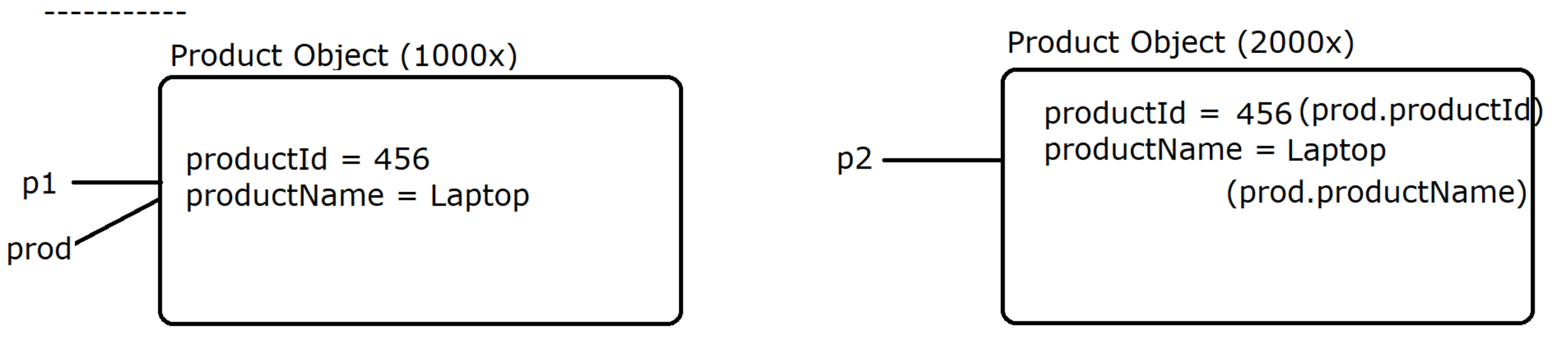
* In the following program the purpose is to copy the content of one object to another object of same type.

Diagram :




```java
package com.ravi.copy_constructor;

class Product
{
    private int productId;
    private String productName;

    public Product(int productId, String productName)
    {
        super();
        this.productId = productId;
        this.productName = productName;
    }

    public Product(Product prod)  //prod = p1
    {
        this.productId = prod.productId;
        this.productName =prod.productName;
    }

    @Override
    public String toString() {
        return "Product [productId=" + productId + ", productName=" + productName +
"]";
    }
}
public class CopyConstructorDemo1 {

    public static void main(String[] args)
    {
        Product p1 = new Product(456, "Laptop");
        System.out.println("Original Data :"+p1);

        Product p2 = new Product(p1);
        System.out.println("Copied Data :"+p2);
    }
}
```

Scenario Based Program (Factory Method + Passing an object reference to the method)

Assignment :

A 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 cusotmerName & creditPoints in that order.

Methods :
    Name of the method : getCreditPoints
    Return Type : int
    Modifier   : 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
    Modifier : 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
    Modifiers: public,static
    Arguments: 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	-	EMI

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

```java
package com.ravi.scenario;

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

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

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

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

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

```java
package com.ravi.scenario;

public class CardType
{
    private Customer customer;
    private String cardType;

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

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

    }
}
```

```java
package com.ravi.scenario;

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

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

```java
package com.ravi.scenario;

import java.util.Scanner;

public class CreditCard {

    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Customer Name :");
        String name = sc.nextLine();
        System.out.print("Enter your Credit Points :");
        int creditPoint = Integer.parseInt(sc.nextLine());

        Customer cust = new Customer(name, creditPoint);

         CardType offeredCard = CardsOnOffer.getOfferedCard(cust);
        System.out.println(offeredCard);
        sc.close();
    }

}
```

Constructor Overloading :
----
* We can write multiple constractors in the BLC class but parameter must be different, If parameters are different then it is called **Constructor Overloading**

```java
public class Overload
{
    public Overload()
    {
    }

    protected Overload(int x)
    {
    }
}
```

* THE FIRST LINE OF ANY CONSTRUCTOR IS ALWAYS MEANT FOR CALLING ANOTHER CONSTRUCTOR EITHER FROM THE SUPER CLASS OR FROM THE CURRENT CLASS. We should always write this() [this of] OR super() [super of] to call the constractor of same class or super class respectively.

  super() : To call Parent class Constructor
  this() : To call current class Overloaded constructor

* In order to call mutiple constructor in the BLC class we need to create only one object and with the help of this() we call multiple overloaded constructor.

* this() and super() both we can use only in constructor as a first statement, We cannot use in method to call a constructor

```java
package com.ravi.constructor_overloading;

class Calculate
{
   public Calculate(int x)
   {
      this(1000,2000);
      System.out.println("Single parameter Constructor :"+x);
   }

   public Calculate(int x, int y)
   {
      this("Java"," is "," best ");
      System.out.println("Two parameterized Constructor :"+(x+y));
   }


   public Calculate(String x, String y, String z)
   {
      System.out.println("Three parameterized Constructor :"+(x+y+z));
   }
}
public class ConstructorOverloading
{
   public static void main(String[] args)
   {
       new Calculate(100); //Anonymous OR Nameless Object
   }
}
```

Modifiers on Constructor :
----
* We can apply all the access modifiers like private, default (private-package), protected and public to the constructor.

* A constructor cannot be decalred as

  a) static : [It is used for object creation]
  b) final : [Constructors are not inherited in java so it is not available in the child class so final is not applicable]
  c) abstract : [abstract method does not have method body]
  d) synchronized : [For accepting a single thread]

Why we should declare a constructor with private access modifier :

We should decalre the constructor with private access modifier in the following two cases :

1) When we designing factory methods [Singleton class, only one object is possible]

```java
package com.ravi.constructor_overloading;

class Singleton
 {
    private Singleton()
    {
        System.out.println("Private Constructor");
    }

    public static Singleton getInstance()
    {
       return new Singleton();
    }


    public void accept()
    {
        System.out.println("Singleton class non static method");
    }
}

public class Main
{
   public static void main(String[] args)
   {
      Singleton obj = Singleton.getInstance();
        obj.accept();
   }

}
```

2) If we want to decalre only static variables and static methods in my class then we can decalre the constructor with private access modifier so external object creation is not possible.

Note : java.util.Arrays and java.lang.Math both contain private constructor because both are having only static variable and static methdos