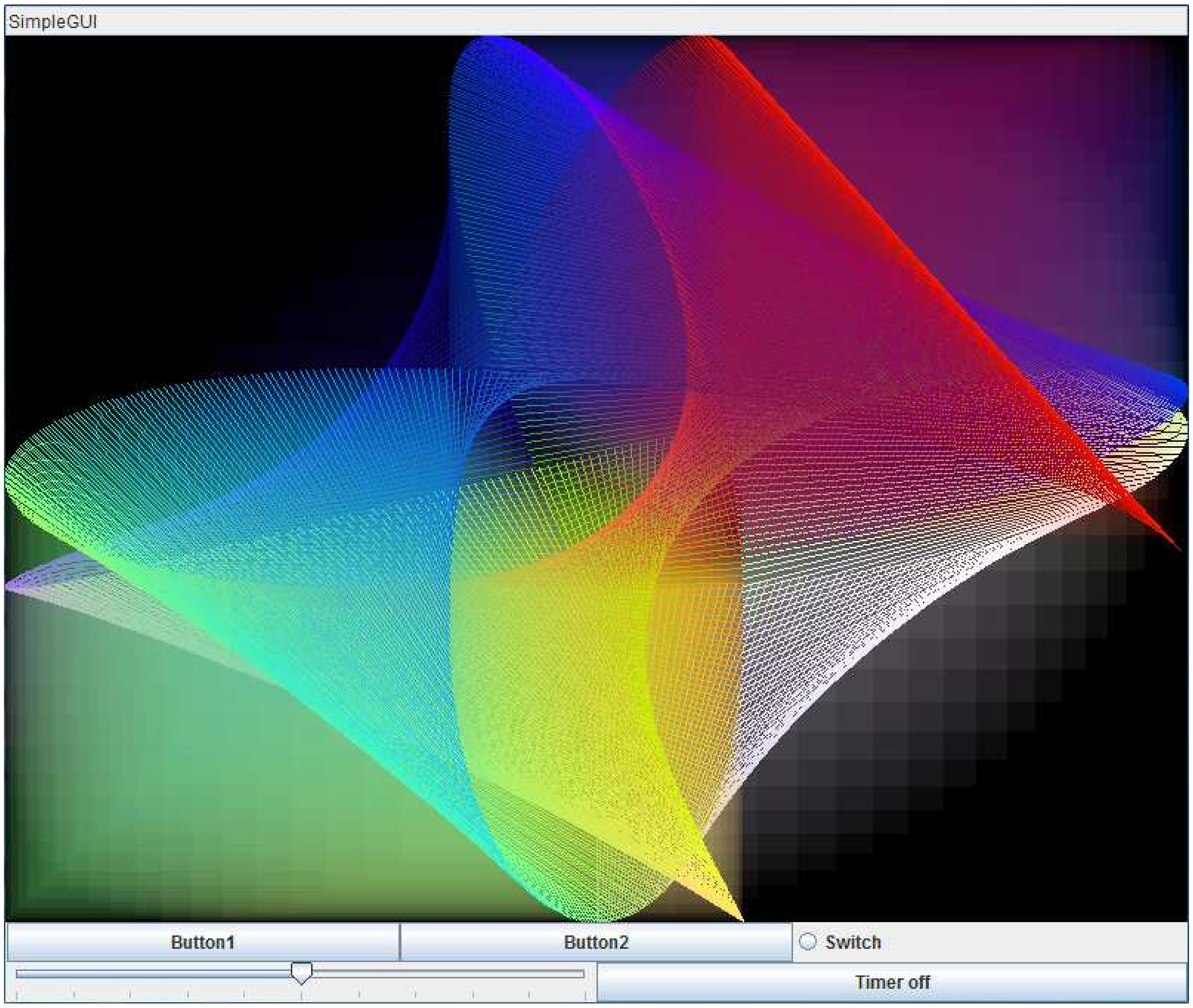
SimpleGUI

Simple access to JAVA graphics and GUI input

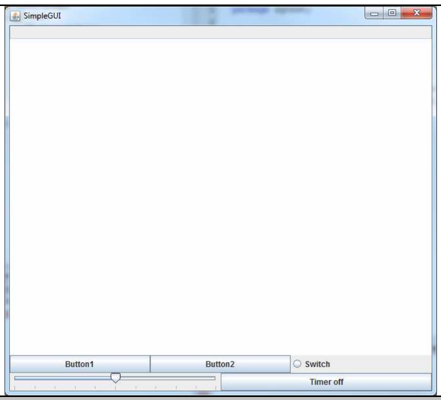


Rolf Lakaemper, Temple University, 2013
lakamper@temple.edu

Section 1: Introduction

What is SimpleGUI?

SimpleGUI offers easy access to basic drawing, GUI, mouse and keyboard interaction in JAVA. With a few lines of code, and without diving into the depths of advanced graphics programming (e.g. Swing, Java2D, JavaFX etc.), it allows programming beginners to participate in the excitement of graphical output, and to experience the advantages of using GUI elements in their own programs.



SimpleGUI

SimpleGUI features

- 2D drawing commands for graphic output
- Text output
- A customizable GUI element section
- Keyboard input
- Mouse input

SimpleGUI contains basic 2D drawing commands (boxes, ellipses, lines, text, images), which are directly and intuitively accessible with parameters like location, size, stroke width, color, transparency.

Using SimpleGUI, a single line of code opens a window featuring a drawing panel and a few customizable GUI elements. GUI elements, the keyboard and the mouse are accessible in techniques of sequential programming (i.e. beginners' style), as well as event based programming (advanced). With a little more effort, SimpleGUI even allows for animation, and low level image processing.

Who should use SimpleGUI?

SimpleGUI is designed as a graphical in-/output tool for JAVA beginners, and for those who need quick access to graphical output. It is designed to be used in beginners' education, to allow for exciting programming assignments, featuring graphical output to replace the standard console IO.

Remarks

This tutorial assumes the usage of the Netbeans IDE. Of course, using the SimpleGUI package is independent of any IDE. However, the step-by-step examples are tailored for Netbeans. If you are a JAVA beginner, and you need these examples, it will be of advantage to switch to Netbeans.

The jar file was compiled using JAVA 1.7. If you run into version problems, recompile the source.

For non-beginners, SimpleGUI should be straightforward and self-explanatory, this manual is mainly written for JAVA beginners.

Just add the jar file to your project and play around!

Demos

SimpleGUI comes with demonstration programs, illustrating the usage from simple drawing up to advanced image processing. The demos are available as source code (like all parts of SimpleGUI). They are also included in the *.jar file, which makes them directly accessible using a single line of code.

Starting Demos using the jar file

To start a demo program with name <name>, just insert

```
demo.<name>.main(null)
```

into your own code (after adding the jar file to your libraries, see section Installation/For Beginners). In Netbeans, typing "demos." + ctrl + space in your code will list all available demos.

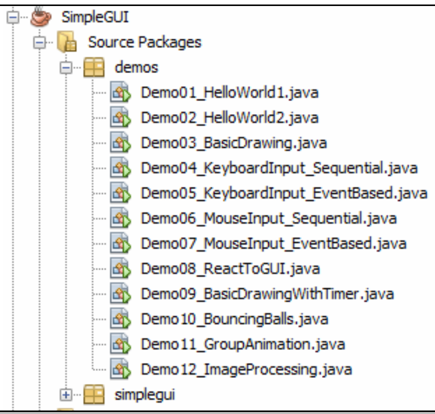
```
public class SGTest {
    // main
    public static void main(String[] args) {
        demos.Demo01_HelloWorld1.main(null); //this line start the demo
program!
    }
}
```

Calling a demo program, here "`Demo01_HelloWorld1`" from the jar file

Starting Demos from Source Files

The SimpleGUI sources come as a Netbeans project. Open the project in Netbeans, open the folder Source Packages in the Project view, open the package folder "demos", and there they are! Open any demo, and run it (shift + F6).

