Method Types and Properties

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

ADAP C01

Agenda

1. Method types
    2. Query methods
    3. Mutation methods
    4. Helper methods
5. Method properties
    6. Implementation related
    7. Inheritance related
    8. Convenience methods
9. Design guidelines

1. Method Types

Method Types

- A method type classifies a method into a particular type
  - The method type is indicative of the main purpose
  - A method may have only one type, not many
  - Thus, a method should have one purpose
- Different method types are orthogonal to each other
  - **Query methods**
  - **Mutation methods**
  - **Helper methods**
- A method type comes with its own conventions
  - Naming conventions, for example, specific leading verbs
  - Specific implementation structures

Main Categories of Method Types

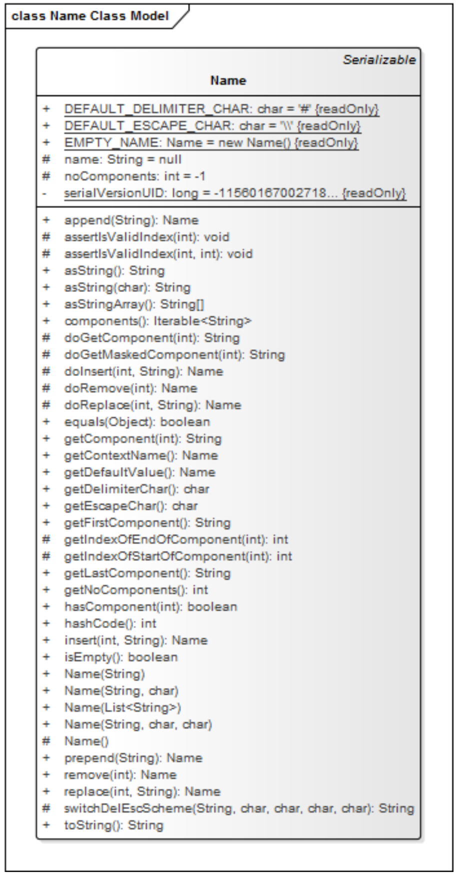
- Query methods
  - Methods that return information about the object but don’t change its state
- Mutation methods
  - Methods that change the object’s state but don’t provide information back
- Helper methods
  - Methods that perform some utility function independent of the object

Categories and Examples of Method Types

Query Method	Mutation Method	Helper Method
get method (getter)	set method (setter)	factory method
boolean query method	command method	cloning method
comparison method	initialization method	assertion method
conversion method	finalization method	logging method
...	...	...

A Simple Class for Homogenous Names

- Homogenous name
  - A multi-component text string
    - of same-type components with
    - a single delimiter character
- Homogenous name examples
  - Domain names (“www.jvalue.org”)
  - Java path names (“org.jvalue.names.Name”)
  - Directory paths (“/home/dirk/docs”)
- Unlike heterogeneous names
  - “http://www.jvalue.org/index.html”



```
class Name Class Model

                                              Serializable
                         Name
+  DEFAULT_DELIMITER_CHAR: char = '#' {readOnly}
+  DEFAULT_ESCAPE_CHAR: char = '\\' {readOnly}
+  EMPTY_NAME: Name = new Name() {readOnly}
#  name: String = null
#  noComponents: int = -1
-  serialVersionUID: long = -11560167002718... {readOnly}

+  append(String): Name
#  assertIsValidIndex(int): void
#  assertIsValidIndex(int, int): void
+  asString(): String
+  asString(char): String
+  asStringArray(): String[]
+  components(): Iterable<String>
#  doGetComponent(int): String
#  doGetMaskedComponent(int): String
#  doInsert(int, String): Name
#  doRemove(int): Name
#  doReplace(int, String): Name
+  equals(Object): boolean
+  getComponent(int): String
+  getContextName(): Name
+  getDefaultValue(): Name
+  getDelimiterChar(): char
+  getEscapeChar(): char
+  getFirstComponent(): String
#  getIndexOfEndOfComponent(int): int
#  getIndexOfStartOfComponent(int): int
+  getLastComponent(): String
+  getNoComponents(): int
+  hasComponent(int): boolean
+  hashCode(): int
+  insert(int, String): Name
+  isEmpty(): boolean
+  Name(String)
+  Name(String, char)
+  Name(List<String>)
+  Name(String, char, char)
#  Name()
+  prepend(String): Name
+  remove(int): Name
+  replace(int, String): Name
#  switchDelEscScheme(String, char, char, char, char): String
+  toString(): String
```

