|  |
| --- |
| Circle Language Spec |

## Basic Diagram Elements

This article attempts to give a more or less formal overview of elements that might appear in the diagrams.

The diagrams might involve set of basic elements, that may include the following:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | |  | |  | |
|  | | |  | |  | |  | |

### Text

Diagrams might also contain text. Shapes might have names that might appear as text. Numbers and dates and other values could also appear as text.

**MyObject**

**My Site**

**7**

**3.141593**

**2005-08-14**

**"Hello"**

**True**

**False**

**On**

**Off**

Perhaps some of the code might not be visualized graphically, but just displayed as text expressions:

**A = B + C**

### Circles & Triangles

Circles and triangles may represent objects, classes or interfaces. An object might serve as the class or prototype of another object. An object could perhaps also describe the interface for another object. A choice was made to have objects, classes and interfaces be depicted by similar shapes.

A circle might be the usual symbol for an object. A triangle might have a particular function related to interfaces, which may be clarified by other text.

### Squares & Diamonds

|  |  |
| --- | --- |
|  |  |

Squares and diamonds may stand for *commands*. A square may denote an *inactive* command, that might not run. Such a square might be use as a *definition* for other commands. A diamond might depict *execution*. A diamond might execute, while a square may not.

### Pentagons



A pentagon could represent a *module*. A module is like a little world in which objects and commands live. A module may be a site, a computer program, a library or any other kind of module. A pentagon could be considered an object just like circles and triangles, but a pentagon may have special purposes and behavior.

### Nonagon



A nonagon may stand for a collection or a list of things. Objects might be placed inside the nonagon, to group them together perhaps. Maybe nonagons might not be drawn with precisely nine corners.

### Object Symbols

Circles, triangles, pentagons and nonagons could be called *object symbols*.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

### Command Symbols

Squares and diamonds could be called *command* *symbols*, since they appeared to represent commands.

|  |  |
| --- | --- |
|  |  |

### Lines

The idea is that, when symbols would be connected with lines, those symbols would have something in common.



Now follows an attempt to summarize how that might work.

### Solid Line



A solid line may point out an *object*. Symbols connected by a solid line might symbolize the same object. Other names for this line might be an object line or a reference line.

### Dashed Line



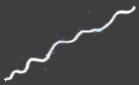
A dashed line could point out a *class* or *definition*. Symbols connected by dashed lines could be thought of as having the same class. These symbols may share similar behavior, while they might still be living as individual objects. Other names for this line might be a class line or a definition line.

### Dotted Line



A dotted line might point out an *interface*. Shapes connected by dotted lines might give them the same interface. The idea behind that is that they may look the same from the outside, but might behave differently on the inside. Another name for this line might be an interface line.

### Wavy Line



A wavy line could represent the concept of *values*. When shapes are connected with a wavy line, it might mean, they have the same value or perhaps soon will get the same value. Another name for this line might be a *value line*.

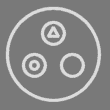
### Cross

A cross might be placed inside a symbol to indicate, that an symbol is *nothing / null*: it might not reference any object: it is empty.



### Relations Between Symbols

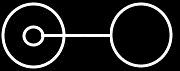
Relations between symbols might be expressed by *containment*:



and then possibly by connecting them with *lines:*



#### Object Reference



This object reference notation may be seen as that the bigger object on the left would contain a reference to the bigger object on the right. It is almost as if the left object contains the other object, but instead of containing it directly it would be pointing out to something on the outside.

### Connecting Object Symbols

Object symbols could be connected by different types of lines.

#### Object Connection

When two object symbols would be connected by a solid line, it might mean that both symbols represent the same object.



One symbol might be considered to represent the actual object while the other symbol might represent an *object reference*. An object line may point out which object an object symbol would point to. The solid line might be called an *object line* in this case, because it may point out the object.

#### Class Connection

When two object symbols would be connected by a dashed line, it might mean that both symbols may share the same class, which could mean they may have the same behavior, while they might not represent the same individual object.



A class line may point out, what class an object might have.

#### Interface Connection

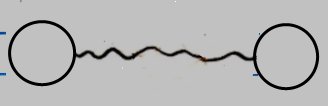
If two object symbols would be connected by a dotted line, this might depict that the symbols would have the same interface, which may mean they could look the same from the outside, but might be different on the inside.



The two symbols might be two separate individual objects, they may also have a different class, so could behave differently, but since they seem to have the same interface, this might mean that from the outside they might look the same.

#### Value Connection

When two object symbols would be connected with a wavy line, it might mean the objects would get the same value.



This could express that the values of the two objects are the same. It might also mean transferring a value from one object to the next.

### Connecting Command Symbols

Connecting command symbols could be something parallel to connections between object symbols. Command symbols might also be connected with different types of lines.

