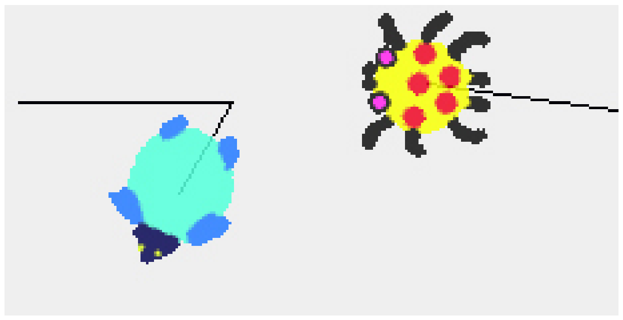
1 Fun with Robots



The basic look of the robots is rather simple. In this chapter I show you how you can change the shape, the pen size and the color of robots. I present how you can draw yourselves the way your robots look like. For example your robot can look like an animal or a monster.

1 Robot Handles

In addition to the fact that we can send messages to robot by clicking on them, you can have access to other functionalities such as duplicating, moving, or changing the look of your robot. These extra functionalities are available via handles that you get when you left-click on a robot. The handles are the round little icons around the robot as shown in Figure 1.1. I will explain the functionalities when needed. Note that if you let your mouse over a handle a balloon will pop up and explain you its purpose. For now try to duplicate the robot by clicking on the green handle, move it by clicking on the black or destroy it using the pink pale crossed handle.

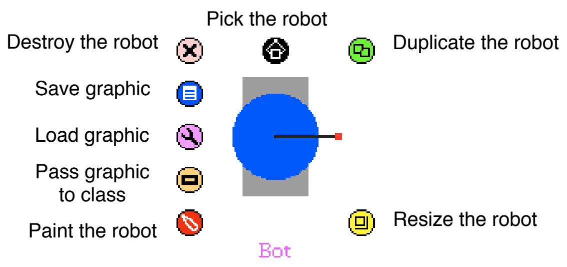


Figure 1.1: Right-click to get the handles.

2 Pen Size and Color

So far the trace left by our robot was black. However, you can change the color of a robot's pen by sending a robot the message penColor: with a color. For example the expression pica penColor: Color blue changes the color of the pen to blue. One of the ways to create a color is to send a message with the name of a color to the class Color, such as Color blue or Color yellow. We will explain colors in the following section.

We can also change the thickness of the robot's pen by sending the message penSize: with a number as argument, for instance pica penSize: 5 make the pen to be 5 pixels wide.

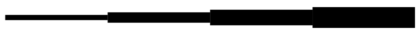
Script 1.1 draws a thick blue line of 5 pixel width. Script 1.2 draws a strange staircase by repeatedly increasing the pen size.

Script 1.1 (*A blue line)*

```
| pica |
pica := Bot new.
pica penColor: Color blue.
pica go: 100.
pica penSize: 5.
pica go: 100
```

Script 1.2 (*Stair)*

```
| pica |
pica := Bot new.
pica go: 40.
pica penSize: 2.
pica go: 40.
pica penSize: 4.
pica go: 40.
pica penSize: 6.
pica go: 40.
```



Changing the color of the robot itself is also possible using the method color:. For instance, the expression pica color: Color yellow changes the color of the robot to yellow. Script 1.3 asks daly to change its color and go forward, while pica does not move and keep its default color.

Script 1.3 (*Two Robots)*

```
| pica daly |
pica := Bot new.
daly  := Bot new.
daly color: Color yellow.
daly go: 100.
```

3 About Colors

As previously mentioned already, Squeak is an environment built from and using objects. Therefore programming in Squeak amounts to creating objects and sending them messages. In particular a *color* is an object created by the class Color. To get a color you send a message to the class Color.

Some color messages are named for the color they get. For example, Color red creates the color red. Here is the list of the predefined color name messages that you can send to the class Color to create that color: black, veryVeryDarkGray, veryDarkGray, darkGray, gray, lightGray, veryLightGray, veryVeryLightGray, white, red, yellow, green, cyan, blue, magenta, brown, orange, lightRed, lightYellow, lightGreen, lightCyan, lightBlue, lightMagenta, lightBrown, lightOrange, paleBuff, paleBlue, paleYellow, paleGreen, paleRed, veryPaleRed, paleTan, paleMagenta, paleOrange, and palePeach.

