

Method Properties

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

ADAP C02

Licensed under [CC BY 4.0 International](#)

Method Properties

- A method property describes a particular property of a method
 - A method may have one property from any one type of method property
 - Different types of method properties should be orthogonal
- A method property comes with its own conventions
 - Naming conventions, for example, specific leading verbs
 - Specific implementation structures
- Like with method types, developers know and use these names

Types of Method Properties

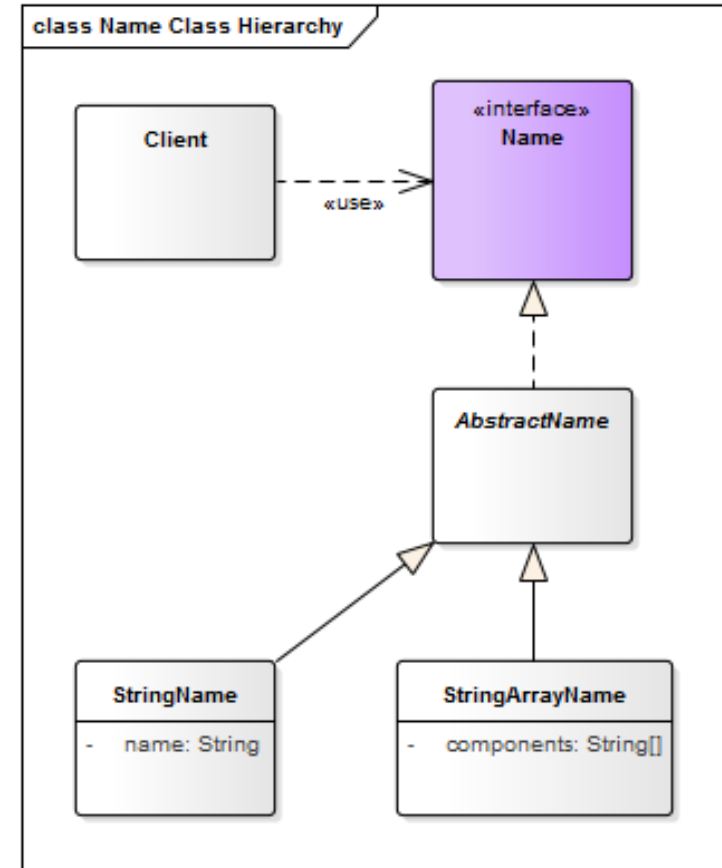
- **Implementation-related**
 - About the internal implementation: { regular, composed, primitive, null }
- **Inheritance-related**
 - About the inheritance interface: { regular, template, hook, abstract }
- **Convenience-related**
 - Making programming easier: { general, constructor, default-value }
- **Meta-level-related**
 - Which meta-level it applies to: { instance, class, meta-class }
- **Visibility-related**
 - Who can see and access: { public, protected, package-protected, private }
- ...

Types and Values of Method Properties

Implementation	Inheritance	Convenience
regular	regular	general
composed	template	constructor
primitive	hook	default-value
null	abstract	
...

A Class Hierarchy for Homogenous Names

- interface Name
 - Captures the Name interface
 - Is client-facing only (no implementation)
- abstract class AbstractName
 - Captures implementation commonalities
 - Defines inheritance interface
- class StringName
 - Represents name in single string
 - Implements inheritance interface
- class StringArrayName
 - Represents name in string array
 - Implements inheritance interface



Composed Method (Implementation)

Definition

A composed method is a method that organizes a task into several subtasks as a linear succession of method calls. Each subtask is represented by another method, primitive or non-primitive.

Also known as

-

JDK example

-

Name example

```
String AbstractName#getComponent(int)
void AbstractName#insert(int, String)
```

Prefixes

-

Comment

Name was taken from [B97].

Composed Method Examples

```
public String[] asStringArray() {
    int max = getNoComponents();
    String[] sa = new String[max];
    for (int i = 0; i < max; i++) {
        sa[i] = getComponent(i);
    }

    return sa;
}
```

```
protected void doInsert(int index, String component) {
    int newSize = getNoComponents() + 1;
    String[] newComponents = new String[newSize];
    for (int i = 0, j = 0; j < newSize; j++) {
        if (j != index) {
            newComponents[j] = components[i++];
        } else {
            newComponents[j] = component;
        }
    }
    components = newComponents;
}
```

Primitive Method (Implementation)

Definition	A primitive method is a method that carries out one specific task, usually by directly using the fields of the object. It does not rely on any (non-primitive) methods of the class that defines the primitive method.
Also known as	-
JDK example	-
Name example	<code>void AbstractName#assertIsValidIndex(int, int)</code> <code>String AbstractName#doGetComponent(int)</code>
Prefixes	basic, do
Comment	Design by Primitive is a key principle of good class design that uses primitive methods.

