

# **Builder DP**

30-Oct-15 9:18 PM

By Vijay – [caretrainings.co.in](http://caretrainings.co.in)

223

## Problem

- A class has 5 instance variables. The constructor must initialize all 5 variables.
  - All five are optional. If no value is specified, use a default value.



30-Oct-15 9:18 PM

By Vijay – caretrainings.co.in

224

Integrity of an object should not be compromised even if the stack unwinds due to an exception.

Always, object should exist in a valid state.

## Problem

- We have to build a vacation planner for Disney Land.
  - The guest can choose a hotel, various types of admission tickets, make restaurant reservations and even book for special events.
  - Each day can have any combination of hotel reservation, ticket meals and special events.
  - A local guest may not book a hotel but he may book a restaurant.

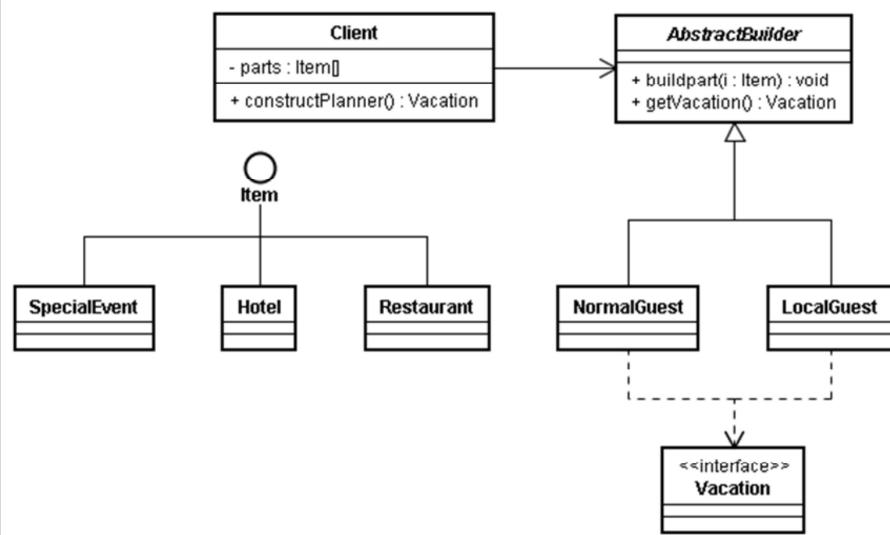


30-Oct-15 9:18 PM

By Vijay – caretrainings.co.in

225

# Solution



30-Oct-15 9:18 PM

By Vijay – caretrainings.co.in

226

It encapsulates the construction of the complex object and allows it to be constructed in steps.

Hides the internal representation, of the product being built, from the client.

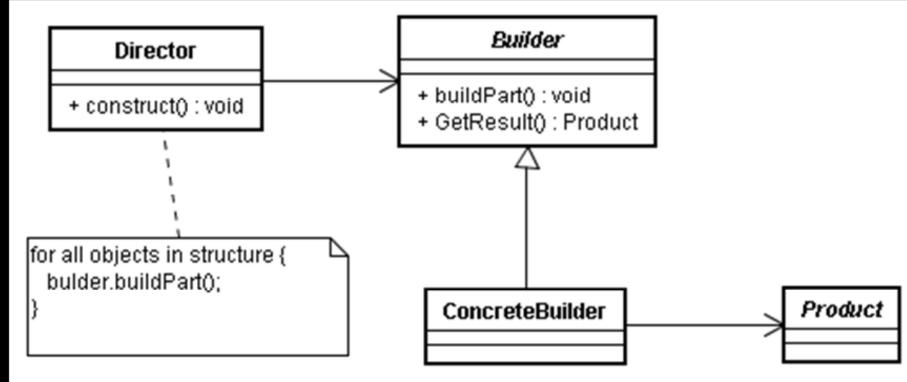
Use the Builder pattern when

The algorithm for creating a complex object should be independent of  
the parts that make up the object and  
how they are assembled.

