CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2022
Doc 17 Prototype, Factory Method, Builder
Nov 10, 2022

Prototype

Prototype

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

Applicability

Use the Prototype pattern when

A system should be independent of how its products are created, composed, and represented; and

When the classes to instantiate are specified at run-time; or

To avoid building a class hierarchy of factories that parallels the class hierarchy of products; or

When instances of a class can have one of only a few different combinations of state.

Insurance Example

Insurance agents start with a standard policy and customize it

Two basic strategies:

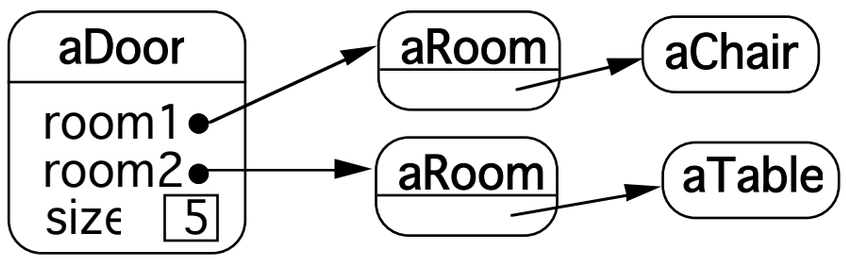
Copy the original and edit the copy

Store only the differences between original and the customize version in a decorator

