Modularizing constructors

Viviana Bono\textsuperscript{1} Jarosław Kuśmierek\textsuperscript{2}

\textsuperscript{1}Torino University \textsuperscript{2}Warsaw University

TOOLS 2007
Zurich - 25 June 2007
Plan

1. Problems
   - Origins of the problems
   - A Java example
   - Existing approaches

2. Our solution
   - General idea
   - An example of code
   - How we create objects

3. Final remarks
   - The bigger picture
   - Comparing measurements of code
Initialization protocol

Describes the process of initialization of an object, which must be performed before the object can be used.
Initialization protocol

Describes the process of initialization of an object, which must be performed before the object can be used.

In most languages: monolithic constructors

- Each possible variant of the initialization protocol is described by a single constructor.
- Very often the constructor used for a certain variant of initialization is chosen via overloading.
Initialization protocol

Describes the process of initialization of an object, which must be performed before the object can be used.

In most languages: monolithic constructors

- Each possible variant of the initialization protocol is described by a single constructor.
- Very often the constructor used for a certain variant of initialization is chosen via overloading.
Copying

Different options of initialization but with common parts: some code needs to be copied.
Copying
Different options of initialization but with common parts: some code needs to be copied.

Exponential growth of the number of constructors
Orthogonal properties with different initialization options cause the number of constructors to grow exponentially.
Problems

Copying
Different options of initialization but with common parts: some code needs to be copied.

Exponential growth of the number of constructors
Orthogonal properties with different initialization options cause the number of constructors to grow exponentially.

Unnecessary dependencies
By copying a list of superclass’ constructors in the subclasses, we needlessly fix the set of options. Further extensions of the set of constructors in the superclass will not affect subclasses accordingly.

Final remarks

Origins of the problems
A Java example
Existing approaches

Viviana Bono Jarosław Kuśmierk
Modularizing constructors
Copying

Different options of initialization but with common parts: some code needs to be copied.

Exponential growth of the number of constructors

Orthogonal properties with different initialization options cause the number of constructors to grow exponentially.

Unnecessary dependencies

By copying a list of superclass’ constructors in the subclasses, we needlessly fix the set of options. Further extensions of the set of constructors in the superclass will not affect subclasses accordingly.

Problems with checked exceptions

When the specification of the parent class constructors is refined with respect to the thrown exceptions, subclasses will not automatically benefit from it.
// Class of points definable by three different coordinate systems
class Point
{
    float x, y; // object state variables

    Point (float x, float y)
    {
        this.x = x; this.y = y;
    }

    Point (Complex comp)
    {
        x = comp.x; y = comp.y;
    }

    // the third parameter is required only for the compiler
    // to distinguish between this one and the (x,y) constructor
    Point (float angle, float rad, boolean PolarDef)
    {
        x = cos(angle) * rad; y = sin(angle) * rad;
    }
}
// Class of colored points whose color is definable by two different color palettes
class ColorPoint extends Point
{
    float r, g, b; // object state variables
}
// Class of colored points whose color is definable by two different color palettes
class ColorPoint extends Point {
    float r, g, b; // object state variables

    // Here there are the constructors for RGB (for three different coord. systems)
    ColorPoint(float x, float y, float r, float g, float b) ... 
    ColorPoint(float comp, float r, float g, float b) ... 
    ColorPoint(float angle, float rad, boolean PolarDef, 
        float r, float g, float b) ...
    }

    // Here there are the constructors for CMYK (...)
    ColorPoint(float x, float y, float c, float m, float y, float k) ... 
    ColorPoint(Complex comp, float c, float m, float y, float k) ... 
    ColorPoint(float angle, float rad, boolean PolarDef, 
        float c, float m, float y, float k) ...
    
    }

Viviana Bono Jarosław Kuśmierek
// Class of colored points whose color is definable by two different color palettes
class ColorPoint extends Point
{
    float r, g, b;  // object state variables

    // Here there are the constructors for RGB (for three different coord. systems)
    ColorPoint (float x, float y, float r, float g, float b) ...
    ColorPoint (float comp, float r, float g, float b) ...
    ColorPoint (float angle, float rad, boolean PolarDef,
                float r, float g, float b) ...

    // While here there are the constructors for CMYK (...)
    ColorPoint (float x, float y, float c, float m, float y, float k) ...
    ColorPoint (Complex comp, float c, float m, float y, float k) ...
    ColorPoint (float angle, float rad, boolean PolarDef,
                float c, float m, float y, float k) ...
}

Viviana Bono Jarosław Kuśmirek
The Java Layers proposal

It allows to declare a set of constructors as a cartesian product of options from a superclass and from a given subclass. It does not allow more complex relations among options.

