Object-Oriented Programming

Method Stereotypes

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Method Stereotypes

- All methods play a *role* or multiple *roles* in a class

- Identifying the role of a method helps with comprehension but also with the original design

- A method has a *primary* role, with some methods a *secondary* role

Method Stereotype Taxonomy

- Detailed Explanation: Reverse Engineering Method Stereotypes (ICSM'06) by Dragan, N., Collard, M.L., Maletic, J.I.

- Updated Taxonomy: Automatic Identification of Class Stereotypes (ICSM'10) by Dragan, N., Collard, M.L., Maletic, J.I.

- Application: Which Method-Stereotype Changes are Indicators of Code Smells (SCAM'18) by Decker, M., Newman, C., Dragan, N., Collard, M.L., Kraft, N.A., Maletic, J.I.

- Application: Automa Source Code with M Stereotypes (DySDo Collard, M.L., Dragan C., Decker, M.

Documentation

```
// @stereotype get
const std::string& getFirstName() const {
    return name;
}

// @stereotype get
const std::string& firstName() const {
    return name;
}

// @stereotype get
bool isDone() const {

    return done;
}
```

- Method stereotypes can be embedded in the source code as a comment

- *documentation* if the identification is manual

- *redocumentation* if the identification is automated

- Useful to be able to update in place

Taxonomy

Stereotype Category	Description
Accessors	Read access to the state of the object, both direct and derived
Mutators	Write access to the state of the object
Creational Methods	Create and manage the lifetime of objects
Collaborational Methods	Work with objects of other classes
Degenerate Methods	No access or changes to state

Accessors

Stereotype	Description
get	Returns the value of a data member
predicate	Returns a Boolean result computed from data members
property	Returns information based on data member values
void-accessor	Returns information about data members through a parameter

Accessor::get

```cpp
// @stereotype get
const std::string& getFirstName() const {
    return name;
}

// @stereotype get
const std::string& firstName() const {
    return name;
}

// @stereotype get
bool isDone() const {

    return done;
}
```

- *Returns the value of a data member*
- The purpose of the method is very simple and primitive
- Direct access to the value of the data member
- C++ Rules:

  Method is const

  Returns a data member

  Return type is primitive or container of a primitive

Accessor::predicate

```
// @stereotype predicate
bool isEmpty() const {
    return v.empty();
}

// @stereotype predicate
bool empty() const {
    return v.empty();
}
```

- *Returns a Boolean result computed from data members*

- Result is not a direct data member but a computation involving data members

- C++ Rules:

  Method is const

  Returns a Boolean value that is not a data member

Accessor::property

```
// @stereotype predicate
int indexOfMin(int index) const {
    int minIndex = 0;
    for (int i = 0; i < data.size(); ++i) {
        if (data[i] < data[minIndex])
            minIndex = i;
    }

    return minIndex;
}
```

- *Returns information based on data member values*
- C++ Rules:

  Method is const

  Does not return a data member

  Return type is primitive or container of primitives

  Return type is not Boolean

Accessor::void-accessor

```
// @stereotype void-accessor
void minElement(Element& element) const {
    int minIndex = 0;
    for (int i = 0; i < data.size(); ++i) {
        if (data[i] < data[minIndex])
            minIndex = i;
    }

    element = data[minIndent];
}
```

- *Returns information about data members through a parameter*

Mutators

Stereotype	Description
set	changes the value of a data member
command	executes a complex change of the object's state
non-void-command	command which returns a value

Mutator::set

```cpp
// @stereotype set
void setName(std::string_view name);
```

- *Directly changes the value of a data member*

- The parameter value is stored in the data member

- C++ Rules:

  Method is not const

  Return type is void or Boolean

  Only one data member is changed

Mutator::command

```
// @stereotype command
void draw(int x, int y);
```

- *Executes a complex change of the object's state*
- The change may involve several data members
- May change the data members either directly or indirectly with another mutator
- C++ Rules:

  Method is not const

  Return type is void or Boolean

  Complex change to the object's state is performed, e.g., more than one data member is changed

Mutator::non-void-command

```
// @stereotype non-void-command
int reshape(int x, int y);
```

- *Command which returns a value*

Collaborational Methods

Stereotype	Description
collaborator	Works on objects of classes different from itself
controller	Works only on objects of classes other than itself

Collaborational::Collaborator

```
// @stereotype collaborator
bool hasControlPoints(const PlotterBase* plotter) const;
```

- *Works on objects of classes different from itself*

- C++ Rules:

Returns void, and at least one of the method's parameters or local variables is an object

Returns a parameter or local variable that is an object

Collaborational::Controller

```
// @stereotype controller
void setBinner(PlotterBase* plotter,
               Axes::Type axis);
```

- *Works only on objects of classes different from itself*

Creational Methods

Stereotype	Example/Description
constructor	`// @stereotype constructor` `DataSource();`
copy-constructor	`// @stereotype copy-constructor` `DataSource(const DataSource&);`
destructor	`// @stereotype destructor` `~DataSource();`
factory	Object creation method

Creational::factory

```cpp
// @stereotype factory
PlotterBase* createDisplay(std::string_view name);
```

- *Object creation method with the object returned to the client*

Degenerate Methods

Stereotype	Description
incidental	Does not read/change the object state, and no calls to other methods of the same class
stateless	Does not read/modify the object state with one call to other methods of the same class
empty	Method with no statements

Degenerate::incidental

```
// @stereotype incidental
Point add(const Point& n1, const Point& n2) {
    return n1 + n2;
}
```

- *Does not read/change the object state, and no calls to other methods of the same class*

Degenerate::stateless

```
// @stereotype stateless
void sort(Point& n1, Point& n2) {
    sort(n1, n2,
         [](auto n1, auto n2){ return n1 < n2; });
}
```

- *Does not read/change the object state with one call to other methods of the same class*

Degenerate::empty

```
// @stereotype empty
void Point::process() const {

}
```

- *Method with no statements*

- Typically part of an early implementation

- Maybe due to functionality moving elsewhere

Clear Design

```
class Stack {
public:

    // @stereotype constructor
    Stack();

    // push an element on top of the stack
    // @stereotype command
    void push(int);

    // remove the element at the top of the stack
    // @stereotype command
    void pop();

    // element at the top of the stack
    // @stereotype property
    int top() const;

    // does the stack have any elements
    // @stereotype predicate
    bool empty() const;

    // @stereotype destructor
    ~Stack();
};
```

- A method should have a single purpose, *single responsibility issue*

- Conversely, a method should have a single primary *role*, i.e., *single role issue*

Pop or Top vs. Pop and Top

```
// remove the element at the top of the stack
// and return it
// @stereotype command property
int pop();

// element at the top of the stack
// @stereotype property
int top() const;
```

```
// remove the element at the top of the stack
// @stereotype command
void pop();

// element at the top of the stack
// @stereotype property
int top() const;
```