# DP

Ahbong Chang

# Builder

- Separate the construction of a complex object from its representation so that the same construction process can create different representations.

我知道正常人不想看這個

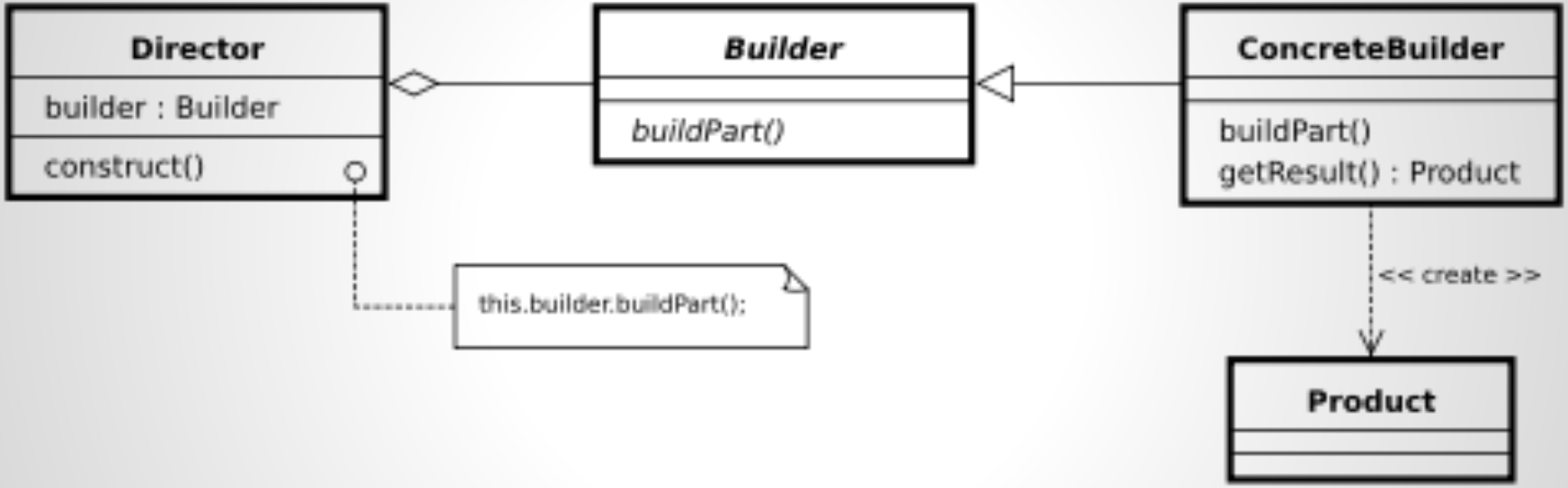
# Builder

- ~~一個____-各自表述 (?)~~
- 一種表述 各種理解 (無誤)
  - 雖然很多時候只有一種理解 *這就夠了*

# Why Builder

More powerful than constructor. You can decide:

- When to build part.
- How to build part.
  - (Have something to do with prototype?)
  - ( immutable object?)


Director
builder : Builder
construct()
Builder
buildPart()
ConcreteBuilder
buildPart()
getResult() : Product
this.builder.buildPart();
<< create >>
Product

# Partial Builders

- Only concrete builder provided. (StringBuilder in Java)
- No director. (Use concrete Builder directly)

# String Concat.

```
List<String> strings;

// Don’t do this!
// Especially for a long list
String longString = “”;
for (String s : strings)
      longString += s;
```

```
List<String> strings;

// This one may be better
int totalLength = 0;
for (String s : strings)
      totalLength += s.length();

char buffer = new char[totalLength];
(*&^@#$(*&
String longString = new String(buffer);
```

# String Concat.

```
List<String> strings;


// How about this one.  Need only concentrate on the representation.
StringBuilder sb; // java.lang.StringBuilder
for (String s : strings)
      sb.append(s);
String longString = sb.toString();
```