Example

class Class1
{
    propagate Class1(String s) {I1;}
    propagate Class1(int i) {I2;}
}
class Class2<T> extends T
{
    propagate Class2(int j) {I3;}
}
Avoiding explicit initialization protocols

Instead of constructors, we write ordinary methods responsible for initialization. Such an approach gives enough flexibility, however lacks any form of verification.
Avoiding explicit initialization protocols

Instead of constructors, we write ordinary methods responsible for initialization. Such an approach gives enough flexibility, however lacks any form of verification.

Support of formal specifications

We can check if the initialization methods are called properly by verifying some formal assumptions (with JML in Java, with Design By Contract in Eiffel). Such specifications gives great flexibility for the declaration of constraints, however they can be hard to write, and cannot be checked automatically in general.
JavaMIP solution
Modularity

The initialization is split into pieces (modules) - each one responsible for the initialization of an independent set of fields according to a certain initialization option.

Extensibility

Each subclass only declares what it changes, what it adds, what it extends, and what it replaces w.r.t. initialization.

Intuitive and simple initialization

Expressions creating objects specify some data, and the initialization modules are chosen and combined depending on the given data.
Modularity

The initialization is split into pieces (modules) - each one responsible for the initialization of an independent set of fields according to a certain initialization option.

Extensibility

Each subclass only declares what it changes, what it adds, what it extends, and what it replaces w.r.t. initialization.

Intuitive and simple initialization

Expressions creating objects specify some data, and the initialization modules are chosen and combined depending on the given data.
Modularity
The initialization is split into pieces (modules) - each one responsible for the initialization of an independent set of fields according to a certain initialization option.

Extensibility
Each subclass only declares what it changes, what it adds, what it extends, and what it replaces w.r.t. initialization.

Intuitive and simple initialization
Expressions creating objects specify some data, and the initialization modules are chosen and combined depending on the given data.
class Point
{
  float x, y;

  // main ini module for Point
  required Point(float x, float y) initializes ()
  { new []; this.x = x; this.y = y; }

  // module declaring that comp can be supplied instead of (x,y)
  optional Point(Complex comp) initializes (x, y)
  { new [x:=comp.x, y:= comp.y]; }
  // those instructions say how the translation from comp to (x,y) is done

  optional Point(float angle, float rad)
  initializes (x, y)
  { new [x:=cos(angle)*rad, y:=sin(angle)*rad]; }
}
class Point
{
float x, y;

// main ini module for Point
required Point(float x, float y) initializes ()
{ new []; this.x = x; this.y = y; }

// module declaring that comp can be supplied instead of (x,y)
optional Point(Complex comp) initializes (x, y)
{ new [x:=comp.x, y:=comp.y]; }
// those instructions say how the translation from comp to (x,y) is done

optional Point(float angle, float rad)
    initializes (x, y)
{ new [x:=cos(angle)*rad, y:=sin(angle)*rad]; }
}
class Point
{
  float x, y;

  // main ini module for Point
  required Point(float x, float y) initializes ()
  { new []; this.x = x; this.y = y; }

  // module declaring that comp can be supplied instead of (x,y)
  optional Point(Complex comp) initializes (x, y)
  { new [x:=comp.x, y:= comp.y]; }
  // those instructions say how the translation from comp to (x,y) is done

  optional Point(float angle, float rad)
  initializes (x, y)
  { new [x:=cos(angle)*rad, y:=sin(angle)*rad]; }
}
class Point
{
    float x, y;

    //main ini module for Point
    required Point(float x, float y) initializes ()
    { new []; this.x = x; this.y = y; }

    //module declaring that comp can be supplied instead of (x,y)
    optional Point(Complex comp) initializes (x,y)
    { new [x:=comp.x, y:= comp.y]; }
    //those instructions say how the translation from comp to (x,y) is done

    optional Point(float angle, float rad)
    initializes (x,y)
    { new [x:=cos(angle)*rad, y:=sin(angle)*rad]; }
}
class ColorPoint extends Point
{
    float r, g, b;

    // main ini module for ColorPoint
    required ColorPoint (float r, float g, float b)
    initializes ()
    { new[];// passes the control “above”
        this.r=r; this.g=g; this.b=b; }

    // optional ini module, initializing color via CMYK palette
    optional ColorPoint (float c, float m, float yc, float k)
    initializes (r,g,b)
    { float R1=...; float G1=...; float B1=...;
        new[r:=R1, g:=G1, b:=B1];
    }
}
Some base conditions are verified

- Each input parameter name occurs only once in each hierarchy.
- Each output parameter of a module references an input parameter of a module declared in a superclass, or above in the same class.
Some base conditions are verified