#### Command Reference Connection

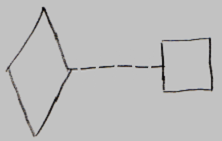
When two command symbols would be connected by a solid line, it might mean that they would represent the same command object.



The object line might point out which command object the other command symbol would represent. A square connected with an object line might depict a *command reference*: a reference to a command definition. A diamond might also get an object line, that may point to another location, where the actual executable object may reside.

#### Command Definition Connection

Command symbols might be connected by a dashed line. This may be used to indicate that both commands would have the same class (or *definition*) which may mean that they would get the same behavior, but might not be the same individual object.



This definition line might point out, what definition a command object would have. The diamond might be a replica of the definition, but it could be that it would be its own individual object: an individual execution of the command. It is not limited to diamonds and squares this way. Dashed lines might be used to connect any two command symbols to each other, to indicate mutuality of definition.

#### Command Interface Connection

Two commands could have the same interface, which might mean that a command has parameters similar to the other command. Command symbols that would have the same interface might be connected using dotted lines:



A dotted line could be called an interface line.

### Straight Mark



(It is supposed to be about the smaller line, that appears to dissect the longer line.)

A straight mark might be used to indicate that a symbol would be **Public**, or accessible.

The straight mark might be connected to a symbol as follows:



It might be used to indicate direction as well.

### Cross Mark

A cross mark:



might be used to indicate, that a symbol would be not accessible from the outside, or **Private**.

Here is how it might look when connected to a symbol:



A cross mark might be left out, if the choice is made that the absence of a straight mark might imply it would be **Private** already, but there could be different ideas about that.

### Triangular Mark

A triangular mark:



may be used to indicate that a symbol that might own the line would be **Protected**. Protected is something that can play a role in the concept of inheritance. In the Circle language it might mean that a symbol might only be accessible if it is placed inside a triangle.

Here is how that might look if it is connected to a symbol:



A triangular mark would not represent an arrow. It might be confusing that a triangular mark looks like an arrow, but isn't, and a different notation for protected might be appropriate and introduced some time.

### Line Direction, Access Marks, Line Ownership

A line might have direction. The direction of a line could be expressed with an access mark:



🡪

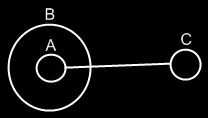
The direction would go from symbol **A** to symbol **B**. The access mark might be placed before the object, that would be accessed.

Admittedly, there is the idea to just draw it as an arrow shape, but currently these access marks are suggested.

But access marks might not always be displayed to indicate line direction. Some suggestions are made here for when direction might be implied without using an access mark.

#### Lines Pointing Outwards

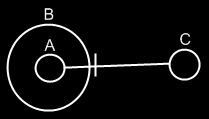
The first suggestion for line direction is that a line might usually point outwards.



🡪

Symbol **A** has a line that might be pointing to Symbol **C**, because lines might tend to point outwards.

If the direction would not be outwards, it might be denoted with an access mark.



🡨

Now symbol **C** would then be like a pointer to symbol **A**.

#### Diamonds More Likely to Redirect

The second suggestion for line direction might be that a diamond might be more likely to redirect to the command definition, than it would be for a command reference to redirect to a diamond.



🡪

The diamond would point to the square.

If the direction would be the other way around, it might be denoted with the use of an access symbol:



🡨

#### Access Mark Placement

When a line crosses symbols’ borders, it might first exit borders and then enter borders.



*First come the exits a and b, and then come the entrances c, d and e.*

An idea is to put the access mark in the section between the exits and entrances. The access mark would then be placed in front of the border that may first be entered.

If the part of the line between exits and entrances would go out of view then perhaps the access mark might be placed where still visible. It could be placed in front of the last border in view that might be entered or exited:



It may be a problem to draw it like this:



This might suggest that the rightmost border would be entered, instead of exited.

#### Line Ownership

Perhaps depending on how these diagrams might be applied, a symbol could only have one object line, one class line and one interface line. A symbol's lines may point *away* from the symbol. They could denote which other symbol may be its object, which other symbol may be its class and which other symbol may be its interface. Other lines connected to symbol could be considered to point *at* the symbol, instead of *away* from the symbol.



It might work similarly for commands. A command might have just one reference line, one definition line and one interface line.

### Object Symbols Drawn with Different Lines

There is an idea for drawing the shapes with different line types, for example: dashed or dotted.

One idea is that an object symbol only used as an object (*not* as a class or an interface) might be drawn with a *solid* line.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

If an object symbol might only used as a class, it might be drawn out with a *dashed* line.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

An idea is also that if an object symbol may only be used as an interface, it could be traced out with a *dotted* line.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

There may be different ideas, for when it could be appropriate to draw shapes with different types of lines.