General Method Naming Rule

verb

(+ optional noun)

then, be as specific as possible

2. Query Methods

Get Method (Query Method)

Definition	A get method is a query method that returns a (logical) field of the object.
Also known as	Getter
JDK example	Class Object#getClass() Object Enumeration#nextElement()
Name example	String getComponent(int) Iterable<String> getComponentIterator()
Prefixes	get
Naming	After the prefix, the name of the field being queried follows.

Get Method (Getter) Examples

```
public class Name {

  protected String name;
  protected int noComponents;

  public int getNoComponents() { // get method
    return noComponents;
  }

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

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

  ...
}
```

Boolean Query Method (Query Method)

Definition	A boolean query method is a query method that returns a boolean value.
Also known as	-
JDK example	boolean Object#equals()
Name example	boolean isEmpty()
Prefixes	is, has, may, can, …
Naming	After the prefix, the aspect being queried follows.

Boolean Query Method Examples

```
public boolean hasComponent(int index) { // boolean query method
  return doHasComponent(index);
}

protected boolean doHasComponent(int index) { // boolean query method
  return (0 <= index) && (index < getNoComponents());
}

public boolean isEmpty() { // boolean query method
  return getNoComponents() == 0;
}
```

Comparison Method (Query Method)

Definition	A comparison method is a query method that compares two objects using an ordinal scale.
Also known as	Comparing method
JDK example	int Comparable#compareTo(Object)
Name example	-
Prefixes	-
Naming	-

Comparison Method Examples

```
public int compareTo(Integer anotherInteger) { // comparison method
  int thisVal = this.value;
  int anotherVal = anotherInteger.value;
  if (thisVal < anotherVal) {
    return -1;
  } else if (thisVal == anotherVal) {
    return 0;
  } else {
    return 1;
  }
}
```

Conversion Method (Query Method)

Definition	A conversion method is a query method that returns a different representation of this object.
Also known as	Converter method, interpretation method
JDK example	String Object#toString() int Integer#intValue()
Name example	String asString() String asStringWith(char)
Prefixes	as, to
Naming	After the prefix, typically the class name being converted to follows.

Conversion Method Examples

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

  return sa;
}
```

3. Mutation Methods

Set Method (Mutation Method)

Definition	A set method is a mutation method that sets a (logical) field of an object to some value.
Also known as	Setter
JDK example	void Thread#setPriority(int)
Name example	void Name#setComponent(int, String)
Prefixes	set
Naming	After the prefix (if any), the field being changed follows.

Set Method (Setter) Examples

```
public void setComponent(int i, String c) { // set method
  assertIsValidIndex(i);
  assertIsNonNullArgument(c);
  int oldNoComponents = getNoComponents();

  doSetComponent(i, c);

  assert c.equals(getComponent(i)) : "postcondition failed";
  assert oldNoComponents == getNoComponents() : "pc failed";
}

protected void doSetComponent(int i, String c) { // set method
  doInsert(i, c);
  doRemove(i + 1);
}
```

Command Method (Mutation Method)

Definition	A command method is a method that executes a complex change to the object's state.
Also known as	-
JDK example	void Object#notify() void JComponent#repaint()
Name example	void insert(int i, String c) void remove(int i)
Prefixes	handle, execute, make
Naming	-

Command Method Examples