- Each input parameter name occurs only once in each hierarchy.
- Each output parameter of a module references an input parameter of a module declared in a superclass, or above in the same class.
// An expression creating a ColorPoint with polar coord. and CMYK color
new ColorPoint [angle:=1.2, rad:=4, c:=0, m:=1, yc:=1, k:=0];
// An expression creating a ColorPoint with polar coord. and CMYK color
new ColorPoint [angle:=1.2, rad:=4, c:=0, m:=1, yc:=1, k:=0];

Execution of an initialization expression

We take the set P of parameters passed by the “client” within the `new`. The hierarchy of a class is scanned bottom-up, and for each module:

- It is checked if all the input parameters of this module are in P.
- If no, it is skipped.
- If yes, it is executed and the output parameters are evaluated.
- The input parameters are removed from P, the output ones are added to P.
// An expression creating a ColorPoint with polar coord. and CMYK color
new ColorPoint [angle:=1.2, rad:=4, c:=0, m:=1, yc:=1, k:=0];

Execution of an initialization expression

We take the set P of parameters passed by the “client” within the new. The hierarchy of a class is scanned bottom-up, and for each module:

- It is checked if all the input parameters of this module are in P.
- If no, it is skipped.
- If yes, it is executed and the output parameters are evaluated.
- The input parameters are removed from P, the output ones are added to P.
// An expression creating a ColorPoint with polar coord. and CMYK color
new ColorPoint [angle:=1.2, rad:=4, c:=0, m:=1, yc:=1, k:=0];

**Execution of an initialization expression**

We take the set $P$ of parameters passed by the “client” within the `new`. The hierarchy of a class is scanned bottom-up, and for each module:

- It is checked if all the input parameters of this module are in $P$.
- If no, it is skipped.
- If yes, it is executed and the output parameters are evaluated.
- The input parameters are removed from $P$, the output ones are added to $P$. 
An expression creating a ColorPoint with polar coord. and CMYK color
new ColorPoint [angle:=1.2, rad:=4, c:=0, m:=1, yc:=1, k:=0];

**Execution of an initialization expression**

We take the set P of parameters passed by the “client” within the `new`.
The hierarchy of a class is scanned bottom-up, and for each module:

- It is checked if all the input parameters of this module are in P.
- If no, it is skipped.
- If yes, it is executed and the output parameters are evaluated.
- The input parameters are removed from P, the output ones are added to P.
An expression creating a ColorPoint with polar coord. and CMYK color
new ColorPoint [angle:=1.2, rad:=4, c:=0, m:=1, yc:=1, k:=0];

// Execution of an initialization expression

We take the set P of parameters passed by the “client” within the `new`. The hierarchy of a class is scanned bottom-up, and for each module:
- It is checked if all the input parameters of this module are in P.
- If no, it is skipped.
- If yes, it is executed and the output parameters are evaluated.
- The input parameters are removed from P, the output ones are added to P.

When the root of the hierarchy is reached, the set of parameters “to be consumed” is empty (this is enforced by the static checks).
class ColorPoint extends Point
{
  float r, g, b;

  // main ini module for ColorPoint
  required ColorPoint (float r, float g, float b)
    initializes ()
  { new[];
    this.r=r; this.g=g; this.b=b; }

  // optional ini module, initializing color via CMYK palette
  optional ColorPoint (float c, float m, float yc, float k)
    initializes (r,g,b)
  { float R1=...; float G1=...; float B1=...;
    new[r:=R1, g:=G1, b:=B1];
  }
}

<table>
<thead>
<tr>
<th>angle</th>
<th>rad</th>
<th>c</th>
<th>m</th>
<th>yc</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
class ColorPoint extends Point
{
  float r, g, b;

  //main ini module for ColorPoint
  required ColorPoint (float r, float g, float b)
    initializes ()
  { new[];
    this.r=r; this.g=g; this.b=b; }

  //optional ini module, initializing color via CMYK palette
  optional ColorPoint (float c, float m, float yc, float k)
    initializes (r,g,b)
  { float R1=...; float G1=...; float B1=...;
    new[r:=R1, g:=G1, b:=B1];
  }
}
class ColorPoint extends Point
{ float r, g, b;

    //main ini module for ColorPoint
required ColorPoint (float r, float g, float b)
    initializes ()
{ new[];
    this.r=r; this.g=g; this.b=b; }

    //optional ini module, initializing color via CMYK palette
optional ColorPoint (float c, float m, float yc, float k)
    initializes (r,g,b)
{ float R1=...; float G1=...; float B1=...;
    new[r:=R1, g:=G1, b:=B1];
}

class ColorPoint extends Point
{
    float r, g, b;

    // main ini module for ColorPoint
    required ColorPoint (float r, float g, float b)
        initializes ()
    {
        new[];
        this.r=r; this.g=g; this.b=b; }

