Another program on parameterized constructor to initialize the non static variable :

```
package com.ravi.constructor;

//BLC
public class Person
{
    private String name;
    private int age;

    public Person(String name, int age)
    {
        super();
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString()
    {
        return "Person [name=" + name + ", age=" + age + "]";
    }
}
```

```
package com.ravi.constructor;

//BLC
public class PersonDemo
{
    public static void main(String[] args)
    {
        Person smith = new Person("Smith", 22);
        System.out.println(smith);
    }

}
```

How to write setter and getter for non static variable ?

```
public class Product
{
  private double price;

  public Product(double price) //Parameterized Constructor to initialize the non static variable price
  {
    this.price = price;
  }

  public void setPrice(double price) //setter [To modify the existing product price]
  {
    this.price = price;
  }

  public double getPrice() //getter [To read the private data value outside of the BLC class]
  {
    return this.price;
  }
}
```

Points :
---

* Setter and getter we can write for non static variables, setter will **modify** the existing Object data on the other hand getter will **read** the private data value outside of the BLC class.

* We should always write a separate pair of setter and getter for each and every non static variable.

* Setter method return type is always void

* Getter method return type is always non void, Actually it depends upon variable data type.

Now while writing the Program the final conclusion is :
---

1) Parameterized Constructor : To initialize the non static variable
2) Setter : To modify the existing Object Data
3) Getter : To read the private data value
4) toString() : To print the Object properties.

```
package com.ravi.setter_getter;

public class Product
{
    private double price;

    public Product(double price)
    {
        super();
        this.price = price;
    }

    public double getPrice()
    {
        return this.price;
    }

    public void setPrice(double price)
    {
        this.price = price;
    }

    @Override
    public String toString() {
        return "Product [price=" + price + "]";
    }

}
```

```
package com.ravi.setter_getter;

import java.util.Scanner;

public class ProductDemo {

    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the product Price :");
        double price = sc.nextDouble();

        Product p1 = new Product(price);
        System.out.println(p1);

        System.out.print("Enter the updated Product Price :");
        double newPrice = sc.nextDouble();

         p1.setPrice(newPrice);
         System.out.println(p1);

    }

}
```

**** Encasulation : [Accessing the **private data via public methods like setter and getter**]

Java Encapsulation

Binding the private data with its associated method in a single unit is called Encapsulation.

Encapsulation ensures that our private data (Object Properties) must be accessible via public methods like setter and getter.

It provides security because our data is private (Data Hiding) and it is only accessible via public methods WITH PROPER DATA VALIDATION.

In java, class is the example of encapsulation.

How to achieve encapsulation in a class :

In order to achieve encapsulation we should follow the following two techniques :

1) Declare all the data members with private access modifiers (Data Hiding OR Data Security)

2) Write public methods to perform read(getter) and write(setter) operation on these private data like setter and getter.

Note : If we decalre all our data with private access modifier then it is called TIGHTLY ENCAPSULATED CLASS. On the other hand if we declare our data other than private access modifier then it is called Loosely Encapsulated class.

```
package com.ravi.encapsulation;

public class Employee
{
    private String name;
    private double salary;

    public Employee(String name, double salary)
    {
        if(name ==null && salary <=0)
        {
            System.err.println("Error");
            System.exit(0);
        }

        this.name = name;
        this.salary = salary;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public double getSalary()
    {
        return salary;
    }

    public void setSalary(double salary)
    {
        if(salary > this.salary)
        {
            this.salary = salary;
        }
        else
        {
            System.err.println("Negative Increment Error");
            System.exit(0);
        }
    }
    @Override
    public String toString()
    {
        return "Employee [name=" + name + ", salary=" + salary + "]";
    }

}
```

```
package com.ravi.encapsulation;

import java.util.Scanner;

public class EmployeeDemo
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the Employee Name :");
        String name = sc.nextLine();
        System.out.print("Enter the Employee Salary :");
        double salary = sc.nextDouble();

        Employee raj = new Employee(name, salary);
        System.out.println(raj);

        System.out.print("Enter the incremented Amount :");
        double increment = sc.nextDouble();

        raj.setSalary(raj.getSalary() + increment);
        System.out.println(raj);

        //Some Criteria whether the Employee is Developer OR Designer OR Tester

         double updatedSalary = raj.getSalary();

         if(updatedSalary >= 100000)
         {
             System.out.println(raj.getName()+" is a Developer!!!");
         }
         else if(updatedSalary >= 50000)
         {
             System.out.println(raj.getName()+" is a Designer!!!");
         }
         else
         {
             System.out.println(raj.getName()+" is a Tester!!!");
         }
         sc.close();
    }

}
```