Initialization order of non static field (Object Properties) :

* The following steps describe the **order of initialization of non static fields during object creation :**

1) With the help of Java compiler, new keyword will allocate the memory and initialized all the non static fields with default value.

2) During Object creation, appropriate constructor will be invoked.

3) The first line of constructor which contains super() [Among all the constructors, One constructor will definetly contain super()] will call Object class (super class) constructor (Init() method OR invokespecial command)

4) IN THE 2ND LINE OF THE CONSTRUCTOR BODY, **non static variable declaration cum initialization** OR non static block both are **having same priority as shown below :**

```
Case 1 :
--------
class Test
{
    {
        x = 300;
    }

    int x = 200;
}

public class ConstructorTest
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        SOP.println(t1.x);  //200
    }
}

Case 2 :
--------
class Test
{
    int x = 200;

    {
        x = 300;
    }
}

public class ConstructorTest
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        SOP.println(t1.x);
    }
}
```

Lab Program [Static Factory Method + Object reference as a parameter] :

A BLC class called Customer is given to you.

The task is to find the applicable Credit card Type and create CardType object based on the Credit Points of a customer.

Define the following for the class.

Attributes :
customerName : String,private
creditPoints: int, private

Constructor :
parameterizedConstructor: for both cuastomerName & creditPoints in that order.

Methods :
Name of the method : getCreditPoints
Return Type :int
Modifiers: public
Task : This method must return creditPoints

Name of the method : toString, Override it,
Return Type : String
Task : return only customerName from this.

Create another class called CardType. Define the following for the class

Attributes :
customer : Customer, private
cardType : String, private
Constructor :
parameterizedConstructor: for customer and cardType attributes in that order

Methods :
Name of the method : toString Override this.
Return type : String
Modifiers: public
Task : Return the string in the following format.
The Customer 'Rajeev' Is Eligible For 'Gold' Card.

Create One more class by name CardsOnOffer and define the following for the class.

Method :
Name Of the method : getOfferedCard
Return type : CardType [static Factory Method]
Access Modifiers: public
Other Modifiers : public static
Parameter : Customer object

Task : Create and return a CardType object after logically finding cardType from creditPoints as per the below rules.
creditPoints cardType
100 - 500 - Silver
501 - 1000 - Gold
> 1000 - Platinum
< 100 - ONE

Create an ELC class which contains Main method to test the working of the above.

```
//Program:
--------
package com.ravi.scenario_based_program;

public class Customer
{
    private String customerName;
    private int creditPoints;

    public Customer(String customerName, int creditPoints)
    {
        super();
        this.customerName = customerName;
        this.creditPoints = creditPoints;
    }

    //getter
    public String getCustomerName()
    {
        return this.customerName;
    }


    //getter
    public int getCreditPoints()
    {
        return this.creditPoints;
    }

    public String toString()
    {
        return this.customerName;
    }

}

package com.ravi.scenario_based_program;

public class CardType
{
    private Customer customer;  //HAS-A relation
    private String cardType;

    public CardType(Customer customer, String cardType)
    {
        super();
        this.customer = customer;
        this.cardType = cardType;
    }

    @Override
    public String toString()
    {
        return "The Customer '"+customer.getCustomerName()+"' Is Eligible For '"+this.cardType+"' Card.";
    }

}

package com.ravi.scenario_based_program;

public class CardsOnOffer
{
    public static CardType getOfferedCard(Customer ob)
    {
        int creditPoint = ob.getCreditPoints();

        if(creditPoint >=100 && creditPoint <=500)
        {
            return new CardType(ob,"Silver");
        }
        else if(creditPoint >500 && creditPoint <=1000)
        {
            return new CardType(ob,"Gold");
        }
        else if(creditPoint > 1000)
        {
            return new CardType(ob,"Platinum");
        }
        else
        {
            return new CardType(ob,"ONE");
        }
    }
}

package com.ravi.scenario_based_program;

public class CreditCardEligibility
{
    public static void main(String[] args)
    {
        String name = SIO.readLn("Enter Customer Name :");
        int creditPoint = Integer.parseInt(SIO.readLn("Enter Credit Points :"));

        Customer cust = new Customer(name, creditPoint);
        CardType cardType =  CardsOnOffer.getOfferedCard(cust);
        SIO.println(cardType);
    }

}
```

JVM Architecture with Class Loader Subsystem :

* The main purpose of JVM to LOAD & EXECUTE the byte code i.e .class file.

* The entire JVM Architecture is divided into 3 parts :

1) Class Loader subsystem
2) Runtime Data Areas (Memory Areas)
3) Execution Engine [Interpreter + JIT compiler]

1) Class Loader subsystem :

* The main purpose of class loader subsystem to load the required .class file into JVM memory from different memory location.

* In order to load the .class file into JVM memory, It uses one algorithm called "Delegation Hierarchy Algorithm".

* Internally, Class loader subsystem performs the following tasks :

1) LOADING
2) LINKING
3) INITIALIZATION