Starting the demos from source has the big advantage that you can single-step through them to see what's happening! You might also learn something looking at them.



```
SimpleGUI
  Source Packages
    demos
      Demo01_HelloWorld1.java
      Demo02_HelloWorld2.java
      Demo03_BasicDrawing.java
      Demo04_KeyboardInput_Sequential.java
      Demo05_KeyboardInput_EventBased.java
      Demo06_MouseInput_Sequential.java
      Demo07_MouseInput_EventBased.java
      Demo08_ReactToGUI.java
      Demo09_BasicDrawingWithTimer.java
      Demo10_BouncingBalls.java
      Demo11_GroupAnimation.java
      Demo12_ImageProcessing.java
    simplegui
```

List of Demos (may change with different versions)

Installation

For those who know already:

SimpleGUI comes as a single jar-file, there's nothing to install. The jar file contains two packages, "simplegui" and "demos". The first one contains all the classes you need for your own programs; this is the one you need to import.

The jar file was compiled using JAVA 1.7. If you run into version problems, recompile the source.

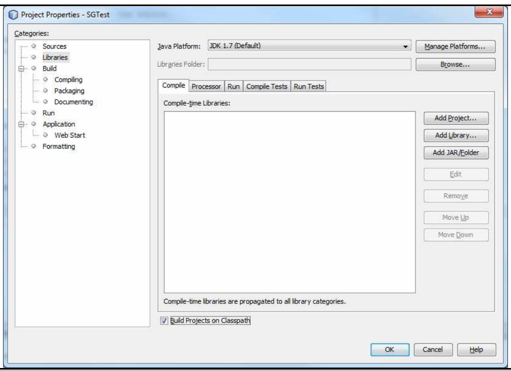
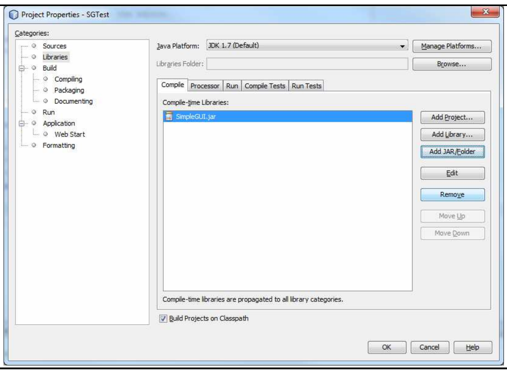
For beginners (how to use a jar-file):

SimpleGUI comes as a single jar-file. Strictly spoken there's nothing to install, except that you need to perform some steps to reveal the secrets hidden in a jar-file, i.e. to use the classes stored in it. These steps are the following:

1. download the file and store it in a useful place
2. tell the JAVA project the location you stored the file (=> add library)
3. in your own code, tell JAVA you want to use classes stored in the jar-file (=> import)

1. Download the file and store it in a useful place
You get the jar-file HERE. A useful place depends on the context you want to use the simplegui package. Let's assume the most frequent case, where you write your own project (e.g. a course assignment) and you want to use simplegui in it. In order to do so, create a Netbeans project, and name it "SGTest". This creates a folder "SGTest" in a default location (you find this location by right clicking on your project in Netbeans, and selecting *properties*. A window appears, somewhere in there you'll find *Project Folder:*, followed by the location). Copy the jar file into this folder.
Remark for the advanced: a nicer way to organize your libraries (jar files) is to first create a subfolder "libraries", and store the simplegui jar-file in there, amongst with all other jar-files (and libraries of other type, e.g. dll files) your project needs.

2. Tell the JAVA project the location you stored the jar-file
Just storing the jar-file in the project folder does not tell the project that it exists. You need to do so explicitly: right-click on your SGTest project, click on *properties*. A window appears. In the *Categories* section, click on *libraries*. The following window appears (left):

Adding a jar file to the project (Netbeans). Left: before the jar file was added. Right: the file is added and can be used in the project.

- Click on "add JAR/Folder"
- Select the file "SimpleGUI" from the location where you just stored it.
- Check "Relative Path" on the right side.
- Click "Open", and you are done.

You should now see the jar-file added to your project, as in Figure 1, right.

3. In your own code, tell JAVA you want to use classes stored in the jar-file, using "import simplegui.*"

The jar file was compiled using JAVA 1.7. If you run into version problems, recompile the source or step up to an appropriate JAVA version for your project.

Initial Test

Demo Programs Related to This Section

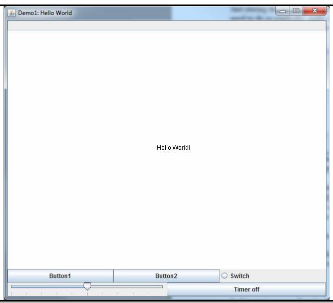
- Demo1_HelloWorld1

For those who know already:

Add `demos.Demo1_HelloWorld1.main(null);`
to your code and run it.

For beginners:

In this section, we will write the most basic code to test if the jar-file is accessible. It's the "Hello World" of SimpleGUI.
We assume, you have created a (Netbeans) project *SGTest*, and added the jar-file to it. We want to create the "Hello World Window" (right).



First steps, "Hello World"

The steps to do so:

- create a class in your SGTest project, e.g. class "SGTest"
- to tell your class that you want to use classes from the package "simplegui", you need to import it. Type "import simplegui.*" where it belongs in your class. The jar-file you added to your project before contains the package simplegui, which in turn contains all the classes (".*") you want to use.
- in the *public static void main(String []args)* method, add the lines:
```
SimpleGUI sg = new SimpleGUI();
sg.setTitle("Demo1: Hello World");
sg.drawText("Hello World!", 300, 240);
```
- run the project (Figure 3 shows the program), and a window as in Figure 2 should appear.

```
package sgtest;
import simplegui.*;

public class SGTest {
    public static void main(String[] args) {
        SimpleGUI sg = new SimpleGUI();
        sg.setTitle("Demo1: Hello World");
        sg.drawText("Hello World!", 300, 240);

    }
}
```

The Hello World program.

→ If it worked: congratulations, we can start!
→ If it didn't: go to the beginner's section above. If it still does not work, ask someone (TA, friend...). At this point, do NOT send me an email yet.

Section 2: Basic Drawing

This section will explain the usage of basic SimpleGUI features. If you just want to draw, this section is all you need.

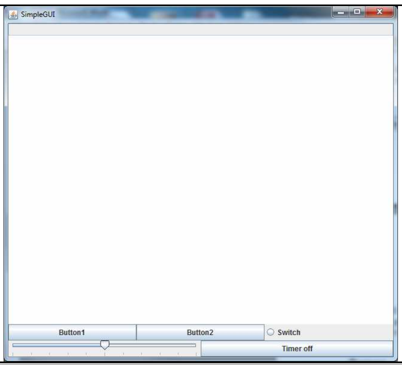
Demo Programs Related to This Section

- Demo2_HelloWorld2
- Demo3_BasicDrawing

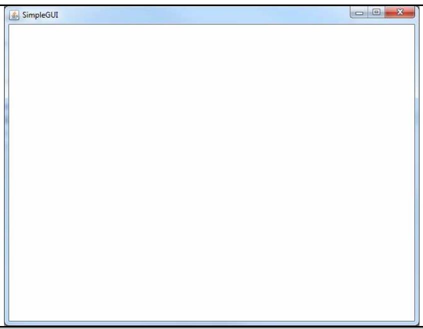
The SimpleGUI class

Creating a Window

The core class that represents the GUI and provides all drawing functionality is the class *SimpleGUI*. To make the GUI window appear, you just need to instantiate an object of this class; it's as simple as that!

