AARHUS UNIVERSITET

Software Engineering and Architecture

Pattern Catalog: Builder

AARHUS UNIVERSITET

Problem

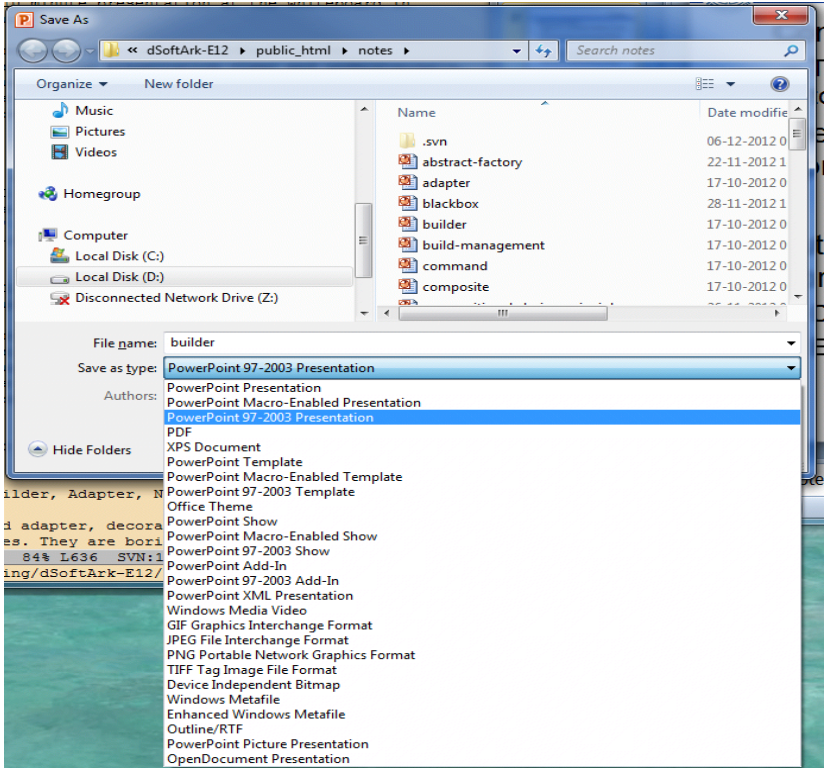
- Consider your favorite
  - Text editor, word processor, spreadsheet, drawing tool

- They allow editing *a complex data structure* representing a document, spreadsheet, etc.

AARHUS UNIVERSITET

Problem

- But they also need to *save* it to a persistent store, typically a hard disk.
  - Converting internal data structure to external format
  - Ex: Binary encoding, XML, HTML, RTF, PDF, …



AARHUS UNIVERSITET

Example

- A document consists of
  - Sections, subsections
  - paragraphs

```
private String section = "The Builder Pattern";
private String subSection1 = "Intent";
private String paragraph1 =
  "Separate the construction of a complex object\n"+
  "from its representation so that the same construction\n"+
  "process can create different representations.";
private String subSection2 = "Problem";
private String paragraph2 = "(The problem goes here)";
```

- We like to output in formats:
- HTML

```
<H1>The Builder Pattern</H1>
<H2>Intent</H2>
<P>
Separate the construction of a complex object
from its representation so that the same construction
process can create different representations.
</P>
<H2>Problem</H2>
<P>
(The problem goes here)
</P>
```

- Or ASCII

```
1. The Builder Pattern
======================
1.1 Intent
Separate the construction of a complex object
from its representation so that the same construction
process can create different representations.
1.2 Problem
(The problem goes here)
```

