Objects of class JFrame

Thus far, you have seen powerpoint slides that showed what an object is, how to call methods of an object, how to reference fields of an object, and how to create a new object using a new-expression. We now make these concepts more concrete using DrJava. You will see the creation of new objects, execution of procedure calls, and evaluation of function calls.

In Java, an object of class `javax.swing.JFrame` is associated with a window on your monitor. We will show you later how the rather long name `javax.swing.JFrame` can be abbreviated to simply `JFrame`. Class `JFrame` comes with every implementation of Java.

The upper right pane of DrJava, give a list of the methods that are in each object of class `JFrame`. What they do is indicated in the method names.

We begin by creating a new object of class `JFrame`, using a new-expression, and assigning it to a variable `x`:

```
x= new javax.swing.JFrame();
```

The upper right pane in DrJava shows variable `x` and the new object.

Calling procedures

An object of class `JFrame` has been created, but we can't see the window that is associated with it. So, we execute method

```
x.show();
```

There is the window, in the upper left corner of the screen! Calling method show of object `x` (or, rather, the object whose name is in `x`), caused the window to appear. Let's drag the window down a bit and make it bigger.

To show you the effect of other method calls, we hide the window and then show it again.

```
x.hide();
x.show();
```

We call one more procedure, to set the title of the window. The desired title is given as the argument of the procedure call:

There, the title now appears in the title bar of the window! Do you see how execution of a call of a procedure in an object has an immediate effect?

Calling functions

We call some functions. Methods `getX()` and `getY()` yield the position, in pixels of the upper-left pixel of the window.

```
x.getX()
x.getY()
```

Let's move the `JFrame` over to the left edge of the window and call `getX` again, so you can see that the value of the function call depends on the position of the window:

```
x.getX()
```

Functions `getWidth` and `getHeight` yields the width and height of the window, in pixels. We leave you to write calls on them later on.

Squaring the window

Procedure `setSize` changes the size of the window. The width becomes the first argument and the height becomes the second argument. Let's use `setSize` to square the window by making the width equal to the height. We first write the call, with no arguments, and then fill in the arguments. The first argument should be changed to the height, which is `x.getHeight()`, and the second argument should also be the height. There we are! Look how the window changed shape!

Objects of class JFrame

Each object has its own methods

We create a second object and store its name in a variable y and show the new window:

```
y=  new javax.swing.JFrame();
y.show();
```

There it is!

We now have *two* objects, which are associated with different windows. We can show you that each object has its own methods by setting the title of the second window to something other than the title in the first:

```
y.setTitle("truth");
```

See? The titles are different. We can also set the first title again:

```
x.setTitle(("love");
```

The point should be clear. Each object has its own methods, which manipulate the fields and other properties of that object in some way.

Aliasing

We make one more point. The upper right pane contains a picture of the two variables `x` and `y` and the two objects. Suppose we write an assignment to store `x` in `y`:

```
y= x;
```

How is this executed? According to the definition of the assignment, the value of the expression on the right, which is `a1`, is stored in variable `y`, so `y` also contains the name `a1`. Therefore, two variables contain the same object name. This is called *aliasing*. Aliasing happens often in object-oriented programming

We evaluate both `x.getTitle()` and `y.getTitle()` to show you that they contain the same value, since x and y contain the same name.

Object `a6` can no longer be referenced, because no variable contains its name. If you were executing this by yourself, drawing objects on a piece of paper, you could erase object `a6` because it can never be used again.