    // optional ini module, initializing color via CMYK palette
    optional ColorPoint (float c, float m, float yc, float k)
        initializes (r,g,b)
    {
        float R1=...; float G1=...; float B1=...
        new[r:=R1, g:=G1, b:=B1];
    }
}
class ColorPoint extends Point
{
  float r, g, b;

  //main ini module for ColorPoint
  required ColorPoint (float r, float g, float b)
    initializes ()
  { new[];
    this.r=r; this.g=g; this.b=b; }

  //optional ini module, initializing color via CMYK palette
  optional ColorPoint (float c, float m, float yc, float k)
    initializes (r,g,b)
  { float R1=...; float G1=...; float B1=...;
    new[r:=R1, g:=G1, b:=B1];
  }
}
class Point
{
    float x, y;

    //main ini module for Point
    required Point(float x, float y) initializes ()
    { new []; this.x = x; this.y = y; }

    //module declaring that comp can be supplied instead of (x,y)
    optional Point(Complex comp) initializes (x,y)
    { new [x:=comp.x, y:= comp.y]; }
    //those instructions say how the translation from comp to (x,y) is done

    optional Point(float angle, float rad)
        initializes (x,y)
    { new [x:=cos(angle)*rad, y:=sin(angle)*rad]; }
}
class Point {
  float x, y;

  // main ini module for Point
  required Point(float x, float y) initializes ()
  { new []; this.x = x; this.y = y; }

  // module declaring that comp can be supplied instead of (x,y)
  optional Point(Complex comp) initializes (x,y)
  { new [x:=comp.x, y:= comp.y]; }
  // those instructions say how the translation from comp to (x,y) is done

  optional Point(float angle, float rad) initializes (x,y)
  { new [x:=cos(angle)*rad, y:=sin(angle)*rad]; }
}

\[
\begin{array}{c|c}
  x & y \\
  \hline
  1.45 & 3.73 \\
\end{array}
\]
class Point
{
    float x, y;

    //main ini module for Point
    required Point(float x, float y) initializes ()
    { new []; this.x = x; this.y = y; }

    //module declaring that comp can be supplied instead of (x,y)
    optional Point(Complex comp) initializes (x,y)
    { new [x:=comp.x, y:= comp.y]; }
    //those instructions say how the translation from comp to (x,y) is done

    optional Point(float angle, float rad)
        initializes (x,y)
    { new [x:=cos(angle)*rad, y:=sin(angle)*rad]; }
}
class Point
{
    float x, y;

    // main ini module for Point
    required Point(float x, float y) initializes ()
    {
        new []; this.x = x; this.y = y;
    }

    // module declaring that comp can be supplied instead of (x,y)
    optional Point(Complex comp) initializes (x,y)
    {
        new [x:=comp.x, y:=comp.y];
    }

    // those instructions say how the translation from comp to (x,y) is done

    optional Point(float angle, float rad)
    initializes (x,y)
    {
        new [x:=cos(angle)*rad, y:=sin(angle)*rad];
    }
}
Formal calculus

A core of the language JavaMIP was modelled as a formal calculus extending Featherweight Java.
A core of the language JavaMIP was modelled as a formal calculus extending Featherweight Java.

**Full language description**

- Support for default parameter values.
- Modular approach to exceptions.
- Backward compatibility with classic constructors (in various ways).
Formal calculus
A core of the language JavaMIP was modelled as a formal calculus extending Featherweight Java.

Full language description
- Support for default parameter values.
- Modular approach to exceptions.
- Backward compatibility with classic constructors (in various ways).
Formal calculus

A core of the language JavaMIP was modelled as a formal calculus extending Featherweight Java.

Full language description

- Support for default parameter values.
- Modular approach to exceptions.
- Backward compatibility with classic constructors (in various ways).
Formal calculus

A core of the language JavaMIP was modelled as a formal calculus extending Featherweight Java.

Full language description

- Support for default parameter values.
- Modular approach to exceptions.
- Backward compatibility with classic constructors (in various ways).

Implementation

By translating JavaMIP into Java source code, containing all the base language features, plus it allows a simple mixing of Java constructor style and JavaMIP ini module style.
<table>
<thead>
<tr>
<th>class name</th>
<th>constructors</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Java</td>
<td>JavaMIP</td>
</tr>
<tr>
<td>java.util.Formatter</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>18</td>
<td>8</td>
</tr>
<tr>
<td>javax.swing.JCheckBox</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>java.awt.Dialog</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>java.sql.Exception</td>
<td>120</td>
<td>4 or 1</td>
</tr>
<tr>
<td>+ 13 subclasses</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
class HSBColorPoint extends ColorPoint {
    optional HSBColorPoint(float h, float s, float b) initializes(r, g, b)
    {
        ...
    }
}