Primitive Method Examples

```
public String getComponent(int index) {  
    assertIsValidIndex(index);  
    return doGetComponent(index);  
}  
  
protected abstract String doGetComponent(int index);
```

```
protected String doGetComponent(int i) {  
    return components[i];  
}
```

```
protected String doGetComponent(int i) {  
    int startPos = getStartPositionOfComponent(i);  
    int endPos = getEndPositionOfComponent(i);  
    String maskedComponent = name.substring(startPos, endPos);  
    return NameHelper.unmaskString(maskedComponent);  
}
```

Template Method (Inheritance)

Definition

A template method is a method that defines an algorithmic skeleton for a task by breaking it into subtasks. Some of the subtasks are deferred to subclasses by means of hook methods.

Also known as

-

JDK example

-

Name example

```
Name Name#getContextName()  
String[] Name#asStringArray()
```

Prefixes

-

Comment

Name was taken from [G+95].

Template Method Examples

```
public String[] asStringArray() {  
    int max = getNoComponents();  
    String[] result = new String[max];  
    for (int i = 0; i < max; i++) {  
        result[i] = getComponent(i);  
    }  
    return result;  
}
```

```
public abstract int getNoComponents();
```

```
public String getComponent(int index) {  
    assertIsValidIndex(index);  
    return doGetComponent(index);  
}
```

```
protected abstract String doGetComponent(int index);
```

```
public String[] asStringArray() {  
    return Arrays.copyOf(components, components.length);  
}
```

Hook Method (Inheritance)

Definition

A hook method is a method that declares a well-defined task and makes it available for overriding through subclasses.

Also known as

-

JDK example

-

Name example

String AbstractName#doGetComponent(int)
Name AbstractName#doInsert(int, String)

Prefixes

-

Comment

-

Hook Method Examples

```
public String[] asStringArray() {
    int max = getNoComponents();
    String[] result = new String[max];
    for (int i = 0; i < max; i++) {
        result[i] = getComponent(i);
    }

    return result;
}

public abstract int getNoComponents();

public String getComponent(int index) {
    assertIsValidIndex(index);
    return doGetComponent(index);
}

protected abstract String doGetComponent(int index);
```

Hook Method Examples

```
public String[] asStringArray() {
    int max = getNoComponents();
    String[] result = new String[max];
    for (int i = 0; i < max; i++) {
        result[i] = getComponent(i);
    }

    return result;
}

public abstract int getNoComponents();

public String getComponent(int index) {
    assertIsValidIndex(index);
    return doGetComponent(index);
}

protected abstract String doGetComponent(int index);
```

Convenience Method (Convenience)

Definition

A convenience method is a method that simplifies the use of another, more complicated method by providing a simpler signature and by using default arguments where the client supplies no arguments.

Also known as

-

JDK example

`String BigInteger::toString()` (wraps `String BigInteger::toString(int radix)`)

Name example

`String Name#getFirstComponent()`
`String Name#asString()`

Prefixes

-

Comment

Name was taken from [H00].

Convenience Method Examples

```
public String getFirstComponent() {  
    return getComponent(0);  
}  
  
public String asString() {  
    return asString(getDelimiterChar());  
}
```


Default-Value Method (Convenience)

Definition

A default-value method is a method that returns a single pre-defined value, like a constant, but changeable by subclasses.

Also known as

-

JDK example

-

Name example

```
public char AbstractName#getDelimiterChar()  
public char AbstractName#getEscapeChar()
```

Prefixes

-

Comment

-

Default-Value Method Examples

```
public static final char DEFAULT_DELIMITER_CHAR = '#';  
public static final String DEFAULT_DELIMITER_STRING = "#";  
public static final char DEFAULT_ESCAPE_CHAR = '\\';  
public static final String DEFAULT_ESCAPE_STRING = "\\";
```

```
public char getDelimiterChar() {  
    return DEFAULT_DELIMITER_CHAR;  
}  
  
public char getEscapeChar() {  
    return DEFAULT_ESCAPE_CHAR;  
}
```

Making Method Properties Explicit in Code

- Annotate in comments using `@MethodProperties` list-of-properties

Review / Summary of Session

- General method properties
 - What are method types?
 - What categories of method properties are there?
- Specific method properties
 - What specific method properties are there? How common are they?
 - How are they defined? What naming convention do they follow?
- Interactions of methods
 - How do methods interact? How is this reflected in their properties?