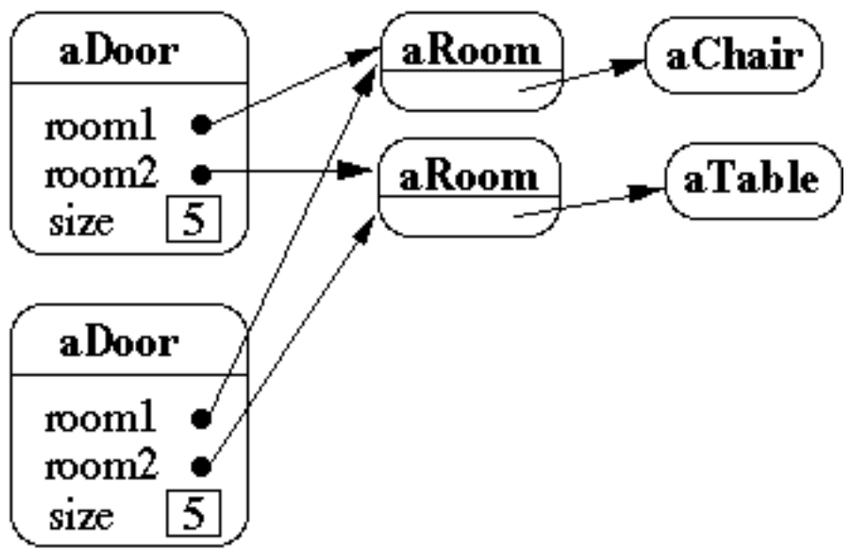
Copying Issues

Shallow Copy Verse Deep Copy

Original Objects

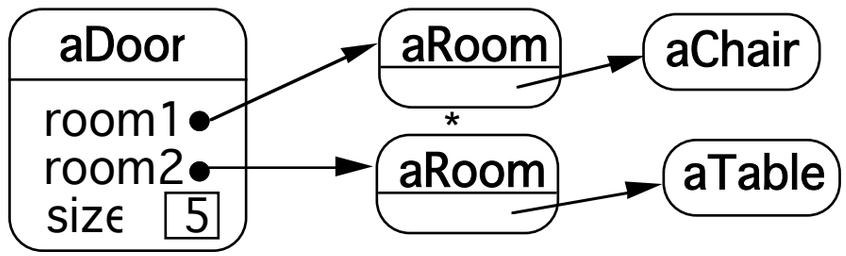


Shallow Copy

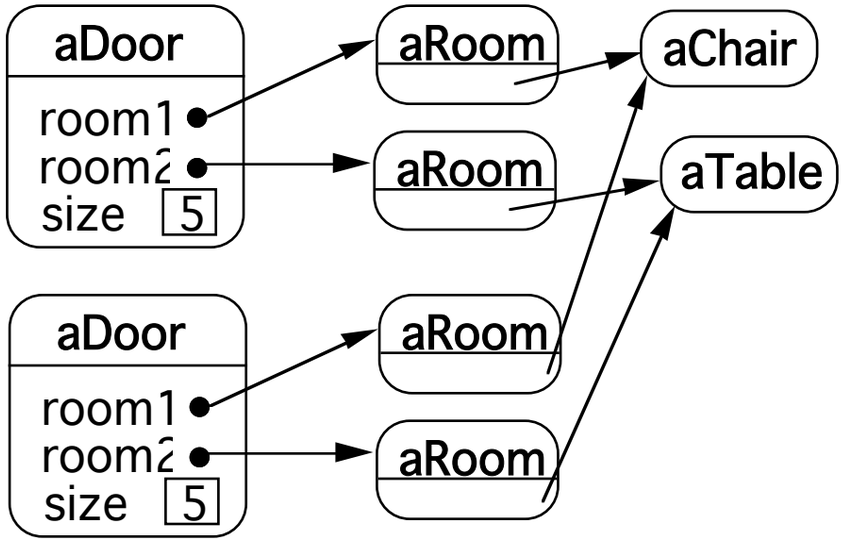


Shallow Copy Verse Deep Copy

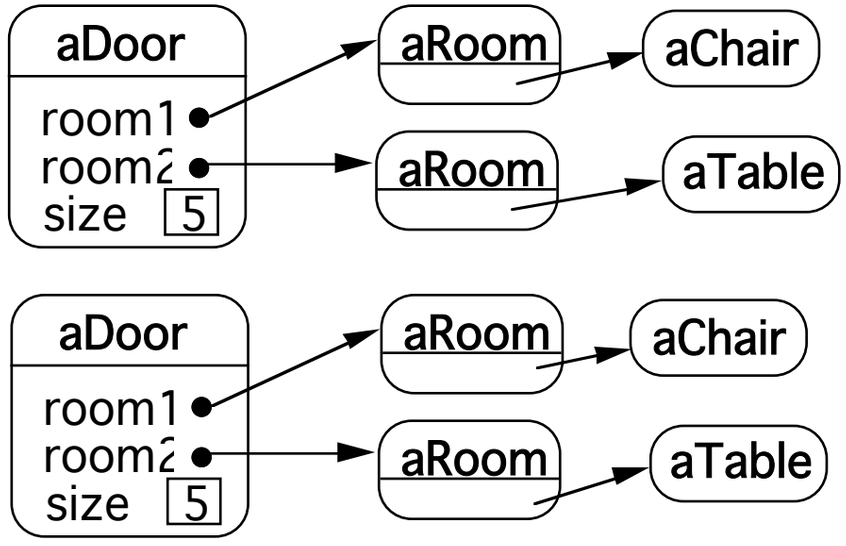
Original Objects



Deep Copy



Deeper Copy



Prototype-based Languages

No classes

Behavior reuse (inheritance)

Cloning existing objects which serve as prototypes

Some Prototype-based languages

Self

JavaScript

Squeak (eToys)

Perl with Class::Prototyped module

JavaScript - Copying

```
var Animal = {
  type: 'Invertebrates',
  displayType: function() {
    console.log(this.type);
  }
};
```

Animal is the Prototype

```
var animal1 = Object.create(Animal);
animal1.displayType();              // Output:Invertebrates

var fish = Object.create(Animal);
fish.type = 'Fishes';
fish.displayType();                 // Output:Fishes

var copy = {...Animal};
```

JavaScript Prototype

Every object has a prototype

When looking for a property or method in an object
- Look in the object if not there
- Look in prototype, if not there
- Look in the prototype's prototype, if not there
- Continue looking in the prototype chain until find it or reach the end

