# OBJECT ORIENTED PROGRAMMING WEEK 4

13 FEB-17 FEB, 2023

Instructor:

**Abdul Aziz**
Assistant Professor
(School of Computing)
National University- FAST (KHI Campus)

## ACKNOWLEDGMENT

- Publish material by Virtual University of Pakistan.

- Publish material by Deitel & Deitel.

- Publish material by Robert Lafore.

## Problems

- How can you create a **constant**?
- How can you declare **data** that is **shared by all instances of a given class**?
- How can you **prevent** a class from being **subclassed**?
- How can you **prevent** a method from being **overridden**?

## Problem

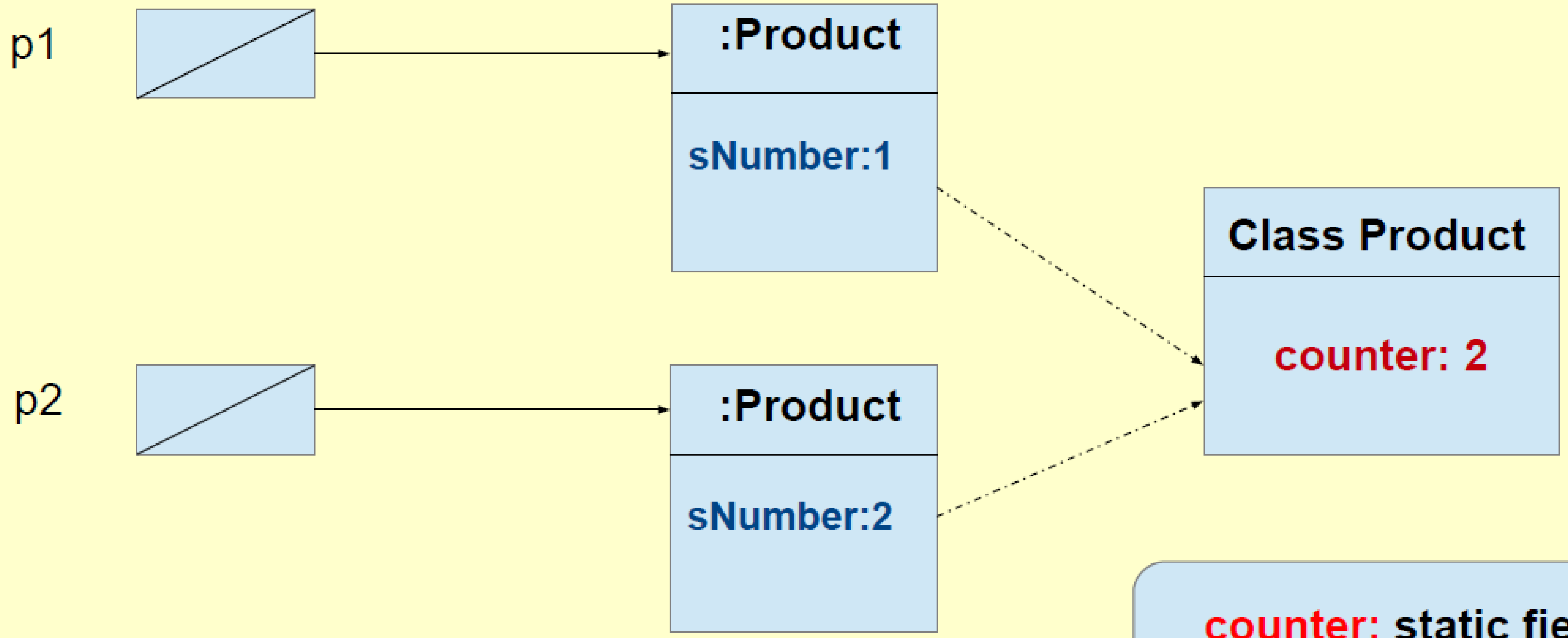
- Create a `Product` class which initializes each new instance with a `serialNumber (1,2, 3,…)`

## Solution

```
public class Product{
    private int sNumber;
    public static int counter = 0;
    public Product() {
        counter++;
        sNumber = counter;
    }
 }
```

## Solution

```
Product p1 = new Product();
Product p2 = new Product();
```



counter: static field
sNumber: instance field

## What's wrong?

```
public class Product{
    private int sNumber;
    public static int counter = 0;
    public Product() {
        counter++;
        sNumber = counter;
    }
}
```

It can be accessed from outside the class!

```
public class AnyClass{
    public void increment() {
        Product.counter++;
    }
}
```

## Better solution

```
public class Product{
    private int sNumber;

    private static int counter = 0;

    public static int getCounter(){
        return counter;
    }

    public Product() {
        counter++;
        sNumber = counter;
    }
}
```

```
System.out.println(Product.getCounter());
Product p = new Product();
System.out.println(Product.getCounter());
```

**Output?**

## Accessing static members

Recommended:

```
<class name>.<member_name>
```

Not recommended (but working):

```
<instance_reference>.<member_name>
```

```
System.out.println(Product.getCounter());
Product p = new Product();
System.out.println(p.getCounter());
```

**Output?**

## Static Members

- Static data + static methods = static members
- Data are allocated at **class load time** → *can be used without instances*
- Instance methods **may use** static data. **Why?**
- Static methods **cannot use** instance data. **Why?**

## The `InstanceCounter` class

```
public class InstanceCounter {
   private static int counter;

   public InstanceCounter(){
       ++counter;
   }

   public static int getCounter(){
       return counter;
   }
}
```

Output?

```
System.out.println( InstanceCounter.getCounter());

InstanceCounter ic = new InstanceCounter();
     System.out.println( InstanceCounter.getCounter());
```

## Singleton Design Pattern

```
public class Singleton {
    private static Singleton instance;

    private Singleton(){
    }

    public static Singleton getInstance(){
        if( instance == null ){
            instance = new Singleton();
        }
        return instance;
    }
}
```

## STATIC INITIALIZERS

```
public class AClass{
private static int counter;
static  {
        }
}
```

## STATIC INITIALIZERS

- Static blocks in Java are executed automatically when the class is loaded in memory.
- Static blocks are executed only once as the class file is loaded to memory.
- Static blocks are executed before the main() method
- A class can have any number of static initialization blocks
- The execution of multiple static blocks will be in the same sequence as written in the program

## EXAMPLE

```
class Main{

    static {
        System.out.println("Static Block 1");
    }

    static {
        System.out.println("Static Block 2");
    }

    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```