The construction process must allow different representations for the object  
that is constructed.

Product implementation can be swapped because the client only sees  
the interface of the product.

## Builder DP



30-Oct-15 9:18 PM

By Vijay – caretrainings.co.in

227

The Product represents the complex object under construction.

The Director constructs the object using the Builder interface.

ConcreteBuilder constructs and assembles parts of the Product.

It also provides an interface for retrieving the product

Consequences:

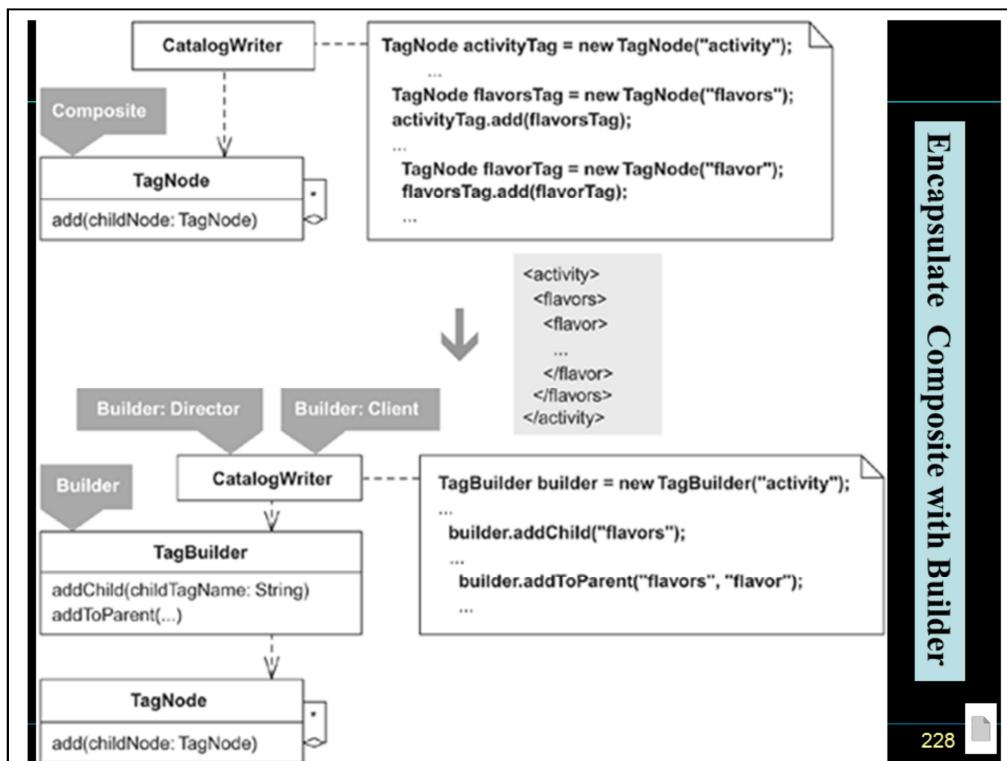
It hides the details of how the product is assembled.

It isolates the steps for construction from the representation of the product.

After the product is finished, the director retrieves it from the builder.

A Builder lets us vary the internal representation of the product it builds.

We can create a new product by adding a new concrete-builder.



A Builder performs burdensome or complicated construction steps on behalf of a client. A common motivation for refactoring to a Builder is to simplify client code that creates complex objects. When difficult or tedious parts of creation are implemented by a Builder, a client can direct the Builder's creation work without having to know how that work is accomplished.