# Prototype

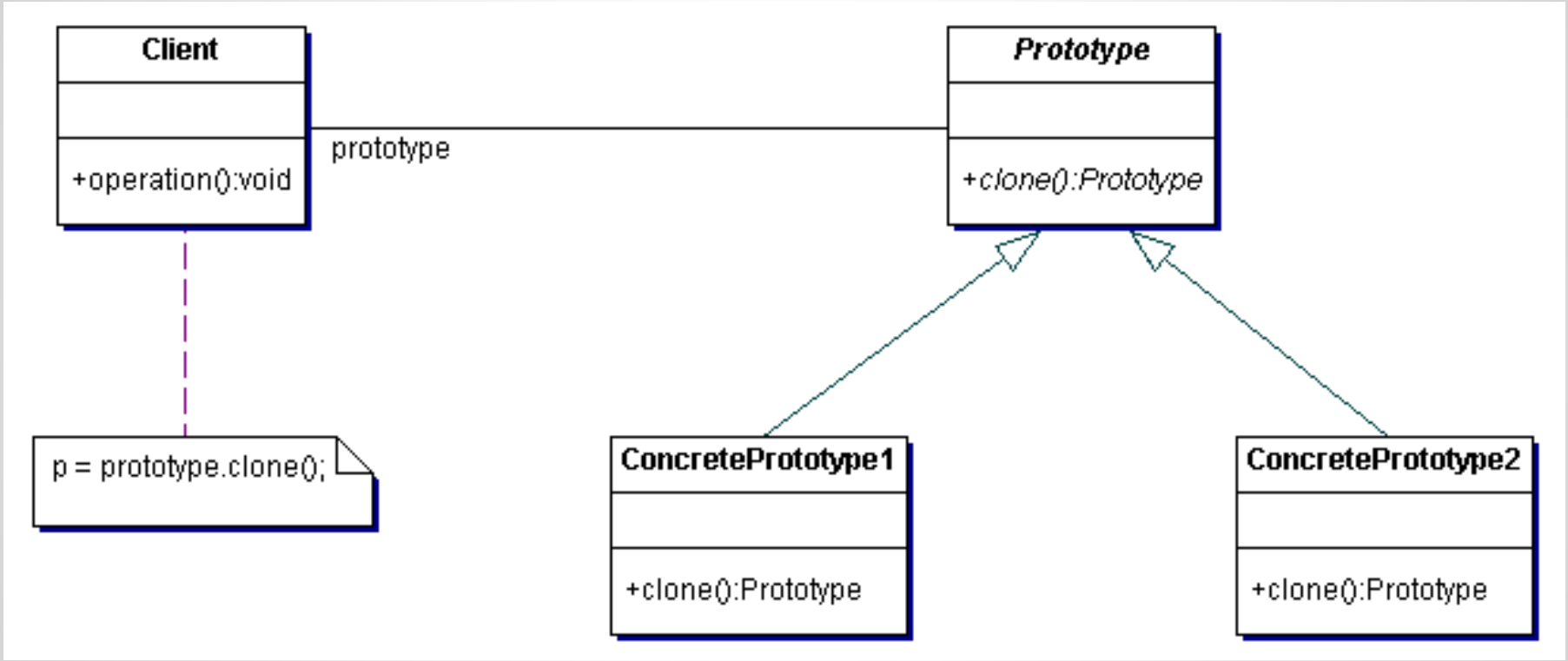
- Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

我知道正常人不想看這個

# Prototype

- 天下文章_____
- ~~下一張投影片的圖~~
- ~~韓國美女~~
- ~~有些人的作業~~
- ~~百度一下，你就知道~~

我不能再多說了


Client
+operation():void
prototype
Prototype
+clone():Prototype
p = prototype.clone();
ConcretePrototype1
+clone():Prototype
ConcretePrototype2
+clone():Prototype

# clone()

That’s all.

# Show Time

Qcl : Director

People : Builder

Ahbong : ConcreteBuilder

GirlFriend : Product

```
{
    Qcl qcl = Qcl.getInstance();
    Person ahbong = Ahbong.getInstance();
    qcl.tellIdealGirlFriendTo(ahbong);
    GirlFriend gf = ahbong.conclude();
}
```

# Show Time

```
public interface Person {
    Person toldHairStyle(String);
    Person toldDress(String);
    …
    GirlFriend conclude();
}
```

```
public class Ahbong implements Person {
    GirlFriend ponyTail = …..;
    public Person toldHairStyle(String style) {
        if (!“ponytail”.equals(style)) {
            throw new NotSupportedException(
                “Only poly tail!”);
        }
    }
    public Person toldDress(String dress) {
        throw new NotSupportedException(
                “Cannot understand.”);
    }
    // Suppose ahbong cannot distinguish different pony tails
    public GirlFriend conclude() {
        return ponyTail.clone();
    }
}
```

# Show Time

```
public final class Qcl {
    // Singleton, will be mentioned….
    public static Qcl getInstance() {/* Return the only Qcl */}

    public void tellIdealGirlFriendTo(Person p) {
        p.toldHairStyle(“ponytail, tied properly.”);
        p.toldDress(“...”);
        ...
    }
}
```