println("e1.x: " +e1.x); // Run time error

// System.out.println("e1.y: " + e1.y); // Run time error

}

}

**Modifying static and non – static variables**

If we modify static variables using one object that modification is affected to all objects, because all objects share same copy of static variable’s memory location.

If we modify non – static variables using one object that modification is not affected to other objects, because every object has its own copy of non – static variable’s memory location.

class MemberAccessDemo

{

static int a = 10;

int b = 20;

public static void main(String[] args)

{

MemberAccessDemo mad1 = new MemberAccessDemo();

MemberAccessDemo mad2 = new MemberAccessDemo();

mad1.a = 30;

System.out.println("mad1.a: " + mad1.a);

System.out.println("mad2.a: " + mad2.a);

System.out.println("MemberAccessDemo.a: " + MemberAccessDemo.a);

mad1.b = 40;

System.out.println("mad1.b: " + mad1.b);

System.out.println("mad2.b: " + mad2.b);

}

}

Output

mad1.a: 30

mad2.a: 30

MemberAccessDemo.a: 30

mad1.b: 40

mad2.b: 20

**Non – Static Methods**

The method which does not have static keyword in its prototype is called non – static method. Like non – static variables, non – static methods also should be called only by using object.

If non – static methods called directly by their names, then it leads to CE: non – static method cannot be referenced from static context.

class Example

{

void m1()

{

System.out.println("In m1() method");

}

public static void main(String args[])

{

System.out.println("In main method");

m1(); // CE

}

}

class Example

{

void m1()

{

System.out.println("In m1() method");

}

public static void main(String args[])

{

System.out.println("In main method");

Example e = new Example();

e.m1();

}

}

**How can we call non – static members from other non – static members?**

Directly by their name

class Example

{

int x = 10;

int y = 20;

void m1()

{

System.out.println("In m1() method");

System.out.println("x: " + x);

System.out.println("y: " + y);

m2();

}

void m2()

{

System.out.println("In m2() method");

}

public static void main(String args[])

{

Example e = new Example();

e.m1();

}

}

**What is the object structure of non – static method? Will non – static method get separate memory for each object like variable?**

No, every object doesn’t have separate copy of non – static method, rather all objects share same copy of non – static method logic.

class Example

{

int x = 10;

int y = 20;

void printXY()

{

System.out.println("x: " + x);

System.out.println("y: " + y);

}

public static void main(String args[])

{

Example e1 = new Example();

Example e2 = new Example();

e2.x = 30;

e2.y = 40;

e1.printXY();

e2.printXY();

}

}

Draw JVM architecture.

**Conclusion**

In a non – static context, if non – static variables are called directly by their name, JVM reads their values from the current object.

**What is current object?**

The object that is used to call a non – static method is called current object.

**Since method logic is shared by all objects, how can JVM retrieve object specific non – static members?**

Using this keyword.

**Local object creation**

**Can we create object in non – static method?**

Yes, we can also create object in non – static method. Such object is called local object.

class Example

{

int x = 10;

int y = 20;

void m1()

{

Example e2 = new Example();

e2.x = 30;

e2.y = 40;

}

public static void main(String args[])

{

Example e1 = new Example();

System.out.println(e1.x + " " + e1.y);

e1.m1();

System.out.println(e1.x + " " + e1.y);

}

}

Draw JVM architecture

**How many referenced variable can an object has pointing to it?**

More than one.

class Example

{

int x = 10;

int y = 20;

public static void main(String args[])

{

Example e1 = new Example();

Example e2 = new Example();

Example e3 = e2;

e2.x = 30;

e3.y = 40;

System.out.println(e1.x + " " + e1.y);

System.out.println(e2.x + " " + e2.y);

System.out.println(e3.x + " " + e3.y);

}

}

Draw JVM architecture

**Can we pass object as an argument to a method?**

Yes, we can pass object as an argument to a method.

To pass an object as an argument to a method, the method parameter type should be that object’s class type. When we pass object as an argument to a method, its reference is passed but not object memory, that reference is stored in parameter memory.

class Example

{

int x = 10;

int y = 20;

void m1(Example e)

{

System.out.println("In m1() before modification");

System.out.println(x + " " + y);

x = 30;

y = 40;

System.out.println("in m1() after modification");

System.out.println(x + " " + y);

}

public static void main(String args[])

{

Example e1 = new Example();

Example e2 = new Example();

System.out.println("e1 object value before calling method");

System.out.println(e1.x + " " + e1.y);

System.out.println("e2 object value before calling method");

System.out.println(e2.x + " " + e2.y);

e1.m1(e2);

System.out.println("e1 object value after calling method");

System.out.println(e1.x + " " + e1.y);

System.out.println("e2 object value after calling method");

System.out.println(e2.x + " " + e2.y);

}

}

Draw JVM architecture

So, till the method execution is completed **current** object and **argument** objects are pointed by two referenced variables. As we learned already, when an object is pointed by multiple referenced variable, if it is modified by one referenced variable other referenced variables are also get the same modified values.

**Can we call a method by passing same object as current object and argument object?**

Yes, we can call method by passing same object as current object and also argument object, provided that method is defined in that current class object’s referenced variable’s type class and with the same class type parameter.

class Example

{

int x;

int y;

void m1(Example e)

{

x = x + 1;

y = y + 2;

e.x = e.x + 3;

e.y = e.y + 4;

}

public static void main(String args[])

{

Example e1 = new Example();

Example e2 = new Example();

e1.m1(e2);

System.out.println(e1.x + " " + e1.y); // 1 2

System.out.println(e2.x + " " + e2.y); // 3 4

e2.m1(e1);

System.out.println(e1.x + " " + e1.y); // 4 6

System.out.println(e2.x + " " + e2.y); // 4 6

e1.m1(e1);

System.out.println(e1.x + " " + e1.y); // 8 12

System.out.println(e2.x + " " + e2.y); // 4 6

e2.m1(e2);

System.out.println(e1.x + " " + e1.y); // 8 12

System.out.println(e2.x + " " + e2.y); // 8 12

}

}

Draw JVM architecture

**Note**

1. Java does not support pass by address, because it does not support pointers.

**Introduction to Non Static Block**

A class level block which doesn’t have prototype is called non – static block.

**Syntax**

{

...................

...................

}

All legal statements except return statement and throw keyword are allowed in non – static block.

**Need of NSB**

We should define non – static block to execute some logic only at the time of object creation irrespective of the constructor used in object creation.