Pola Desain Perangkat Lunak

[Week] 4 – Creational Pattern, Prototype

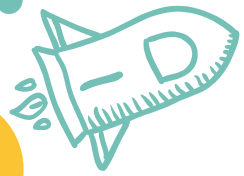
Prepared by: Tifanny Nabarian

Design Patterns Category

Creational Patterns	Creational patterns prescribe the way that objects are created.
Structural Patterns	• Structural patterns are concerned with how classes and objects are composed to form larger structures
Behavioral Patterns	• Behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects.
Concurrency Patterns	• Concurrency patterns prescribe the way access to shared resources is coordinated or sequenced

Design Patterns Scope

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	• Factory method	• Adapter	• Interpreter • Template method
	Object	• Abstract factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Fasad • Flyweight • Proxy	• Chain of responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

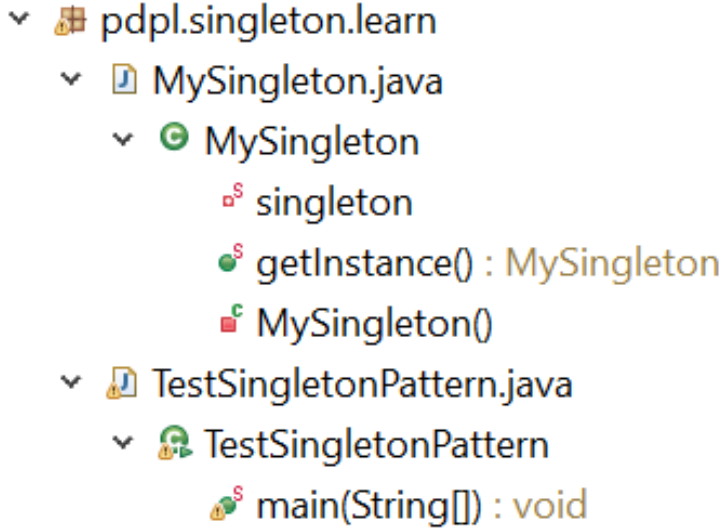


Creational Pattern

Review Singleton

Review Singleton

- Perhatikan implementasi singleton pada code berikut:



Review Singleton

- Perhatikan implementasi singleton pada code berikut:

```
public class MySingleton {
    private static MySingleton singleton = new MySingleton();

    private MySingleton(){
        System.out.println("object created...");
    }

    public static MySingleton getInstance() {
        return singleton;
    }
}
```

Eager Inializations

Review Singleton

- Perhatikan implementasi singleton pada code berikut:

```
public class TestSingletonPattern {
    public static void main(String[] args){

        MySingleton s1 = MySingleton.getInstance();
        MySingleton s2 = MySingleton.getInstance();
        MySingleton s3 = MySingleton.getInstance();

    }
}
```

Review Singleton

- Result:

```
3  public class TestSingletonPattern {
4      public static void main(String[] args){
5
6          MySingleton s1 = MySingleton.getInstance();
7          MySingleton s2 = MySingleton.getInstance();
8          MySingleton s3 = MySingleton.getInstance();
9
10     }
```

Problems @ Javadoc Declaration Console

```
<terminated> TestSingletonPattern [Java Application] C:\Program Files\Java\jre1.8
object created...
```

Eager Inializations VS Lazy Inializations

```
// Lazy initialization
if (captain == null)
{
    captain = new Captain();
    System.out.println("New captain is elected for your
    team.");
}
```

- In simple terms, lazy initialization is a technique through which you delay the object creation process. It says that you should create an object only when it is required. This approach can be helpful when you deal with expensive processes to create an object.

Eager Inializations VS Lazy Inializations

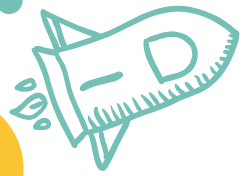
Eager Inializations

Pros

- It is straightforward and cleaner.
- It is the opposite of lazy initialization but still thread safe.
- It has a small lag time when the application is in execution mode because everything is already loaded in memory.

Cons

The application takes longer to start (compared to lazy initialization) because everything needs to be loaded first



Creational Pattern

Prototype

Prototype Concept

Definisi GoF

- Menentukan jenis objek yang akan dibuat menggunakan instance yang bersifat prototipe, dan membuat objek baru dengan menyalin prototipe ini.

Konsep

- Secara umum, membuat *instance* baru dari awal adalah operasi yang “mahal”. Menggunakan pola prototipe, Anda dapat membuat instance baru dengan menyalin atau mengkloning *instance* yang sudah ada. Pendekatan ini menghemat waktu dan uang untuk membuat instance baru dari awal.