`SimpleGUI sg = new SimpleGUI();`	
Instantiating an object of SimpleGUI...	...makes the GUI appear!

The GUI consists of two areas, the drawing area, and the GUI input area with buttons etc. If you only want to draw, you might want to hide the buttons:

`SimpleGUI sg = new SimpleGUI(false);`	
Creating a GUI with drawing area only.	

You can also change the size of the drawing area at creation time.

The default size of the drawing area is 640x480.

There are four constructors for a SimpleGUI object, they should be self explanatory:

Constructors for SimpleGUI
SimpleGUI()
SimpleGUI(boolean createGUIElements)
SimpleGUI(int width, int height)
SimpleGUI(int width, int height, boolean createGUIElements)

Basic Drawing
The SimpleGUI object provides all methods to draw into the drawing area. Example methods are:

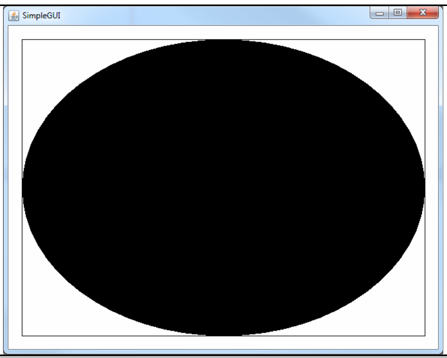
Example Drawing Methods of SimpleGUI	
drawBox	...draws a box (rectangle) as a non filled frame
drawDot	...plots a dot
drawEllipse	...draws an ellipse, non filled
drawFilledBox	...filled version of drawBox
drawFilledEllipse	...filled version of drawEllipse
drawImage	...draws a (photographic) image, e.g. a jpg-file
drawLine	...draws a line
drawText	...writes text

Each of these methods come with different parameters, determining
- location
- size
- color
- <other parameters>
- name (this will be explained later)

of the elements to draw. Location (and size, if applicable) must always be determined, while the remaining parameters usually have default values, which can be used for convenience.

For example, to draw an empty box with a filled ellipse inside, in default color, just type:

```
SimpleGUI sg = new SimpleGUI(false);
sg.drawBox(20,20,600,440);
sg.drawFilledEllipse(20,20,600,440);
```



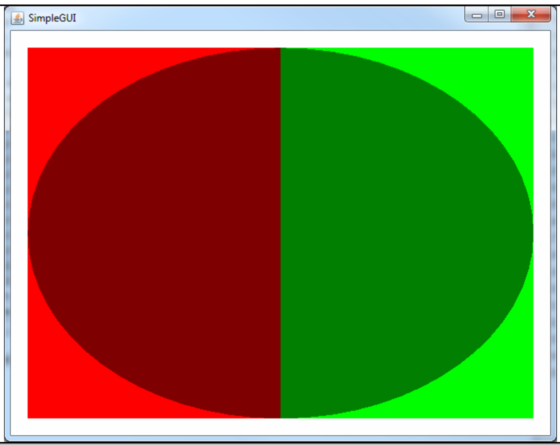
Drawing an empty box with a filled ellipse inside

Boxes and Ellipses are specified at least by location (x,y) and size (width, height). How to draw other elements, please see section "Drawables".

Color and Transparency
It is possible to draw in different colors and even transparency. In order to do so, either the default color must be changed, or the desired color must be specified directly as a parameter. In both cases, a *Color*-object must be specified (java.awt.Color). This can be either done by pre-defined color names (e.g. "Color.red"), or by so called RGB-values RGB = Red Green Blue (e.g. new Color(130,255,34)). Transparency is a double value between 0.0 (fully transparent) and 1.0 (opaque).

The following example illustrates how to use color and transparency in the method call:

```
SimpleGUI sg = new SimpleGUI(false);
sg.drawFilledBox(20, 20, 300, 440, Color.red, 1.0, null);
sg.drawFilledBox(320, 20, 300, 440, Color.green, 1.0, null);
sg.drawFilledEllipse(20,20,600,440, Color.black, 0.5, null);
```



Two colored filled boxes with a transparent black filled ellipse on top.

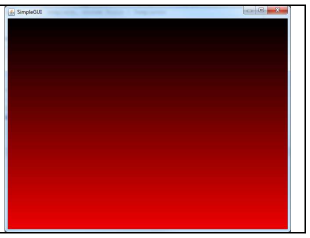
To set different *default* color and transparency values, use the method *setColorAndTransparency(Color c, double t).*

RGB Colors
Colors other than the predefined, named colors in the class java.awt.Color can be composed using the red-green-blue (rgb) color model. It uses additive color mixing to generate colors. Please read details at http://en.wikipedia.org/wiki/RGB_color_model .
For the JAVA Color class, colors are defined by three integer values between 0...255, defining the intensity of red, green and blue:

```
        Color c = new Color(r,g,b); // r,g,b are integers in 0..255
```

The following example draws 48 horizontal stripes with increasing red intensity:

```
SimpleGUI sg = new SimpleGUI(false);
for (int i = 0; i<48; i++){
   Color c = new Color(i*5,0,0);   //red between 0 and 240
   sg.drawFilledBox(0, i*10, 640, 10, c, 1.0, null);
}
```



Drawables
We call each element that we can add to the drawing area a "drawable", i.e. boxes, lines ellipses etc. are drawables. They differ in shape and parameters. The following table contains all drawables along with the methods to generate them. Handling of drawables is straightforward. Here are some additional notes:

- All methods are of return type *void*.
- Non fully parameterized methods use default values.
- Each drawable has a simple convenience, non fully parameterized version, and a completely parameterized version.
- Line based drawables (box, ellipse, line) can be given a different thickness, using the "strokeWidth" parameter.
- Images can be drawn using their filename. If not a full path is given, the parent folder is the project folder (if started from Netbeans).
- Images can be resized.
- drawables are placed into the drawing area in the order they are created!
- The (String) "name" parameter will be explained in the advanced section. For basic drawing it can be set to "null"
- For detailed method descriptions please see the javadoc that comes with the project

Note: The origin (0,0) of the drawing area is, in contrast to coordinate systems you know from math, in the TOP LEFT corner!