A parent class's prototype is added to the subclasses prototype chain

But prototype chains are dynamic
- They can be changed at runtime

JavaScript Class

```
class Rectangle {
    constructor(height, width) {
      this.height = height;
      this.width = width;
    }

  }
```

```
const q = new Rectangle(10, 45);
q.height;
q['height'];
```

Prototype Example

```
class Cat {
  constructor(name) {
    this.name = name;
  }
}

class Dog {
  constructor(name) {
    this.name = name;
  }
}
```

```
let fluffy = new Cat("Fluffy");
let spot = new Dog('Spot');
fluffy['food'];                    //undefined

 Object.color = 'red';
 Object.prototype.food = "mouse";
 Cat.prototype.size = 'small';

  fluffy['food'];                  // 'mouse'
  fluffy.size;                     // 'small'
  fluffy.color;                    // undefined
  spot.food;                       // 'mouse'

  Dog.prototype.food = 'rabbit';
  spot.food;                       // 'rabbit'
  fluffy.food;                     // 'mouse'
```

JavaScript Classes are Functions

```
class Rectangle {
    constructor(height, width) {
      this.height = height;
      this.width = width;
    }
 }

 console.log(typeof Rectangle);                    // function

 console.log(Object.getOwnPropertyNames(Rectangle));        // [ 'length', 'prototype', 'name' ]

 console.log(Rectangle.name);                      // Rectangle

 console.log(Rectangle.length);                    // 2
```

Most Things are Objects in JavaScript

Object

- Function
- Array
- Date
- RegExp

```
function adder(x, y) {
  return x + y;
}

console.log(Object.getOwnPropertyNames(adder))
```

```
[ 'length', 'name', 'arguments', 'caller', 'prototype' ]
```

Factory Method

Factory Method

A template method for creating objects

```
public class Example {
   protected Bar bar() { return new Bar(); }

    public void foo() {
      blah
      Bar soap = bar();
      blah;
   }
```

Maze Game Example

Maze Game Example

```
class MazeGame{
    public Maze makeMaze() { return new Maze(); }
    public Room makeRoom(int n ) { return new Room( n ); }
    public Wall makeWall() { return new Wall(); }
    public Door makeDoor() { return new Door(); }

    public Maze CreateMaze(){
        Maze aMaze = makeMaze();
        Room r1 = makeRoom( 1 );
        Room r2 = makeRoom( 2 );
        Door theDoor = makeDoor( r1, r2);

        aMaze.addRoom( r1 );
        aMaze.addRoom( r2 );
        etc

        return aMaze;
        }
}
```

```
class BombedMazeGame extends MazeGame {

    public Room makeRoom(int n )  {
        return new RoomWithABomb( n );
    }

    public Wall makeWall() {
        return new BombedWall();
    }
}
```

Don't repeat your self

```
public class LinkedList extends Collection {
   public OrderedLinkedList() {
      this(defaultOrder());
   }

   public LinkedList(Order listOrder ) {
      this(listOrder, new OrderedCollection());
   }

   public LinkedList(Collection items) {
      this(defaultOrder(), items);
   }

   protected Order defaultOrder() {
      return new RandomOrder();
   }

   public LinkedList(Order listOrder, Collection items) {
      blah
```

Implementation Variation

```
class Hershey {

    public Candy makeChocolateStuff( CandyType id ) {
        if ( id == MarsBars ) return new MarsBars();
        if ( id == M&Ms ) return new M&Ms();
        if ( id == SpecialRich ) return new SpecialRich();

        return new PureChocolate();
    }

class GenericBrand extends Hershey {
    public Candy makeChocolateStuff( CandyType id ) {
        if ( id == M&Ms ) return new Flupps();
        if ( id == Milk ) return new MilkChocolate();
        return super.makeChocolateStuff(id);
    }
}
```

Smalltalk Variant

Return the class, caller creates an object

```
chocolateStuff
        ^SpecialRich
```

```
some code
candy := (self chocolateStuff) new
mode code
```

Use Factory Method When

A class can't anticipate the class of objects it must create

A class wants its subclasses to specify the objects it creates

You want to localize the knowledge of which help classes is used in a class

But when is this?