- A classic variability problem

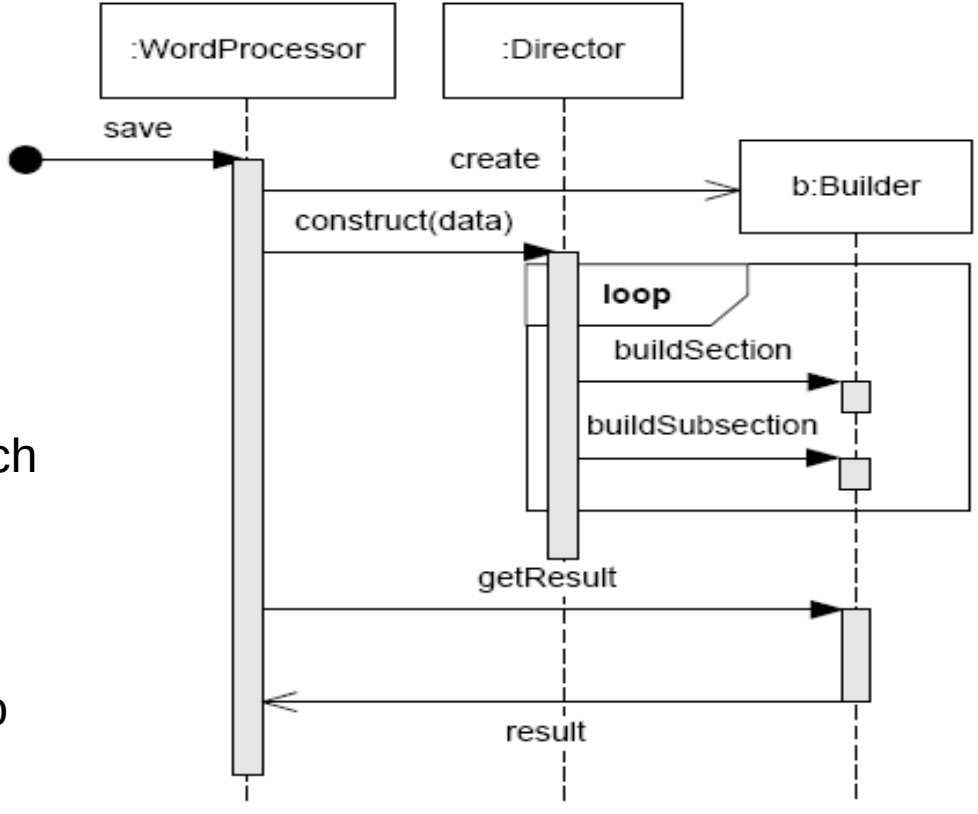
③ All outputs consist of the same set of "parts" (section, subsection, paragraphs, etc.) but how the parts are built varies. That is, concrete construction of the individual node is variable.

① I encapsulate the "construction of parts" in a **builder** interface. A builder interface must have methods to build each unique part: in our case methods like buildSection, buildQuote, etc. Instances realizing this interface must be able to construct concrete parts to be used in the data structure.

② I write the data structure iterator algorithm once, the **director**, and let it request a delegate builder to make the concrete parts as it encounters them.

Dynamics

- Create the builder
  - User chose ‘html’ or ‘ascii’
- The common part is the **director** that knows the *structure and iterates over all its parts.*
- The **builder** handles building each part for the particular output format
- Output data structure is known to the client



AARHUS UNIVERSITET

A *Fake Object* WordProcessor

```
/** This class plays the director role of the builder pattern */
class WordProcessor {
  /** The document
    1. The Builder Pattern
    ====================
      1.1 Intent
        Separate the construction of a complex object
        from its representation so that the same construction
        process can create different representations.
      1.2 Problem
        (The problem goes here)

    is coded as named strings to avoid defining a large
    datastructure.
  */
  private String section = "The Builder Pattern";
  private String subSection1 = "Intent";
  private String paragraph1 =
    "Separate the construction of a complex object\n"+
    "from its representation so that the same construction\n"+
    "process can create different representations.";
  private String subSection2 = "Problem";
  private String paragraph2 = "(The problem goes here)";

  public void construct(Builder builder) {
    /* a real constructor would iterate over a
     * data structure, here I have hardcoded the
     * document to keep the code small */
    builder.buildSection(section);
    builder.buildSubsection(subSection1);
    builder.buildParagraph(paragraph1);
    builder.buildSubsection(subSection2);
    builder.buildParagraph(paragraph2);
  }
}
```

Note: Also plays the **Director role!**
(Quite often the case)

A real 'construct()' would iterate the document structure! ☺

AARHUS UNIVERSITET

Builder part

```
/** This is the Builder role, the interface that
 * defines the parts that can be built */
interface Builder {
  public void buildSection(String text);
  public void buildSubsection(String text);
  public void buildParagraph(String text);
}
```

```
/** A concrete builder implementing a ASCII format */
class AsciiBuilder implements Builder {
  private String result;
  int sectionCounter, subSectionCounter;
  public AsciiBuilder() {
    result = new String();
    sectionCounter = subSectionCounter = 0;
  }
  public void buildSection(String text) {
    sectionCounter++;
    result += ""+sectionCounter+". "+text+"\n";
    result += "======================\n";
  }
  public void buildSubsection(String text) {
    subSectionCounter++;
    result += ""+sectionCounter+"."+subSectionCounter+" "+text+"\n";
  }
  public void buildParagraph(String text) {
    result += text + "\n";
  }
  public String getResult() {
    return result;
  }
}
```