Thank you! Questions?

dirk.riehle@fau.de – <http://osr.cs.fau.de>

dirk@riehle.org – <http://dirkriehle.com> – [@dirkriehle](#)

Credits and License

- Original version
 - © 2012-2019 [Dirk Riehle](#), some rights reserved
 - Licensed under [Creative Commons Attribution 4.0 International License](#)
- Contributions
 - ...

Method Properties

Prof. Dr. Dirk Riehle
Friedrich-Alexander University Erlangen-Nürnberg

ADAP C02

Licensed under [CC BY 4.0 International](#)

It is Friedrich-Alexander University Erlangen-Nürnberg – FAU, in short.
Corporate identity wants us to say “Friedrich-Alexander University”.

Method Properties

- A method property describes a particular property of a method
 - A method may have one property from any one type of method property
 - Different types of method properties should be orthogonal
- A method property comes with its own conventions
 - Naming conventions, for example, specific leading verbs
 - Specific implementation structures
- Like with method types, developers know and use these names

Types of Method Properties

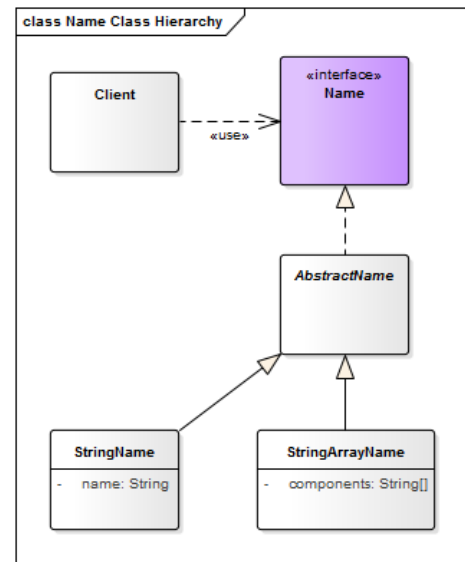
- **Implementation-related**
 - About the internal implementation: { regular, composed, primitive, null }
- **Inheritance-related**
 - About the inheritance interface: { regular, template, hook, abstract }
- **Convenience-related**
 - Making programming easier: { general, constructor, default-value }
- **Meta-level-related**
 - Which meta-level it applies to: { instance, class, meta-class }
- **Visibility-related**
 - Who can see and access: { public, protected, package-protected, private }
- ...

Types and Values of Method Properties

Implementation	Inheritance	Convenience
regular	regular	general
composed	template	constructor
primitive	hook	default-value
null	abstract	
...

A Class Hierarchy for Homogenous Names

- interface Name
 - Captures the Name interface
 - Is client-facing only (no implementation)
- abstract class AbstractName
 - Captures implementation commonalities
 - Defines inheritance interface
- class StringName
 - Represents name in single string
 - Implements inheritance interface
- class StringArrayName
 - Represents name in string array
 - Implements inheritance interface



Composed Method (Implementation)

Definition	A composed method is a method that organizes a task into several subtasks as a linear succession of method calls. Each subtask is represented by another method, primitive or non-primitive.
Also known as	-
JDK example	-
Name example	String AbstractName#getComponent(int) void AbstractName#insert(int, String)
Prefixes	-
Comment	Name was taken from [B97].

Composed Method Examples

```
public String[] asStringArray() {
    int max = getNoComponents();
    String[] sa = new String[max];
    for (int i = 0; i < max; i++) {
        sa[i] = getComponent(i);
    }

    return sa;
}
```

```
protected void doInsert(int index, String component) {
    int newSize = getNoComponents() + 1;
    String[] newComponents = new String[newSize];
    for (int i = 0, j = 0; j < newSize; j++) {
        if (j != index) {
            newComponents[j] = components[i++];
        } else {
            newComponents[j] = component;
        }
    }
    components = newComponents;
}
```

Primitive Method (Implementation)

Definition	A primitive method is a method that carries out one specific task, usually by directly using the fields of the object. It does not rely on any (non-primitive) methods of the class that defines the primitive method.
Also known as	-
JDK example	-
Name example	<code>void AbstractName#assertIsValidIndex(int, int)</code> <code>String AbstractName#doGetComponent(int)</code>
Prefixes	basic, do
Comment	Design by Primitive is a key principle of good class design that uses primitive methods.