CS 580 Example - Testing a Server

```
public class SDTwitterServer {
    public void run(int port) throws IOException {
        ServerSocket input = new ServerSocket( port );

        while (true) {
            Socket client = input.accept();
            processRequest(
                client.getInputStream(),
                client.getOutputStream());
            client.close();
        }
}

void processRequest(InputStream in,OutputStream out) {
    do a bunch of stuff
}

etc.
```

Using Factory Method

```
public class SDTwitterServer {
    public void run(int port) throws IOException {
        ServerSocket input = this.serverSocket( port );

        while (true) {
            Socket client = input.accept();
            processRequest(
                client.getInputStream(),
                client.getOutputStream());
            client.close();
        }
    }

    ServerSocket serverSocket( int port) {
        return new ServerSocket(port);
    }

    etc.
```

TestServer

```
public class TestServer extends SDTwitterServer {
   MockServerSocket testSocket;

   ServerSocket serverSocket( int port) {
      return testSocket;
   }
}
```

Other than using a different type of socket it performs the operations as the parent class

```
public class Tests extends Testcase {
   public void testLogin() {
      TestServer server = new TestServer();
      server.testSocket = new MockServerSocket("client command to login");
      server.run();
      assertTrue(server.testSocket.serverResponse() = "the correct response here");
   }
}
```

MockServerSocket

Returns a fake (Mock) client connection

Fakes client connection

- Does not use network
- Contains fixed requests
- Records server responses

Dependency Injection

```
public class SDTwitterServer {
    ServerSocket input;
    public SDTwitterServer(ServerSocket input) {
        this.input = input;
    }

    public void run(int port) throws IOException {
        while (true) {
            Socket client = input.accept();
            processRequest(
                client.getInputStream(),
                client.getOutputStream());
            client.close();
        }
}
```

Dependency Injection

"One object (or static method) supplies the dependencies of another object"
Wikipedia

