2019

The other art of computer programming. Milestone 4: Smalltalk. 1980s

Melanaie Tarr
Edith Cowan University

10.25958/5e3a76555f8e8

Milestone

Smalltalk

The Other Art of Computer Programming
by Melanie Tarr

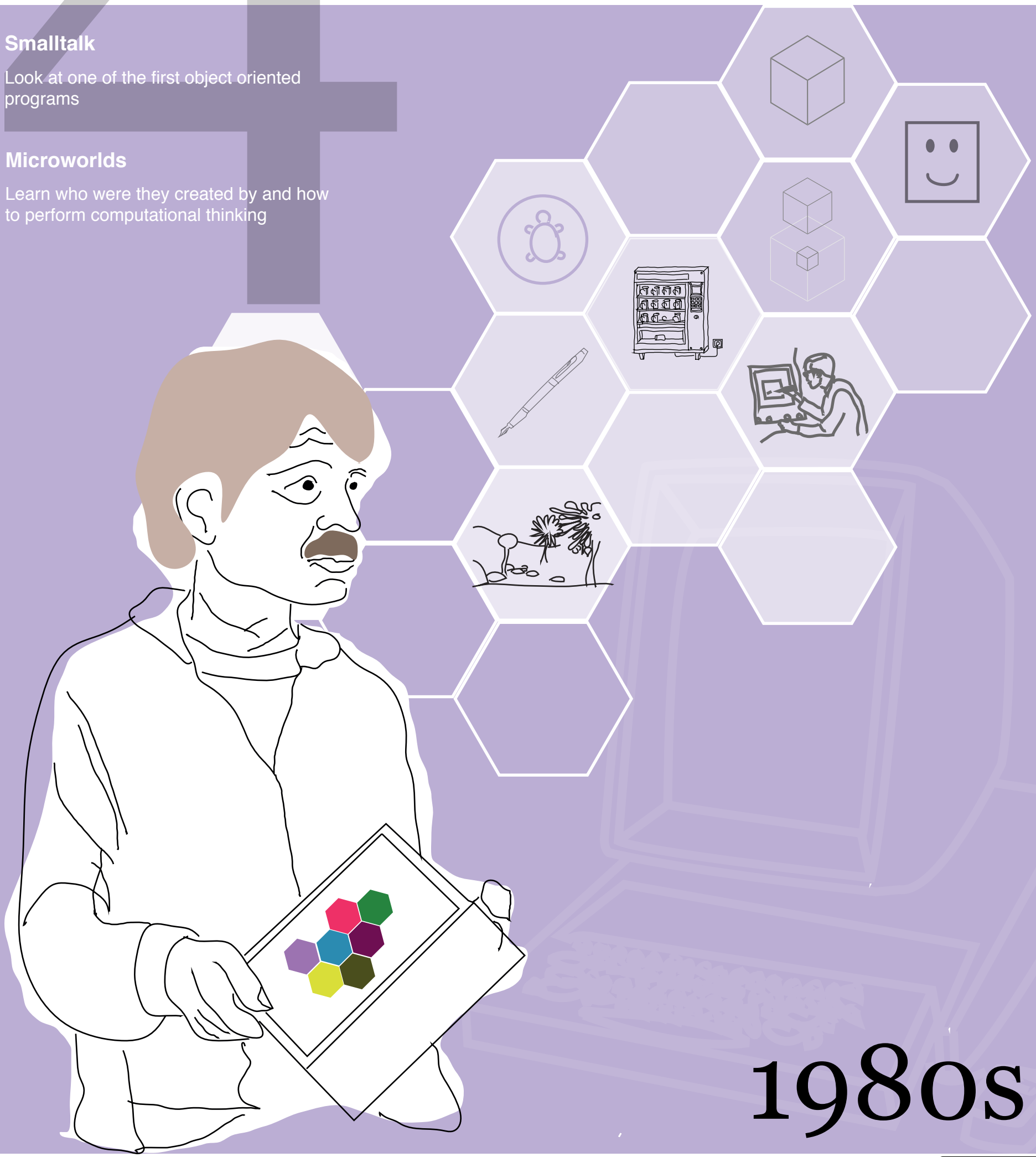
4

Smalltalk

Look at one of the first object oriented programs

Microworlds

Learn who were they created by and how to perform computational thinking



1980s

DOI
10.25958/5e3a76555f8e8

1980

Smalltalk

WHEN WE STARTED TEACHING SMALLTALK, THE FIRST OBJECT ORIENTATED LANGUAGE, WE TAUGHT COMPUTER SIMULATION METHODS, GRAPHIC TECHNIQUES AND GEOMETRY.

Table of Contents

- ACTDIK015, ACTDIP027, ACTDIK024 - requires an appreciation of binary numbers
- ACTDIK001 - identify hardware components in a system

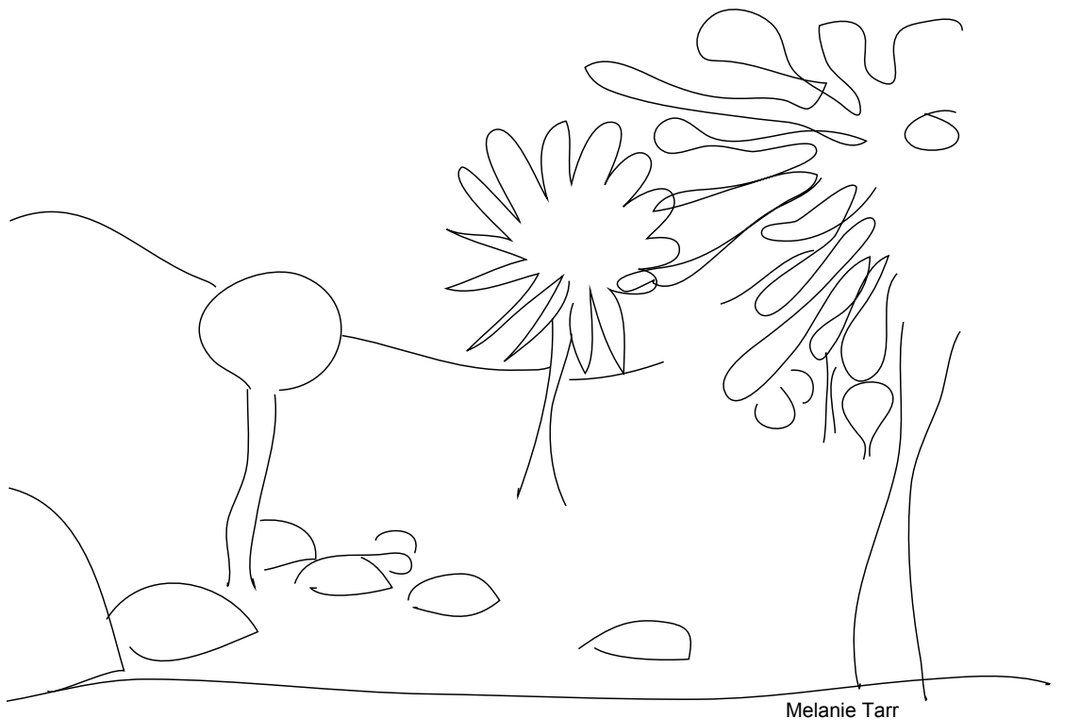
Note: This lesson may offer useful explanations for the above elaborations in the Australian National Curriculum.

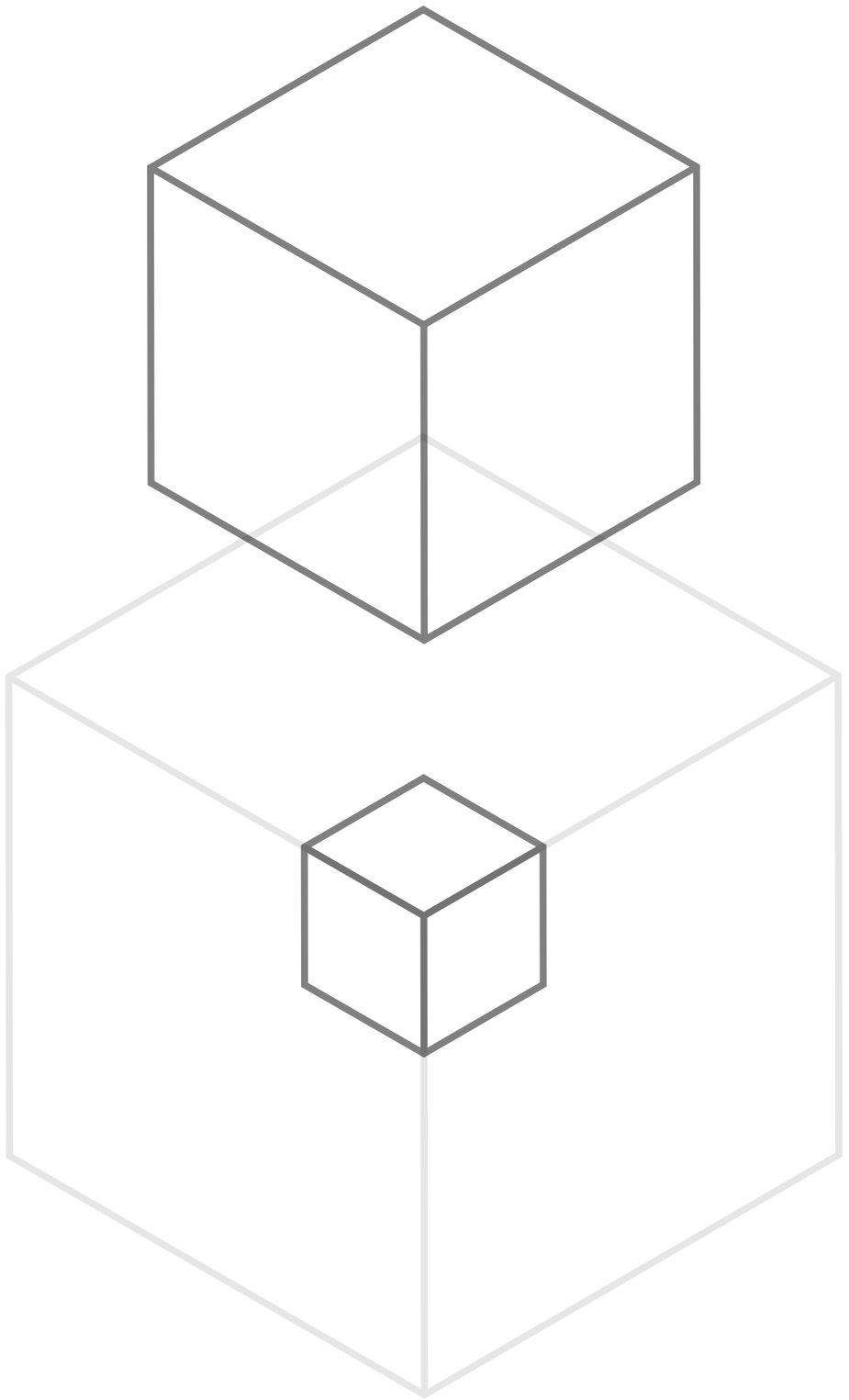
Microworlds in Smalltalk

Microworlds were created by the MIT Logo Group to provide a programming space for exploring software. Within a microworld, a structure exists to allow a learner to exercise computational ideas and intellect. Papert used microworlds as tools for computational thinking.

> a subset of reality or constructed reality whose structure matches that of a given cognitive mechanism so as to provide an environment where the latter can operate effectively. -Papert

A Microworld can also be thought of as a subset of reality that is computable.





Everything is an object

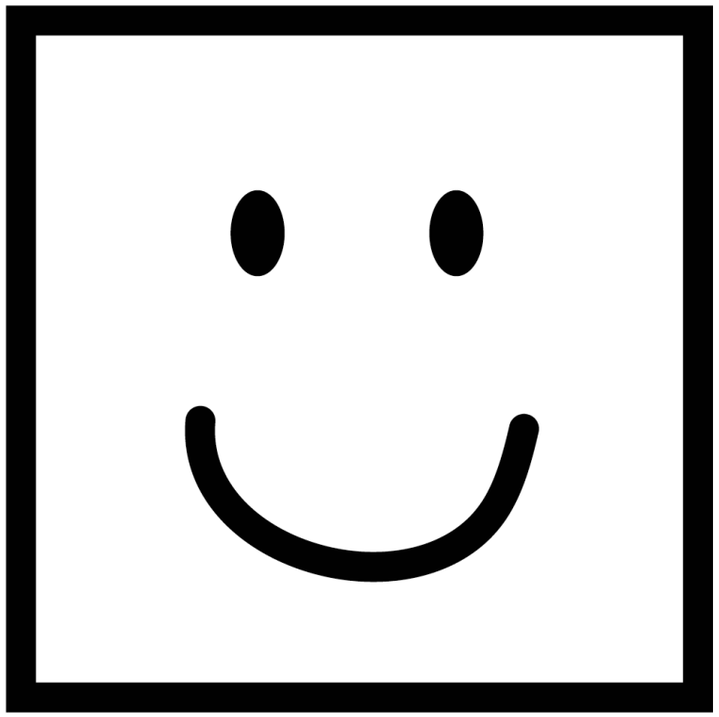
Every object is an instance of a class

Every class has a superclass

Everything happens by message sending

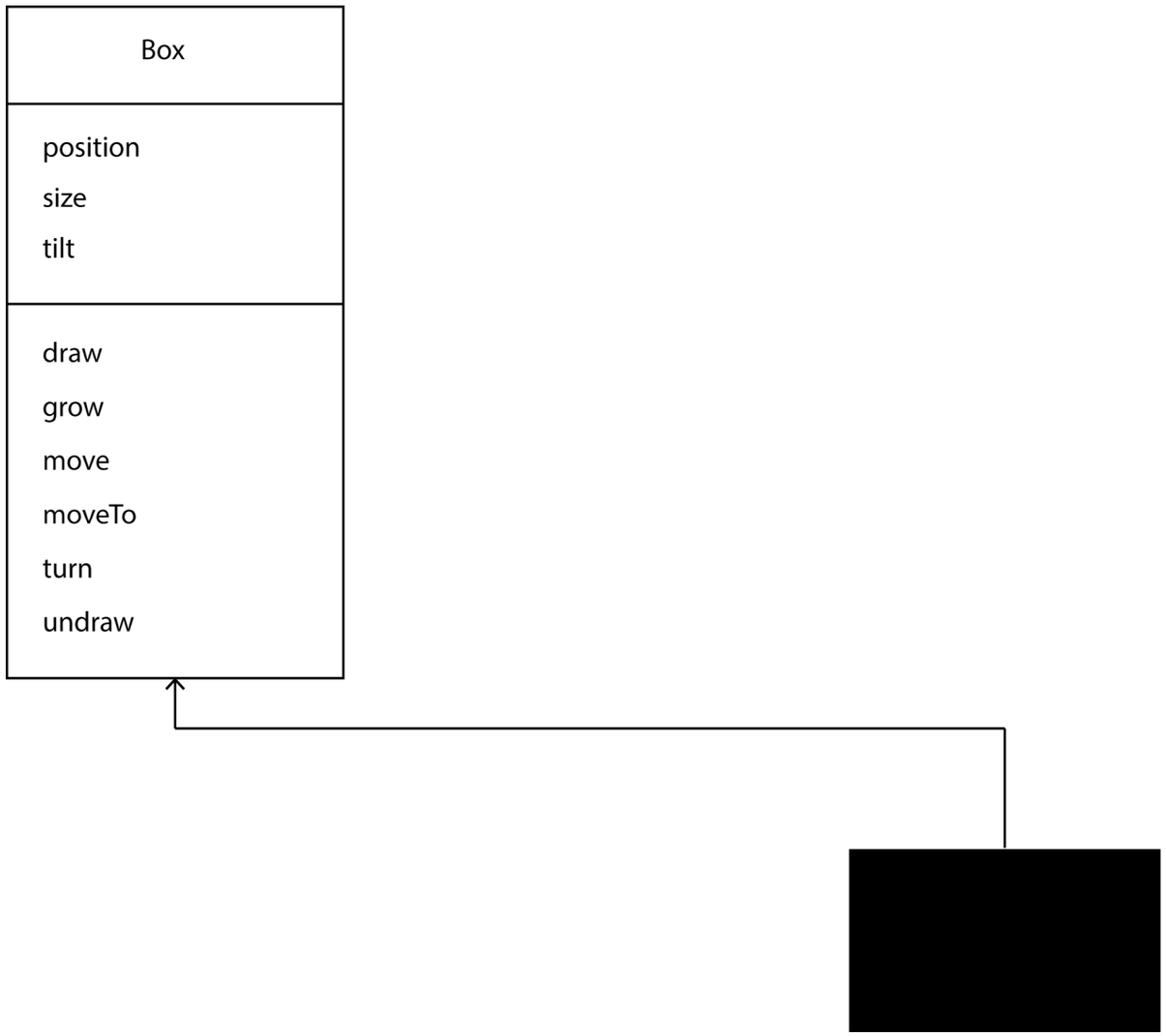
Rules of Smalltalk

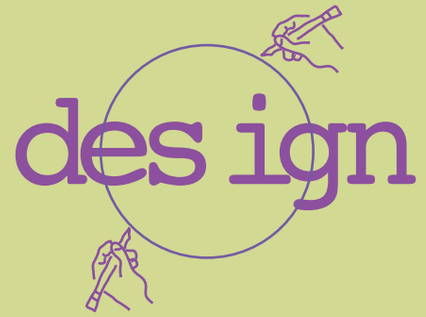
Joe the box



Object orientated programming was taught successfully to children by Adele Goldberg and Alan Kay in the late 1970s in the Smalltalk programming language. Some companies still use Smalltalk today.

Designing software in Smalltalk





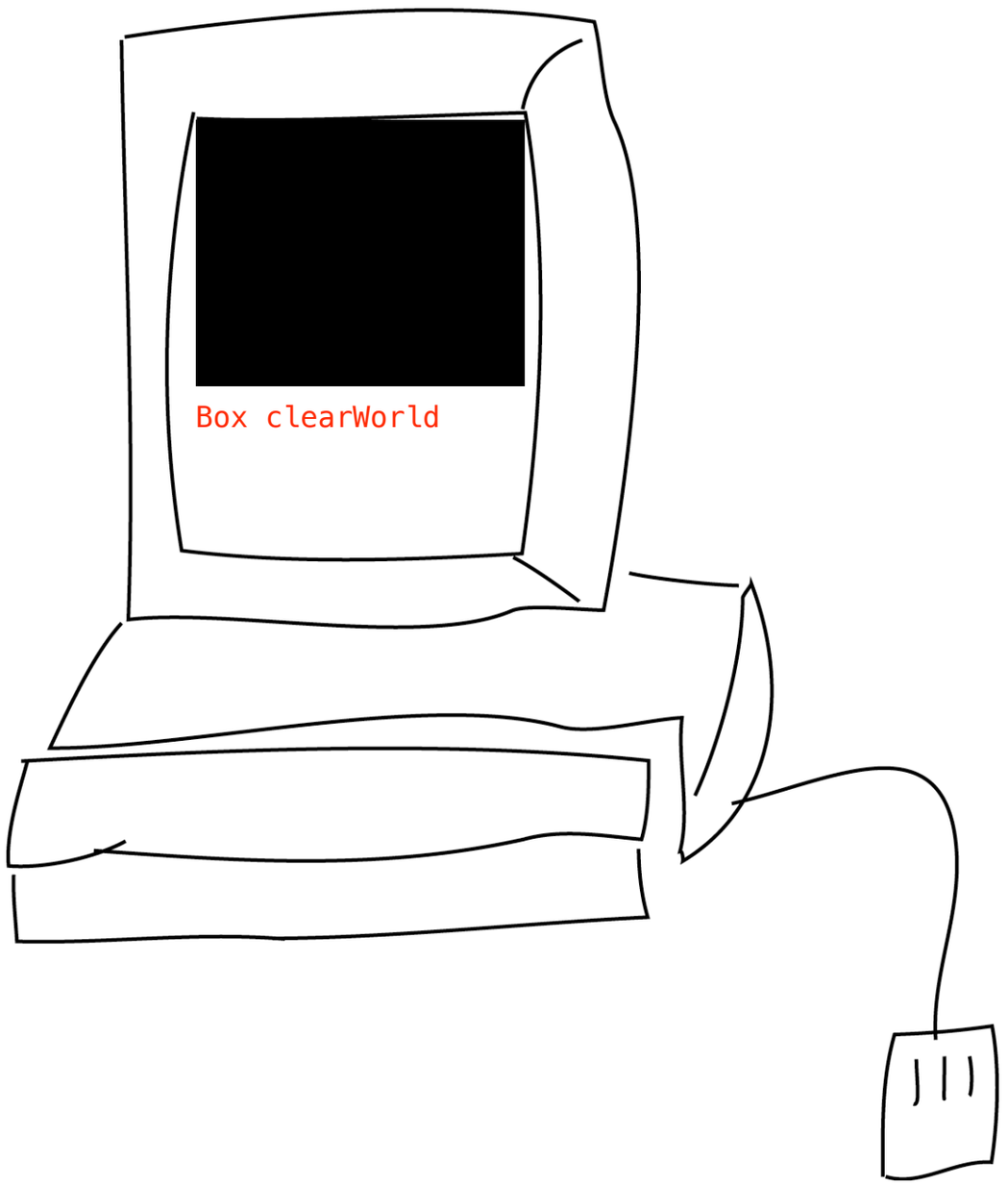
Computer scientists use Universal Modelling Language (UML) diagrams to design their software. UML is formally defined as a way of visualising software programs using a collection of diagrams. UML diagrams help to visualise the inheritance chain that occurs in software development.