هَلۡ أَتَىٰ عَلَى ٱلۡإِنسَٰنِ حِينٞ مِّنَ ٱلدَّهۡرِ لَمۡ يَكُن شَيۡـٔٗا مَّذۡكُورًا

Bukankah telah datang atas manusia satu waktu dari masa, sedang dia ketika itu belum merupakan sesuatu yang dapat disebut?

QS. Al-Insan Ayat 1

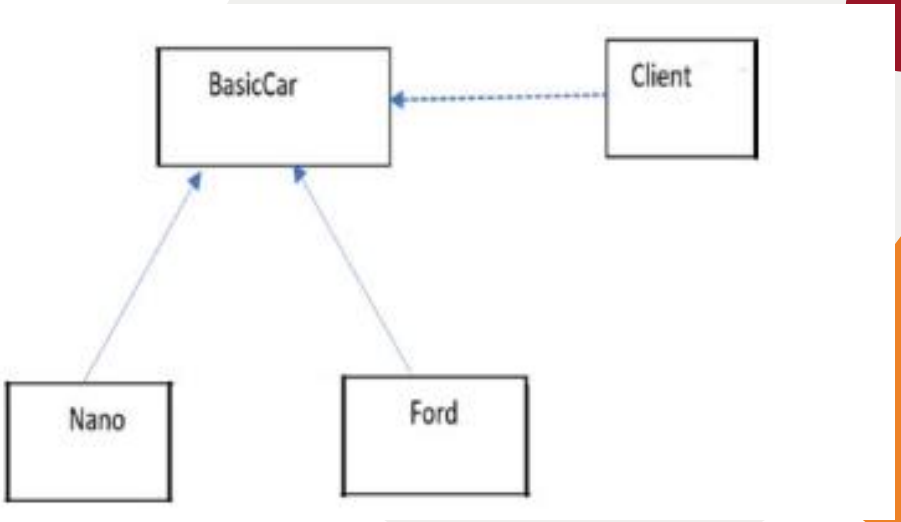
Prototype – Real World Example

Prototype – Computer World Example

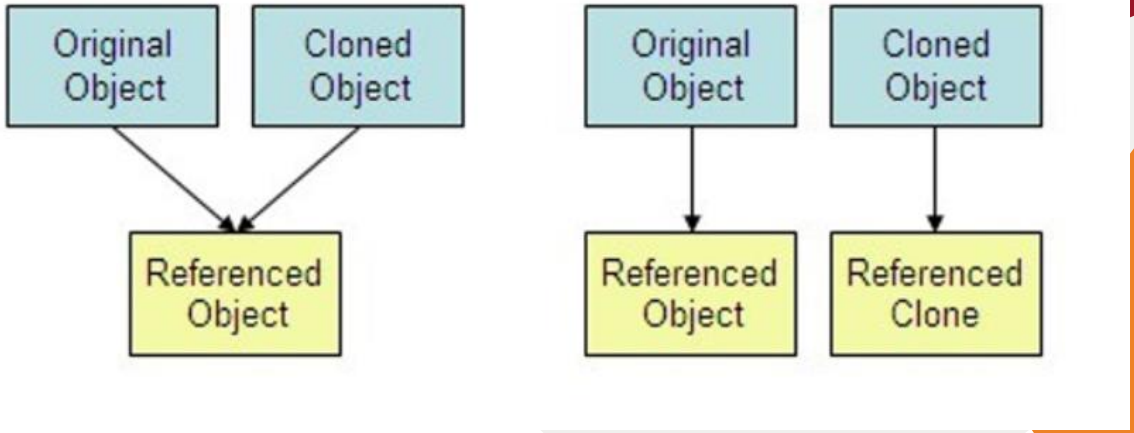
Let's assume that you have an application that is **very stable**. In the future, you may want to update the application with some small modifications. So, you start with a **copy of your original application**, make changes, and analyze further. Surely, to **save your time and money**, you do not want to start from scratch


Any questions?
CHARACTER BUILDING CAMPUS
STT - NF
NURUL FIKRI

Let's Play with the code!