Builders often encapsulate Composites because the construction of Composites can frequently be repetitive, complicated, or error-prone. For example, to add a child node to a parent node, a client must do the following:

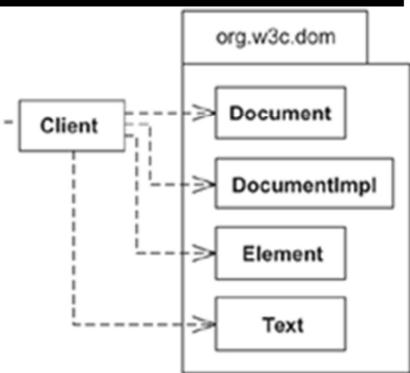
## Instantiate a new node

Initialize the new node

Correctly add the new node to the right parent node

# Identify the problem

```
Document doc = new DocumentImpl();  
Element orderTag = doc.createElement("order");  
orderTag.setAttribute("id", order.getOrderId());  
Element productTag = doc.createElement("product");  
productTag.setAttribute("id", product.getID());  
Text productName =  
    doc.createTextNode(product.getName());  
productTag.appendChild(productName);  
orderTag.appendChild(productTag);
```



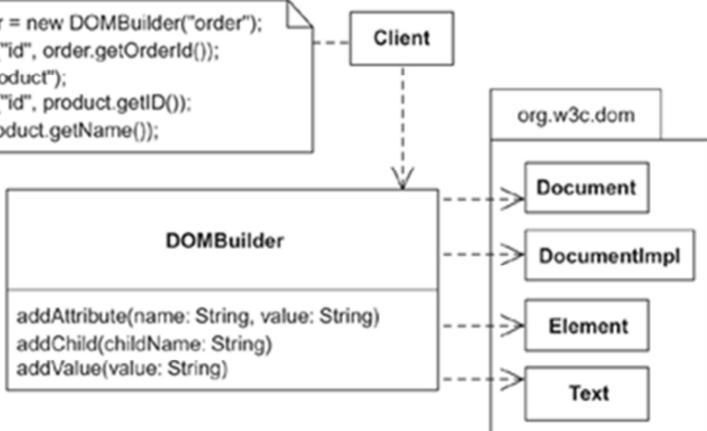
30-Oct-15 9:18 PM

By Vijay – caretrainings.co.in

229

# Decoupling

```
DOMBuilder orderBuilder = new DOMBuilder("order");
orderBuilder.addAttribute("id", order.getOrderId());
orderBuilder.addChild("product");
orderBuilder.addAttribute("id", product.getId());
orderBuilder.addValue(product.getName());
```