Creating a virtual world.

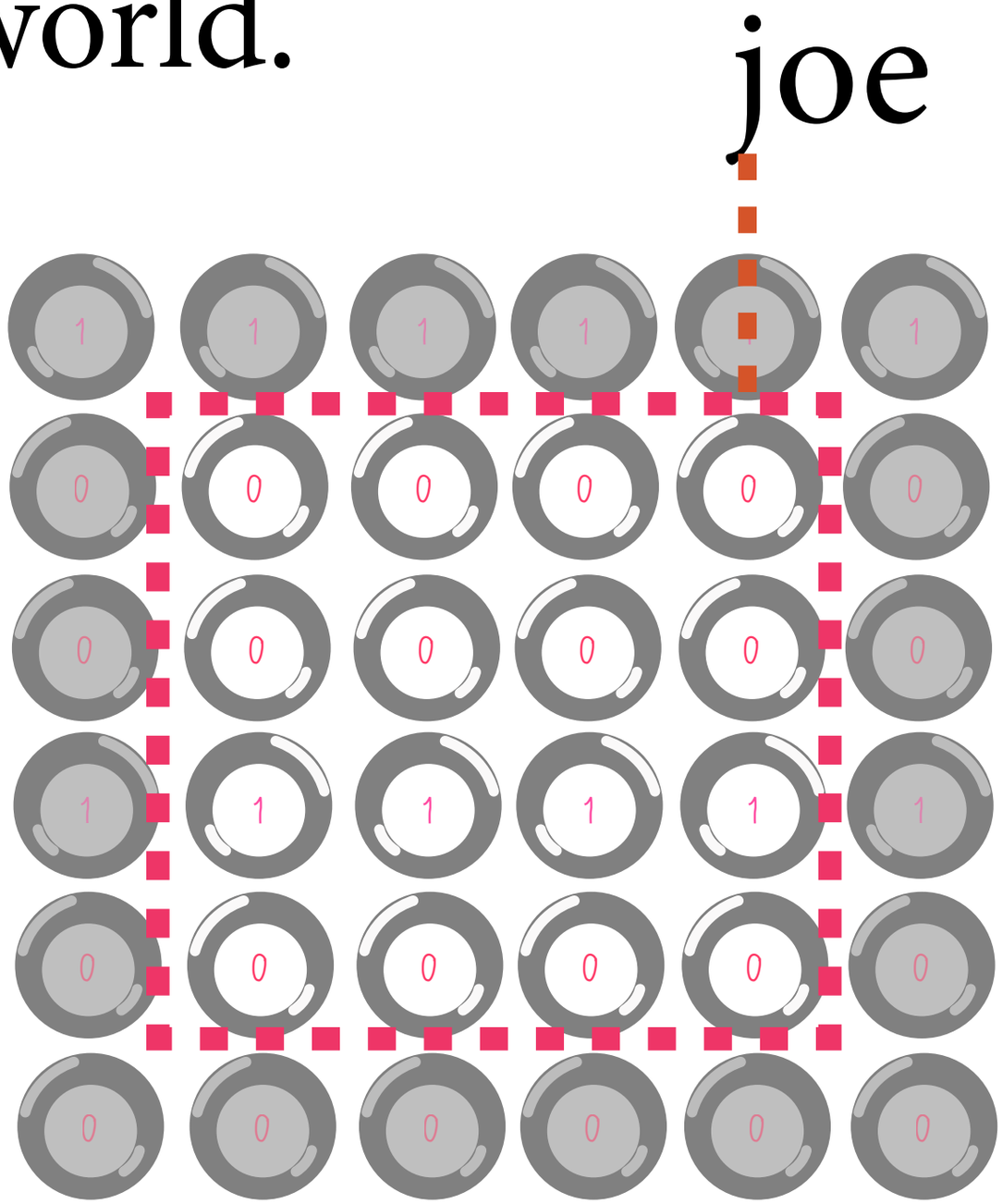
By typing the command:

```
Box clearWorld
```

on the screen you make a space on the screen to draw objects. The objects we will draw are boxes.



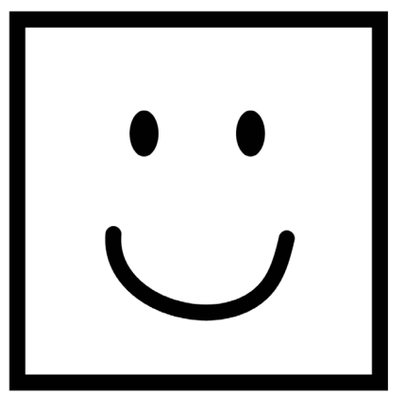
Creating a virtual world.



The command:

```
joe ← Box new
```

creates the new object called “Joe” and allocates the new class of object in computer memory.
Joe is now a virtual object that references a physical area in computer memory



Joe is now an instance of the class Box.

Creating a virtual world.

But Joe is an object that we can not see. Like the Turing test, we need to ask Joe questions to find out information. In the Small-talk, we ask questions in the software by sending *messages*.

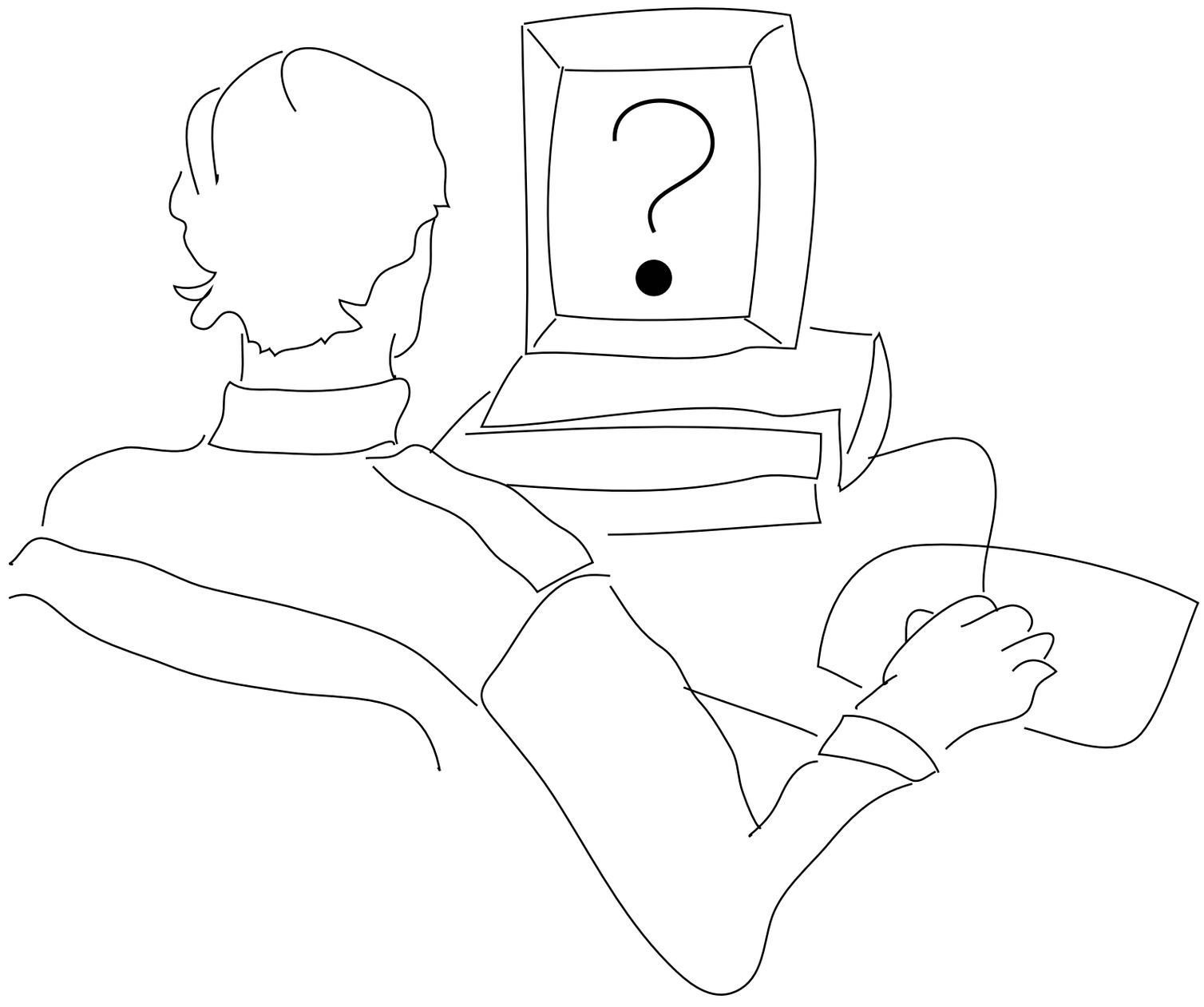
By asking the question:

```
Joe class print
```

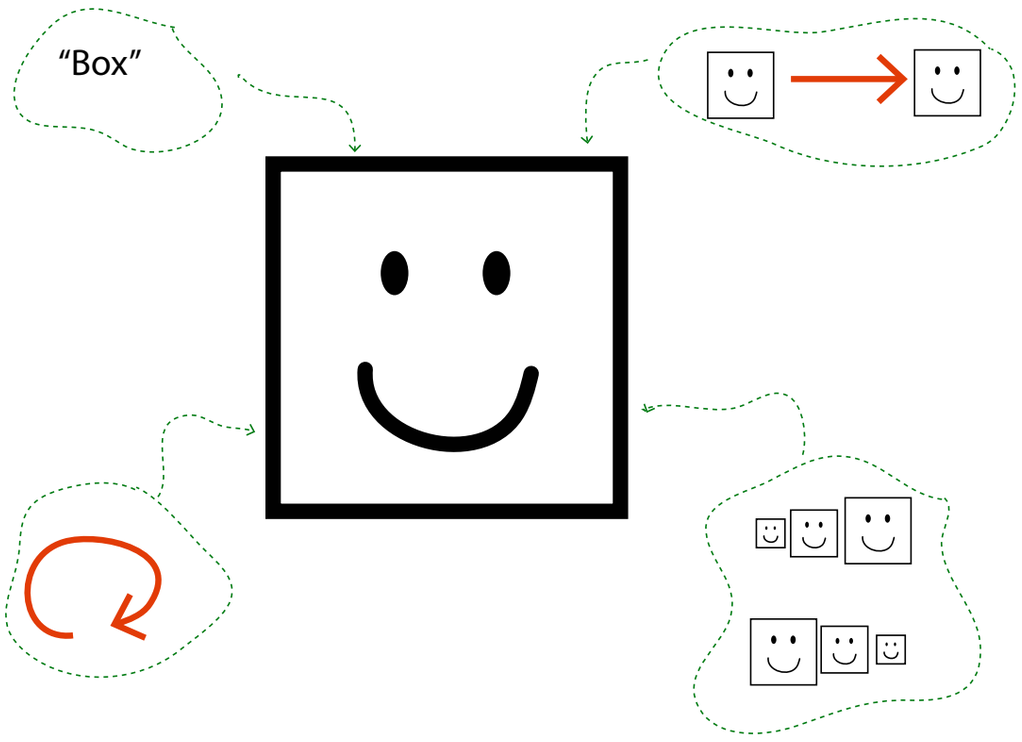
The machine will answer:

```
Box
```

This is called sending a *message*.

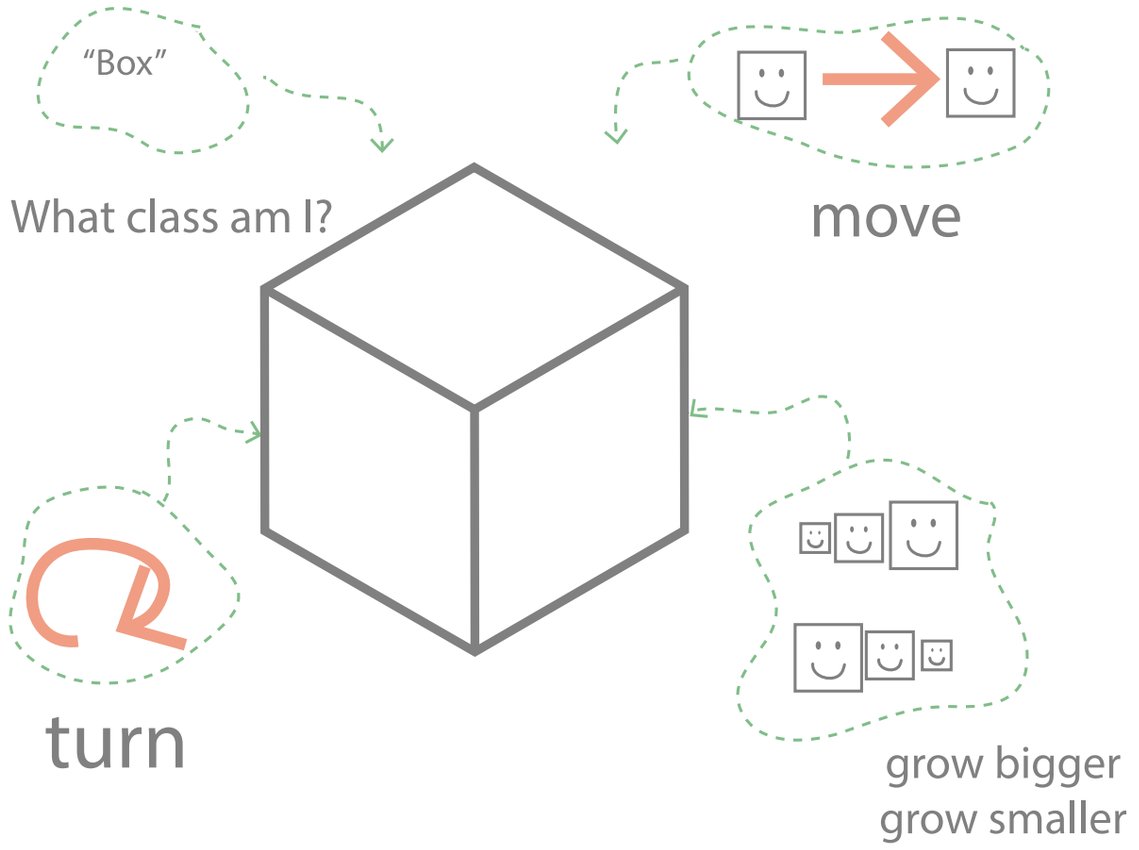


What can Joe do?



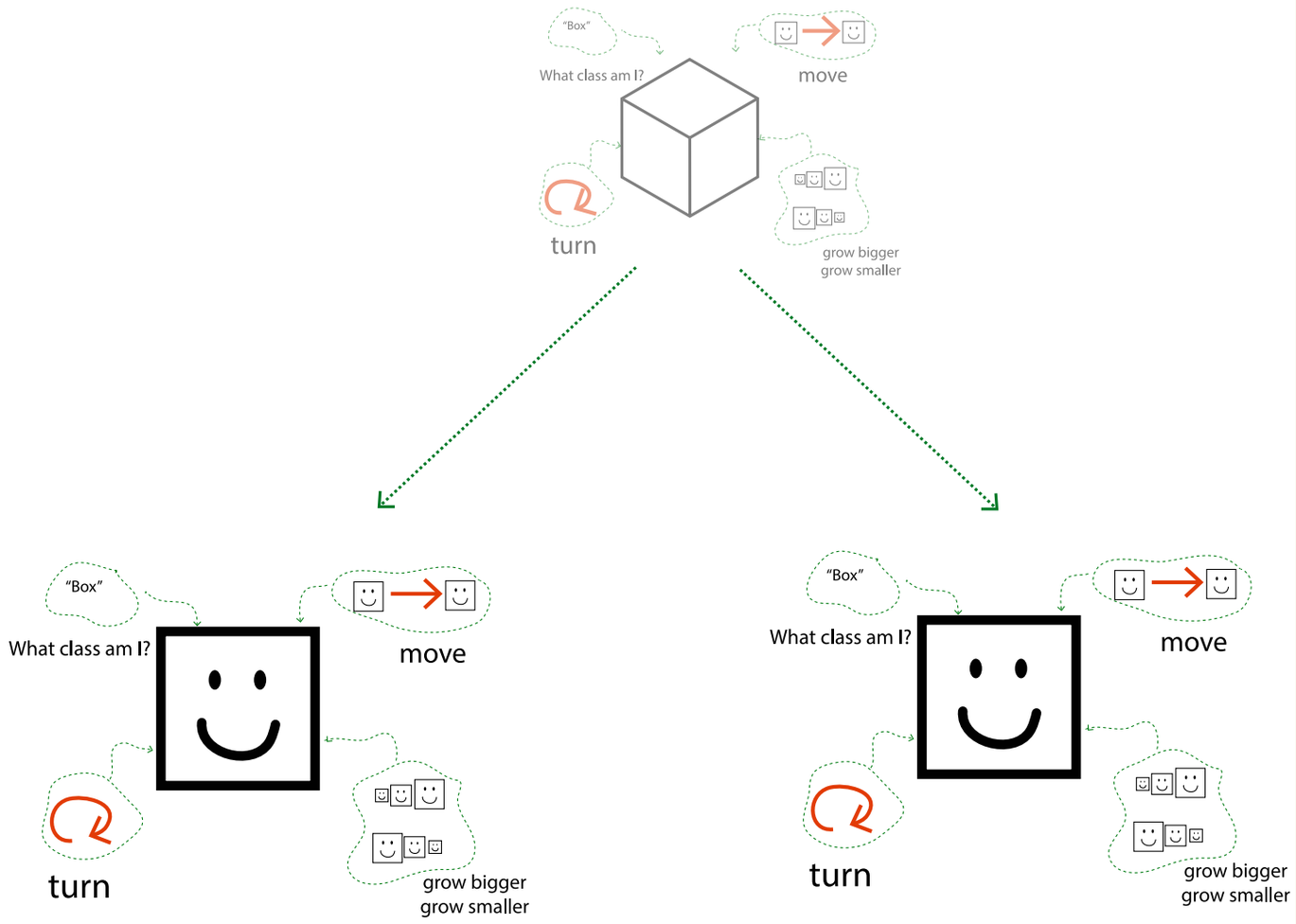
The virtual object we have created, "Joe" can do a number of activities.
The instance Joe knows what class it is, it can turn, it can move, grow bigger and smaller and it can also move to where the mouse points to on the screen.