Box	**drawBox**(double x, double y, double width, double height) **drawBox**(double x, double y, double width, double height, java.awt.Color c, double transparency, int strokeWidth, java.lang.String name)
Dot	**drawDot**(double x, double y, double radius) **drawDot**(double x, double y, double radius, java.awt.Color c, double transparency, java.lang.String name)
Ellipse	**drawEllipse**(double x, double y, double width, double height) **drawEllipse**(double x, double y, double width, double height, java.awt.Color c, double transparency, int strokeWidth, java.lang.String name)
FilledBox	**drawFilledBox**(double x, double y, double width, double height) **drawFilledBox**(double x, double y, double width, double height, java.awt.Color c, double transparency, java.lang.String name)
FilledEllipse	**drawFilledEllipse**(double x, double y, double width, double height) **drawFilledEllipse**(double x, double y, double width, double height, java.awt.Color c, double transparency, java.lang.String name)
Image	**drawImage**(java.lang.String filename, double x, double y) **drawImage**(java.lang.String filename, double x, double y, double width, double height, java.lang.String name) **drawImage** (DrwImage image, double x, double y, double width, double height, java.lang.String name) //**[advanced method, see section Image processing!]**
Line	**drawLine**(double xStart, double yStart, double xEnd, double yEnd) **drawLine**(double xStart, double yStart, double xEnd, double yEnd, java.awt.Color c, double transparency, int strokeWidth, java.lang.String name)
Text	**drawText**(java.lang.String text, double x, double y) **drawText**(java.lang.String text, double x, double y, java.awt.Color c, double transparency, java.lang.String name)

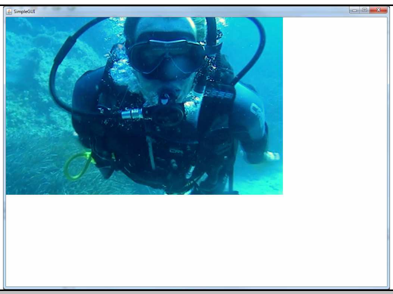
Additional Methods
A few additional methods can be used to change the appearance of the window and drawing panel:

centerGUIonScreen(), maximizeGUIonScreen(), setLocationOnScreen(int x, int y)	Window sizing and positioning
eraseAllDrawables()	Clear drawing area
getHeight(), getWidth()	Get dimension of drawing area
setBackgroundColor(Color c)	Define default background color

Drawing Images

You can add (photographic) images to the drawing panel just as easy as you draw a box or ellipse! You just need the filename of the image.

```
import simplegui.SimpleGUI;
public class SGTest {
    // main
    public static void main(String[] args) {
        SimpleGUI sg = new SimpleGUI(false);
        sg.drawImage("test.jpg", 0,0);
    }
}
```

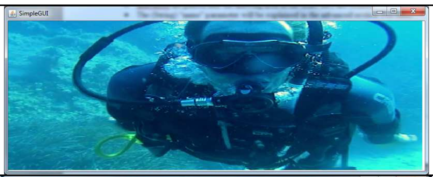


Basic Image Printing

The fully parameterized image drawing method allows you to rescale the image, adding the target width and height. *If you set width and height to -1, the image will automatically rescale to the drawing panel dimensions.*

Rescaled to 800,300:

```
sg.drawImage("test.jpg",
0,0,800,300,null);
```



Rescaled to drawing panel dimensions:

```
sg.drawImage("test.jpg",
0,0,-1,-1,null);
```

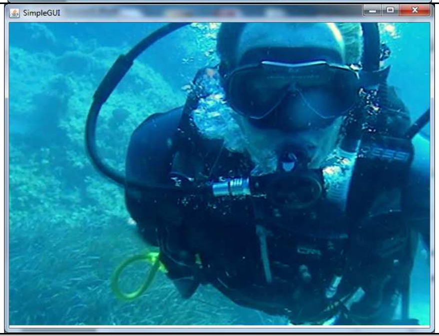


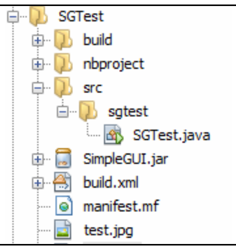
Image Printing with Rescaling

Hint: Images are just normal drawables. You can mix them with lines, ellipses etc. It can be nice to use an image as a background! Please have in mind, that drawables are printed in the order you create them; hence drawing an image first makes it the background. If you draw on images with transparent drawables, you can create really nice images/animations.

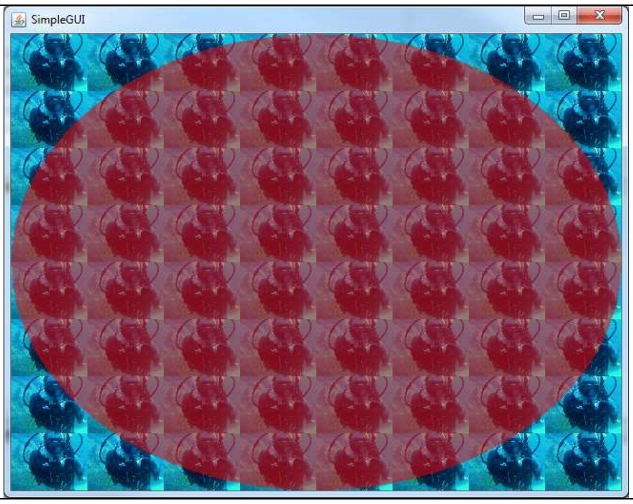
*Permitted file formats are *.jpg, *.bmp, *.png, *.gif (png images can even contain transparent layers)*

Where should I store my image?
In Netbeans, the root folder (i.e. the start for each file-search java performs) is the project folder. If you store your image there, you just need the image name for java to find it. You can either do so by drag and drop into Netbeans, or by copying the file manually. You find

the location of your project, if you right-click on your project in Netbeans, choose properties, and the project location will appear. However, it is nicer to create your own folder "images" inside the project folder, and store your images there. In this case, you need to address the images using "images/<filename>".

 SGTest build nbproject src sgtest SGTest.java SimpleGUI.jar build.xml manifest.mf test.jpg	SGTest Project with test.jpg image file in root folder (please note that SimpleGUI.jar is also stored in the same folder)

To conclude, here is a little program that uses an image as background tiles, and draws a transparent ellipse on top (you will see that this program builds up the screen relatively slowly. This can be changed by using the advanced drawing method **`drawImage`** `(DrwImage image, double x, double y, double width, double height, java.lang.String name),` which will be described in the advanced section`).`



```
import java.awt.Color;
import simplegui.SimpleGUI;
public class SGTest {
    // main
    public static void main(String[] args) {
        SimpleGUI sg = new SimpleGUI(false);
        for (int r = 0; r<480; r+=60){
            for (int c=0; c<640; c+=80){
                sg.drawImage("test.jpg", c,r,80,60,"null");
            }
        }
        sg.drawFilledEllipse(0, 0, 640, 480,Color.red,0.5,"null");
    }
}
```

Section 3: Keyboard Input

Demo Programs Related to This Section

- Demo04_KeyboardInput_Sequential
- Demo05_KeyboardInput_EventBased

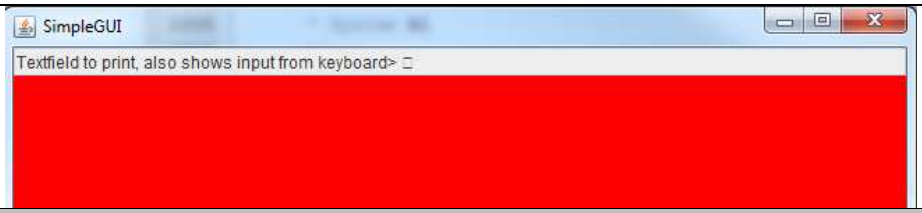
For Beginners: Sequential Programming Style

SimpleGUI makes it very simple to read the keyboard. The following routines wait for keyboard input, in sequential programming style (beginners: just use these routines):

String keyReadString()	//waits for an <enter> terminated input, returns a String
char keyReadChar()	// waits for a single key, returns char
double keyReadDouble()	// waits for <enter> terminated input of a double-format number

Sequential Keyboard Input Methods

All key-input is visible in the textfield above the drawing panel.



The textfield for keyboard input is located above the drawing panel.