You can also make a color by telling the Color factory how to make it by mixing red, green and blue. script 1.4 shows how to create colors this way using the method r: red g: green b: blue. The arguments of method r:g:b: should be decimal numbers between 0 and 1.0. For example the expression Color r: 1 g: 0 b: 0 creates the same pure red color that you get from Color red.

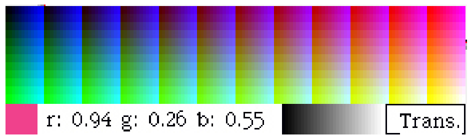
Finally the method fromUser lets you pick a color from a palette on the screen, and then shows you that color's ingredients. For that you need to execute the expression Color fromUser using the **print it** menu to get the result of the selection printed. Print the result by using the **print it** menu when executing the expression Color fromUser.

Script 1.4 (*Other ways to create colors)*

```
"Produce a pure red"
Color r: 1 g: 0 b: 0

"Produce a light gray"
Color r: 0.1 g: 0.1 b: 0.1

"To choose your color from a palette"
Color fromUser
```



4 Changing Robot Shape

Another aspect of a robot that you can change is its shape. Two different shapes – a circle, and a triangle are built into the Bot factory. But you can also draw the robot shape with a drawing tool as shown later in Section 5.

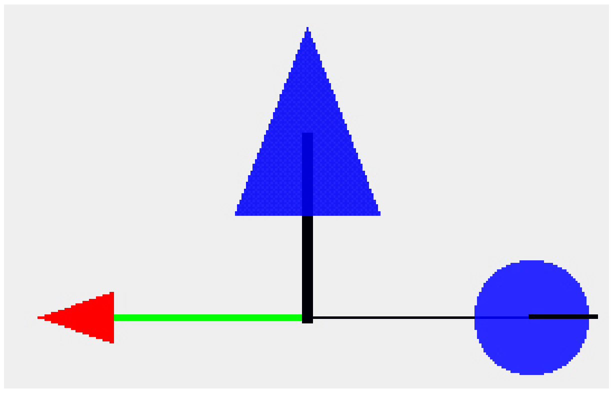


Figure 1.2: Different robot shapes and sizes.

script 1.5 shows how to change the shape of a robot as shown in Figure 1.2. The message look-

LikeTriangle gives a triangular shape to a robot. The message gives a circular shape to a robot. The default shape is produced by sending the message lookLikeBot.

Script 1.5 (*Creating Robots of Different Sizes and Shapes)*

```
| pica daly  bigpica |
pica := Bot new.
pica lookLikeTriangle.
pica west.
pica color: Color red.
pica penColor: Color green.
pica penSize: 3.
pica go: 100.
daly := Bot new.
daly extent: 60@60.
daly east.
daly go: 100.
bigpica := Bot new.
bigpica lookLikeTriangle.
bigpica extent: 100@150.
bigpica penSize: 5.
bigpica north.
bigpica go: 80.
```

Robot Size. The second aspect you can change is the size of a robot using the message extent: widthAndHeight, where the values of widthAndHeight represent the width and height of the rectangle in which the robot is drawn. The argument widthAndHeight is a pair of numbers also called a point in Squeak. It is composed of two numbers separated by the @ symbol. For example, the point 50@100 represents a rectangle of 50 pixels by 100.

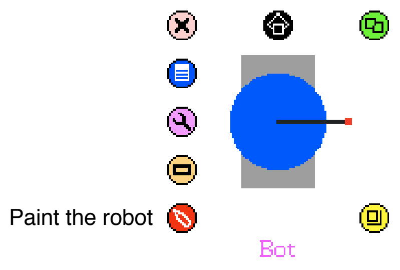


Figure 1.3: Right-click to get the handles. Getting the painting editor from the red handle.

5 Drawing Your Own Robot

Squeak lets you draw the robot itself and get robots that look like the figure in the heading of this chapter. Now we describe step by step how you can draw your own robot.