Box messages



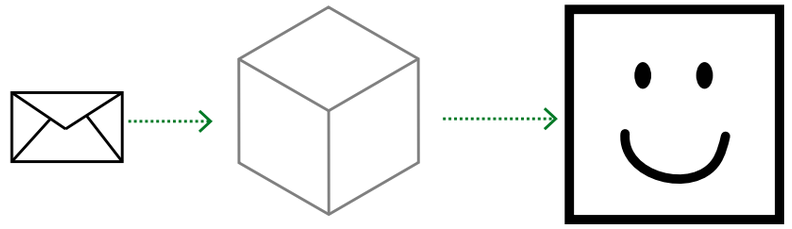
Joe inherited his characteristics from the "Box" class.

Inheritance



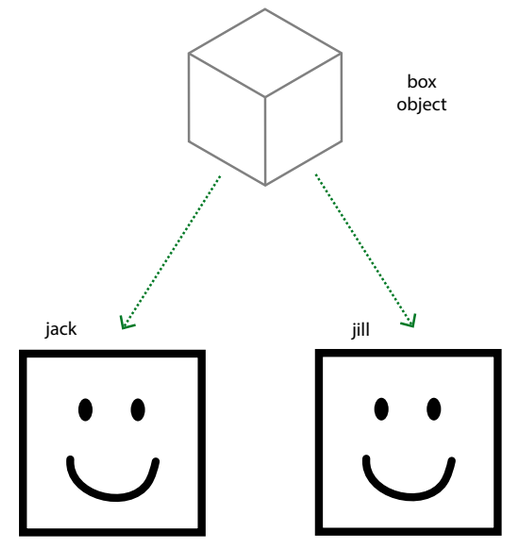
When new objects are created, they inherit the *messages* of the parent object. Here the parent object is "Box" and a superclass. The child objects are Joe and Jill who have inherited all of their parent's characteristics or messages.

Create Jill



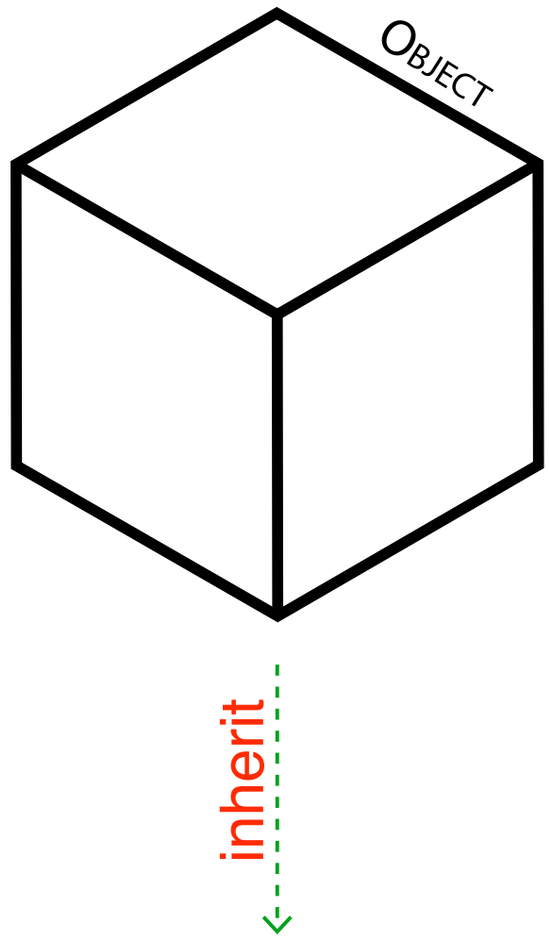
Send a new *message* to Box to create a second Box instance called `Jill`.

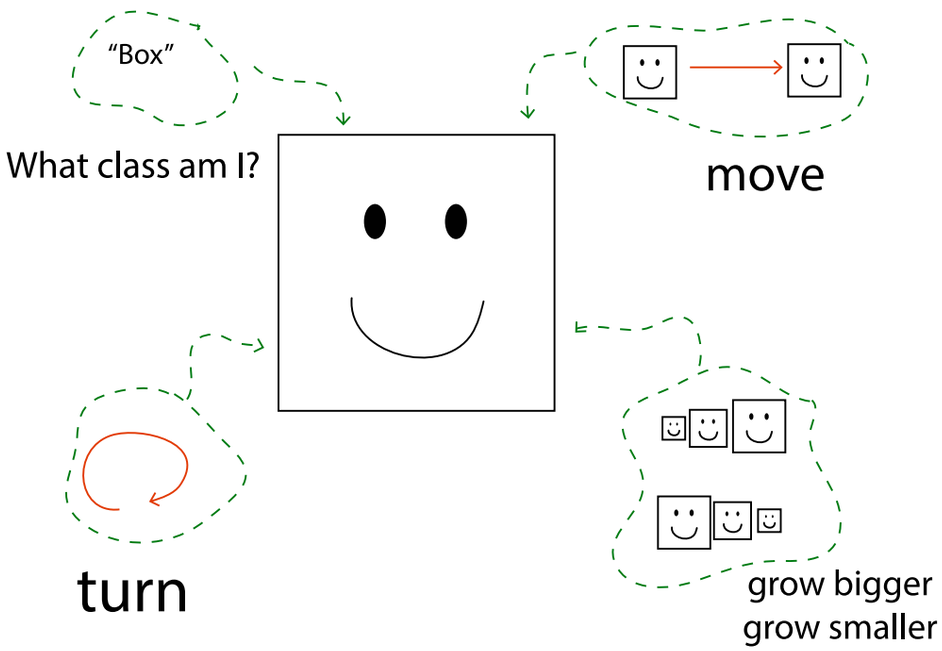
```
jill ← Box new
```



Jill and Joe are *instance* of the same *class*. They belong to the same *class*, that is, Box.

Messages





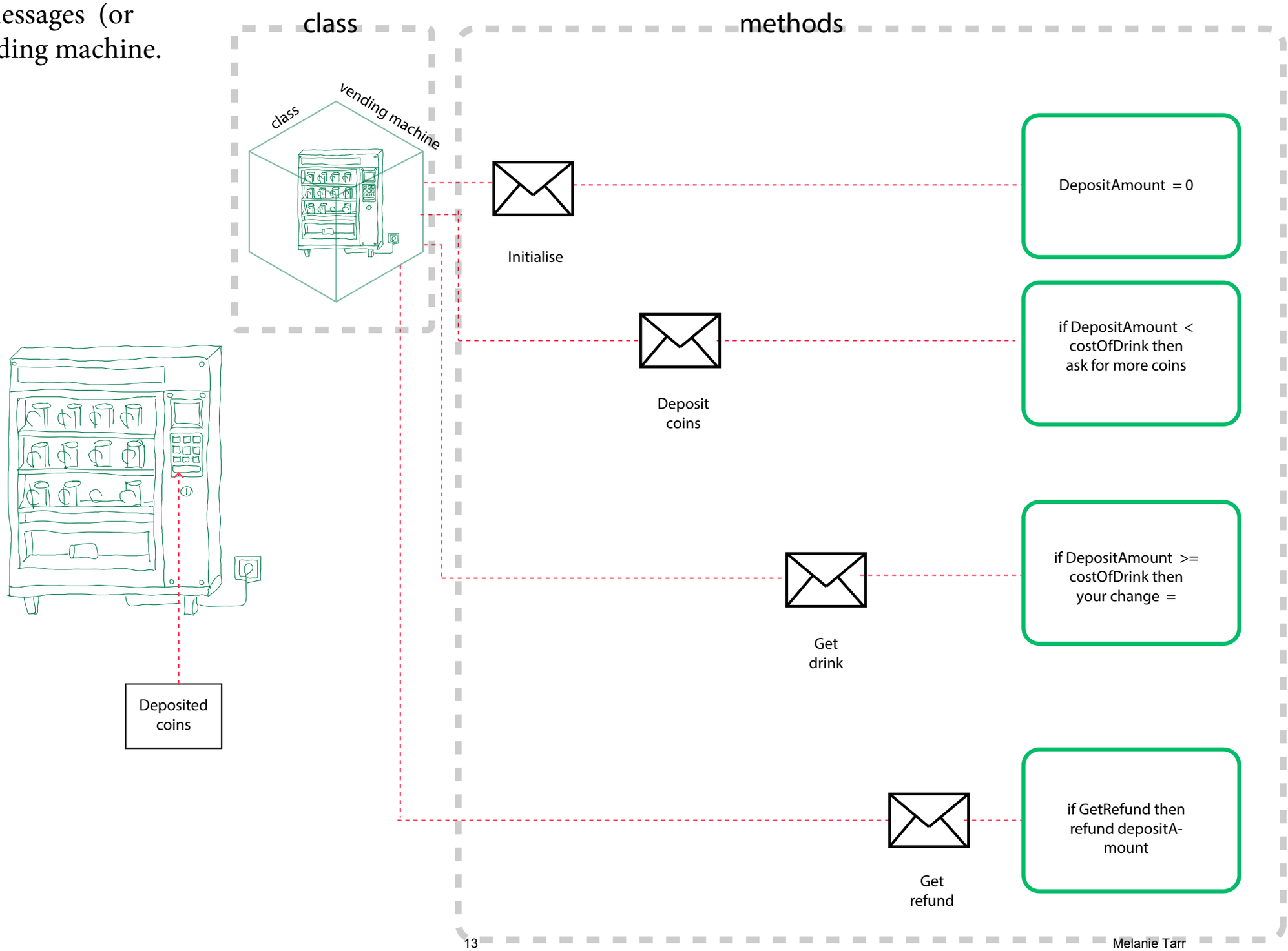
The *services* were inherited

Because Joe and Jill belong to the same family or class, the have the same set of characteristics. The Object Oriented Programming word for characteristics is *services*. Jack and Jill look the same and inherit the same *services* as they are part of the same family. The *services* were inherited from the Box class.

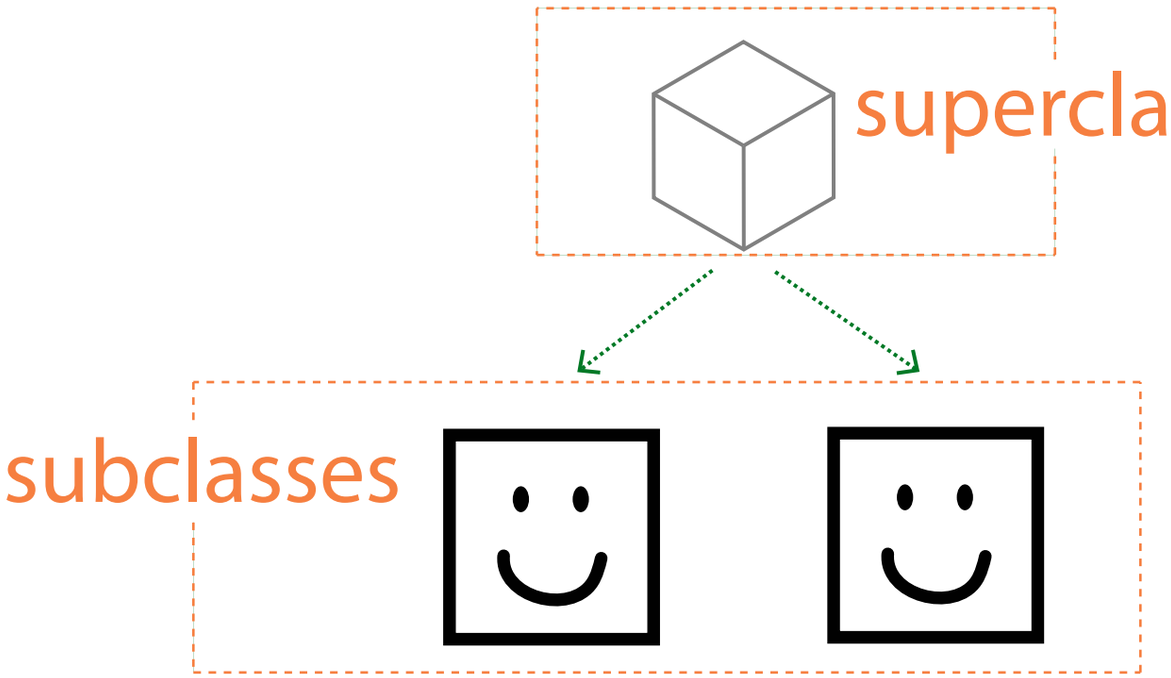
Vending machine example

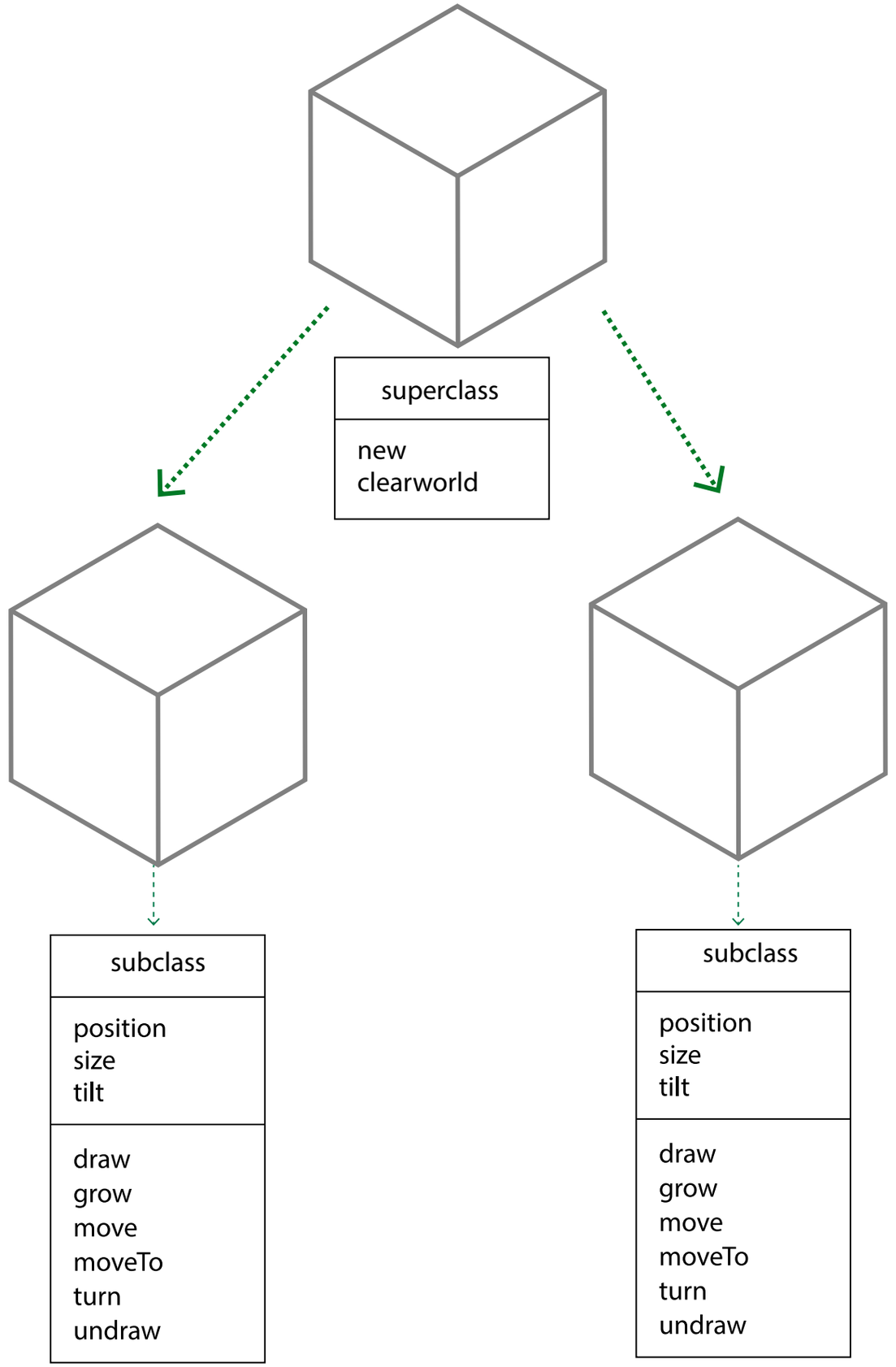
A vending machine can be used as an everyday example of inheritance. Here the vending machine is programmed as a class and the methods are what happens when the machine is operated. There is one class, called vending maching, and four messages (or services) that operate the vending machine. These messages are:

- Initialise
- Deposit coins
- Get drink
- Get refund



Inheriting services

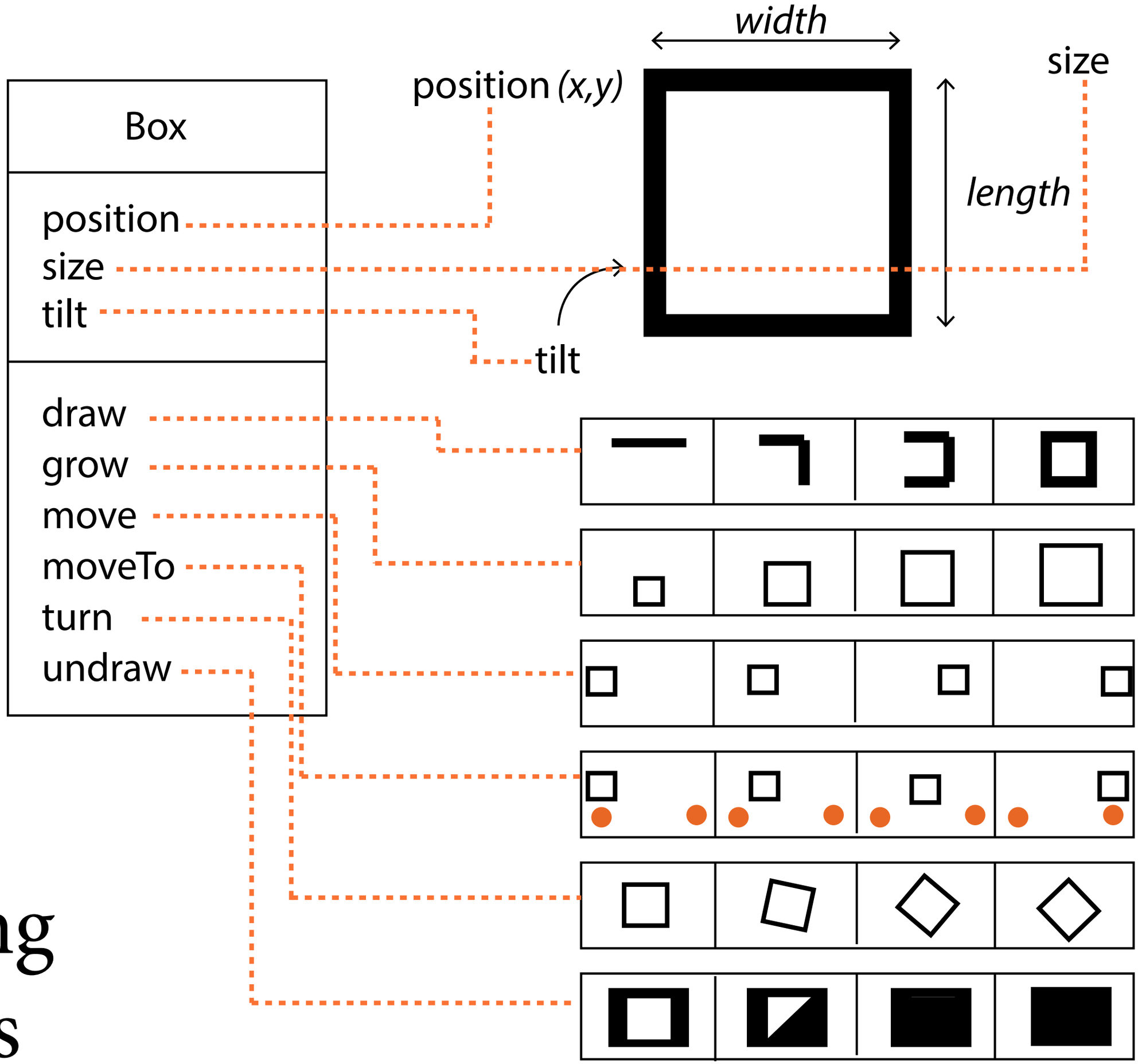


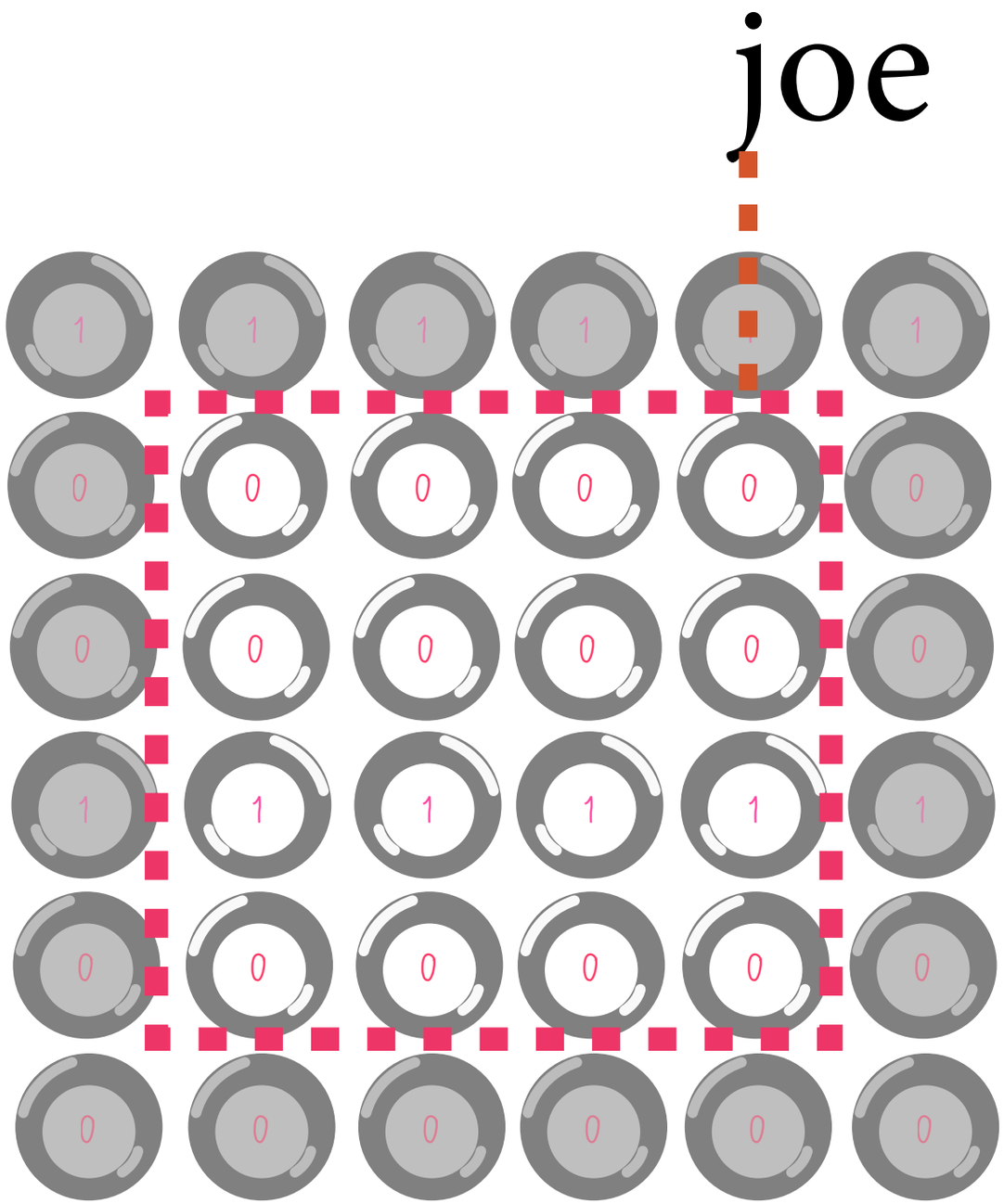


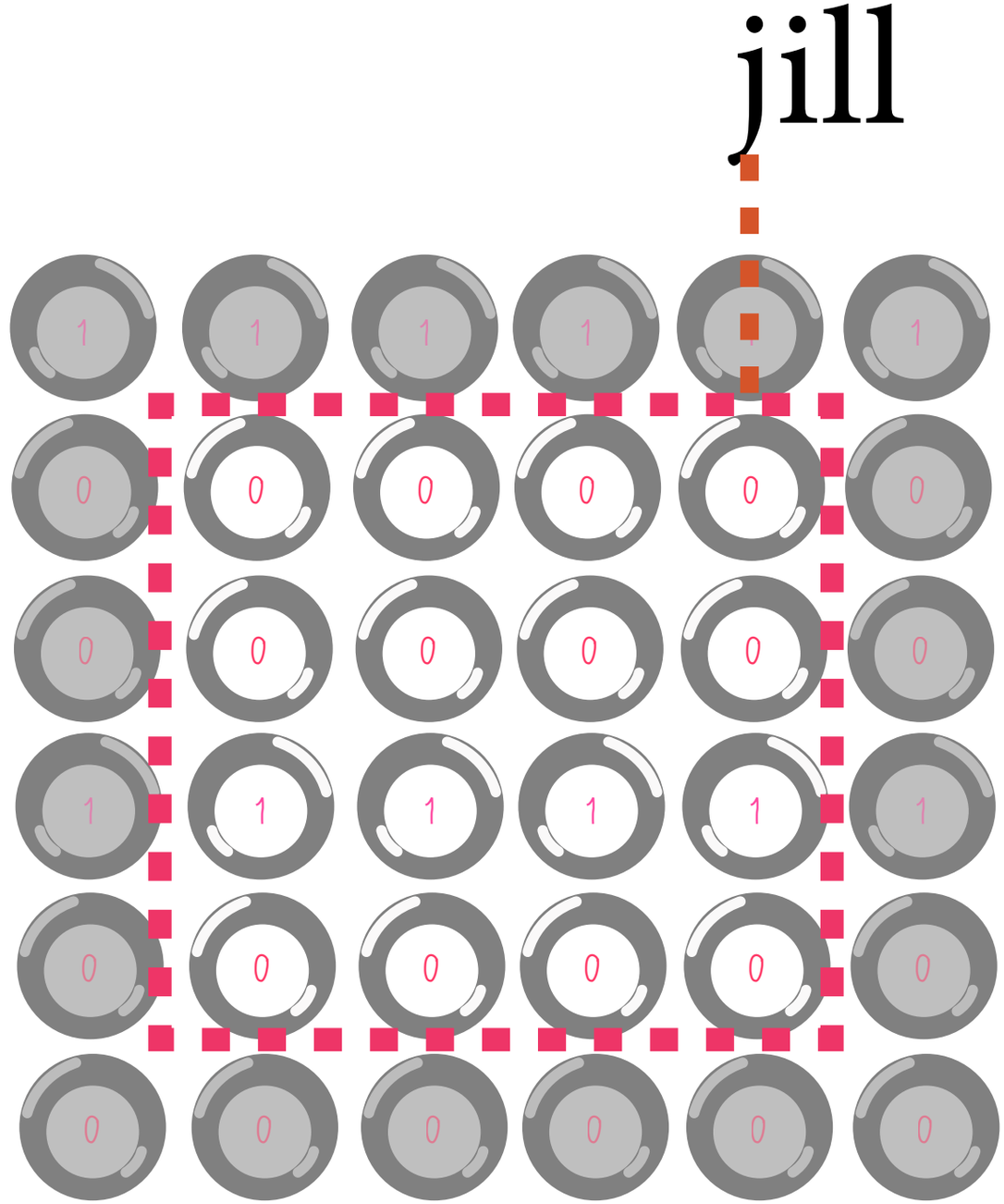
As Joe and Jill have the same services, it is best to abstract them as common placing them in the superclass. Now there is less code to write for the programmer.

The services for the subclasses are inherited from the superclass.

Inheriting messages

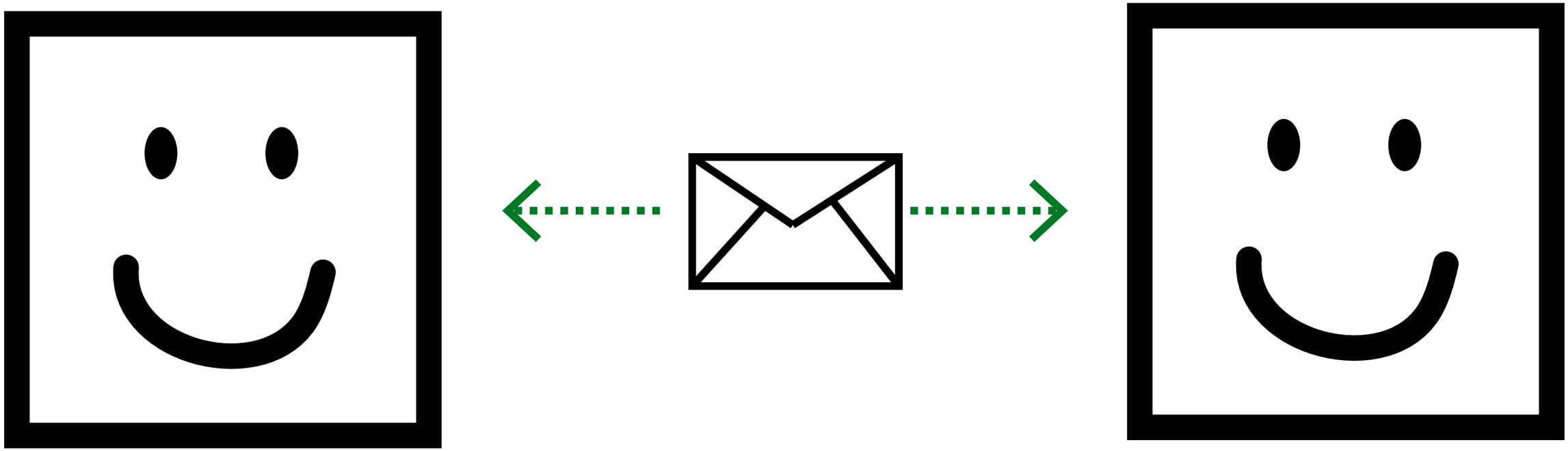


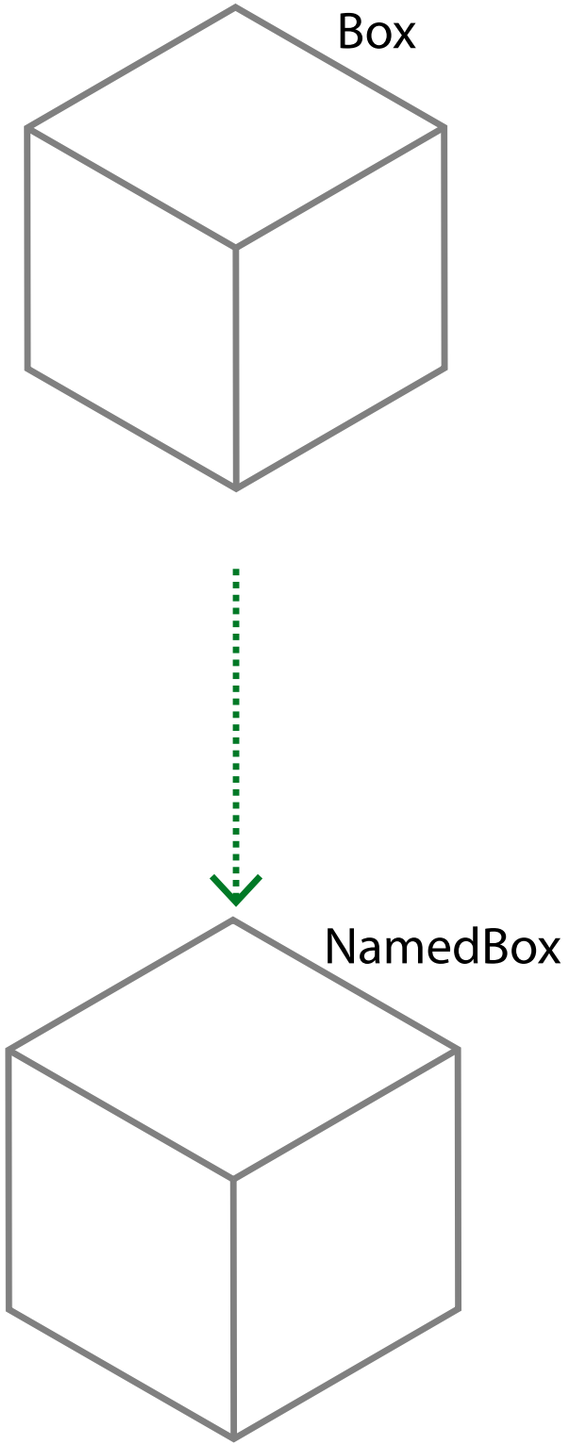




Different objects respond to messages differently because they occupy different memory space.

These objects can understand messages to and from each other.





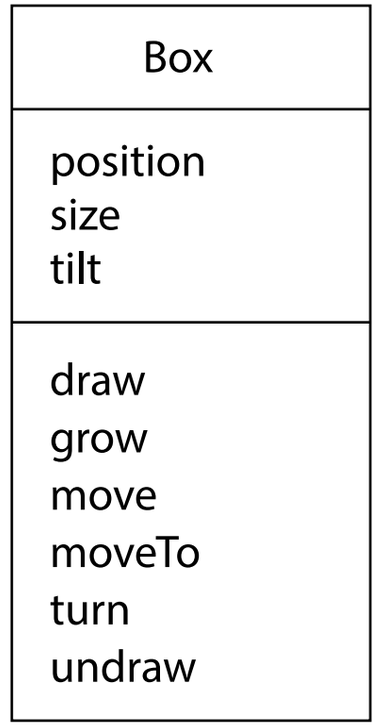
Create a named box object

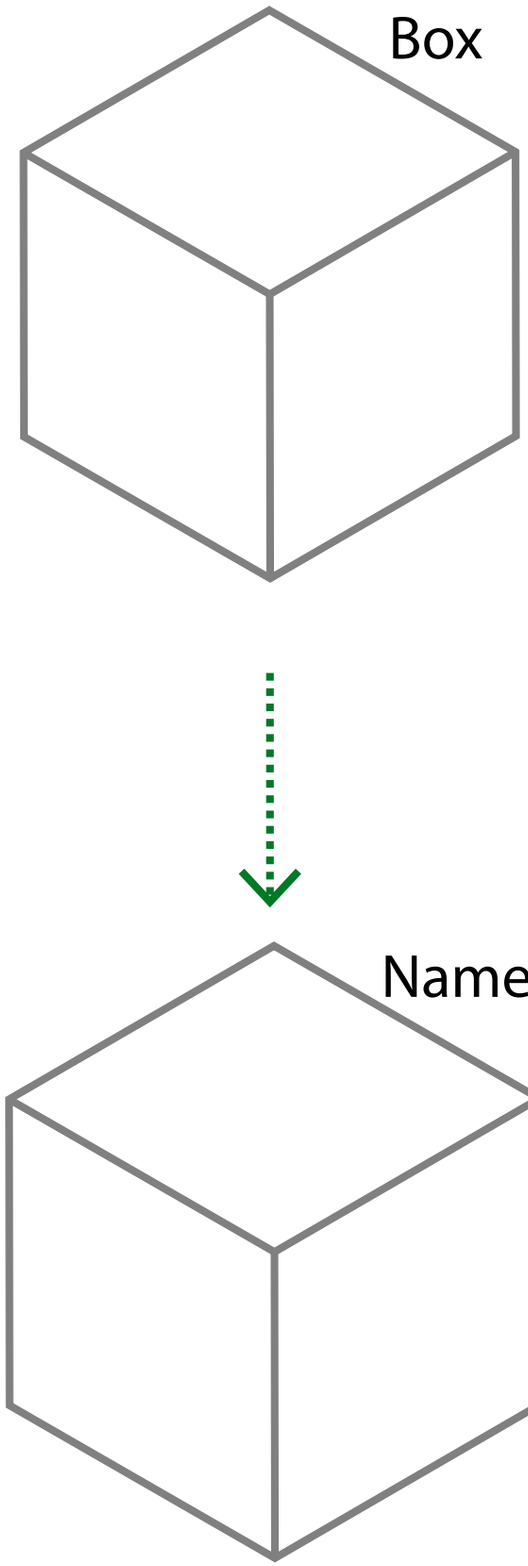
If we wanted a new kind of Box to be displayed on the screen that has the object name printed above it, we design a new object with an added service that is a kind of Box.

Extending the box definition to create NamedBox

The attribute name has been added to the NamedBox object. When the NamedBox object is created, a name is stored in the area along with position, size and tilt in memory.

```
Box subclass: #NamedBox
   instanceVariableNames: 'name '
   classVariableNames: ''
   poolDictionaries: ''
   category: 'Boxes'!
```

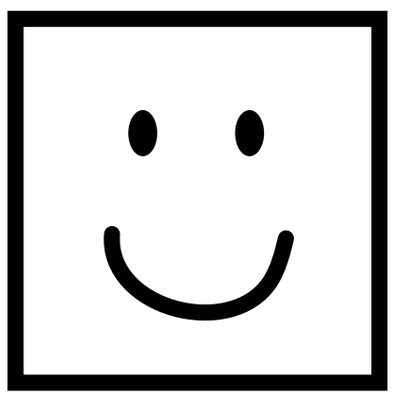




Creating a namedBox object

When the new NamedBox object is created, a call to the super class occurs and a box is created. Then the box is removed from the screen with the `undraw` message. A name is assigned to the NamedBox and the new Box is `drawn`. The memory location of the new Box is passed back to the main program.