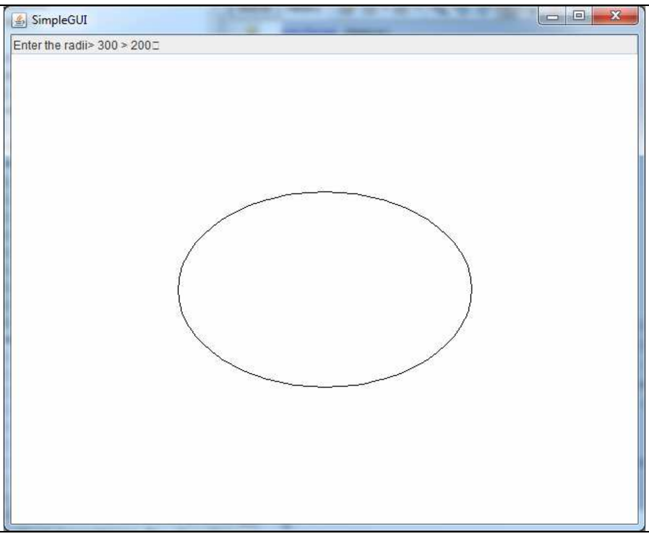
A boolean parameter can be passed to the input methods to erase previously printed text:

- `keyReadString(boolean eraseText),`
- `keyReadChar(boolean eraseText),`
- `keyReadDouble(boolean eraseText))`

Note:

- Text can be printed into the textfield using the method `sg.print(String text).`
- The textfield can be cleared using `sg.print("");`
- SimpleGUI reacts to keyboard input whenever it is in focus, i.e. the main window. There is no need to click the textfield.

The following example reads two double numbers, which are used as radii for an ellipse (available in package demos):



Enter the radii> 300 > 200

```
import simplegui.SimpleGUI;
public class Demo_KeyboardInput_Sequential {
    public static void main(String []args) {
        SimpleGUI sg = new SimpleGUI(false);
        while (true) {
            sg.print("Enter the radii");
            // read radii
            double radius1 = sg.keyReadDouble();
            double radius2 = sg.keyReadDouble();
            // compute top left corner of bounding box
            double startX = (sg.getWidth() - radius1) / 2;
            double startY = (sg.getHeight() - radius2) / 2;
            // draw
            sg.eraseAllDrawables();
            sg.drawEllipse(startX, startY, (int)radius1, (int)radius2);
        }
    }
}
```

Keyboard input demo: defining radii of an ellipse

Advanced: Event Based Keyboard Input

In this mode, the keyboard notifies the program when a key is pressed/a line is entered. The paradigm behind this programming style is called "event based programming". Tutorials on event based programming are available online. The event/callback mechanism is implemented straightforward: In order to react to the keyboard, the code

1. has to implement the Interface "KeyboardListener". This guarantees the existence of certain methods in your class. These are the methods which are used by a background process started by the SimpleGUI object to notify you of key events. These methods are called "callback methods".
2. has to register to the SimpleGUI using "registerToKeyboard(KeyListener kl)". Registration tells the SimpleGUI object, that your class wants to be notified about keyboard events.

The Interface KeyboardListener

The interface declares 2 callback methods:

void reactToKeyboardEntry(String input) void reactToKeyboardSingleKey(String input)
Event based keyboard methods

After registering a KeyboardListener to the SimpleGUI (using registerToKeyboard(KeyboardListener kl)), these methods are called when a key is typed. There is no specific method to read numbers. Please see the following demo how to change a String to a double.

The following demo program illustrates the usage of event based keyboard input. It is the event based version of the previous demo, drawing an ellipse with user defined radii. Please note:

- The class implements KeyboardListener, i.e. it declares to do so in the class declaration (`public class SGTest implements GUIListener`), and implements the callback methods defined in the interface.
- All interface methods must be implemented. If they are not needed, they should be implemented empty "`{ }`" (like `reactToSingleKey`).
- The object SGTest is registered to the SimpleGUI object (`sg.registerToKeyboad(kl)`). In order to do register, there must be an instance of SGTest, i.e. it is not possible anymore to just have a static main method without objects.

The demo program is also available in the package "demos".

```
import simplegui.KeyboardListener;
import simplegui.SimpleGUI;

public class Demo5_KeyboardInput_EventBased implements KeyboardListener {

    int radiusID; // determines which radius to change
    double radius1, radius2;
    private static SimpleGUI sg;

    public Demo5_KeyboardInput_EventBased() {
        radius1 = radius2 = 10.0;
        radiusID = 0;
    }

    @Override
    public void reactToKeyboardEntry(String input) {
        try {
            // convert string to double
            // (this might throw a NumberException format. That's why
            // this block is surrounded by try/catch)
            double radius = (new Double(input)).doubleValue();
            if (radiusID==0){
                radius1 = radius;
            } else {
                radius2 = radius;
            }
            // toggle radii
            radiusID = (radiusID+1)%2;
            sg.print("Radius "+(radiusID+1)+" >");
            // compute top left corner of bounding box
            double startX = (sg.getWidth() - radius1) / 2;
            double startY = (sg.getHeight() - radius2) / 2;
            // draw
            sg.eraseAllDrawables();
            sg.drawEllipse(startX, startY, (int)radius1, (int)radius2);
        }catch(NumberFormatException e){
            sg.print("Wrong format. Radius "+(radiusID+1)+" >");
        }
    }

    @Override
    public void reactToKeyboardSingleKey(String input) {
    }

    // ------------------------------------
    public static void main(String[] args) {
        sg = new SimpleGUI(false);
        Demo5_KeyboardInput_EventBased demo = new
Demo5_KeyboardInput_EventBased();
        sg.registerToKeyboard(demo);
        sg.print("Enter radii> ");
    }

}
```

Keyboard input demo: defining radii of an ellipse (Event Based)

Section 4: Mouse Input

Mouse input reacts to clicks in the drawing panel. SimpleGUI detects when and where (in the drawing panel) the mouse was clicked.

Demo Programs Related to This Section

- Demo06_MouseInput_Sequential
- Demo07_MouseInput_EventBased

For Beginners: Sequential Programming Style

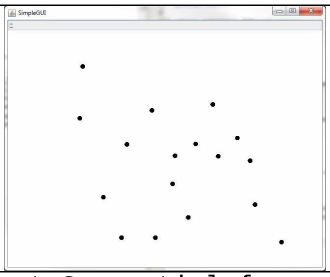
Reacting to mouse clicks in sequential programming is straightforward. There is exactly one method,

```
public int[] waitForMouseClick()
```

Reading the mouse, sequential version

The method waits until a mouse button was pressed (it does not wait for the release), and returns an int-array containing the x and y mouse position (remember: top left is (0,0)).

The following demo, which is also available in the demo package, draw dots wherever the mouse was clicked.



```
public class Demo06_MouseInput_Sequential {
    public static void main(String []args){
        SimpleGUI sg = new SimpleGUI(false);
        while(true){
            int[]xy = sg.waitForMouseClick();
            sg.drawDot(xy[0], xy[1], 5);
        }
    }
}
```

Mouse Demo, Sequential Version

Note: sequential mouse programming is for beginners only. It is strongly advised to use the event based version!