Primitive Method Examples

```
public String getComponent(int index) {  
    assertIsValidIndex(index);  
    return doGetComponent(index);  
}
```

```
protected abstract String doGetComponent(int index);
```

```
protected String doGetComponent(int i) {  
    return components[i];  
}
```

```
protected String doGetComponent(int i) {  
    int startPos = getStartPositionOfComponent(i);  
    int endPos = getEndPositionOfComponent(i);  
    String maskedComponent = name.substring(startPos, endPos);  
    return NameHelper.unmaskString(maskedComponent);  
}
```

Template Method (Inheritance)

Definition	A template method is a method that defines an algorithmic skeleton for a task by breaking it into subtasks. Some of the subtasks are deferred to subclasses by means of hook methods.
Also known as	-
JDK example	-
Name example	Name Name#getContextName() String[] Name#asStringArray()
Prefixes	-
Comment	Name was taken from [G+95].

Template Method Examples

```
public String[] asStringArray() {
    int max = getNoComponents();
    String[] result = new String[max];
    for (int i = 0; i < max; i++) {
        result[i] = getComponent(i);
    }
    return result;
}

public abstract int getNoComponents();

public String getComponent(int index) {
    assertIsValidIndex(index);
    return doGetComponent(index);
}

protected abstract String doGetComponent(int index);

public String[] asStringArray() {
    return Arrays.copyOf(components, components.length);
}
```

Hook Method (Inheritance)

Definition	A hook method is a method that declares a well-defined task and makes it available for overriding through subclasses.
Also known as	-
JDK example	-
Name example	String AbstractName#doGetComponent(int) Name AbstractName#doInsert(int, String)
Prefixes	-
Comment	-

Hook Method Examples

```
public String[] asStringArray() {
    int max = getNoComponents();
    String[] result = new String[max];
    for (int i = 0; i < max; i++) {
        result[i] = getComponent(i);
    }

    return result;
}

public abstract int getNoComponents();

public String getComponent(int index) {
    assertIsValidIndex(index);
    return doGetComponent(index);
}

protected abstract String doGetComponent(int index);
```

Hook Method Examples

```
public String[] asStringArray() {
    int max = getNoComponents();
    String[] result = new String[max];
    for (int i = 0; i < max; i++) {
        result[i] = getComponent(i);
    }

    return result;
}

public abstract int getNoComponents();

public String getComponent(int index) {
    assertIsValidIndex(index);
    return doGetComponent(index);
}

protected abstract String doGetComponent(int index);
```

Convenience Method (Convenience)

Definition	A convenience method is a method that simplifies the use of another, more complicated method by providing a simpler signature and by using default arguments where the client supplies no arguments.
Also known as	-
JDK example	<code>String BigInteger::toString()</code> (wraps <code>String BigInteger::toString(int radix)</code>)
Name example	<code>String Name#getFirstComponent()</code> <code>String Name#asString()</code>
Prefixes	-
Comment	Name was taken from [H00].

Convenience Method Examples

```
public String getFirstComponent() {  
    return getComponent(0);  
}  
  
public String asString() {  
    return asString(getDelimiterChar());  
}
```

Default-Value Method (Convenience)

Definition	A default-value method is a method that returns a single pre-defined value, like a constant, but changeable by subclasses.
Also known as	-
JDK example	-
Name example	<code>public char AbstractName#getDelimiterChar() public char AbstractName#getEscapeChar()</code>
Prefixes	-
Comment	-

Default-Value Method Examples

```
public static final char DEFAULT_DELIMITER_CHAR = '#';  
public static final String DEFAULT_DELIMITER_STRING = "#";  
public static final char DEFAULT_ESCAPE_CHAR = '\\';  
public static final String DEFAULT_ESCAPE_STRING = "\\\";
```

```
public char getDelimiterChar() {  
    return DEFAULT_DELIMITER_CHAR;  
}  
  
public char getEscapeChar() {  
    return DEFAULT_ESCAPE_CHAR;  
}
```


Making Method Properties Explicit in Code

- Annotate in comments using `@MethodProperties` list-of-properties

Review / Summary of Session

- General method properties
 - What are method types?
 - What categories of method properties are there?
- Specific method properties
 - What specific method properties are there? How common are they?
 - How are they defined? What naming convention do they follow?
- Interactions of methods
 - How do methods interact? How is this reflected in their properties?

Thank you! Questions?

dirk.riehle@fau.de – <http://osr.cs.fau.de>

dirk@riehle.org – <http://dirkriehle.com> – [@dirkriehle](#)

DR

Credits and License

- Original version
 - © 2012-2019 [Dirk Riehle](#), some rights reserved
 - Licensed under [Creative Commons Attribution 4.0 International License](#)
- Contributions
 - ...