Box subclass: #NamedBox

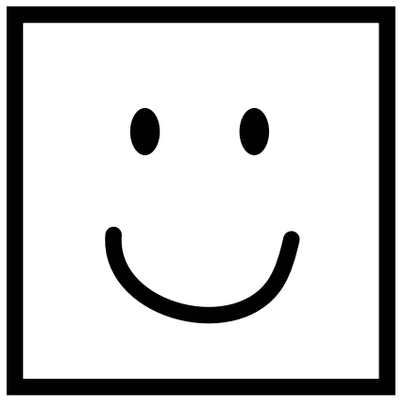


```
newBox := super
    new.
```

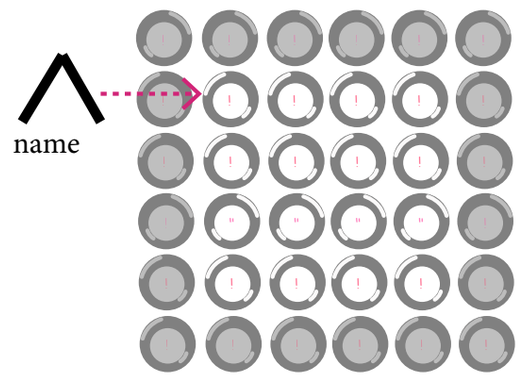


```
newBox undraw.
```

Jane



```
newBox name: aName.
newBox draw.
```




```
^ newBox
```

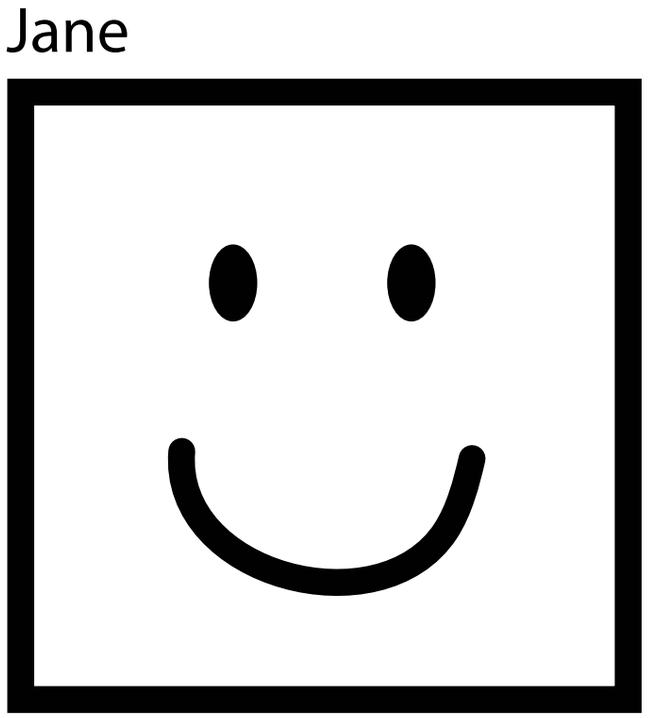
Added message

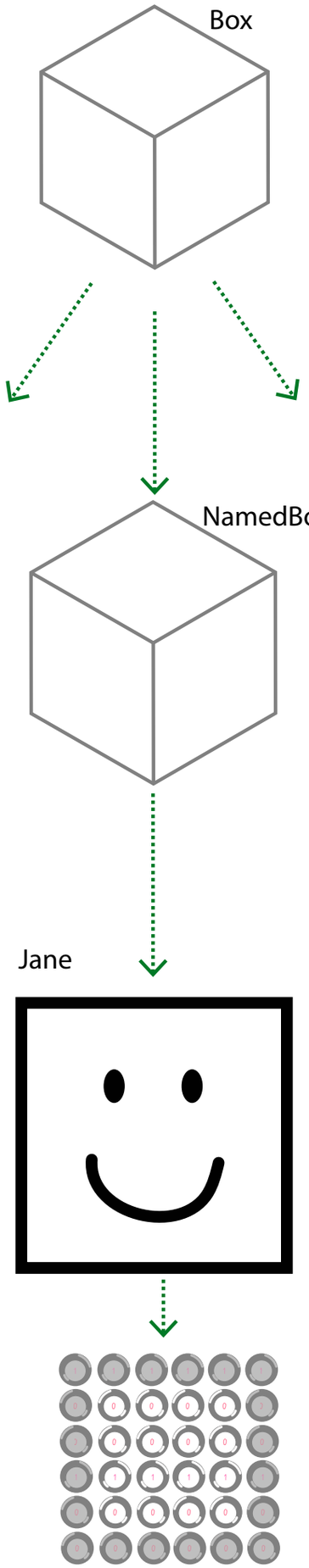
"Jane" the box prints her name on the screen. She can also do everything that Joe and Jill do because she has inherited her attributes and services from the parent class.

```
jane ⟵ isKindOf:  Box "PrintIt to see true"
```

The code below also draws "Jane"

```
jane := NamedBox new.
jane name: 'Jane'.
jane draw.
```





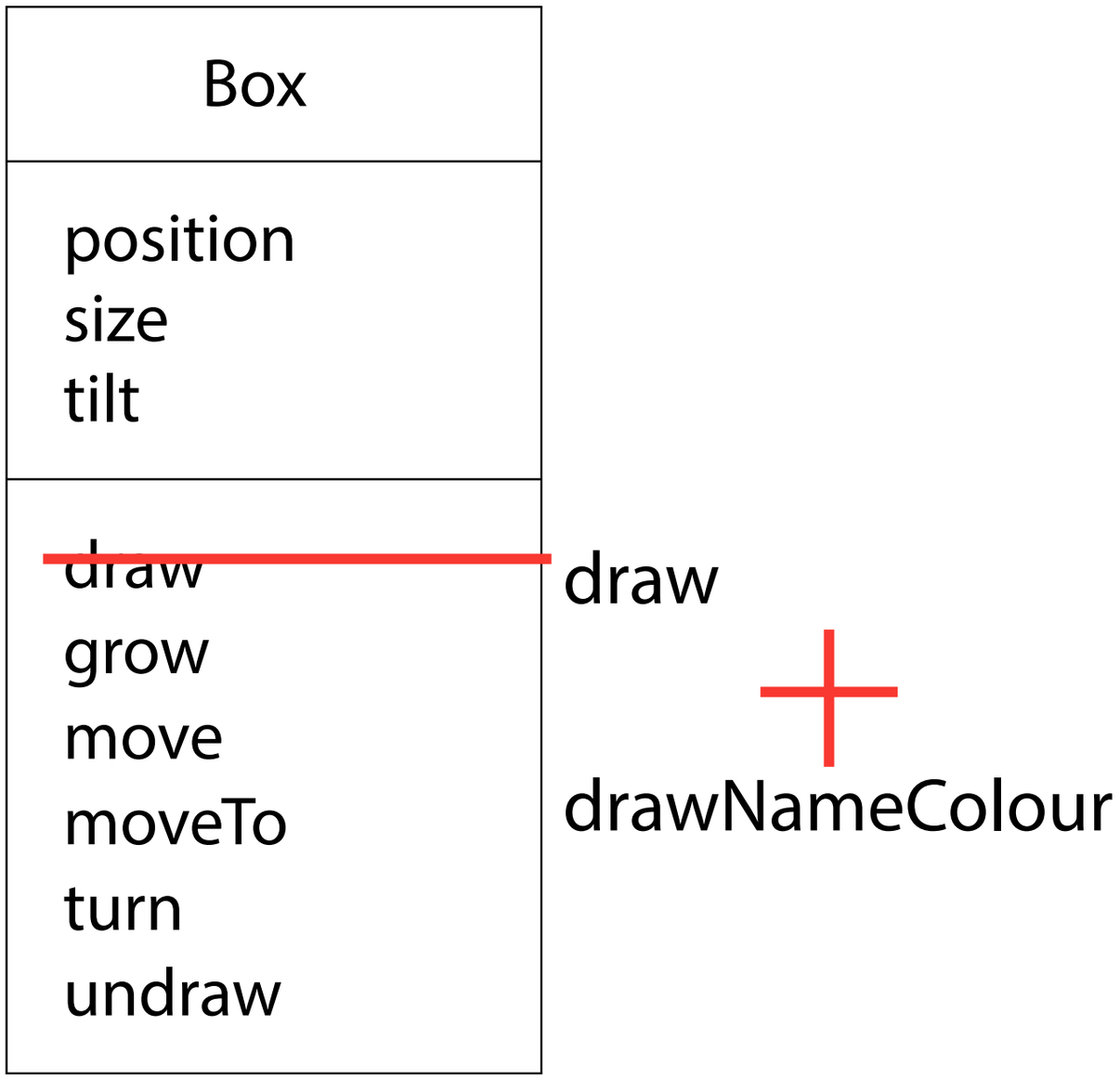
Box
position size tilt
~~draw~~ draw grow move moveTo turn undraw drawNameColour

Jane inherits services and attributes

When Jane is created, all services from the box class are passed down the inheritance chain to her.

Redefining Jane's inherited services

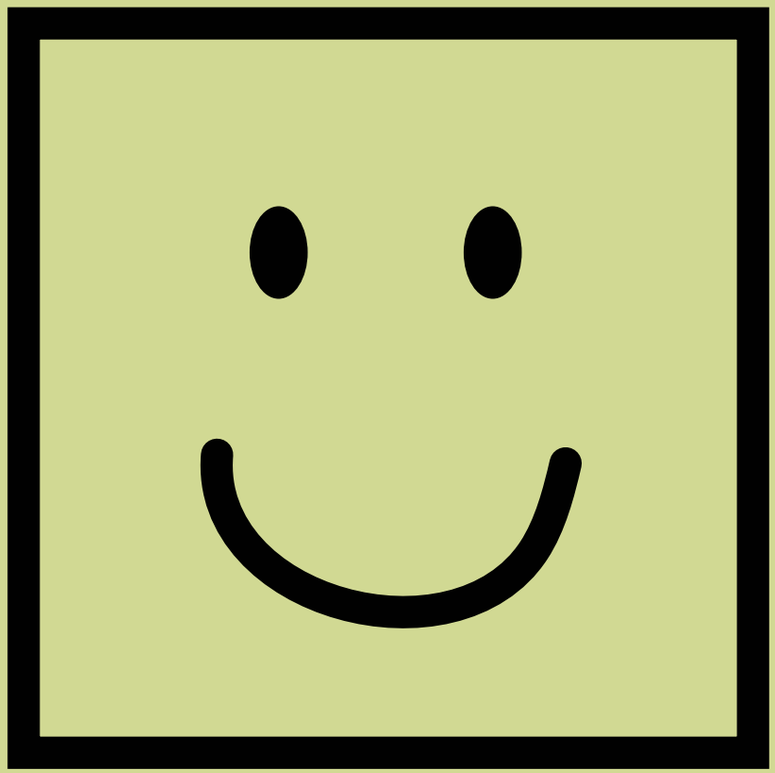
Jane has inherited all the same services as Joe and Jill however her services have evolved from her parents and she does some things differently. Jane overrides the message `draw` and also uses colour. She adds the message drawNameColour.



Box
position size tilt
~~draw~~ draw grow move moveTo turn undraw drawNameColour

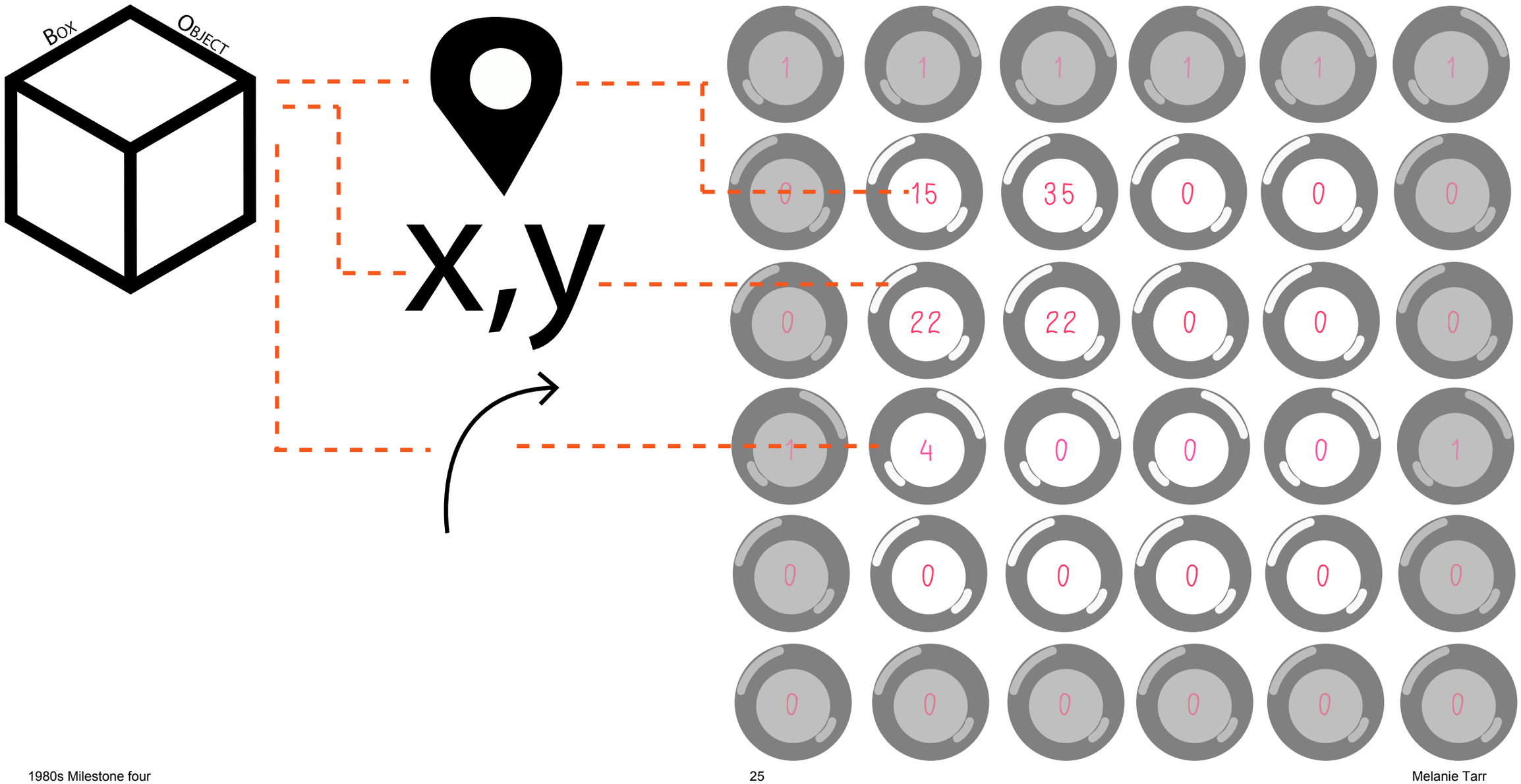
Jane can still do everything Joe and Jill do.

Jane



What does the box look like?

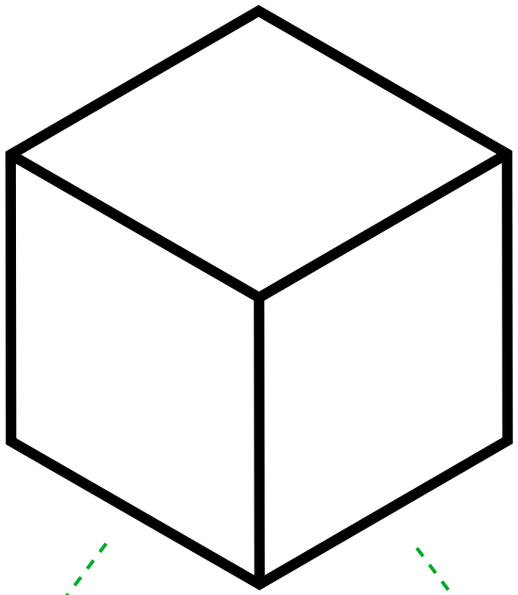
Box is defined as having the following attributes:

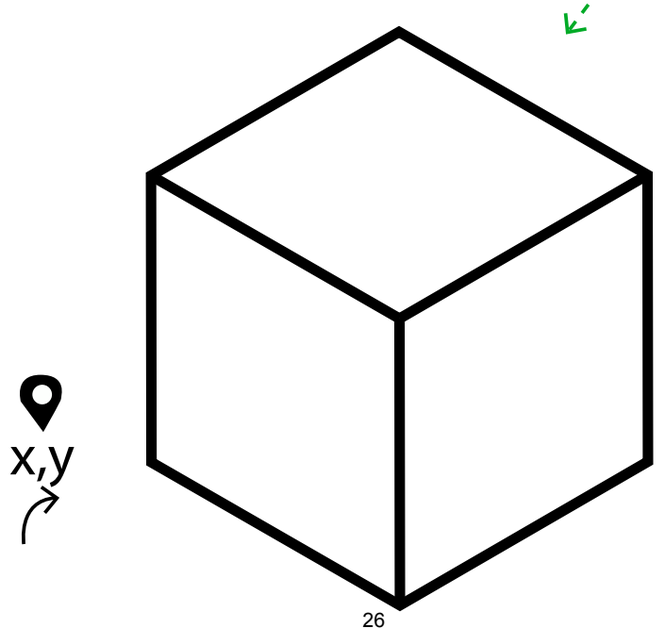


All inherited classes from Box have these attributes

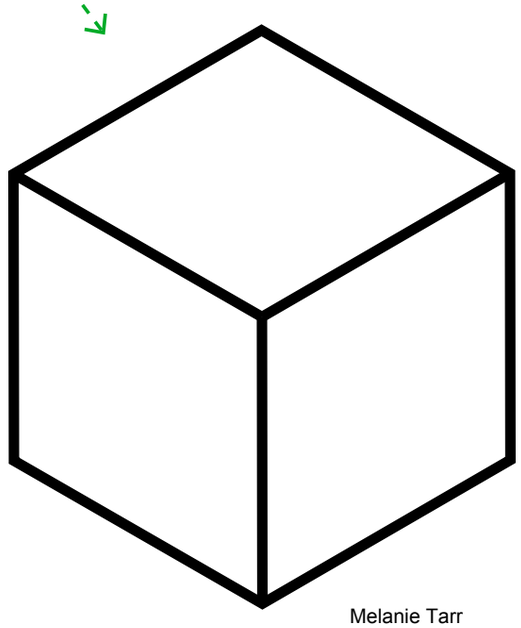
x,y

I have a position
I have a size
I have a tilt

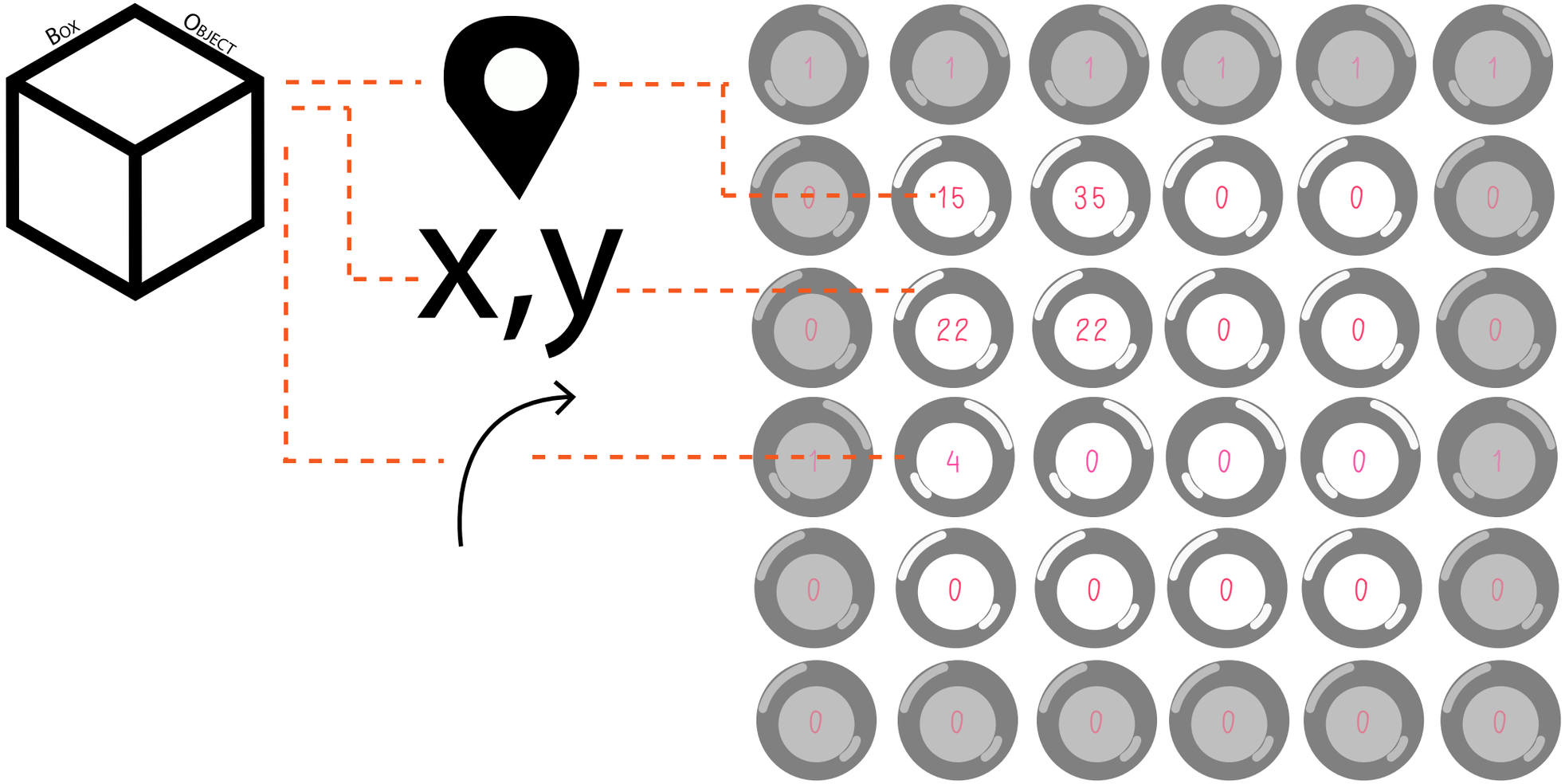




x,y



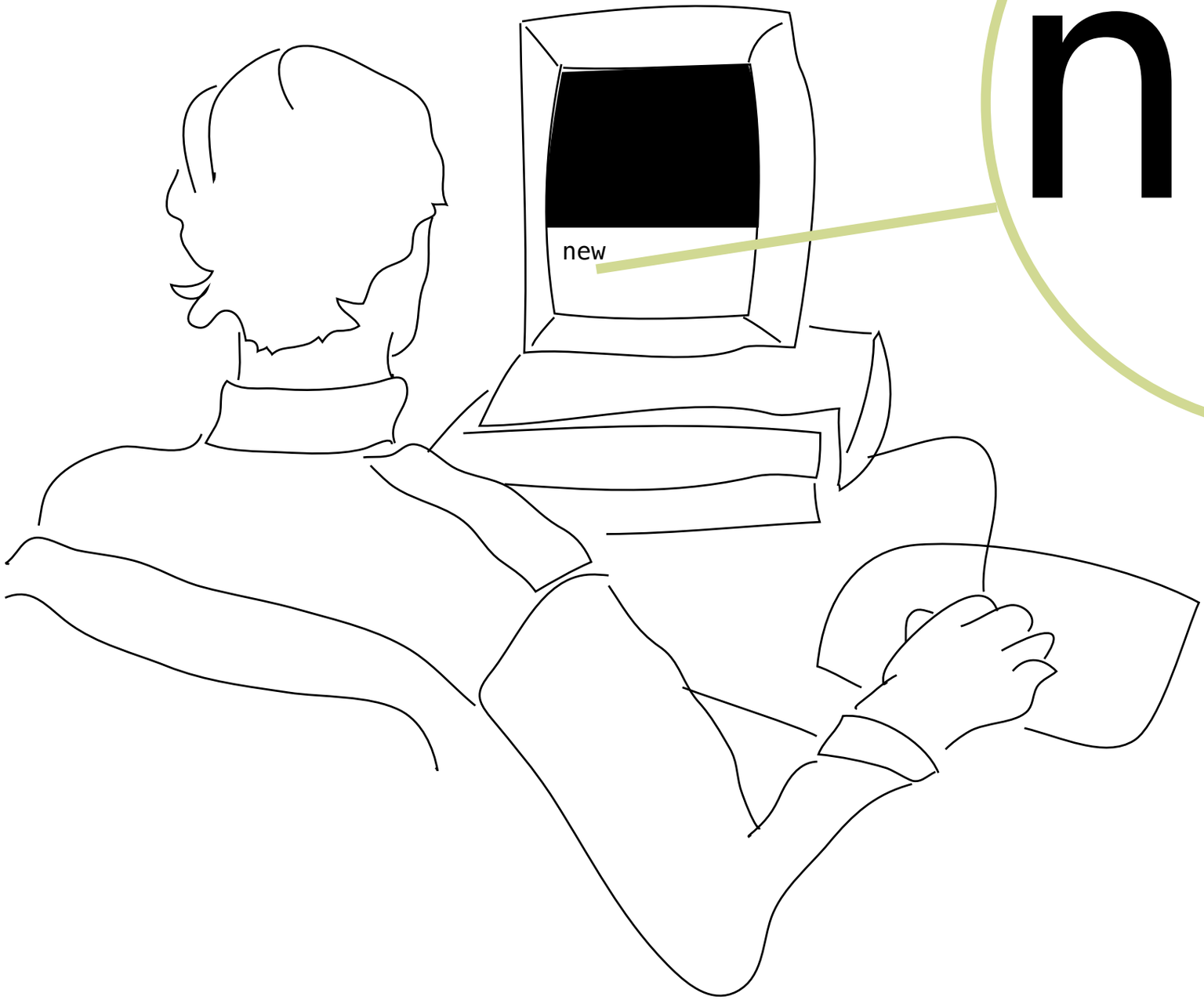
x,y



The following code defines a box

```
Object subclass: #Box
  instanceVariableNames: ‘position size tilt ‘
  classVariableNames: ‘’
  poolDictionaries: ‘’
  category: ‘Boxes’
```

Sending a new message



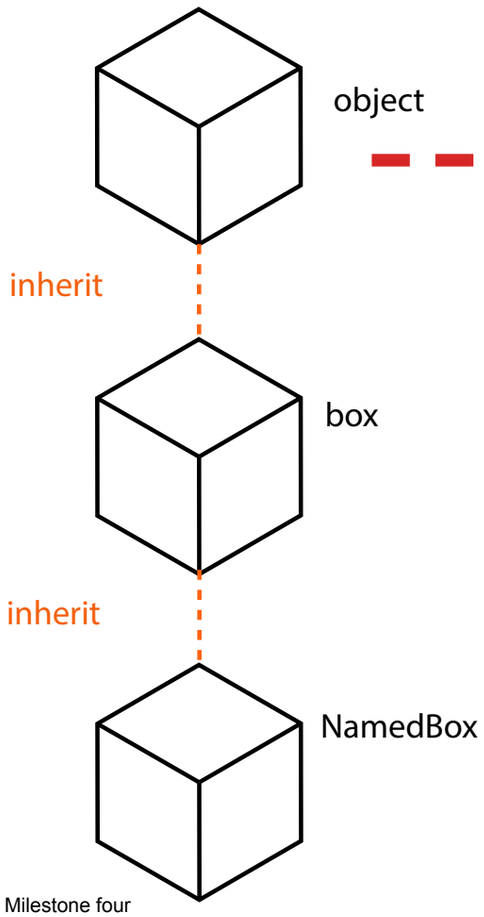
Once a programmer sends the message New to the Box object, the new class is created.

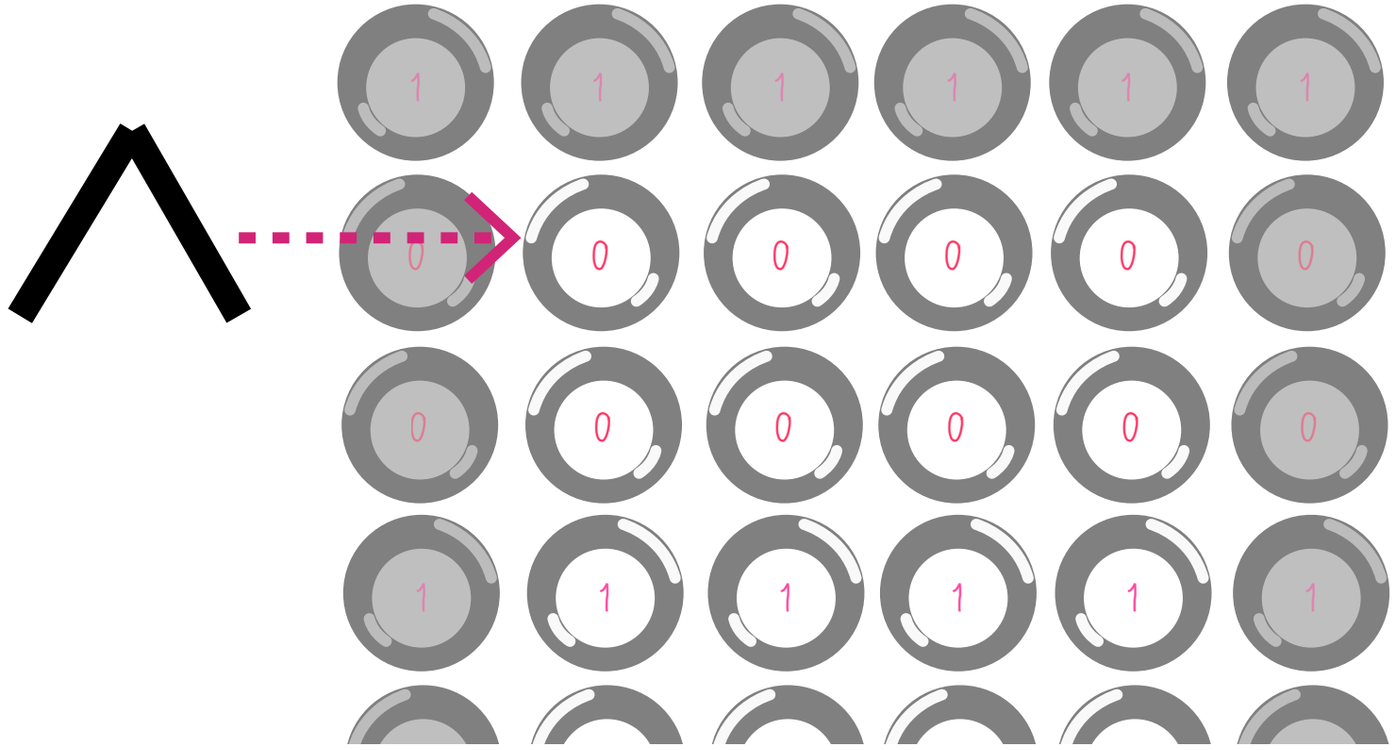
The inheritance chain execution

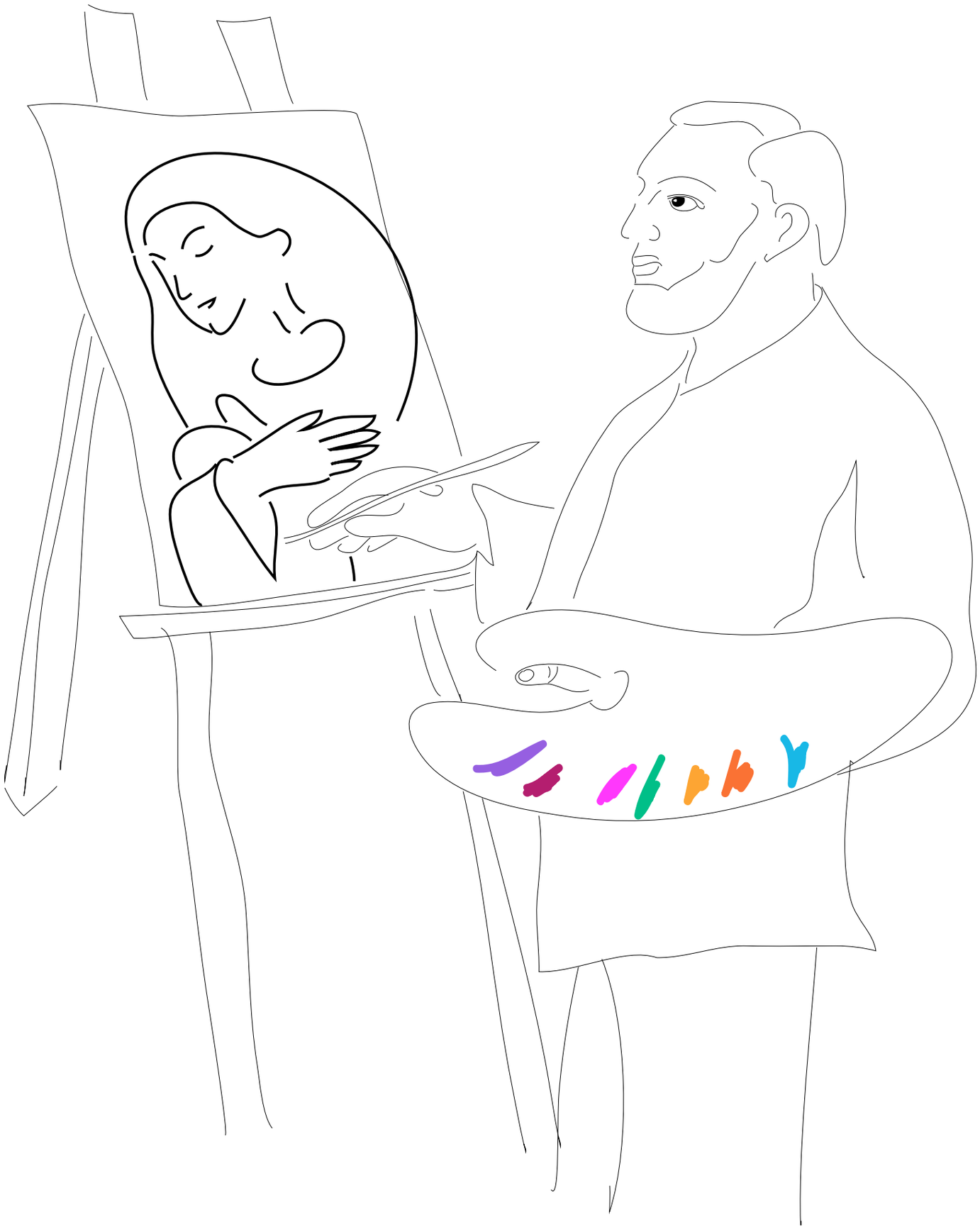
```
new
   ^super new initialize
```

The code above is executed automatically when the `new` message is entered by the programmer.

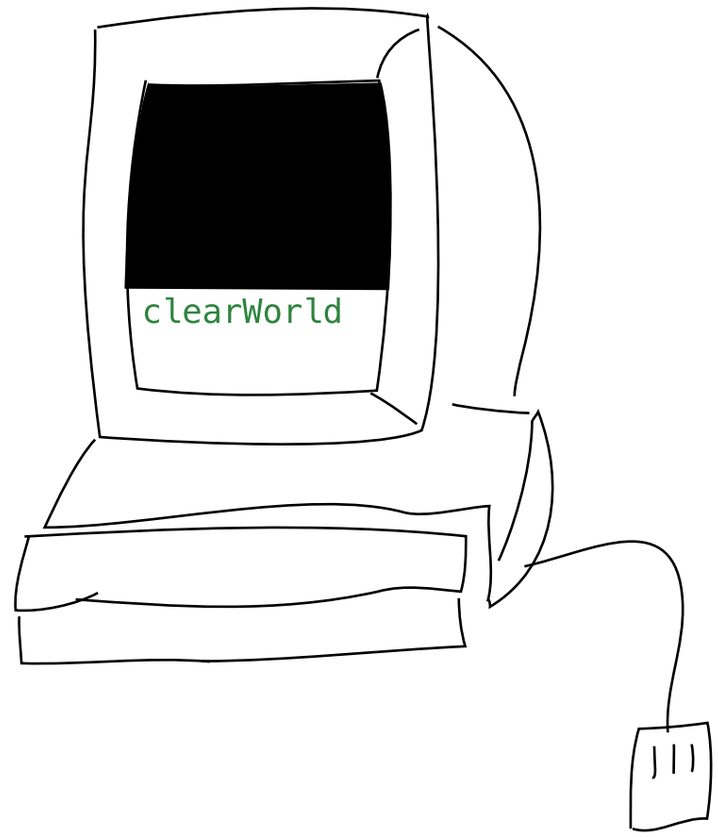
The ^ returns the position in memory of the current object in the inheritance chain.







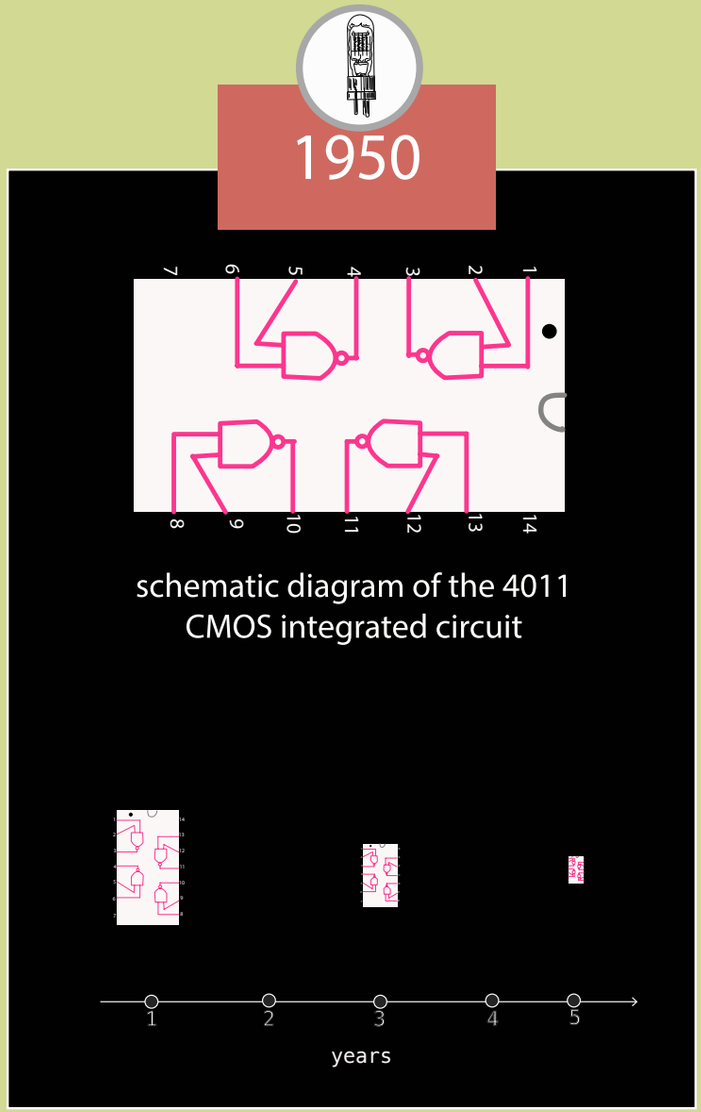
What is a clear world?