Advanced: Event Based Mouse Input

Event based mouse input is very similar to event based keyboard input or event based GUI handling. Again, in this mode, the mouse notifies the program when a mouse-button is pressed. The paradigm behind this programming style is called "event based programming". Tutorials on event based programming are available online. The event/callback mechanism is implemented straightforward: In order to react to the mouse, the code

1. has to implement the Interface "SGMouseListener". This guarantees the existence of certain methods in your class. These are the methods which are used by a background process started by the SimpleGUI object to notify you of mouse events. These methods are called "callback methods".
2. has to register to the SimpleGUI using "registerToMouse(SGMouseListener ml)". Registration tells the SimpleGUI object, that your class wants to be notified about mouse events.

The Interface SGMouseListener

Note: the interface is not just named "MouseListener" to not confuse it with the java.awt.event.MouseListener interface.

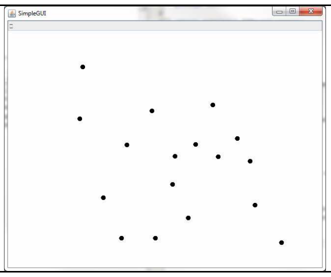
The interface declares only 1 callback method:

void reactToMouseClick(int x, int y)

Event based mouse method

After registering an SGMouseListener to the SimpleGUI, this method is called when a mouse button is pressed.

The following demo program illustrates the usage of event based mouse input. Like the sequential version, it draws dots where the mouse is clicked. The demo program is also available in the package "demos".



```
public class Demo07_MouseInput_EventBased implements SGMouseListener {

    SimpleGUI sg;

    public Demo07_MouseInput_EventBased() {
        sg = new SimpleGUI(false);
        sg.registerToMouse(this);
    }

    @Override
    public void reactToMouseClick(int x, int y) {
        sg.drawDot(x, y, 5);
    }

    public static void main(String args[]) {
        new Demo07_MouseInput_EventBased();
    }
}
```

Mouse Demo, Event Based Version

Section 5: GUI Elements, Interaction

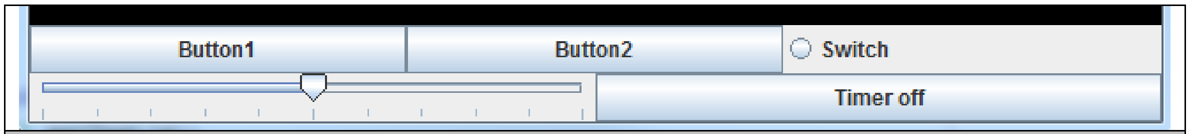
Demo Programs Related to This Section

- Demo08_ReactToGUI
- Demo09_BasicDrawingWithTimer (timer)
- Demo10_BouncingBalls (timer)

SimpleGUI allows for simple, customized interaction. SimpeGUI supports sequential-programming type interaction (synchronous methods) and event-driven interaction (asynchronous methods). SimpleGUI offers as customizable elements (see below):

- two buttons
- a switch (radio button)
- a slider

(An additional non customizable button is dedicated to control a timer, see section "Using the Timer")



The GUI element section of SimpleGUI

Beginner's GUI Interaction (Sequential Programming)

In this section, the most simple (and also restricted) form of interaction will be explained. In this form, the GUI elements are queried directly in the user's code ("read slider value", "wait for button"). Changes in the GUI elements are therefore only detected when the code checks for it. Depending on the application, this drawback is often balanced with the simplicity of the code.

The GUI elements can be accessed with the following methods:

`void waitForButton1()`	Waits until button 1 is clicked*1
`void waitForButton2()`	Waits until button 2 is clicked*1
`boolean getSwitchValue()`	Returns true if the radio button is selected/activ
`int getSliderValue()`	Returns a value in 0...100*2
`void print(String s)`	Print a string in *textfield*. This does NOT print on the panel. To print on the panel, use drawText()

(Synchronous) GUI methods

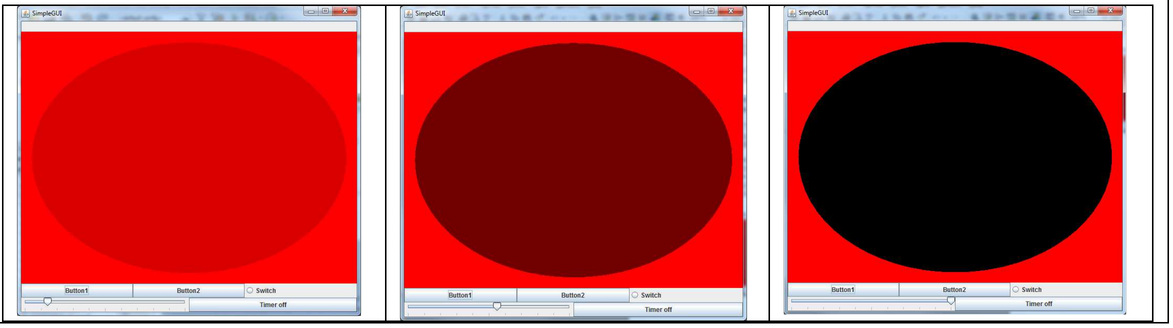
**1: waiting for a button is not a suggested programming style. If you need buttons, please consider to advance to event based programming!*

**2: Transforming the Slider Value to a Different Interval*
The method "getSliderValue" always returns a value in the interval 0..100. Often, this value needs to be transformed to a different interval. As an example, let's assume the slider should determine an x-position between min = 50 and max = 320, i.e. slider value 0 must be mapped to 50, slider value 100 must be mapped to 320, intermediate values linearly in between.
The formula to achieve this is: *transformedValue = sliderValue*(max-min)/100.0 + min*

Example Program

The example program changes the transparency of a filled ellipse based on the slider value. The change is performed each time button1 is pressed.

```
SimpleGUI sg = new SimpleGUI();
sg.setBackgroundColor(Color.red);
sg.drawFilledEllipse(20,20,600,440, Color.black, 1.0, null);
while(true){
  sg.waitForButton1();
  int sliderValue = sg.getSliderValue();
  double transparency = (double)sliderValue/100.0;
  sg.eraseAllDrawables();
  sg.drawFilledEllipse(20,20,600,440, Color.black, transparency, null);
}
```



Custom Button Labeling

The button labels can be changed using the following methods:

`labelButton1(String s)`
`labelButton2(String s)`
`labelSwitch(String s)`

The appearance of the slider can not be changed.

Advanced GUI Interaction (Event Based)

In this mode, the GUI elements can notify the program when they are activated/changed. The paradigm behind this programming style is called "event based programming". Tutorials on event based programming are available online. *It is strongly suggested to use event based programming when interacting with GUI elements*. The event/callback mechanism is implemented straightforward: In order to react to GUI elements, the code

3. has to implement the Interface "GUIListener". This guarantees the existence of certain methods in your class. These are the methods which are used by a background process started by the SimpleGUI object to notify you of changes in the GUI elements. They are called "callback methods".
4. has to register to the SimpleGUI using "registerToGUI". Registration means to tell the SimpleGUI object, that your class wants to be notified about changes.

The Interface GUIListener

The interface declares 4 callback methods:

- reactToButton1()
- reactToButton2()
- reactToSwitch(boolean switchValue)
- reactToSlider(int sliderValue)

After registering a GUIListener to the SimpleGUI (using registerToGUI(GUIListener gl)), these methods are called on the respective GUI interaction.

