Static Members

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

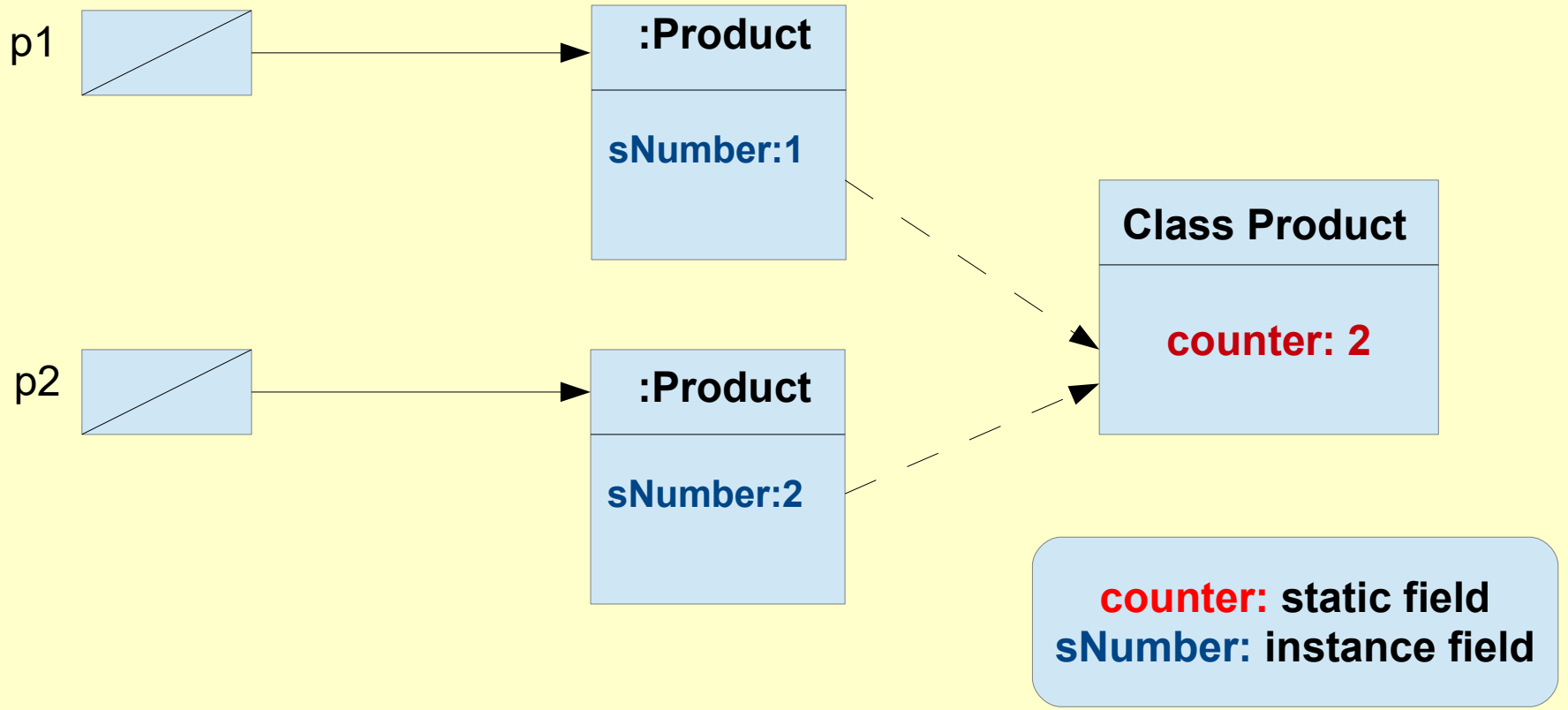
- Create a `Product` class which initializes each new instance with a `serialNumber (1,2, 3,…)`

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();
```



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;
    }
}
```

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 {
     // e.g. read counter from a file
  }
}
```

The `final` Keyword

- **Class**
  - You cannot subclass a `final` class.
- **Method**
  - You cannot override a `final` method.
- **Variable**
  - A `final` variable is a constant.
  - You can set a `final` variable only once.
  - Assignment can occur independently of the declaration (*blank final variable*).

Blank Final Variables

```
public class Employee{
    private final long ID;

    public Employee(){
      ID = createID();
    }

    private long createID(){
       //return the generated ID
    }
    …
}
```