Step1. Getting the Painting Tool via the red Handle. The first step is to open the painting tool that is included in Squeak. Right click or press command and click for Mac to get the halos around the robot you want to paint as shown by Figure 1.3. Click on the red handle with the pen inside. This should open the paint tool shown in Figure 1.4. Do not worry about the other handle right now I explained them just after. Note that if you already drew a graphic, the graphic will be shown inside the painting tool.

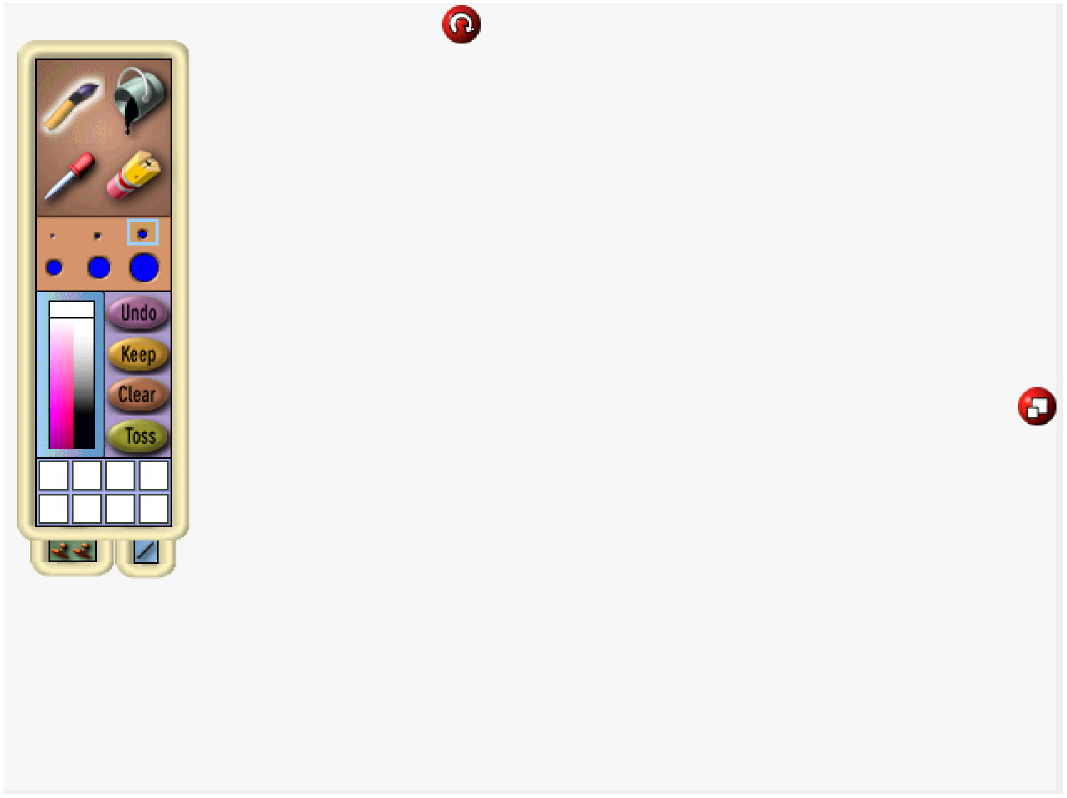


Figure 1.4: The opened painting editor.

Step2. Drawing the New Graphic. The second step is to draw a new graphic for your robot. Draw your robot pointing to the right, as shown in Figure 1.5. The painting editor has the usual features like choosing the brush size, filling a region, repeating a selected region, and selecting the painting color. The painting tool also has two buttons (shown in Figure 1.6) to rotate and zoom your drawing.