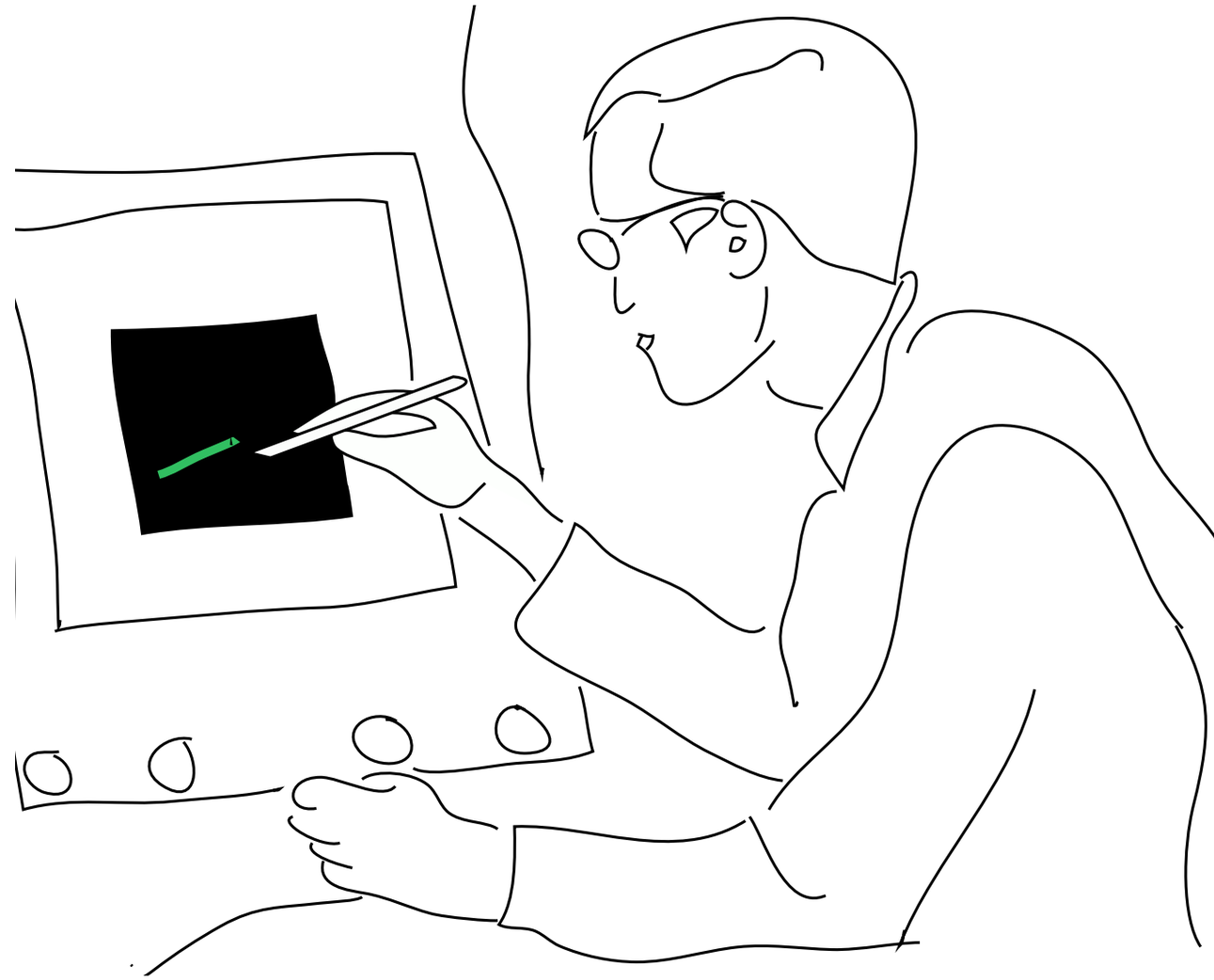


A clear world command creates a blank canvas on the computer screen to draw on.

```
clearWorld
```

Drawing in Smalltalk



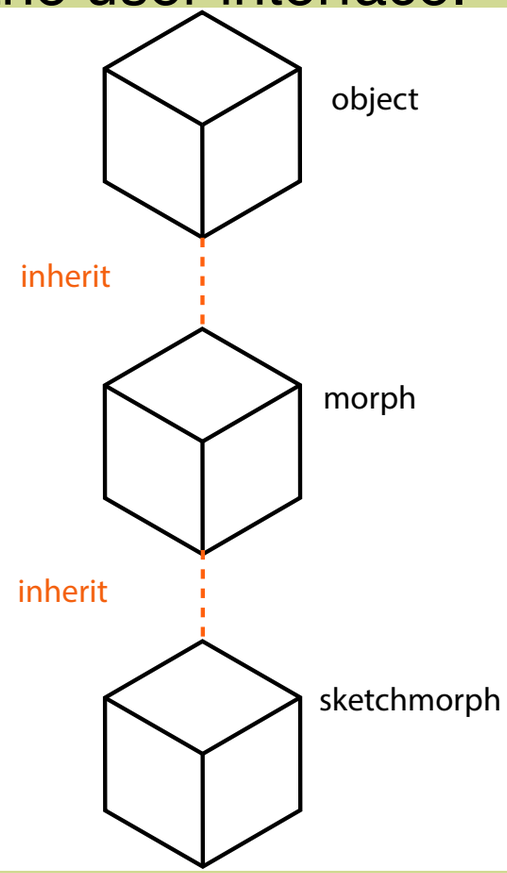


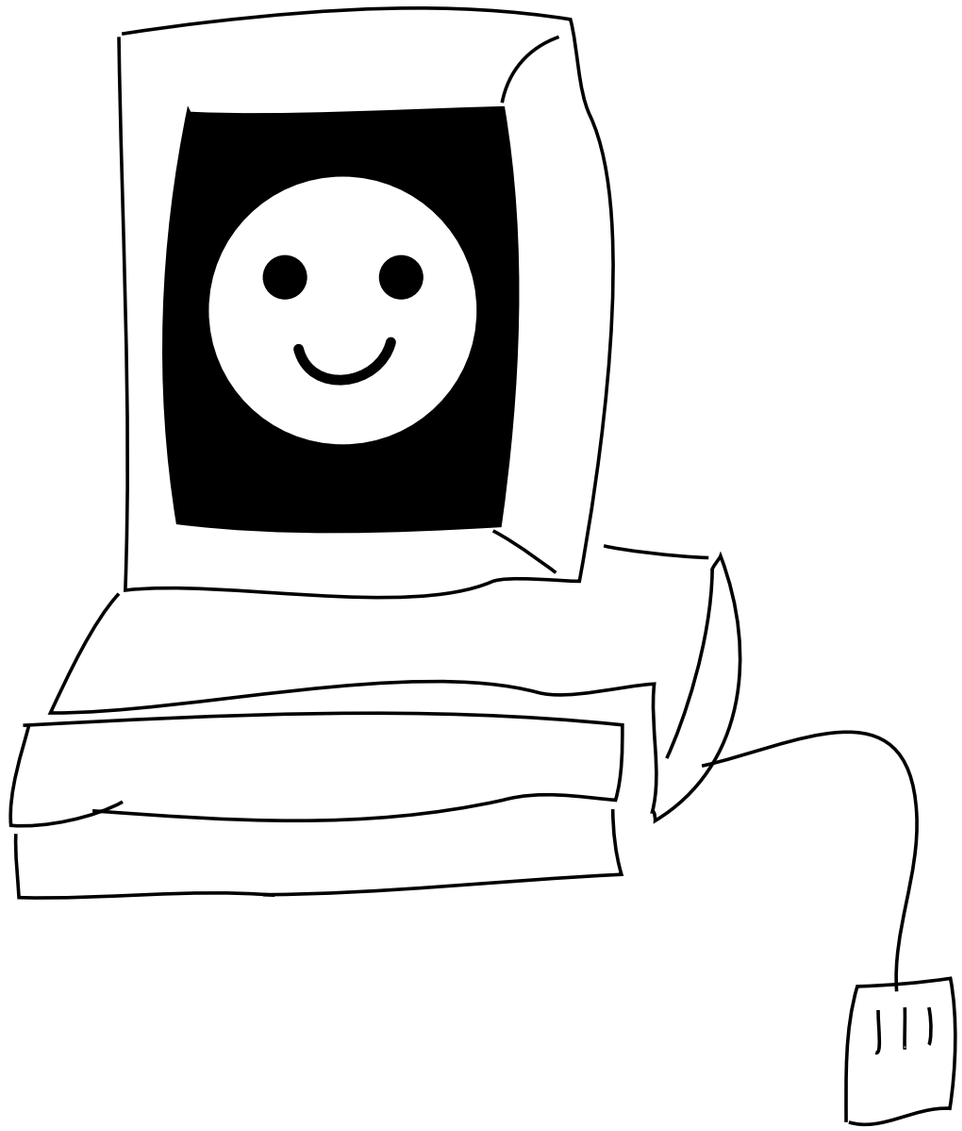
Due to Moore's Law processing capacity continued to increase in computers in the 1970s - 1980s. The increased processing capacity allowed drawing on the screen to be performed. Drawing involved complex calculations to be performed by the computer. Sketchpad and CAD systems were possible because of the increased processing power.

Forms

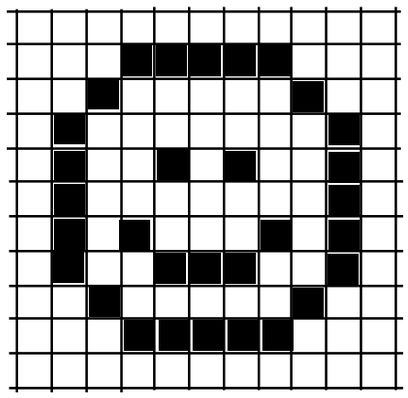
Forms are basic shapes in Smalltalk and are bitmap graphics. They are created in three ways:

1. user drawing;
2. coding;
3. through the user interface.





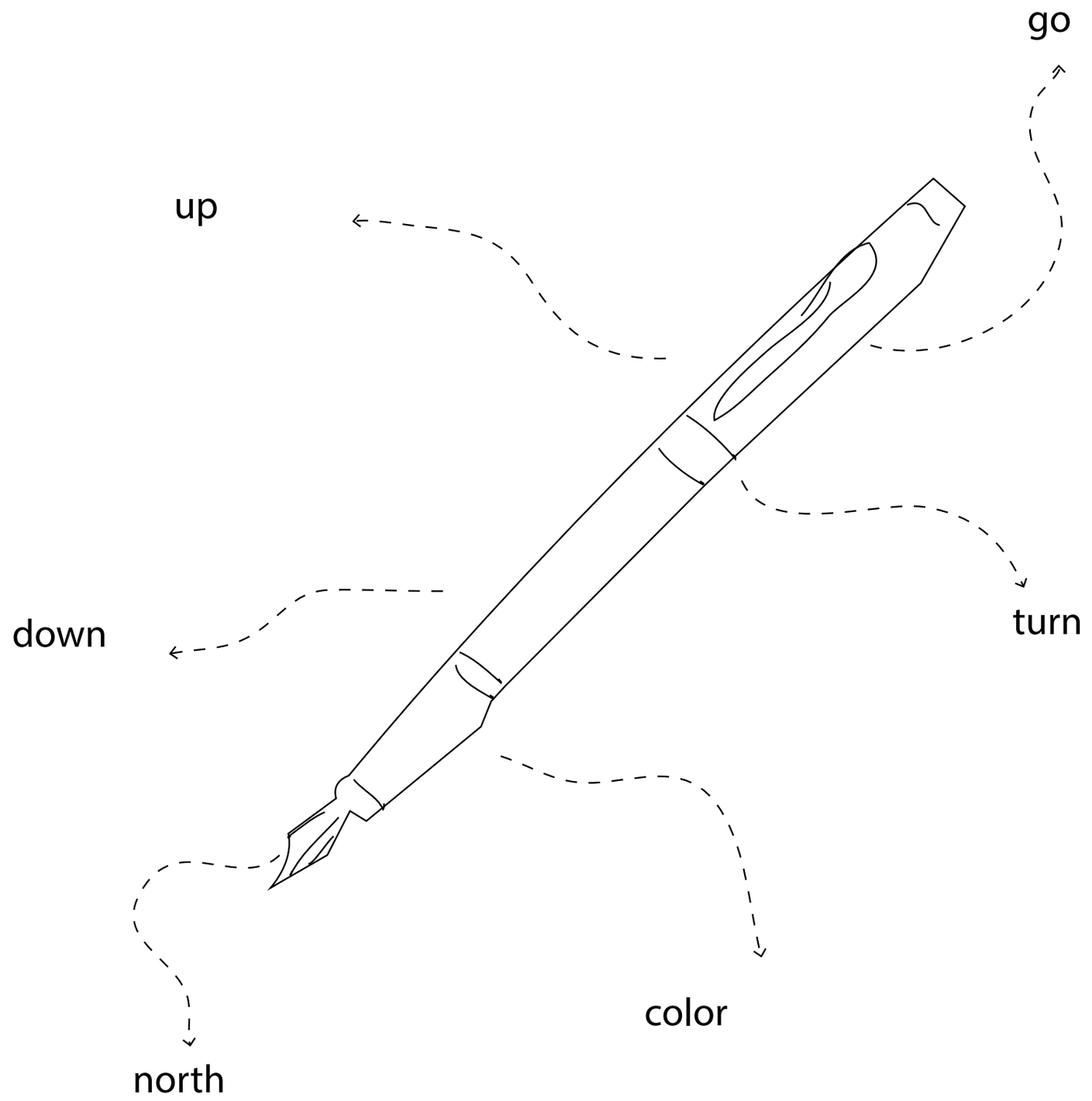
Through the programming of the sketchmorph object, a person can draw the form they want on the screen using the mouse.



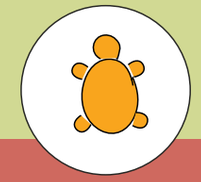
A picture element or pixel is the basic unit of programmable colour on a computer display. A bitmap graphic is composed of pixels. The file size of a bitmap graphic is large as details are stored about each pixel.

Teaching boxes to draw themselves

Although a box can now be created on the screen, it would be useful to see the box object drawn on the screen rather than have it simply appear. To do that, we write some code that creates a new object called `myPen`. `myPen` inherits services and attributes from the object Pen.



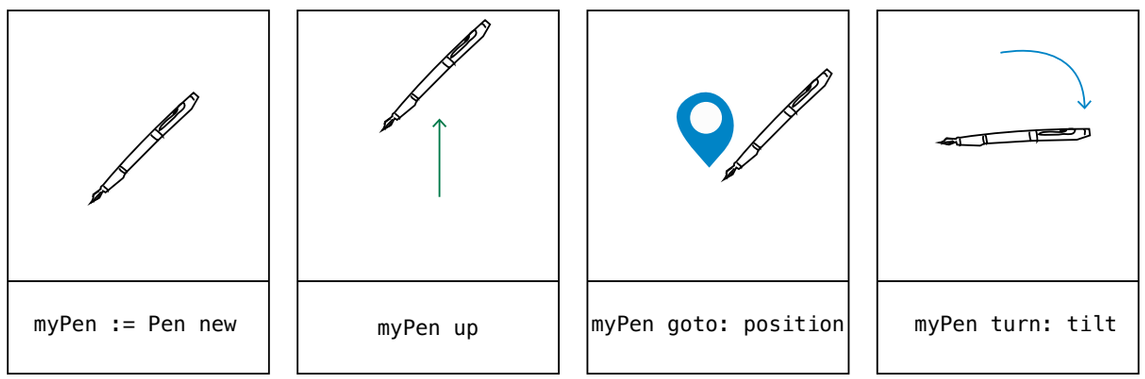
Pens come from turtles



1967

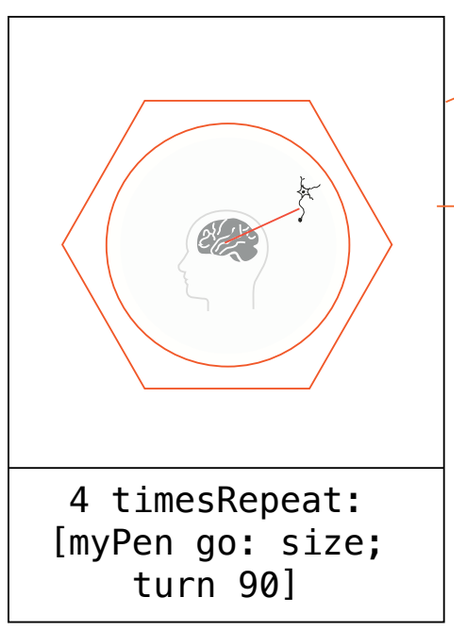
Pens are like turtle objects that are used to draw. The `draw` message for the box object creates the pen to draw.

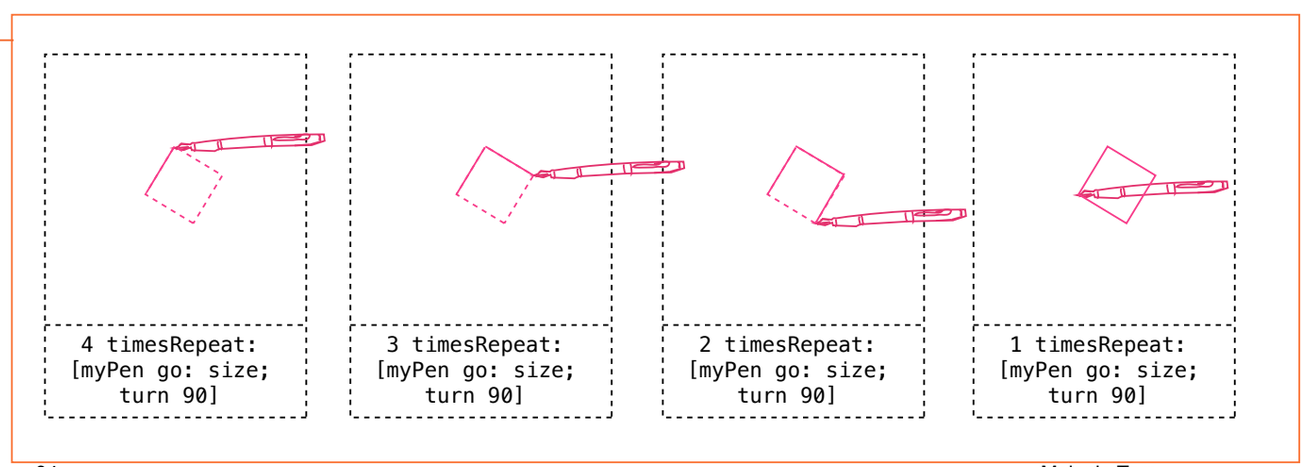
|myPen|



Abstraction is a core skill in computational thinking. Once a pattern, in this case, a line repeats itself, it can be abstracted into the one line of code.

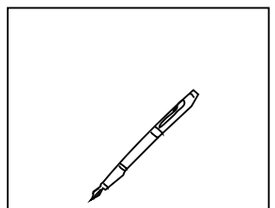
abstraction



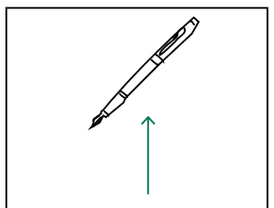


The `undraw` method is the same as the draw method only the pen writes over the form in white.

|myPen|

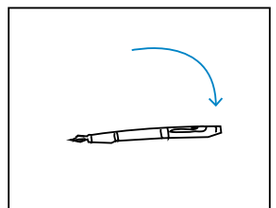


```
myPen := Pen new
```

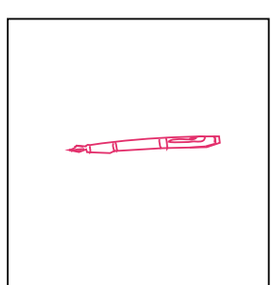


```
myPen up
```

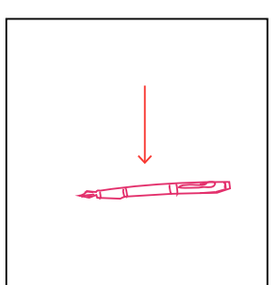
```
myPen goto: position
```



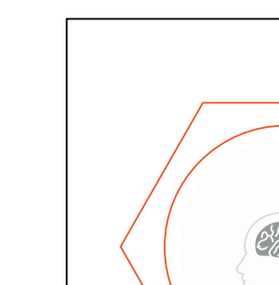
```
myPen turn: tilt
```



```
myPen color:
(colorPink)
```

