Effective Java
Item 1. Consider Static Factory methods

Consider using static Factory methods instead of constructors

Java String class

```java
public static String valueOf(boolean b) {
    return b ? "true" : "false";
}
```

```java
public static String valueOf(char c) {
    char data[] = {c};
    return new String(data, true);
}
```
Advantages of Static Factory methods

They have names

Don’t need to create a new object each time

They can return an Object of any type
Java Boxing of Primitives

```java
Integer x = new Integer(5);
Boolean y = new Boolean(true);
```
Objective C Boxing of Primitives

Uses static factory methods in Number

    Number x = Number.value(5);
    Number y = Number.value(true);

Programmers only need to know Number class

    Class Cluster
Smalltalk

No constructors

Just static factory methods
Item 12 Minimize accessibility

Rule of thumb

Make each class or member as inaccessible as possible
Item 13 Favor Immutability

Immutable objects are simple

Immutable objects are thread-safe

Immutable objects can be shared freely

Immutable objects are good building blocks for other objects
Item 13 Favor Immutability

Don’t provide any methods that modify the object (setters)

Ensure that no methods may be overridden

Make all fields final

Make all fields private

Ensure exclusive use to any mutable components

Make defensive copies of data provided/given to client
public final class Period {
  private final Date start;
  private final Date end;

  public Period(Date start, Date end) {
    if (start.compareTo(end) > 0 )
      throw new IllegalArgumentException(start + " is after " + end);
    this.start = start;
    this.end = end;
  }

  public Date start() {
    return start;
  }
}
Item 24 Make Defensive Copies when Needed

```java
public final class Period {
    private final Date start;
    private final Date end;

    public Period(Date start, Date end) {
        this.start = new Date(start.getTime());
        this.end = new Date(end.getTime());
        if (this.start.compareTo(this.end) > 0 )
            throw new IllegalArgumentException(start + " is after " + end);
    }

    public Date start() {
        return start.clone();
    }
}
```
Item 14 Favor Composition over Inheritance

Inheritance breaks encapsulation

Safe to use inheritance when

Superclass and subclass in same package

When superclass is designed for inheritance
Item 16 Prefer Interfaces to Abstract Classes

Existing classes can be modified to implement a new interface

Interfaces are ideal for defining mixins

Interfaces allow construction of nonhierarchical frameworks

Provide skeletal implementation class to go with nontrivial interface
Item 30 Know and use the Libraries
Item 32 Avoid strings if other types are better

String compoundKey = name + "#" + i.next();

What happens if "#" is in name?

Create CompoundKey class
Item 34 Refer to objects by their Interfaces

Your code will be more flexible

✅ List subscribers = new Vector();

❌ Vector subscribers = new Vector();

If no interface exists then ok to refer to object via class
Builder
Builder

Separate construction of a complex object from its representation

So same construction process can create different representations
Builder

Client | Director | Builder

```
new ConcreteBuilder
new Director(aBuilder)
Add(aLineShape)
BuildPartA()
BuildPartB()
BuildPartC()
GetResult
```
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.
            }
        }
    }
}

Thursday, April 16, 15
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
<?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 atts)
    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.
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
```ruby
(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) }
      )
    )
```

Thursday, April 16, 15
Simplified Builder Pattern

More common the standard Pattern

Used to set multiple fields

Replaces using constructor with many parameters
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;

    public Person(
        final String newLastName,
        final String newFirstName,
        final String newMiddleName,
        final String newSalutation,
        final String newSuffix,
        final String newStreetAddress,
        final boolean newIsEmployed) {
        this.lastName = newLastName;
        this.firstName = newFirstName;
        ...
    }
}
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;
    }
}
public Person createPerson() {
    return new Person(newLastName, newFirstName, newMiddleName,
    newSalutation, newSuffix, newStreetAddress, newCity, newState, newIsFemale,
    newIsEmployed, newIsHomeOwner);
}

The rest of the set methods

class PersonBuilder {
    //...

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

    //...

    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
Strategy vs Builder