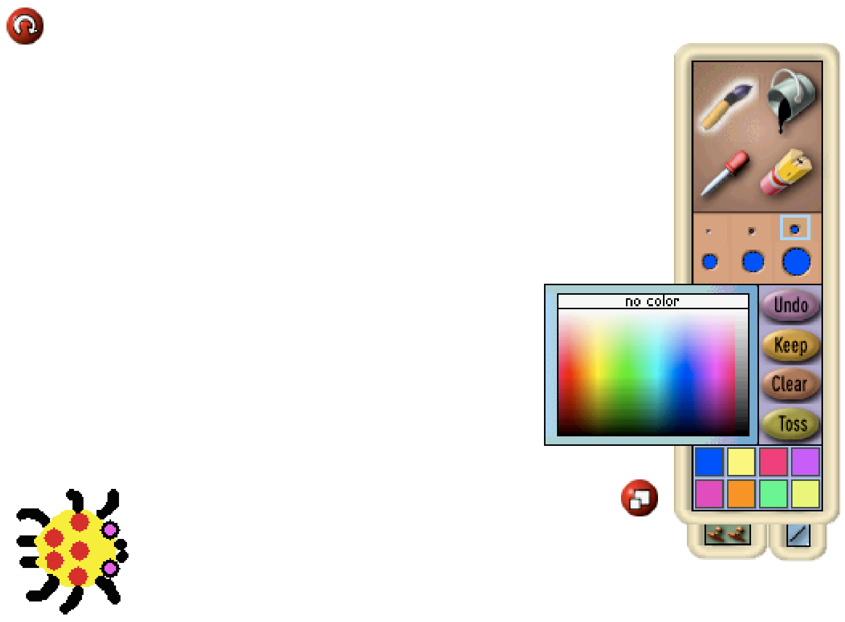


Figure 1.5: Drawing a cool spider.

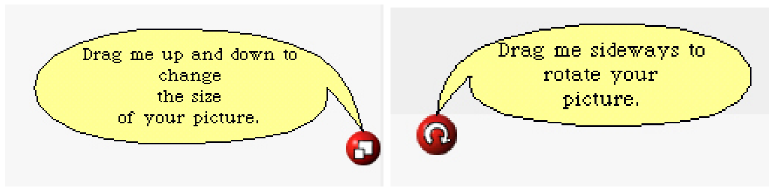

Drag me up and down to change the size of your picture.
Drag me sideways to rotate your picture.

Figure 1.6: The zoom and rotate buttons.

Step3. Keeping the Graphic. Once you are satisfied with your drawing, you should press the button **keep**. This closes the painting tool and your robot know is looking like graphic. Note that you can still get the other shapes such as the robot, the triangle or the circle sending the messages I presented above.

6 Saving and Restoring Graphics

Once you have spent lot of time in drawing a robot, you can save it on a file. This way you will be able to load it in several environments, to exchange it with friends and you can to build a library of graphics over time. I will show you now how load save and load a graphic, then I will show how you can associate a graphic to a single robot or to a class so that all the newly created robots will look like the graphics you drew. I will start by showing you how to do all these manipulations by interacting with the robots directly but also how we can write scripts to do that automatically.

6.1 Using Handles

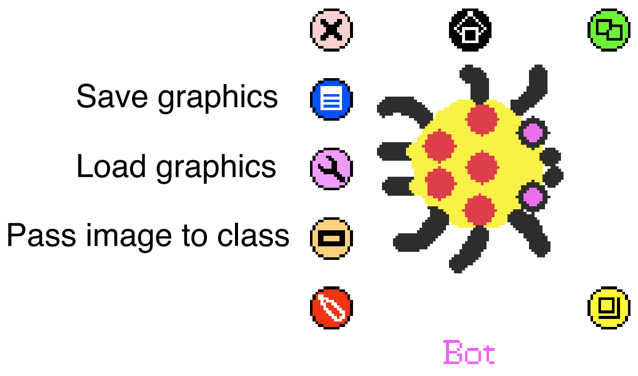


Figure 1.7: The robot is now looking like a spider and is pointing to the right.

To save a graphic simply click on the blue handle with a kind of file logo (Figure 1.7). I chose the color blue by analogy with ice thinking that my robot would get frozen. The system will ask you to give a name to the saved graphic as shown in Figure 1.8. This operation save your graphic on a file close to the Squeak image with the name you entered and with the extension .frm.