Example Program

The following demo program will illustrate the usage of button 1 and 2. The program reacts to button 1 by drawing an ellipse on the left window side (position 20,20), whereas button 2 triggers the program to draw an ellipse on the right side (position 320,20). Please note the following important program features:

- SGTest implement GUIListener, i.e. it declares to do so in the class declaration (`public class SGTest implements GUIListener`), and implements the four callback methods defined in the interface GUIListener.
- Methods relating to unused GUI elements (slider, switch) must be implemented (with an empty body).
- On instantiation, the object SGTest registers to the SimpleGUI object (`sg.registerToGUI(this)`). In order to do register, there must be an instance of SGTest, i.e. it is not possible anymore to just have a static main method without objects. For the very beginners amongst you, this is the irreversible good bye to non-object-oriented programming. Welcome to OOP.

```
import simplegui.*;

public class SGTest implements GUIListener {
    private SimpleGUI sg;

    // main ------------------------------------
    public static void main(String[] args) {
        SGTest test = new SGTest();
    }

    // Constructor -----------------------------
    SGTest(){
        sg = new SimpleGUI();
        sg.registerToGUI(this);
    }

    // Callback methods ------------------------
    @Override
    public void reactToButton1() {
        sg.eraseAllDrawables();
        sg.drawFilledEllipse(20,20,300,440);
    }

    @Override
    public void reactToButton2() {
        sg.eraseAllDrawables();
        sg.drawFilledEllipse(320,20,300,440);
    }

    @Override
    public void reactToSwitch(boolean bln) {
         // not used, implement empty!
    }

    @Override
    public void reactToSlider(int i) {
         // not used, implement empty!
    }
}
```

Demo program showing ellipses at different positions, based on which button is pressed.

Using the Timer

The timer fires timer-events with a given frequency (default: 1000ms). It can be used to repeatedly trigger certain actions, e.g. to show the time, or for any other animation. In sequential programming, you would program a loop with some kind of sleep or pause command inside. This has huge disadvantages if you have multiple tasks in parallel, some of them not belonging into the loop (or even belonging into another timed loop). A timer is a convenient solution to such a situation. Think of it as an alarm clock, that goes off every so often (e.g. every second) to remind you to do something. You can also think it's an invisible button, that is automatically pressed every second.

Timer usage is, again, event based programming. Hence, to use the timer, the code

1. has to implement the Interface "TimerListener"
2. has to register to the SimpleGUI using "registerToTimer".

These are exactly the same steps that were needed for event based reaction to the GUI elements (see above), yet this time we react to different events.

The interface TimerListener declares only one method, "reactToTimer(long time)". Whenever the timer goes off, it calls this method in every class that is registered. The parameter "long time" is the system time in milliseconds when the timer went off.

The following demo program will move a box from left to right over the screen. Please note the following features:

- the timer pause time is set to 20ms (that's 50 frames per second) in the timerStart(int time) method call
- the timer needs to be started explicitly using the timerStart() method!
- the GUI timer-button can pause the timer, causing the box to stop moving
- this is the simplest form of animation. It might cause some flickering. For mode advanced animation, go to section "Animation".
- most of the code is in the callback method. That's typical for event based programming.
- the position (positionX variable) is computed based on the system time (l % sg.getWidth()). "%" is the module operator, i.e. the remainder of an integer division. Setting the position this way guarantees the same speed of the box, independent of the pause time (try it!)

```
package sgtest;
import simplegui.*;

public class SGTest implements TimerListener {
    private SimpleGUI sg;

    // main
    public static void main(String[] args) {
        SGTest test = new SGTest();
    }

    // Constructor
    SGTest(){
        sg = new SimpleGUI();
        sg.registerToTimer(this);
        sg.timerStart(20);  //important! Timer needs to be started!
    }

    // Callback method
    @Override
    public void reactToTimer(long l) {
        sg.eraseAllDrawables();
        int positionX = (int)(l%sg.getWidth());
        sg.drawFilledBox(positionX, sg.getHeight()/2, 30, 30);
    }
}
```

Timer Example: moving a box across the screen.

SimpleGUI contains the following **timer related methods:**

registerToTimer(TimerListener l)
removeFromTimer(TimerListener l)
	Registration

timerStart()
timerStart(long pauseTime)
	Start

timerPause()	Pause

Timer Methods

Section 6: Advanced Topics

Animation

Demo Programs Related to This Section

- Demo11_GroupAnimation

Moving Drawables
SimpleGUI has built in methods to perform basic animation, i.e. methods to relocate drawables. There are no methods to change the appearance of drawables with respect to other parameters (size, color, shape etc.).
In general, the technique used in (computer) graphics to make elements seemingly move is to redraw them at a different, sufficiently near position with a sufficiently high frequency (>20 frames per second). Hence, the animation methods in SimpleGUI can address select drawables and change their location parameter. This, in combination with timer usage, creates the effect of movement.
Drawables are selected by their name, i.e. the string passed as *name-parameter* in the fully parameterized drawing methods. Selection is performed by the following rules

- If a single drawable should be selected, the first (in drawing order) drawable with fitting name is selected
- if multiple drawables (group animation) should be selected, all drawables with fitting name are selected.

There are four animation related methods:

animMoveTo(int x, int y, String name)	Move single drawable, absolute
animMoveRel(int x, int y, String name)	Move single drawable, relative
animMoveAllRel(int x, int y, String name)	Move drawable group, absolute
repaintPanel()	Repaint drawables at new locations

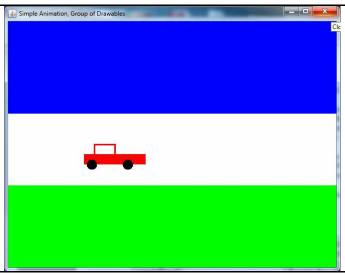
Animation Methods

Single drawables can be relocated to an absolute position, or relative to their current position. A group of drawables can, naturally, only be moved relative to their current position.

Important: the "animXXXX" methods only change the location parameter of drawables, other than the drawXXX methods, they do not refresh the drawpanel. This has to be done explicitly using the repaintPanel() method. Otherwise, no effect or strange effects will occur!

Not refreshing the drawing panel avoids flickering effects when moving multiple elements.

The following example program moves a (simple) car across the screen (also available in the demo package as demo GroupAnimation):



```
import java.awt.Color;
import simplegui.AbstractDrawable;
import simplegui.SimpleGUI;
import simplegui.TimerListener;

public class GroupAnimation implements TimerListener {

    SimpleGUI sg;

    public GroupAnimation() {
        sg = new SimpleGUI(640,480,false);
        sg.setTitle("Simple Animation, Group of Drawables");
        createBackground();
        createCar();
        sg.registerToTimer(this);
        sg.timerStart(50);
    }

    private void createBackground() {
        sg.drawFilledBox(0, 0, 640, 180, Color.blue, 1.0, "");
        sg.drawFilledBox(0, 320, 640, 160, Color.green, 1.0, "");

    }

    private void createCar() {
        sg.drawBox(20, 240, 40, 20, Color.red, 1, 3, "car");
        sg.drawFilledBox(0, 259, 120, 20, Color.red, 1, "car");
        sg.drawDot(15, 279, 10, Color.black, 1, "car");
        sg.drawDot(85, 279, 10, Color.black, 1, "car");
    }

    @Override
    public void reactToTimer(long time) {
        AbstractDrawable car1 = sg.getDrawable("car");
        if (car1.posX > 640) {
            sg.animMoveAllRel(-640, 0, "car");
        } else {
            sg.animMoveAllRel(4, 0, "car");
        }
        sg.repaintPanel(); // important!
    }

    public static void main(String[] args) {
        new GroupAnimation();
    }
}
```

- The program utilizes the timer to constantly relocate of a group of drawables. The actual animation is triggered in "reactToTimer"
- all animated elements are given the same name, "car". Group animation (animMoveAllRel) selects all elements named "car".
- all other drawables remain unchanged.

