**---untill anything(data) loaded into the RAM (primary memory) it will not be available to the CPU for the execution.**

**static loading :- a block of code would be loaded into the RAM before it is executed, after being loaded into the RAM it may or may not get executed.**

**dynamic loading:- a block of code would be loaded into the RAM only when it is required to be executed.**

**Note: - static loading take place in the execution of structural programing languge, like c program,**
**where as Java follows dynamic loading.**

**ex:-**

**c program:{**

**1000 variable**
**10000 methods/function**

**}**

**app1.c ---while execution, entire application will be loaded into the RAM**

**ex2: Java application**

**class Demo{**

**1000 variables**

**10000 methods/function**

**}**

**Demo.java -----> compile(java compiler)----->Demo.class---while executing ,**
**only required part will be loaded into the RAM, remaining part will be there inside the harddisk in the form bytecode.**

**--when even it is required , at that time we can load other part of the application into the RAM dynamically and execute them.**

**--here the required parts are static members.**

**1. how many members are there in a class:**

**a. variables**
**b. methods**

**--after counting the members inside a class, we need to identify the types of the members ( i.e static or non-static)**

**--the members where 'static' keyword is applied is known as static member and for which it is not applied is known as non-static.**

**--while executing a java application, all the static members will be loaded into the RAM and they will be available to the CPU , where as non-static will be there inside the harddisk in the form of byte code.**

**--to access the non-static members in main method, we need to load the non-static members inside the RAM dynamically by creating object of that class.**

**requirement of creating an object: -- to load non-static memebers of a class into the RAM dynamically we need to create instance/object of that class.**

**Ex:**

**Demo.java:**
**----------**

```
package com.masai;

public class Demo {

	//instance variable
	int i=100;

	//static or class variable
	static  int j =200;

	//non-static
	void fun1() {

		System.out.println("inside fun1 of Demo");
		//10000
	}
```

```
	static void fun2() {
		System.out.println("inside fun2 of Demo");

	}


	public static void main(String[] args) {

		//local variable
		//int x= 10;

		//System.out.println(j);

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

		//creating object of a class
		//instantiating a class
		Demo d1 = new Demo();

		System.out.println(d1);

		System.out.println(d1.i);

		d1.fun1();

		//fun1();

		//fun2();


	}

}
```

**functionality of 'new' operator:**
**==========**

**1. whenever jvm encounters 'new' operator, it will reserve a memeory space inside the heap area of RAM.**

**2. inside that memory space, it will load all the non-static members of that class (variable + methods)**

*****while loading the methods, it will load only the method name/signature and its address,**
**it will not load method body, method body will be loaded at run time when we call that method.**

*****while loading the non-static variables, inside that reserved memory space, if variable are not initialized then jvm will provide the default value to those non-static variables.**

**Note: default value will not be given to the local variables, we can not use uninitialized local variable in our application, it will raise an compilation error.**

**Note: we can create multiple objects for a perticular class,and modification done in one obj will not reflect another object.**

*****technically, an obj of a class is a memory space inside the heap area where non-static elements are loaded.**

**Demo d1 = ?**

**--for a class ref variable 3 possible values are there:**

**1. its own class object.**

**Demo d1 = new Demo();**

**2. its child class object:-**

**Demo d1 = new DemoChild(); // it is possible only if DemoChild class is the child class of Demo class.**

**--this concept is also known as super class ref and child class object.**

**3. default value for any ref variable. i.e null.**

**Demo d1 = null;**

**example:**

**Demo.java:**

**----------**

```
package com.masai;

public class Demo {

	//instance variable
	 int i;

	//static or class variable
	static  int j =200;

	//non-static
	 void fun1() {

		System.out.println("inside fun1 of Demo");
		//10000
	}

	static void fun2() {
		System.out.println("inside fun2 of Demo");

	}


	public static void main(String[] args) {



		Demo d1 = new Demo();//here d1 is a reference variable, d1 is pointing to the Demo class obj.

		d1.i=100;

		System.out.println(d1.i);//100

		Demo d2 = d1;

		System.out.println(d2.i); //100

	}

}
```

**Note: one obj(memory space) can be refered by multiple variables simulteniously, but one variable can not refer multiple object simulteniously.**

**--the obj which is not referenced by any ref variable will be treated as garbage, and in Java there is a seperate thread running continiusly, called Garbage collector, the duty of this garbage collector is to kill that un-referenced obj and free the RAM.**

**--if any one variable holding the address of any object then that object is not treated as garbage.**

**--from the null, if we try to access any value(non-static) or call any method(non-static), then we will get a runtime exception called NullPointerException.**

**example:**

```
Demo d1 = new Demo();//here d1 is a reference variable, d1 is pointing to the Demo class obj.

        d1.i=100;

        Demo d2 = d1;

        d1= null;

        System.out.println(d2.i);//100

        // till this line no any obj is eligible for GC.
```

**primitive variables having thier seperate copies:**

**ex:**

```
        int x = 10;

        int y =x;

        System.out.println(x); // 10
        System.out.println(y); //10
```

```
x = 200;

System.out.println(x); // 200
System.out.println(y); // 10
```

**where as object if reassigned to diff variables then all variables will points to the  s same copy.**

**ex2:**

```
Demo d1 = new Demo();

Demo d2 = d1;

System.out.println(d1.i); //0
System.out.println(d2.i); //0

d1.i=500;

System.out.println(d1.i); //500
System.out.println(d2.i); //500
```

**ex3:**

```
Demo d1 = new Demo();

Demo d2 = d1;

System.out.println(d1.i); //0
System.out.println(d2.i); //0

d2.i=500;

System.out.println(d1.i); //500
System.out.println(d2.i); //500
```

**example :**

```
Demo d1 = new Demo();
```

```
        d1.fun1();

        //another approach of calling a method
        new  Demo().fun1();

        d1.fun1();

        d1.fun1();

        System.out.println(d1.i);
```

**State of an object:  data present inside that object at that instance of time is known(what an obj knows)**
**as state of the object.**

**behaviour of an object: functionality that are applicable to that object (what an object can do) is known as behaviour of an object**

**Song.java:**
**----------**

```
package com.masai;

public class Song {

    String artist;
    String title;

    void play() {

        System.out.println(artist+" is singing "+title);
    }


    public static void main(String[] args) {

        Song track1 = new Song();
        track1.artist="Lata";
        track1.title="Wande Matram";

        track1.play();
```

```
        Song track2 = new Song();
        track2.artist="Sukhwindar";
        track2.title="Jai Ho";

        track2.play();

    }

}
```

**Note: we can have an empty class also. and we can generate the .class file also for this class:**

**ex:**

**A.java:**
**----------**

```
public class A {

}
```

**A.java ---- > A.class**

**Has-A relationship:**
**==========**

**inside a class, as a instance member if we define any other class object then it is called as a Has-A relationship.**

**ex:**

**A.java:**
**----------**

```
package com.masai;

public class A {

    int i = 10;
```

```
	void funA() {

		System.out.println("inside funA of A");
	}


}
```

Demo.java:

```
---------

package com.masai;

public class Demo {

	int x = 100;

	A a1 = new A();


	public static void main(String[] args) {

		Demo d1 = new Demo();
		System.out.println(d1);//address of Demo obj


		System.out.println(d1.x);//100

		System.out.println(d1.a1);//address of A obj

		d1.a1.funA();



	}

}
```

You problem:

---

```
package com.masai;

public class Demo {

	int x = 100;

	Demo d1 = new Demo();


	public static void main(String[] args) {

		Demo d1 = new Demo();

		System.out.println(d1);
	}

}
```