Constructor injection
Setter injection
Interface injection

Builder

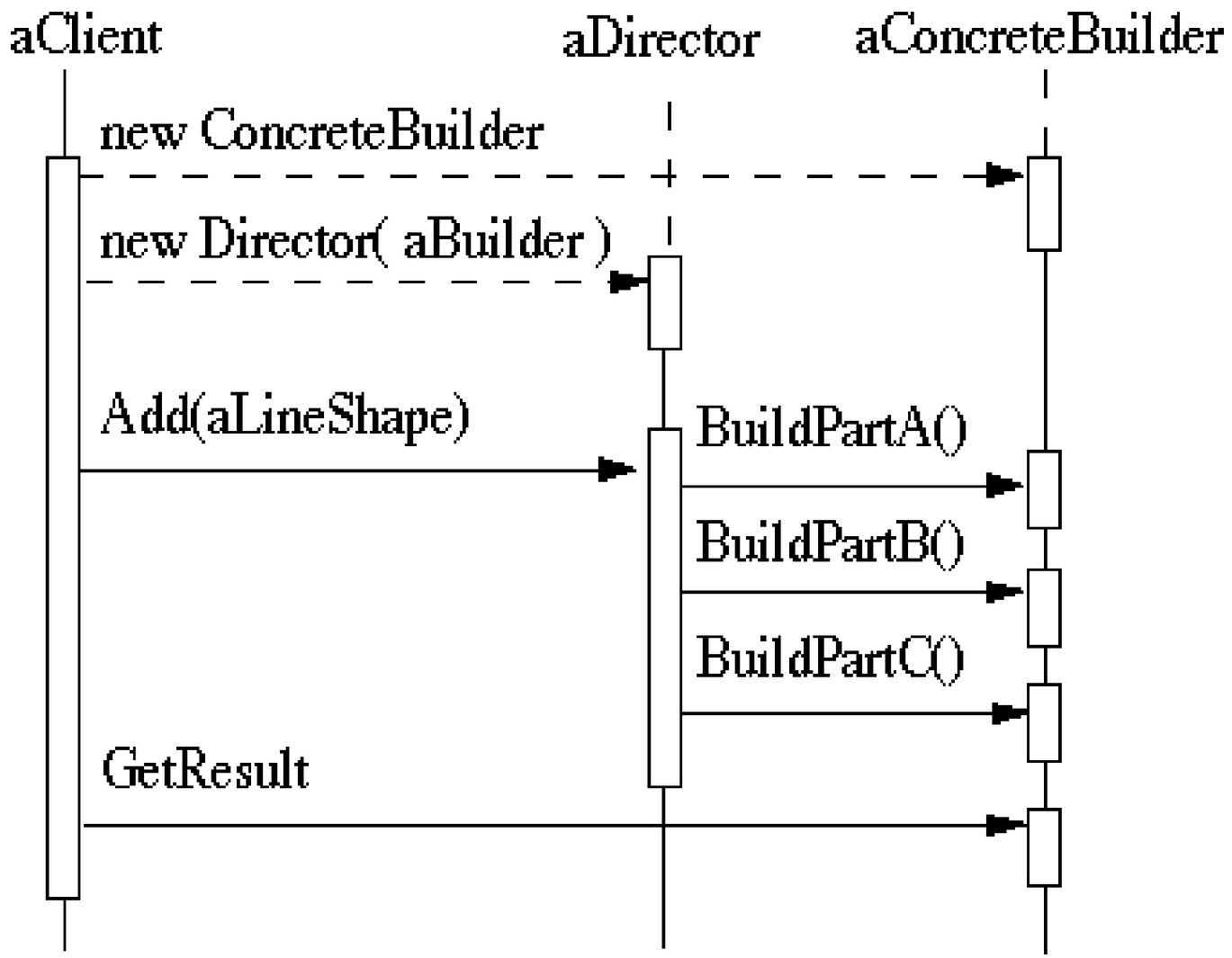
Builder

Separate construction of a complex object from its representation

So same construction process can create different representations

Builder

Client Director Builder



RTF Converter

A word processing document has complex structure

How to convert Rich Text Format (RTF) to

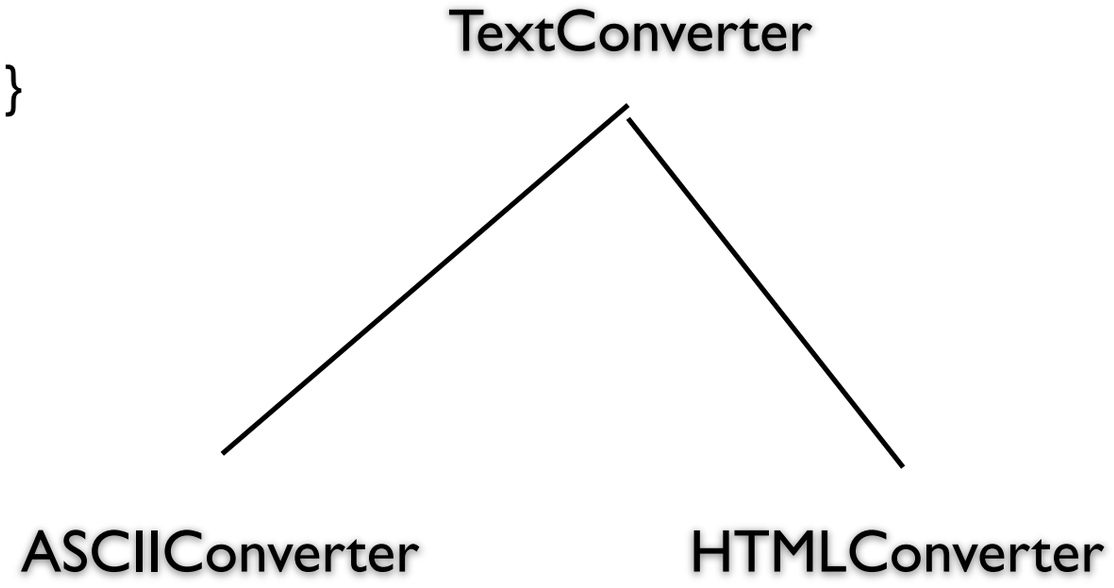
- TeX
- html
- PDF
- etc.