Builder decouples client from Composite code

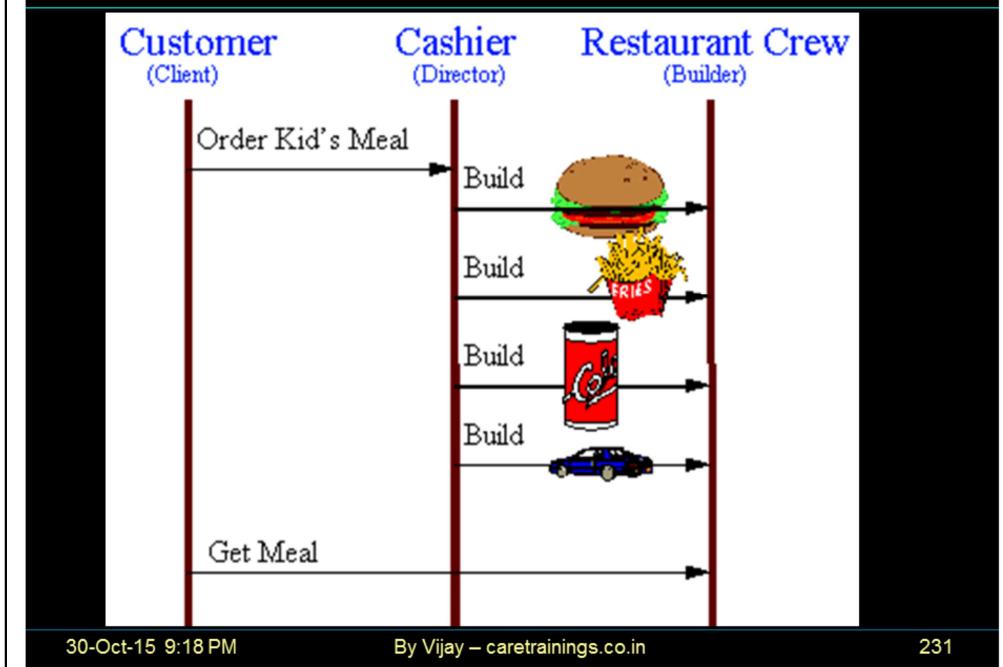
30-Oct-15 9:18 PM

By Vijay – caretrainings.co.in

230

- +Simplifies a client's code for constructing a Composite.
- +Reduces the repetitive and error-prone nature of Composite creation.
- +Creates a loose coupling between client and Composite.
- +Allows for different representations of the encapsulated Composite or complex object.
- May not have the most intention-revealing interface.

# Real World Builder



## Compare

- Builder with Abstract Factory
- Builder with Factory

30-Oct-15 9:18 PM

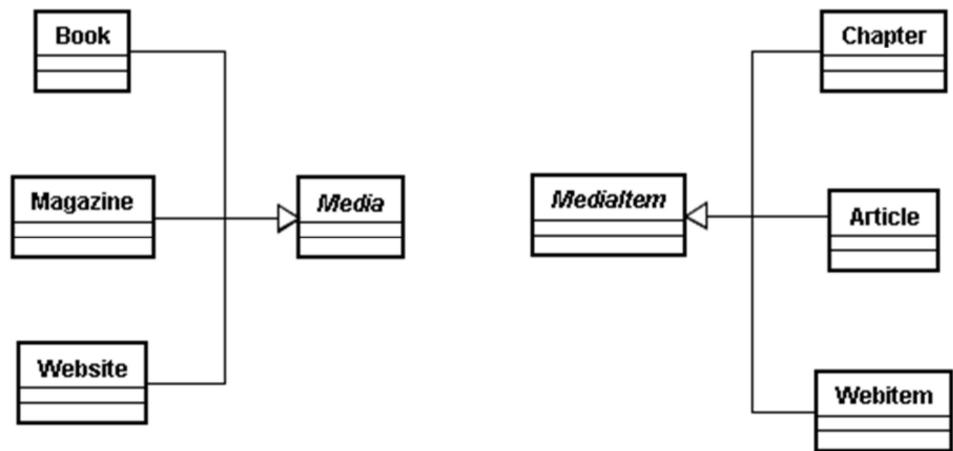
By Vijay – caretainings.co.in

232

Abstract Factory DP emphasizes on product families, while Builder DP constructs a complex object step by step.

Builder returns the completed product at the end of construction, while Factory DP returns it immediately