General Animation (color, size, etc.)

The animXXXX methods of the previous section are a convenient way to move drawables. This is, however, not the only way to animate drawables. Any remodeling of a drawable over time, that is performed in a way that the human eye/mind catches as a change of the same element (in contrast to appearance of a new element) is animation. We can change the thickness of a line, the color of a filled box, the radii of an ellipse etc. over time. Carefully performed, this will generate the effect of animation.
SimpleGUI is too simple to cover all these parameter changes, but of course you can manage your drawables yourself and change their parameters over time to create animation. It is suggested to use SimpleGUI's timer.
Animation on your own is done in the following steps, typically performed in the "reactToTimer" method:

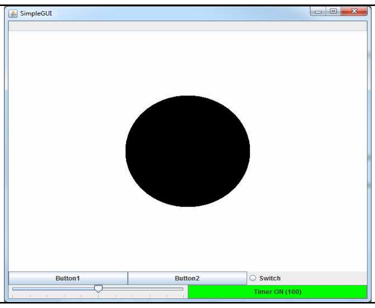
1. recompute the new parameters for all drawables
2. erase the old drawables from the drawing panel
3. redraw all drawables
4. goto 1

I you don't erase the old drawables, you will keep all versions, old and new, even if the new ones might entirely cover the old ones. After a while this will lead to an overload of drawables in SimpleGUI, slowing down your program.
There is one dangerous transition here: between erasing and redrawing the screen is empty. If you show an empty screen even ever so shortly between your scenes, you create an annoying flickering effect. To avoid flickering, you need to make sure that the scene is only shown when it is entirely re-created. SimpleGUI has a simple mechanism to guarantee this.

To avoid flickering, you need to
1. turn off auto-repainting (use `setAutoRepaint(false)` at the beginning of your code)
2. repaint the scene after redrawing all drawables (using `repaintPanel()`)

That's it. Try your own animation without turning off auto repaint, and see the difference when you animate multiple drawables!

The following example changes the size of an ellipse over time in a random way, which makes it wobble.



```
import simplegui.*;

public class SGTest implements TimerListener {
    private SimpleGUI sg;
    double radius = 100;

    // main
    public static void main(String[] args) {
        SGTest test = new SGTest();
    }

    // Constructor
    SGTest(){
        sg = new SimpleGUI();
        sg.setAutoRepaint(false); //<=== important!!!
        sg.registerToTimer(this);
        sg.timerStart(100);
    }

    // Callback method
    @Override
    public void reactToTimer(long l) {
        sg.eraseAllDrawables();
        radius += Math.random()*10-5;
        radius = Math.max(radius, 10); // never go below 10.
        sg.drawFilledEllipse(320-radius,240-radius, radius*2,
radius*2);
        sg.repaintPanel();        //<=== important!!!
    }
}
```

Image Processing

SimpleGUI does not only allow for drawing and scaling images, but gives full access to the image content, i.e. the pixel colors. The rgb values of each single pixel can be read and written. Hence without too much effort, basic image processing routines (e.g. color enhancement) can be written.

Demo Programs Related to This Section

- Demo12_ImageProcessing

Making Images Accessible

For image processing, the image-drawable must be accessible to our own code. The DrwImage object offers a constructor which just takes the image filename:

```
DrwImage image1 = new DrwImage("testImage.jpg");
```

Now we have access to certain methods of DrwImage:

int getHeight() // image height in pixel
int getWidth() //image width in pixel
RGB getPixelRGB(int x, int y) //get pixel color, returns an RGB object (see below)
void setPixelRGB(int x, int y, int r, int g, int b) // set color of pixel
void setPixelRGB(int x, int y, RGB rgb) // set color of pixel

Methods for image access

Once a DrwImage object is instantiated (and probably processed), it can be drawn using the SimpleGUI method

drawImage (DrwImage image, double x, double y, double width, double height, java.lang.String name)
Drawing a DrwImage Object

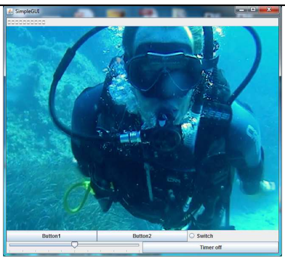
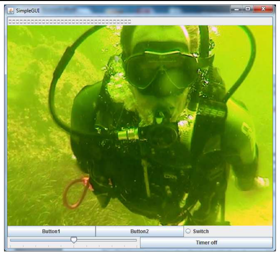
The class RGB
The class is a convenience class that just holds the red, green, blue and brightness value of a pixel. It explains itself:

```
package simplegui;
public class RGB {
    public int r, g, b;
    public int brightness;

    public RGB() {
        r = g = b = 0;
    }

    public RGB(int r, int g, int b) {
        this.r = r;
        this.g = g;
        this.b = b;
        brightness = (int) Math.round(0.2989 * r + 0.5870 * g + 0.1140 * b);
    }
}
```

Class RGB

The weights 0.2989, 0.5870, 0.1140 mirror the relative sensitivity of the human eye to the respective colors red, green, blue. The brightness value is an integer in 0..255.
Knowing the image's dimensions (getHeight(), getWidth()), a simple nested loop can retrieve all RGB values, and, as in the following example, exchange them to different colors.

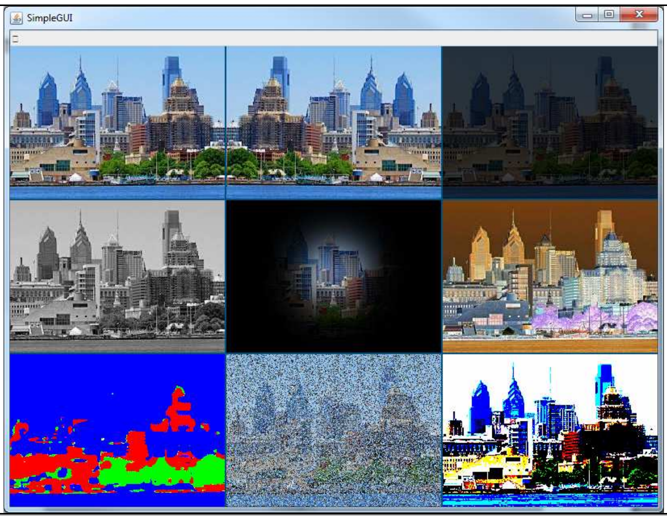
The example exchanges the r,g and b values of each pixel, i.e. r<-g, g<-b, b<-r after button 1 is pressed.

 changing to 

```
import simplegui.DrwImage;
import simplegui.RGB;
import simplegui.SimpleGUI;

public class SGTest {
    public static void main(String[] args) {
        SimpleGUI sg = new SimpleGUI();
        DrwImage im = new DrwImage("test.jpg");
        sg.drawImage(im, 0, 0, -1, -1, null);
        sg.waitForButton1();
        for (int c=0; c<im.getWidth(); c++){
            for (int r = 0; r<im.getHeight(); r++){
                RGB rgb = im.getPixelRGB(c, r);
                im.setPixelRGB(c,r,rgb.g,rgb.b,rgb.r);
            }
        }
        sg.drawImage(im, 0, 0, -1, -1, null);
    }
}
```

Throwing a diver into murky waters with a single press of a button: Color Exchange Example

For more exciting image processing examples, please look at the ImageProcessing example in package demos.



Output of the Image Processing Demo in Package "demos"