Pseudo Solution

```
class RTF_Reader {
   TextConverter builder;
   String RTF_Text;

   public RTF_Reader( TextConverter aBuilder, String RTFtoConvert ){
      builder = aBuilder;
      RTF_Text = RTFtoConvert;
      }

   public void parseRTF(){
      RTFTokenizer rtf = new RTFTokenizer( RTF_Text );

      while ( rtf.hasMoreTokens() ){
         RTFToken next = rtf.nextToken();

         switch ( next.type() ){
            case CHAR:    builder.character( next.char() ); break;
            case FONT:    builder.font( next.font() ); break;
            case PARA:    builder.newParagraph( ); break;
            etc.
      }
```

Builder Classes

```
abstract class TextConverter {
   public void character( char nextChar ) { }
   public void font( Font newFont )   { }
   public void newParagraph() {}
}
```



Sample Program

```
main(){
   ASCII_Converter simplerText = new ASCII_Converter();
   String rtfText;

   // read a file of rtf into rtfText

   RTF_Reader myReader =
      new RTF_Reader( simplerText, rtfText );

   myReader.parseRTF();

   String myProduct = simplerText.getText();
}
```

The Hard Part

The builder interface

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<RootElement param="value">
    <FirstElement>
        Some Text
    </FirstElement>
    <SecondElement param2="something">
        Pre-Text <Inline>Inlined text</Inline> Post-text.
    </SecondElement>
</RootElement>
```

SAX - Builder Pattern

Director	XMLReader
Builder	ContentHandler

ContentHandler Interface

void characters(char[] ch, int start, int length)
Receive notification of character data.

void endDocument()
Receive notification of the end of a document.

void endElement(java.lang.String uri, java.lang.String localName, java.lang.String qName)
Receive notification of the end of an element.

void endPrefixMapping(java.lang.String prefix)
End the scope of a prefix-URI mapping.

void ignorableWhitespace(char[] ch, int start, int length)
Receive notification of ignorable whitespace in element content.

void processingInstruction(java.lang.String target, java.lang.String data)
Receive notification of a processing instruction.

void setDocumentLocator(Locator locator)
Receive an object for locating the origin of SAX document events.

void skippedEntity(java.lang.String name)
Receive notification of a skipped entity.

void startDocument()
Receive notification of the beginning of a document.

void startElement(java.lang.String uri, java.lang.String localName, java.lang.String qName, Attributes att
Receive notification of the beginning of an element.

void startPrefixMapping(java.lang.String prefix, java.lang.String uri)
Begin the scope of a prefix-URI Namespace mapping.

Simple API XML (SAX)

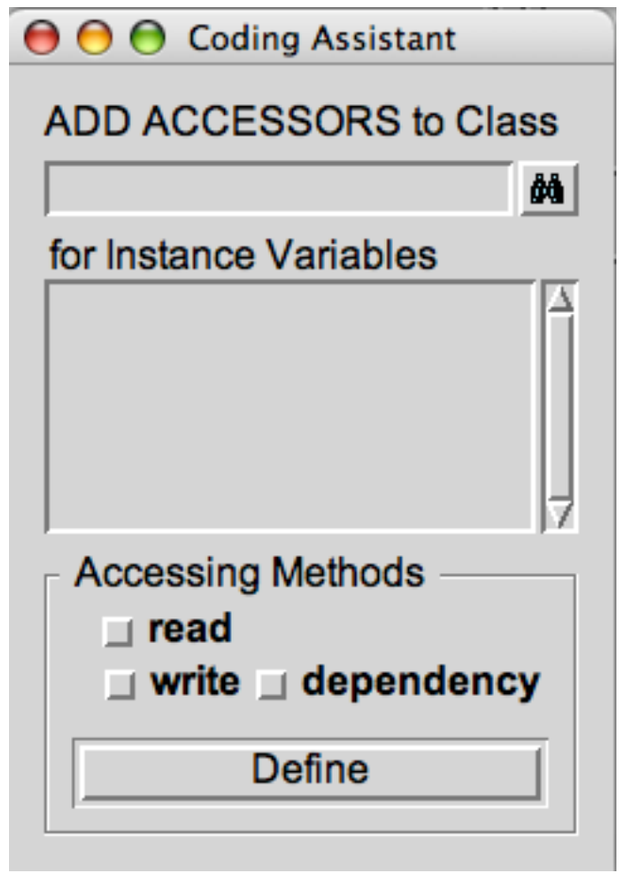
```
public static void main (String args[]) throws Exception {
  XMLReader director = XMLReaderFactory.createXMLReader();
  ContentHandler builder = new MySAXApp();
  director.setContentHandler(builder);
  director.setErrorHandler(builder);