```java
public void insert(int i, String c) { // command method
  assertIsValidIndex(i, getNoComponents() + 1);
  assertIsNonNullArgument(c);
  int oldNoComponents = getNoComponents();

  doInsert(i, c);

  assert (oldNoComponents + 1) == getNoComponents() : "pc failed";
}

protected void doInsert(int index, String component) { // command method
  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;
}
```

Initialization Method (Mutation Method)

Definition	An initialization method is a mutation method that sets some or all fields of an object to an initial value.
Also known as	-
JDK example	void LookAndFeel#initialize()
Name example	-
Prefixes	init, initialize
Naming	If prefixed with init, typically the name of the object part being initialized follows.

Initialization Method Examples

```
public class NameTest {

  protected Name defaultName;
  protected Name emptyDefaultName;
  protected Name compactName;
  protected Name emptyCompactName;

  protected void initNames(String[] arg) { // initialization method
    defaultName = new StringArrayName(arg);
    compactName = new StringName(arg);
  }

  protected void initEmptyNames() { // initialization method
    emptyDefaultName = new StringArrayName();
    emptyCompactName = new StringName();
  }

  ...

}
```

4. Helper Methods

Factory Method (Helper Method)

Definition	A factory method is a helper method that creates an object and returns it to the client.
Also known as	Object creation method
JDK example	String String#valueOf(int) String String#valueOf(double)
Name example	-
Prefixes	create, make, build, (new, getNew)
Naming	After the prefix, the product name follows.

Factory Method Example

```
public class PhotoManager extends ObjectManager {

  protected static final PhotoManager instance = new PhotoManager();

  ...

  public Photo createPhoto(String fn, Image ui) throws Exception { // factory method
    PhotoId id = PhotoId.getNextId();
    Photo result = PhotoUtil.createPhoto(fn, id, ui);
    addPhoto(result);
    return result;
  }

  ...
}
```

Quiz: Naming a Factory Method

1. You need a method to create a new photo. Which name is best?
    1. PhotoFactory.make()
    2. PhotoFactory.newPhoto()
    3. PhotoFactory.createPhoto()
    4. PhotoFactory.createNewPhoto()

More Object Creation Methods

- Cloning method: Clone object at hand
- Factory method: Create related object
    - Copying method: Create by copying argument
    - Trading method: Create by resolving specification

- ...

Naming Object Creation Methods

- Creation methods prefixed with “create”
  - Should create the object (sans abnormal situation)
- Creation methods prefixed with “ensure”
  - Should ensure a particular object exists, possibly creating it

Assertion Method (Helper Method)

Definition	An assertion method is a helper method that tests a condition. If the condition holds, it returns silently. If it does not, an exception is thrown.
Also known as	-
JDK example	void AccessControlContext#checkPermission(Permission) throws AccessControlException
Name example	void Name#assertIsValidIndex(int) throws IndexOutOfBoundsException
Prefixes	assert, check, test
Naming	After the prefix, the condition being checked follows.

Assertion Method Examples

```
protected void assertIsValidIndex(int i) // assertion method
  throws IndexOutOfBoundsException {
  assertIsValidIndex(i, getNoComponents());
}

protected void assertIsValidIndex(int i, int upperLimit) // assertion method
  throws IndexOutOfBoundsException {
  if ((i < 0) || (i >= upperLimit)) {
    throw new IndexOutOfBoundsException("invalid index = " + i);
  }
}