AARHUS UNIVERSITET

Demo Code

```
public class BuilderDemo {
  public static void main(String[] args) {
    System.out.println( "======= Demonstration of Builder =======");
    WordProcessor wp = new WordProcessor();

    // This code act as the client role that
    // creates the concrete builders and instruct
    // the director to construct objects.
    AsciiBuilder asciiBuilder;
    asciiBuilder = new AsciiBuilder();
    wp.construct(asciiBuilder);
    System.out.println( "--- The ASCII Builder output ---");
    System.out.println( asciiBuilder.getResult() );

    HTMLBuilder htmlBuilder;
    htmlBuilder = new HTMLBuilder();
    wp.construct(htmlBuilder);
    System.out.println( "--- The HTML Builder ---");
    System.out.println( htmlBuilder.getResult() );

    CountBuilder countBuilder;
    countBuilder = new CountBuilder();
    wp.construct(countBuilder);
    System.out.println( "--- Counting types ---");
    System.out.println( "Sections    : "+countBuilder.getSectionCount() );
    System.out.println( "Subcections: "+countBuilder.getSubSectionCount() );
    System.out.println( "Paragraphs : "+countBuilder.getParagraphCount() );
  }
}
```

```
D:\proj\Book\src\chapter\builder>java BuilderDemo
======= Demonstration of Builder =======
--- The ASCII Builder output ---
1. The Builder Pattern
======================
1.1 Intent
Separate the construction of a complex object
from its representation so that the same construction
process can create different representations.
1.2 Problem
<The problem goes here>

--- The HTML Builder ---
<H1>The Builder Pattern</H1>
<H2>Intent</H2>
<P>
Separate the construction of a complex object
from its representation so that the same construction
process can create different representations.
</P>
<H2>Problem</H2>
<P>
<The problem goes here>
</P>

--- Counting types ---
Sections    : 1
Subcections: 2
Paragraphs : 2
```

AARHUS UNIVERSITET

Builder part

```
/** This is the Builder role, the interface that
 * defines the parts that can be built */
interface Builder {
  public void buildSection(String text);
  public void buildSubsection(String text);
  public void buildParagraph(String text);
}
```

```
/** A concrete builder implementing a HTML format */
class HTMLBuilder implements Builder {
  private String result;
  public HTMLBuilder() {
    result = new String();
  }
  public void buildSection(String text) {
    result += "<H1>"+text+"</H1>\n";
  }
  public void buildSubsection(String text) {
    result += "<H2>"+text+"</H2>\n";
  }
  public void buildParagraph(String text) {
    result += "<P>\n"+text + "\n</P>\n";
  }
  public String getResult() {
    return result;
  }
}
```

AARHUS UNIVERSITET

Demo Code

```
public class BuilderDemo {
  public static void main(String[] args) {
    System.out.println( "======= Demonstration of Builder =======");
    WordProcessor wp = new WordProcessor();

    // This code act as the client role that
    // creates the concrete builders and instruct
    // the director to construct objects.
    AsciiBuilder asciiBuilder;
    asciiBuilder = new AsciiBuilder();
    wp.construct(asciiBuilder);
    System.out.println( "--- The ASCII Builder output ---");
    System.out.println( asciiBuilder.getResult() );

    HTMLBuilder htmlBuilder;
    htmlBuilder = new HTMLBuilder();
    wp.construct(htmlBuilder);
    System.out.println( "--- The HTML Builder ---");
    System.out.println( htmlBuilder.getResult() );

    CountBuilder countBuilder;
    countBuilder = new CountBuilder();
    wp.construct(countBuilder);
    System.out.println( "--- Counting types ---");
    System.out.println( "Sections    : "+countBuilder.getSectionCount() );
    System.out.println( "Subcections: "+countBuilder.getSubSectionCount() );
    System.out.println( "Paragraphs : "+countBuilder.getParagraphCount() );
  }
}
```

```
D:\proj\Book\src\chapter\builder>java BuilderDemo
======= Demonstration of Builder =======
--- The ASCII Builder output ---
1. The Builder Pattern
======================
1.1 Intent
Separate the construction of a complex object
from its representation so that the same construction
process can create different representations.
1.2 Problem
(The problem goes here)
```

```
--- The HTML Builder ---
<H1>The Builder Pattern</H1>
<H2>Intent</H2>
<P>
Separate the construction of a complex object
from its representation so that the same construction
process can create different representations.
</P>
<H2>Problem</H2>
<P>
(The problem goes here)
</P>
```

```
--- Counting types ---
Sections    : 1
Subcections: 2
Paragraphs : 2
```

AARHUS UNIVERSITET

Builder part

```
/** This is the Builder role, the interface that
 * defines the parts that can be built */
interface Builder {
  public void buildSection(String text);
  public void buildSubsection(String text);
  public void buildParagraph(String text);
}
```

```
/** A concrete builder that simply counts parts */
class CountBuilder implements Builder {
  private int section, subsection, paragraph;
  public CountBuilder() {
    section = subsection = paragraph = 0;
  }

  public void buildSection(String text) { section++; }
  public void buildSubsection(String text) { subsection++; }
  public void buildParagraph(String text) { paragraph++; }

  public int getSectionCount() { return section; }
  public int getSubSectionCount() { return subsection; }
  public int getParagraphCount() { return paragraph; }
}
```