## Example of Builder

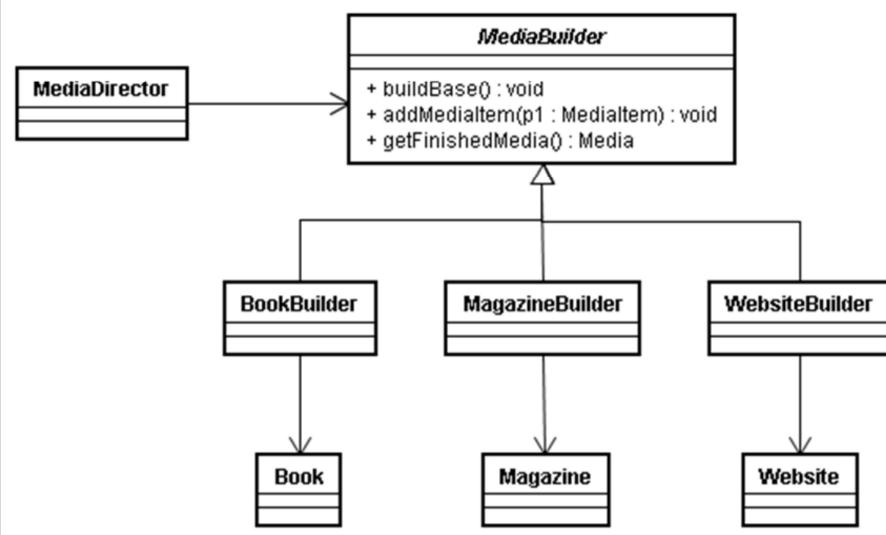


30-Oct-15 9:18 PM

By Vijay – caretainings.co.in

233

## Example of a Builder



30-Oct-15 9:18 PM

By Vijay – caretainings.co.in

234

## Assignment

- A Word document needs to be converted to pdf, rtf and text.

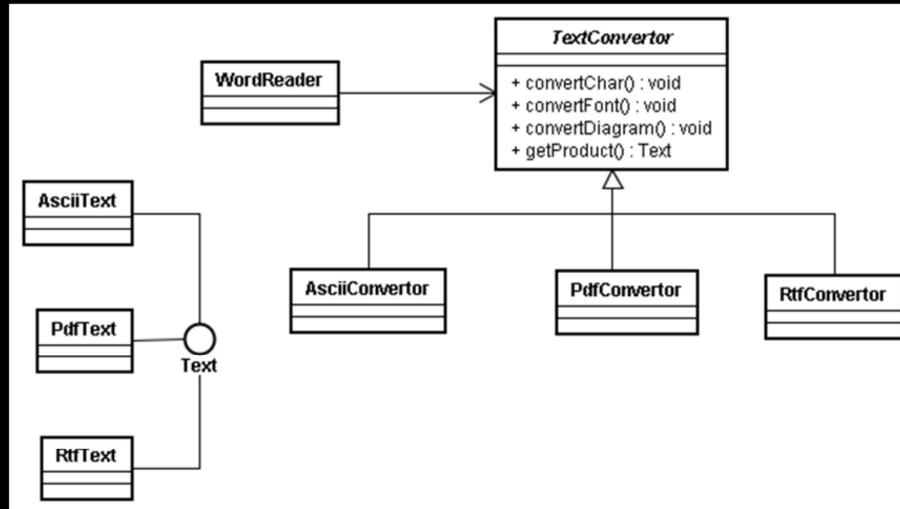


30-Oct-15 9:18 PM

By Vijay – [caretrainings.co.in](http://caretrainings.co.in)

235

# Solution



30-Oct-15 9:18 PM

By Vijay – caretrainings.co.in

236

## Assignment

- We have all the bus routes and distance between two bus stops in files.
  - Find all the routes that exist between the source and destination.
  - Is the Builder DP useful here?

# Assignment

- A function takes five parameters
  - String link, String name, int answer, int zip and String file
- All five parameters are optional
  - i.e. Sometimes we pass 0, sometimes 3 and sometimes any 4 parameters.
- The function signature is

```
void doSomething3(ParameterBuilder pb)
```
- Design this function and write code for any other class used.

30-Oct-15 9:18 PM

By Vijay – caretrainings.co.in

238

```
doSomething3(new ParameterBuilder() .name("Alfred E.  
Neumann").link("http://blog.schaunderhaft.de").ultimateAnswer(42).tempFile("c:\\tem  
p\\x.txt").zip(23));
```

```
public class ParameterBuilder {  
    private String link;  
    private String name;  
    private int answer;  
    private int zip;  
    private String file;  
    public ParameterBuilder name(String aName) {      name = aName;  
        return this; }  
    public ParameterBuilder link(String aLink) {      link = aLink;  
        return this; }  
    public ParameterBuilder ultimateAnswer(int anAnswer) {      answer =  
        anAnswer;      return this; }  
    public ParameterBuilder zip(int aZip) {      zip = aZip;      return this;  
    }  
    public ParameterBuilder tempFile(String aFile) {      file = aFile;
```

```
    return this;  }
        // ... getters for the fields omitted.
}
```