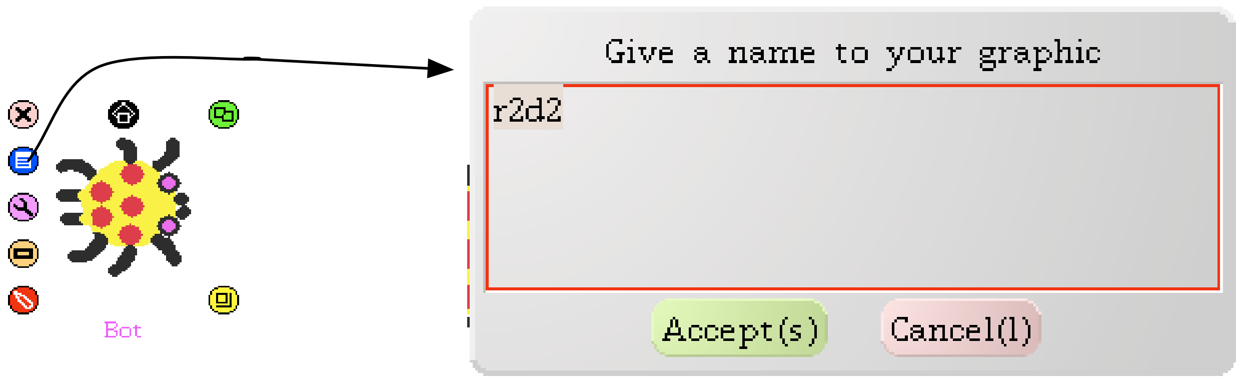


Figure 1.8: Clicking on the blue handle and getting prompted for a name.

Doing a similar operation, you can load a graphic, by clicking on the pink halo with a tool of a robot. I used the color pink with the idea that the color pink would mean bringing back to life your robot. When you click on the pink halo the system asks you for the name of the graphic you want to load. When the operation succeeds your robot will look like the graphic you just loaded.

Involving the Class. Now even if you drew a wonderful insect, when you will ask the class to create a new robot, you will get the default graphic. But what you can do is to tell your robot to pass its graphic to the class using the message passImageToClass. Now if you create a new robot and ask it to look as the image, it will look as the graphic you just draw.

Note that if you send the message lookLikeImage or any of the lookLike message to the *class* itself, the class will be configured so that the new robots will look following the class configuration. For example, if you send the message lookLikeCircle to the *class* Bot, all the new robots like look like a circle. Therefore if you want to have a class creating spiders, you have to (1) get a robot, (2) draw the spider, (3) pass the spider image to the class. Then all the new robots will look like a spider as shown by Figure 1.9.