Shallow Copy VS Deep Copy



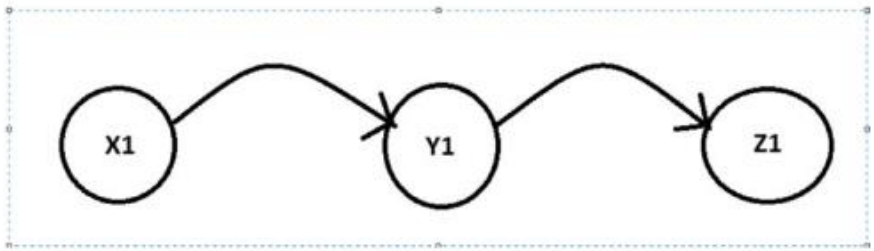
Shallow Copy

Deep Copy

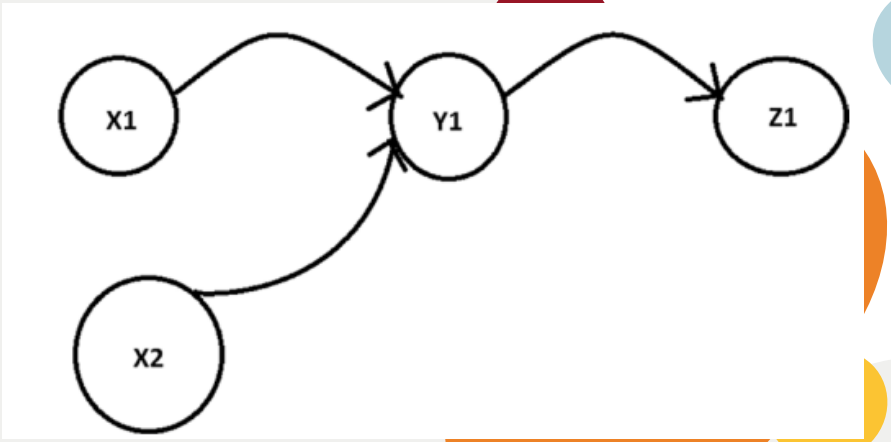
Shallow Copy VS Deep Copy

- A shallow copy creates a new object and then copies various field values from the original object to the new object.
- So, it is also known as a field-by-field copy.
- If the original object contains any references to other objects as fields, then the references of those objects are copied into the new object, (i.e., **you do not create the copies of those objects).**

Shallow Copy VS Deep Copy

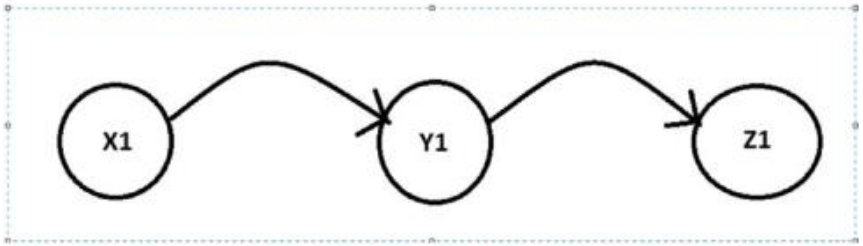


Before Shallow Copy

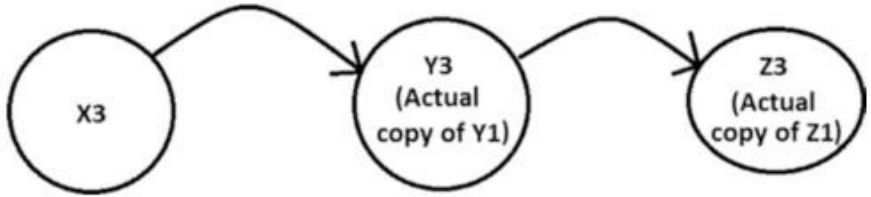


After Shallow Copy (X1 →X2)

Shallow Copy VS Deep Copy



Before Deep Copy



After deep copy (X1 →X3)

Any discussions?

Implementasi Shallow Copy

```
public Box clone(){
    Box b = null;
    try{
        b = (Box)super.clone();
    }catch(Exception e){

    }
    return b;
}
```

```
System.out.println("Box 3 di clone dengan Box 1");
Box box3 = box1.clone();
System.out.println("Box 3 : "+box3.getColor());
```

Implementasi Deep Copy

```java
public abstract class BasicCar implements Cloneable {
    public String modelName;
    public int basePrice,onRoadPrice;
    public String getModelname() {
    return modelName;
    }
    public void setModelname(String modelname) {
    this.modelName = modelname;
    }
    public static int setAdditionalPrice()
    {
    int price = 0;
    Random r = new Random();
    //We will get an integer value in the range 0 to 100000
    int p = r.nextInt(100000);
    price = p;
    return price;
    }
    public BasicCar clone() throws CloneNotSupportedException
    {
        return (BasicCar)super.clone();
    }
}
```

```java
public class Ford extends BasicCar{
    //A base price for Ford
    public int basePrice=100000;
    public Ford(String m)
    {
    modelName = m;
    }
    @Override
    public BasicCar clone() throws CloneNotSupportedException
    {
    return (Ford)super.clone();
    }
}
```

When do you choose a shallow copy over a deep copy?

- ✓ A **shallow copy** is faster and less expensive. It is always better if your target object has the **primitive fields** only.
- ✓ A **deep copy** is expensive and slow. But it is useful if your target object contains many fields that have **references to other objects**.

Any discussions?
Let's Practice!

Thank You!

Subhaanakallohumma wa bihamdika, asy-hadu alla ilaha illa anta, as-tagh-firuka wa atuubu ilaik