protected void assertIsNonNullArgument(Object o) { // assertion method
  if (o == null) {
    throw new IllegalArgumentException("received null argument");
  }
}
```

5. Method Properties

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

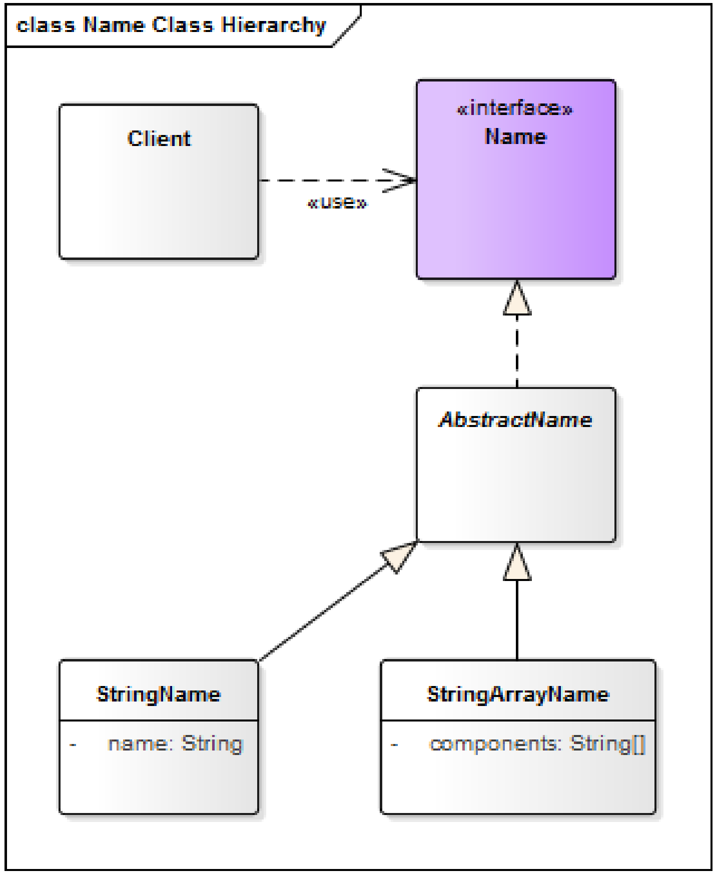
- Implementation-related
  - About the internal implementation: { regular, composed, primitive, null }
- Inheritance-related
  - About the inheritance interface: { regular, template, hook, abstract }
- Convenience-related
  - Making programming easier: { general, constructor, default-value }
- Meta-level-related
  - 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



6. Implementation Related

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() { // composed method
  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) { // composed method
  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	void AbstractName#assertIsValidIndex(int, int) String AbstractName#doGetComponent(int)
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); // primitive method declaration
```

```
protected String doGetComponent(int i) { // primitive method implementation
  return components[i];
}
```

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

7. Inheritance Related

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() { // template method
  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(); // hook method declaration

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

protected abstract String doGetComponent(int index); // hook method declaration
```

8. Convenience Methods

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() { // convenience method
  return getComponent(0);
}

public String asString() { // convenience method
  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() { // default-value method
  return DEFAULT_DELIMITER_CHAR;
}

public char getEscapeChar() { // default-value method
  return DEFAULT_ESCAPE_CHAR;
}
```

9. Design Guidelines

Quiz: java.lang.Object Method Names

1. How do you classify these methods?
   - protected Object clone()
   - boolean equals(Object obj)
   - protected void finalize()
   - Class<?> getClass()
   - int hashCode()
   - void notify()
   - void notifyAll()
   - String toString()
   - void wait()
   - void wait(long timeout)
   - void wait(long timeout, int nanos)

Single-Purpose Rule

- Definition of single-purpose rule
  - A method should serve one main purpose
  - Derives from single method-type rule
- Benefit of single-purpose rule
  - Make the method more easy to understand
  - Makes overriding methods easier

Exceptions to the Single-Purpose Rule

- Critical sections
  - Increment and return
    - Special case: Iterators
- Lazy initialization

Making Method Types Explicit in Code

- Annotate (in comments) using @MethodType method-type

Making Method Properties Explicit in Code

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

Summary

1. Method types
   2. Query methods
   3. Mutation methods
   4. Helper methods
5. Method properties
   6. Implementation related
   7. Inheritance related
   8. Convenience methods
9. Design guidelines

Thank you! Questions?

dirk.riehle@fau.de – https://oss.cs.fau.de

dirk@riehle.org – https://dirkriehle.com – @dirkriehle

Legal Notices