Perhaps if object symbols would be used in several ways, for instance as an object, as well as another object’s class, the symbol might be given multiple borders. Here might be an object symbol that could serve as both an object and a class:



That way an object symbol could also have three borders at the same time or maybe more.

It is not a hard rule to use different line types for symbols. But it might work intuitively.

### Command Symbols Drawn with Different Lines

The idea for symbols drawn with different line types might also be applied to *command* symbols.

A diamond, which might represent a command call, may regularly be an instance of a square, that could represent a command definition. When the diamond is an instantiation, it might be seen as somewhat of an *object*. But a diamond might sometimes be used as a definition, more like a *class*. When a diamond is used as a definition, it might be seen as both an object and a class, so it might be given both a solid border as well as a dashed border, indicating that it might be both an instance, and a definition.



If a diamond would not used as a definition, it might only get a solid border.



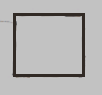
It might be that a diamond would regularly be considered an instance, so it might often have just a solid border. It may become uncommon for it to be drawn with just a dashed line:



A square may be used often as a definition. The idea was proposed that if something may be used as a definition, it might be drawn with a dashed line:



A choice could be to not do this, because otherwise you might mostly see dashed squares.



It depends the choices of how to use this notation of borders with different line styles.

There may be a particular case where you might draw a square with a dashed line, but the suggestion for this may have to wait.

Both command calls and command definitions may function as the interface for another command symbol. In that case the idea is that the command symbol may get an extra dotted border.



It could be an idea that if a square functions solely as an interface for a command, the square might be drawn with just a dotted border:



It is suggested here, that a diamond might so often be thought of as an instance, that you might choose to always show the solid line and not draw it with just a dotted border, so not like this:



But it may depend on specific notation choices.

### More Suggestions for Dashed Lines

#### Commands with Dashed Lines

Previous text tried to show that command symbols might scarcely be drawn with dashed lines.

However, in a certain case it might be drawn with a dashed line. When drawing out the diagrams, the class structure inside a module could be drawn out with dashed lines. To also draw out the command structure with dashed lines, the *target definitions* of commands might be drawn out with a dashed line. Later text may show what a target definition might be.



Lines that connect commands to classes might be drawn with a dashed line.

#### Object Structure Solid, Class Structure Dashed

Drawing dashed lines that way might make the structure elements and their relations be drawn out with dashed lines. It might be more intuitive that way: the class structure would be dashed, while an object structure would be drawn with solid lines.

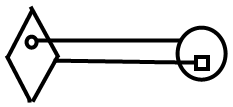
#### Other Ideas

However, there might be different ideas of how to apply dashed lines, for instance one where dashed lines are reserved for the expression of the concept of *static*. Things are open like that.

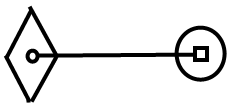
### Line Merge

A suggestion is made here that sometimes two lines might be merged together to form one. This might be called a line merge.

The lines in the following diagram might be very closely related:



They may merge together to one line:



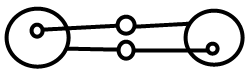
Lines might merge together under different circumstances, that may come to light in other articles.

There may be some reservations about using this notation, because it might introduce some ambiguity or the suggestion of ambiguity, but the idea may still appear here and there.

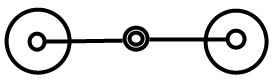
### Symbol Merge

A symbol merge would be closely related to a line merge.

In the following diagram:



Some lines and circles might merge together:



The idea of circles merging together might be called a symbol merge. How this might be applied may come to light in other articles. It could be applied to the concept of *automatically synchronized bidirectional relationships*, if you happen to have an idea of what that is.

## Loose Ideas

Classes,

2008-11-13

If you can see object usage, you can not see class-sub-object usage.

You'd have to look at the usage of the sub-objects of the objects of that class,

to see the class's sub-object usage. Indirectly you will be able to see the dependency on a class's sub-object.

Doesn't a sub-object have a reference to the class's sub-object or does the parent

object only have a reference to the class?

> 2020-06-13: I now think, that a sub-object of an object does not have a reference to the sub-object of a class. I guess the relationship is implied by form/shape/name/aspect correspondence. I get how that might be bothersome. Then there might be relationships in the diagram that are expressed with neither lines nor containment. Maybe it is something to not be bothered by. Maybe it is something to mention as another type of relationship between symbols. There might be a spot for it in Basic Diagram Elements for that, next to the other types of relationships between symbols that might be named there. Maybe 'it is just a notation' might be the way to think about it. There might not be a precise mathematical definition of why shapes are used and the choice of containment vs association is also left as an arbitrary design choice so why would this not be ok as something arbitrary about the notation? If it turns out to be bothersome, maybe a different kind of line to express a different kind of correspondence could be used. But that may be something not to make part of the initial release of Circle language spec.

JJ