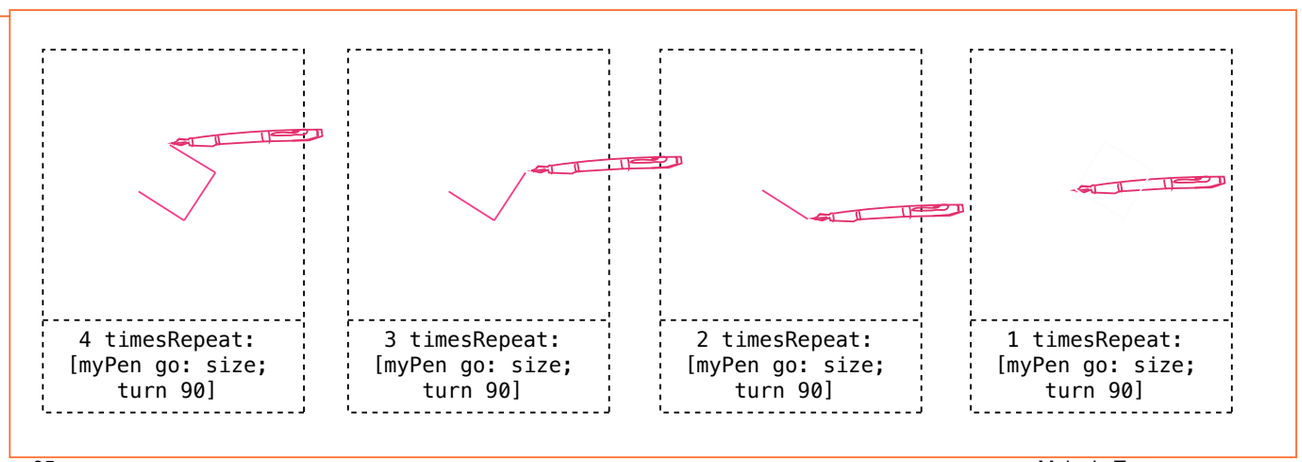


```
myPen down
```

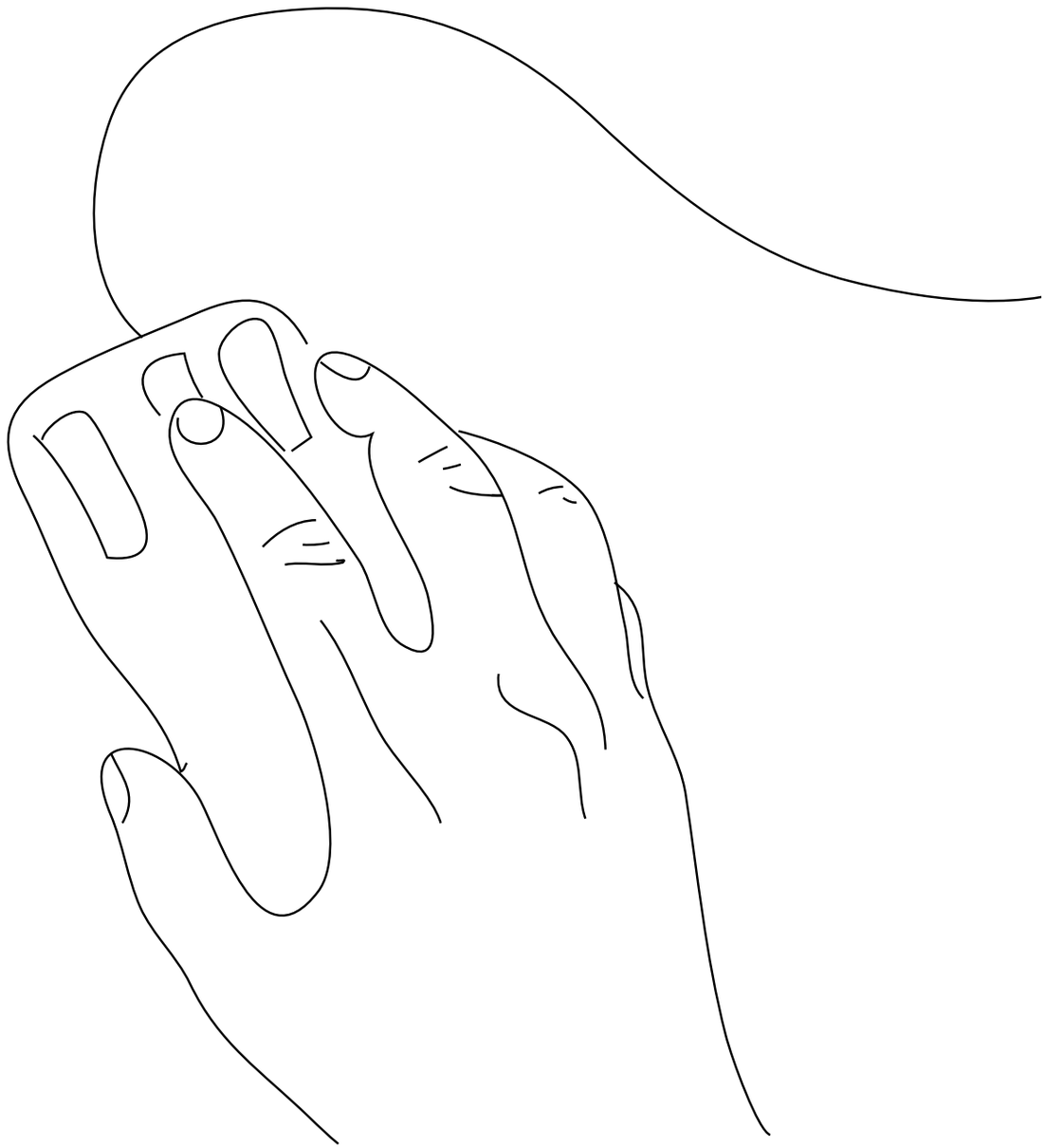


```
4 timesRepeat:
[myPen go: size;
turn 90]
```

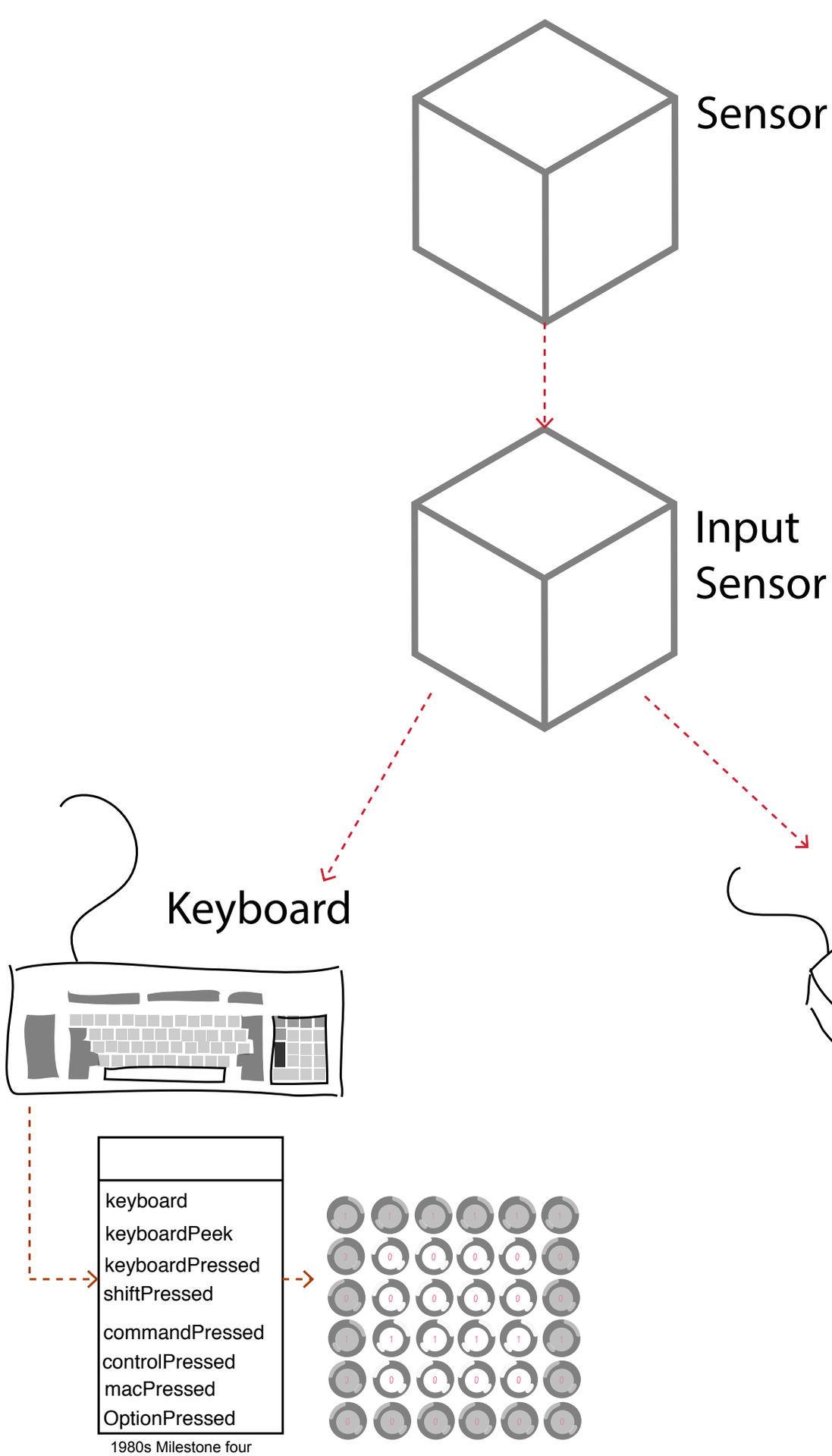
abstraction



Getting input from the User



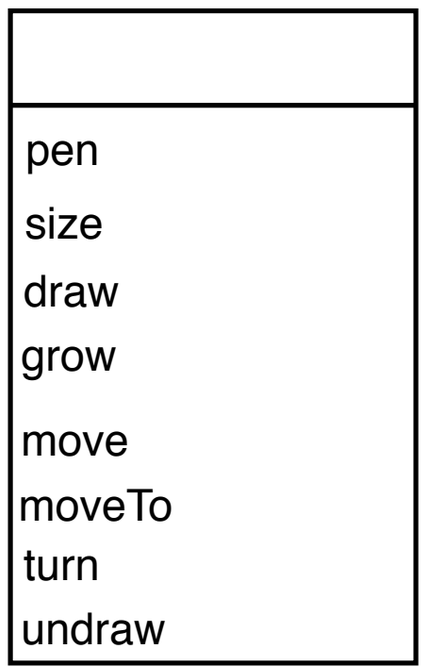
The sensor variable is triggered whenever the mouse or the keyboard is activated in Smalltalk. The `Sensor` message gets input from the user.

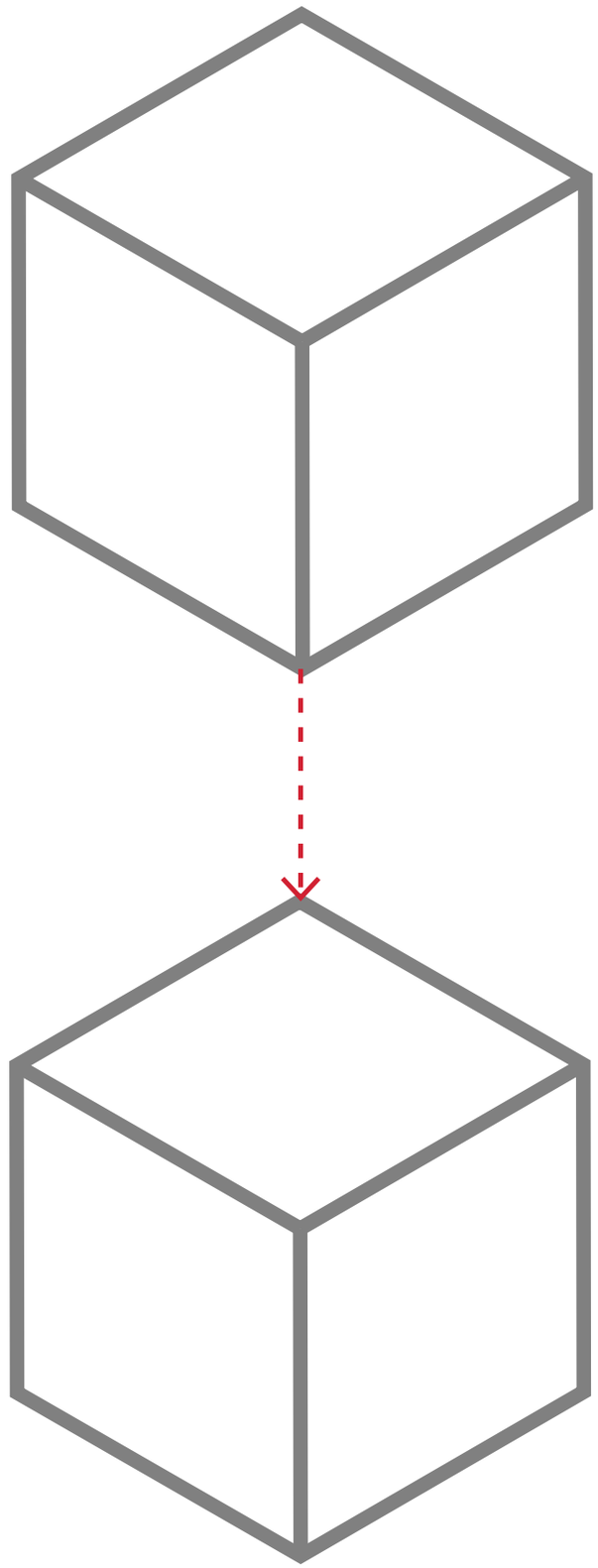


The `Sensor` variable is triggered whenever the mouse or the keyboard is activated in Smalltalk. The `Sensor` message gets input from the user through the mouse or the keyboard.

Delegation: Improving boxes by drawing better

The activity of asking another object to perform a service for the object who receives the message is called delegation. Sometimes in this software the box instance delegates services to the pen instance.





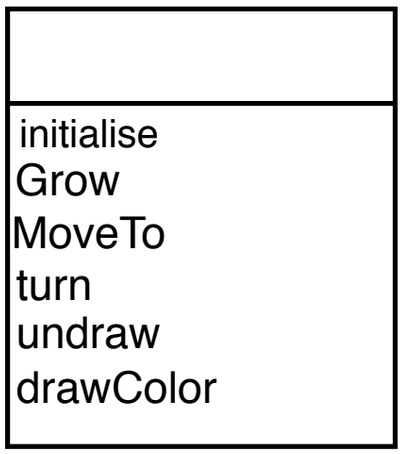
Box

tilt

position

handling

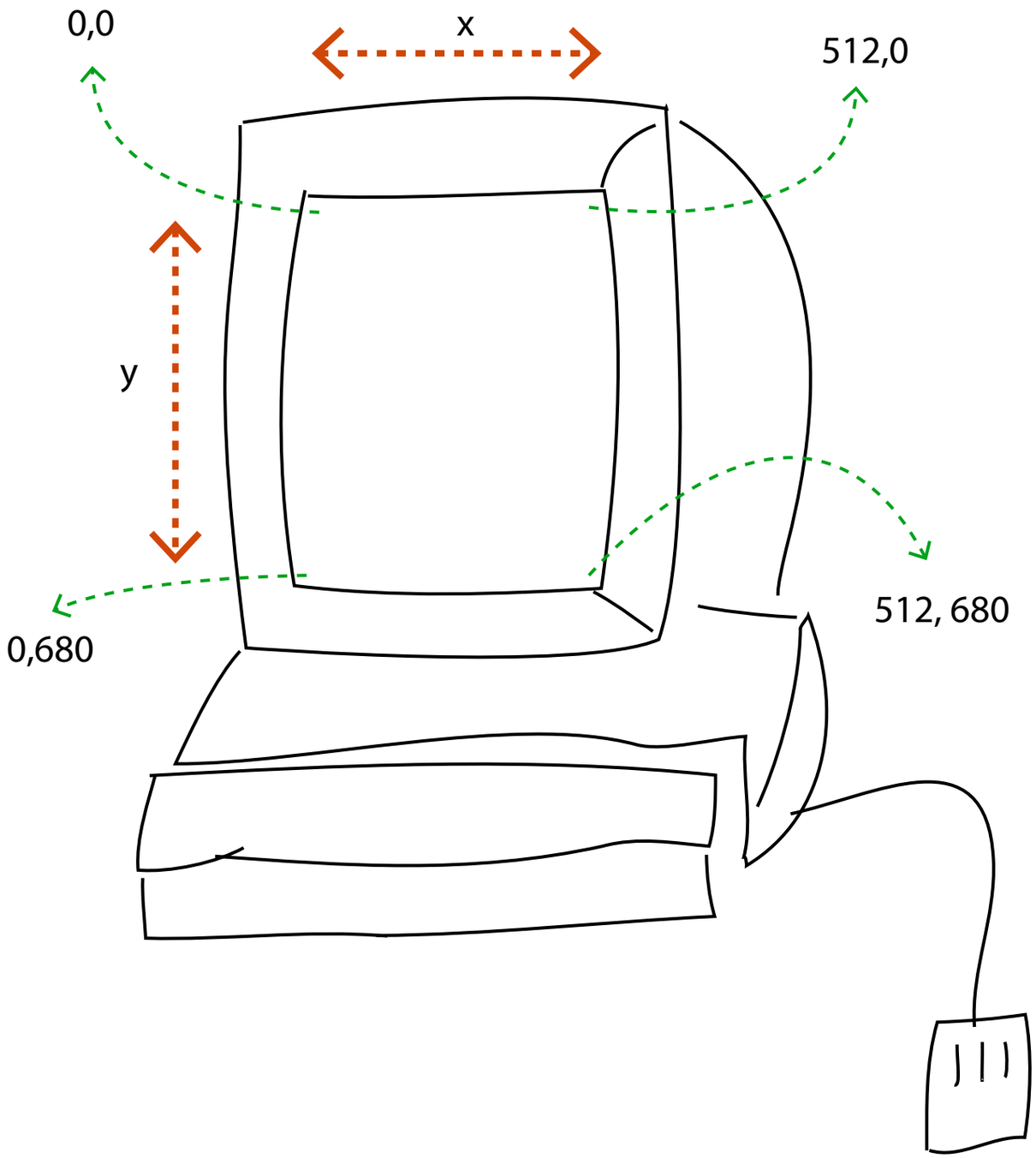
location



Pen instance

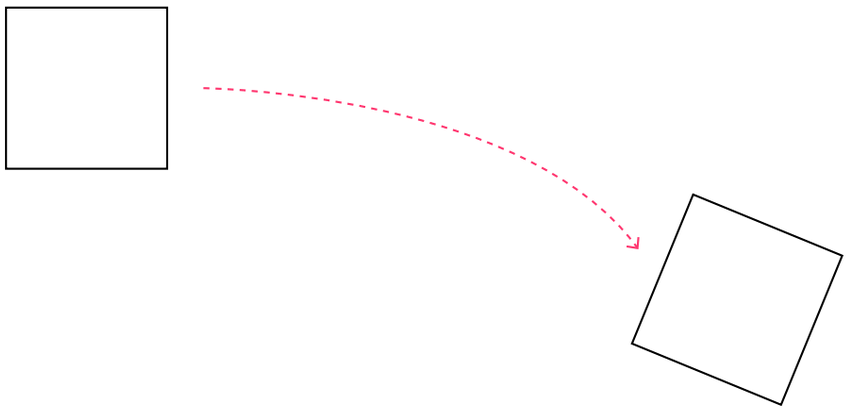
size

Animation



Due to Moore's Law, computer's operate so fast that calculations need to be slowed down when the box is moving so that animation can be seen. The program needs to be delayed to 30 frames per second so our eyes can register a static image.

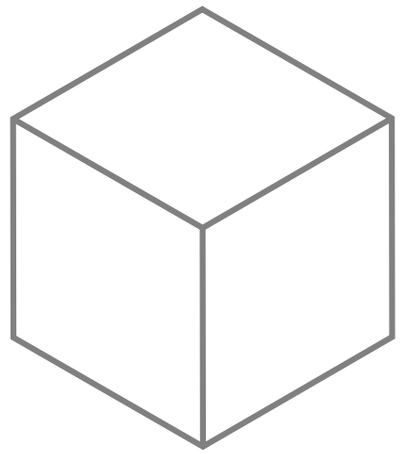
By adding the following code to the `draw` service, an animation can be

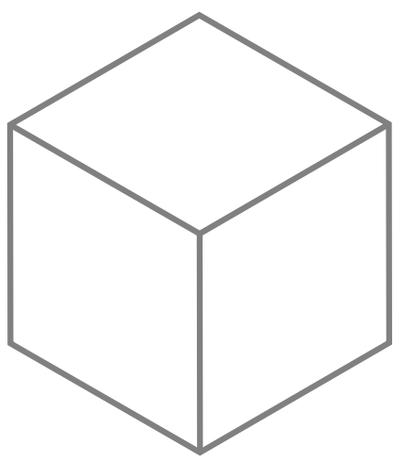


```
draw
  self drawColor: (Color black).
  (Delay forSeconds: (1/30)) wait.
```

By creating the joe and jane box objects

```
joe ← Box new
jane ← Box new
```





And typing the following sequence of code

```
30 timesRepeat: [jane turn: 12. joe turn: 10.
jane move: 3@4. joe move: 2@3].
```

The boxes move around the screen at a rate the eyes can follow.