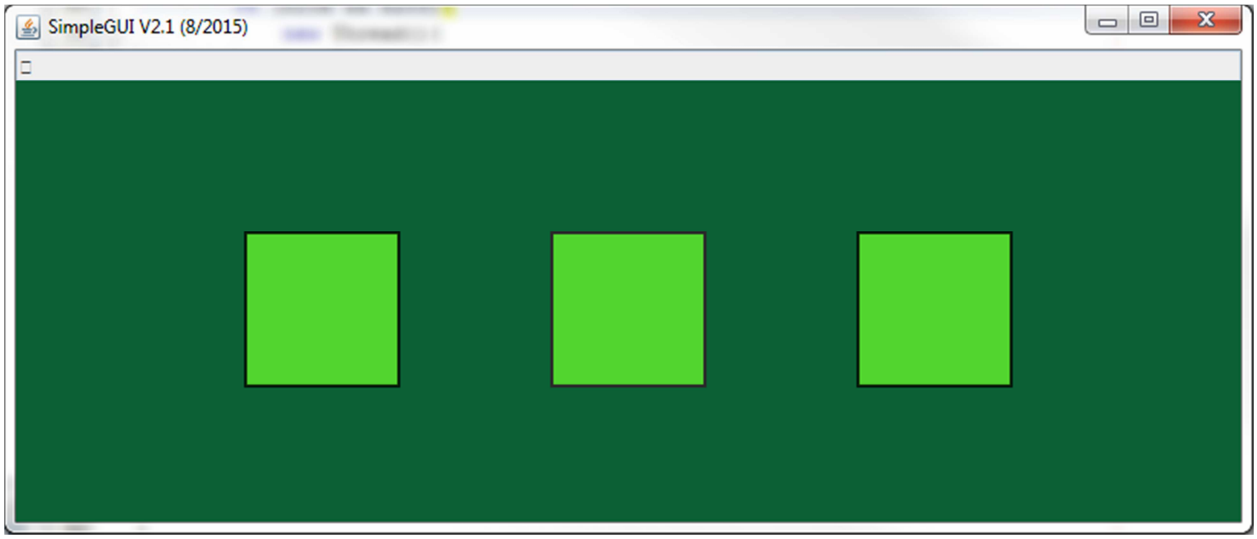
Assignment: Surprising Buttons

Topic: Inheritance, Abstract classes, Polymorphism



This assignment will let you experience the beauty of OOP (object oriented programming), and three of the main OOP concepts, **inheritance, method overriding** and **polymorphism**.

Many of the OOP concepts are motivated by software design. A project structured by inheritance modularizes functionalities: everything that is common to children is put into parent classes, yet methods/fields distinguishing children from parents (and siblings) are placed into the respective children.

Note: be aware that inheritance structures can go deep, grand-children, grand-grand-children etc. In large projects this is not unusual at all.

With the principle of overriding, an additional significant advantage of inherited structures, apart from pure software-structure, comes into play: children can change, or influence, their inheritance, i.e. methods can act differently than they either were supposed to (that's important if a parent method has a bug, for example, or needs to be adjusted to a special setting), or they can act differently than the software designers could even imagine. This assignment will explore the latter case (hence the name, Surprising Buttons).

Surprising Buttons is a collaboration between you and me: I provide a button class (using SimpleGUI), "ParentButton", which provides basic button functionality: the button is put on the screen, it changes its border color when clicked etc. However, usually a button triggers some action. That part is not

implemented, but only prepared. This is where your task starts. By overriding a certain (abstract) method, you can add any action for the button you wish. Your part clearly might go beyond my imagination, but my software is prepared to deal with it. Do you see the advantage of OOP here?

Note: these are custom buttons. They have nothing to do with SimpleGUI's oder JAVA's built in GUI-elements.

Your Task

Write a program that displays three buttons. A click on each of these buttons must trigger a different action. To achieve this, you must write three own classes, which extend from "ParentButton", which is a class that is provided. ParentButton is an abstract class, that provides (among other methods) the abstract method "processButton()". When the button realizes that it is clicked, it will call this method automatically (more later), triggering the action you programmed into your own versions of "processButton()".