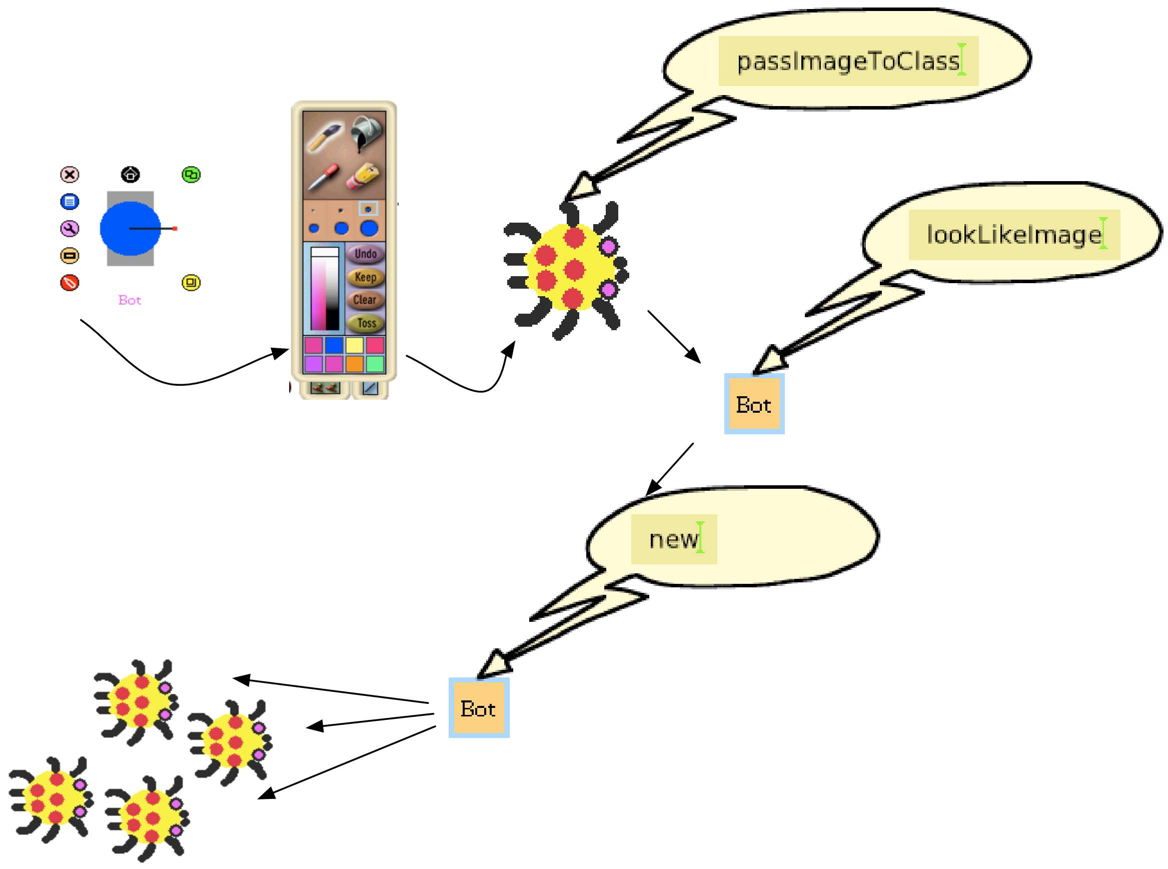


Figure 1.9: The steps to associate a new graphic to all the robots that will be created afterwards.

6.2 Using Scripts

You can also write scripts to load and save graphics, associating them to a single robot or to a class itself.

Script 1.6 creates a robot and load a new graphic. Note that you can also use the message loadImage: specifying the name of the file to load a specific graphic as shown by the last message which loads the graphic named spider. Note that when you use the message loadImage: asString you have to specify the name of the graphic with a string that is the name surrounded by single quotes. You can save the image using the methods save or saveImage: aString as shown in Script 1.6.

Script 1.6 (*Loading and saving a graphic to a single robot)*

```
| pica |
pica := Bot new.
pica loadImage.
"the method loadImage prompts the user for the name of the image to load"

pica loadImage: 'spider'
"the method loadImage: requires as parameter the name of the graphic

pica saveImage: 'spider2'
"you can save the image using the methods save or saveImage:"
```

Once you passed the graphics to the class, script 1.7 shows how to use the method saveImage: aString to save a graphic into a file named aString. This script creates a file named 'spider.frm' at the same location that your Squeak image. You can also use the method saveImage that will prompt you for a name.

Script 1.7 (*Saving the graphic associated with the class)*

```
Bot saveImage: 'spider2'
```

The following scripts (1.8, and 1.9) have the assumption that you have saved three images named luth, spider, and airplane. These files are included in the distribution of the environment used in this book. The expression Bot clearImage clears the previously defined graphic that could be associated with the class Bot so that you can reproduce exactly the same scenarios. Using this expression reset your environment as if you would just started it for the first time.

Script 1.8 (*Changing the image of one single robot)*

```
| pica daly |
Bot clearImage.

pica := Bot new.
pica lookLikeImage.
"No image loaded or created so nothing changes"

pica loadImage: 'luth'.
pica lookLikeImage
"load an image and ask to look like it"

pica loadImage: 'spider'.
"load another image and as it automatically looks like
the image the new image is shown"

daly := Bot new.
"look like a robot"
daly lookLikeImage.
"No image loaded in the class so nothing changes"
```