AARHUS UNIVERSITET

Demo Code

```
public class BuilderDemo {
  public static void main(String[] args) {
    System.out.println( "======= Demonstration of Builder =======");
    WordProcessor wp = new WordProcessor();

    // This code act as the client role that
    // creates the concrete builders and instruct
    // the director to construct objects.
    AsciiBuilder asciiBuilder;
    asciiBuilder = new AsciiBuilder();
    wp.construct(asciiBuilder);
    System.out.println( "--- The ASCII Builder output ---");
    System.out.println( asciiBuilder.getResult() );

    HTMLBuilder htmlBuilder;
    htmlBuilder = new HTMLBuilder();
    wp.construct(htmlBuilder);
    System.out.println( "--- The HTML Builder ---");
    System.out.println( htmlBuilder.getResult() );

    CountBuilder countBuilder;
    countBuilder = new CountBuilder();
    wp.construct(countBuilder);
    System.out.println( "--- Counting types ---");
    System.out.println( "Sections    : "+countBuilder.getSectionCount() );
    System.out.println( "Subcections: "+countBuilder.getSubSectionCount() );
    System.out.println( "Paragraphs : "+countBuilder.getParagraphCount() );
  }
}
```

```
D:\proj\Book\src\chapter\builder>java BuilderDemo
======= Demonstration of Builder =======
--- The ASCII Builder output ---
1. The Builder Pattern
======================
1.1 Intent
Separate the construction of a complex object
from its representation so that the same construction
process can create different representations.
1.2 Problem
(The problem goes here)

--- The HTML Builder ---
<H1>The Builder Pattern</H1>
<H2>Intent</H2>
<P>
Separate the construction of a complex object
from its representation so that the same construction
process can create different representations.
</P>
<H2>Problem</H2>
<P>
(The problem goes here)
</P>

--- Counting types ---
Sections    : 1
Subcections: 2
Paragraphs : 2
```

AARHUS UNIVERSITET

Exercise

- Why is there no *getResult* method defined in the interface???

```
/** This is the Builder role, the interface that
 * defines the parts that can be built */
interface Builder {
  public void buildSection(String text);
  public void buildSubsection(String text);
  public void buildParagraph(String text);
}
```

AARHUS UNIVERSITET

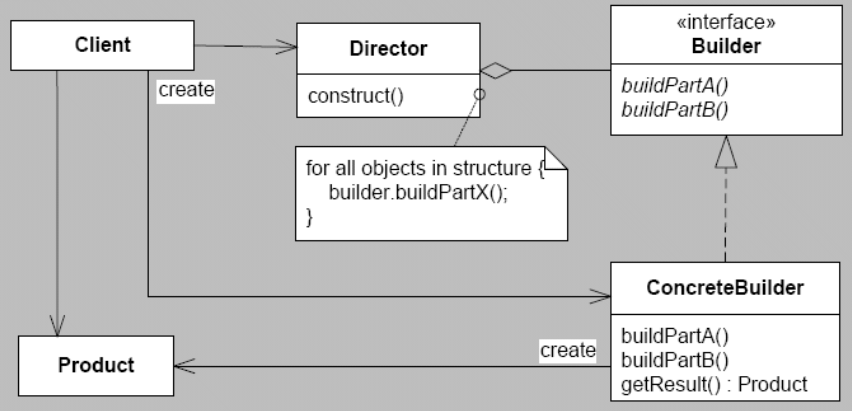
[22.1] Design Pattern: Builder

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

Problem You have a single defined construction process but the output format varies.

Solution Delegate the construction of each part in the process to a builder object; define a builder object for each output format.

Structure:



Roles **Director** defines a building process but constructing the particular parts is delegated to a **Builder**. A set of **ConcreteBuilder**s is responsible to building concrete **Product**s.

Cost - Benefit It is *easy to define new products* as you can simply define a new builder. The *code for construction and representation is isolated*, and thus multiple directors can use builders and vice versa. Compared to other creational patterns (like ABSTRACT FACTORY) products are not produced in "one shot" but stepwise meaning you have *finer control over the construction process.*

Aspects

- Benefits are
  - Fine grained control over the building process
    - Compare to Abstract Factory
  - Construction process and part construction decoupled
    - *Change by addition* to support new formats
    - Many-to-many relation between directors and builders
      - *Reuse the builders in other directors…*
- Liabilities
  - Client must know the product of the builder as well as the concrete builder types