Attention! The breakdown into steps below makes this assignment seem complicated. It is not at all. Just follow each step meticulously yet not stupidly. Think deeply about what each step does.

Detailed Description

You are given a Netbeans project. The following steps build upon that project.

Steps:

1. In the package "surprisingbuttons", Create three different classes, each extending "ParentButton". Since ParentButton resides in a different package, you must import ParentButton (Netbeans: ctrl-shift-i). For this example I will use the class names ChildButtonA, ChildButtonB, ChildButtonC. Please come up with something more descriptive.
2. In each of your classes, implement the requested method "public void processButton()". For testing purposes, just put a simple println in there, so you will see if things work.
3. In your Main class, create an array "buttons[3]" of ParentButton (please think here for a moment. This is an array of ParentButton, but it will contain references to instances of your class, i.e. ChildButtonA, ChildButtonB, ChildButtonC. Why is this possible, and what are the consequences?)
4. Instantiate one object of each class from ChildButtonA, ChildButtonB, ChildButtonC, and store (the references to) them in the array.
5. Miraculously, a SimpleGUI will pop up, although you never created an instance of it. The reason: the class "ParentButton()" has a static(!) instance of SimpleGUI itself. That's the one you see, and that's the one you MUST use. You have access to it through a method "public static ParentClass.getSimpleGUI()". Usage: SimpleGUI sg = ParentButton.getSimpleGUI().
6. For each of your button, set the position on the screen. This is done using the inherited method "setPosition(double x, double y)" which comes from the parent class ParentButton(). Note: setPosition() lives in the parent class, but you call it using the references to your own (child-) classes! You have never declared such method in your class, but you can use it since you

inherited it. If you don't perform this step, all three buttons of yours will reside on top of each other in (0,0), the upper left corner.

7. Write a loop that waits for a mouse-click. Attention! The class "ParentButton()" defines its own SimpleGUI, as mentioned above (step 5). You MUST access the mouse-method of that specific SimpleGUI instance, in the following way:

   Int[] xy = ParentButton.getSimpleGUI().waitForMouseClick().

   WaitForMouseClick() returns an int[] array, which contains the x and y position of the mouse pointer (relative to the its SimpleGUI panel).
8. On each of the buttons in the button[]-array, call the inherited method "hitButton(double x, double y)", using the mouse position parameters gained in step 7. hitButton() is inherited from ParentButton, it checks if the mouse pointer was clicked inside the respective button. If so, the method "processButton()" will be called automatically. THIS STARTS THE ACTION YOU PROVIDED!
9. By now, you should see your println output on the screen. If that works, become creative and do something more exciting. Hint: The SimpleGUI jar includes a package simplesound, which can play sounds.

Methods in ParentButton():

- public ParentButton(): Constructor. Sets the button location to (0,0).
- public void setPosition(double x, double y): move the button to position (x,y). This is a relative displacement.
- Public static SimpleGUI getSimpleGUI(): returns reference to a static SimpleGUI object.
- Public void hitButton(double x, double y): checks if the x,y input coordinates are inside the button. If so, activates border blinking and calls the method processButton(). hitButton() is the method that triggers the action!
- Public abstract void processButton(): method called by hitButton(). This is the method you must override in your own class(es). This method defined the action that is triggered by the button.

Remark: this button implementation is unusual for 2 reasons:

1. It uses abstract classes instead of interfaces. The reason is simply that we did not talk about interfaces. A professional implementation would have utilized interfaces. The advantage: an interface does not use up the single parent connection a class is allowed to have. In our case: your buttons extend ParentButton, now they cannot inherit from anything else anymore. That's a huge drawback compared to Interfaces.
2. In a professional setting, a button would be implemented event-based, whatever that means. We will handle that later.

These 2 notes are useful for interviews, especially note 1 covers a topic frequently asked in interviews.

The TA has the solution code. You might want to visit him to see it, in case you are lost.