In script 1.8, first there is no graphic imported so asking the robot pica to look like an image does not produce any change. Once you load an graphic to pica sending the message loadImage: 'luth' where 'luth' represents the name of the graphic file, asking pica to look like an image produces the excepted effect. Now you control a robot that looks like a luth turtle. Note that if you read another image such as shown in the last line of the script, this image gets automatically used. When you create a new robot, daly, the class does not have any graphic loaded so daly looks like a robot and if you ask it to look like an image, nothing changes since it does not have an image.

script 1.9 shows how to notify the class that all newly created robots can have a new graphic. Note that contrary to the situation described in script 1.8 the message loadImage: aString is sent to the class Bot itself and not to a particular robot. The same message loadImage: has different behavior depending whether it is received by different objects here a class and an instance. On a class it loads and associates the graphic so that newly created instances can use the new graphics, on a robot only the robot receiving the message can use this graphic.

Script 1.9 (*Associating a graphic to the class)*

```
| bot1 bot2 bot3 |
Bot loadImage: 'spider'.
bot1 := Bot new.
bot1 lookLikeImage
"bot1 looks now like a spider"

bot2 := Bot new.
bot2 lookLikeImage.
"bot2 looks also now like a spider"

bot3 := Bot new.
bot3 loadImage: 'luth'.
bot3 lookLikeImage.
"But a specific robot can still change its own graphics"

"To get the image of the class back"
bot3 getImageFromClass.

Bot loadImage: 'luth'.
Bot lookLikeImage.
luth := Bot new.
"Now the class will create looking like luth turtles"
```

Script 1.9 starts by loading a new graphic from a file and associates it with the class itself. Then it creates a new robot bot1 and asks it to use the new graphic. Creating another robot bot2 and performing the same procedure creates another robot with the newly imported graphic.

All the robots created will be able to look like a spider. However a particular robot, such as the robot bot3 in the script which look like a luth, can still have its own graphic by loading one. The method getImageFromClass allows one to restore the graphic associated with the class. The last sequence of messages shows that we can associate a new graphic to a class. Sending the message loadImage: to the class Bot itself associates the graphic to the class, then sending the message lookLIkeImage makes sure that the newly created robots will look like the graphic of the class per default. Hence the robot luth looks like the turtle.

Summary

Method	Description	Example
lookLikeCircle	Change the shape of the receiver to a circle	Bot new lookLikeCircle
lookLikeBot	Change the shape of the receiver to a robot	Bot new lookLikeBot
lookLikeTriangle	Change the shape of the receiver to a triangle	Bot new lookLikeTriangle
lookLikeImage	Change the appearance of the receiver to the graphic you painted	Bot new lookLikeImage
lookLikeCircle	Send to the class makes sure that the created robots will have a circle as shape	Bot lookLikeCircle
lookLikeBot	Send to the class makes sure that the created robots will have a robot as shape	Bot lookLikeBot
lookLikeTriangle	Send to the class makes sure that the created robots will have a triangle as shape	Bot lookLikeTriangle
lookLikeImage	Send to the class makes sure that the created robots will have a the graphic you painted or loaded as shape	Bot lookLikeImage
loadImage: aString	Load the image named aString in the class or the robot	Bot loadImage: 'spider' or aBot loadImage: 'spider'
loadImage	Load the image named aString in the class or the robot	Bot loadImage or aBot load-Image
saveImage: aString	Save the image of the class or the robot in the file named aString	Bot saveImage: 'spider' or aBot saveImage: 'spider'
saveImage	Save the image of the class or the robot by prompting the user for a name	Bot saveImage or aBot saveImage
penColor: aColor	Change the color of the pen	Bot new penColor: Color blue
penSize: aNumber	Change the size of the pen. The default size is 1.	Bot new penSize: 3
color: aColor	Change the color of the receiver to the specified color	Bot new color: Color yellow
extent: aPoint	Change the size of the receiver. aPoint (w@h) specifies the new size of the receiver as the size of the rectangle. The first number is the width and the second the height.	Bot new extent: 80@100
passImageToClass	Pass the graphic of the receiver to the class. After this message, the robots created by the class will have as graphic the graphic that the current robot has.	aBot passImageToClass
getImageFromClass	Get the graphic of the class. After this message, the receiver will look like the robots that would be created by the class.	aBot getImageFromClass