  FileReader source = new FileReader("Foo.xml");
  director.parse(new InputSource(source));
  handler.getResult();
  }
```

Examples - VW Smalltalk

ClassBuilder
MenuBuilder
UIBuilder

UIBuilder

```
#(#{UI.FullSpec}
    #window:
    #(#{UI.WindowSpec}
      #label: #(#{Kernel.UserMessage} #key: #CodingAssistant
            #defaultString: 'Coding Assistant' #catalogID: #UIPainter)
      #min: #(#{Core.Point} 242 320 )
      #max: #(#{Core.Point} 242 320 )
      #bounds: #(#{Graphics.Rectangle} 279 140 521 460 ) )
    #component:
    #(#{UI.SpecCollection}
      #collection: #(
        #(#{UI.LabelSpec}
          #layout: #(#{Graphics.LayoutOrigin} 14 0 12 0 )
          #label: #(#{Kernel.UserMessage} #key: #ADDACCESSORSToClass
            #defaultString: 'ADD ACCESSORS to Class' #catalogID: #UIPainter) )
        #(#{UI.LabelSpec}
          #layout: #(#{Graphics.LayoutOrigin} 16 0 65 0 )
          #label: #(#{Kernel.UserMessage} #key: #forInstanceVariables
              #defaultString: 'for Instance Variables' #catalogID: #UIPainter) )
```



Simplified Builder Pattern

More common the standard Pattern

Used to set multiple fields

Replaces using constructor with many parameters

Person Example

```
public class Person
{
   private final String lastName;
   private final String firstName;
   private final String middleName;
   private final String salutation;
   private final String suffix;
   private final String streetAddress;
   ...
   private final boolean isEmployed;
```

PersonBuilder

```
public class PersonBuilder
{
   private String newLastName;
   private String newFirstName;
   private String newMiddleName;
   private String newSalutation;
   private String newSuffix;
   private String newStreetAddress;
   ...
   private boolean newIsEmployed;

   public PersonBuilder setLastName(String newLastName) {
      this.newLastName = newLastName;
      return this;
   }

   public PersonBuilder setFirstName(String newFirstName) {
      this.newFirstName = newFirstName;
      return this;
   }
```

PersonBuilder - Continued

```
public PersonBuilder setMiddleName(String newMiddleName) {
      this.newMiddleName = newMiddleName;
      return this;
   }
```

The rest of the set methods

```
public Person createPerson() {
      return new Person(newLastName, newFirstName, newMiddleName, newSalutation,
newSuffix, newStreetAddress, newCity, newState, newIsFemale, newIsEmployed,
newIsHomeOwner);
   }
```

Building a Person

```
Person test = new PersonBuilder().
        setLastName("Whitney").
        setFirstName("Roger").
        ...
        setIsEmployed(true).
        createPerson();
```

Improvements

Make Builder an inner class (Java)

Group fields into separate classes

- Name Class
  - firstName
  - lastName
  - middleName
  - salutation
  - suffix

Android Example

Building a Notification

```
Notification note = new Notification.Builder(mContext)
        .setContentTitle("New mail from " + sender.toString())
        .setContentText(subject)
        .setSmallIcon(R.drawable.new_mail)
        .setLargeIcon(aBitmap)
        .build();
```

Strategy
vs
Builder