Arrays:

Feste Größe! Falls man dynamische Größe möchte, dann s. ArrayList!!!

String[] cars = new String[3]; cars[0] = „VW“;

Entweder wie oben, oder die Kurzschreibweise:

String[] cars = {„VW“, „Audi“};

Achtung Zuweisung ist hier {};

2d Array:

String[][] cars = new String[3][3];  
  
cars[0][0] = "Camero";  
cars[0][1] = "VW";  
cars[0][2] = "Audi";  
cars[1][0] = "Ford";  
cars[1][1] = "VW";  
cars[1][2] = "Tesla";  
cars[2][0] = "Ferrari";  
cars[2][1] = "Hey";  
cars[2][2] = "Whatsapp";  
  
for(int i=0; i<cars.length; i++){  
 System.*out*.println();  
 for(int j=0; j<cars[i].length; j++){  
 System.*out*.print(cars[i][j] + " ");  
 }  
}

Kurzschreibweise: String[][] cars = { {„Camero“,“VW“,“Audi“}, {„…“}, {„…“} }

Immer an Rows und Columns denken!!!

Wrapper Classes:

Boolean a = true;  
Character b = 'a';  
Integer c = 123;  
Double d = 3.14;  
String e = "Bro";

Anfangsbuchstabe groß, dann hat man dasselbe wie bei String, d.h. Reference data types.  
Aber diese sind langsamer!!!

ArrayList:

ArrayList<String> food = new ArrayList<String>();  
  
food.add("Burger");  
food.add("Pizza");  
food.add("Kalamari");  
  
food.set(2, "Sushi");  
food.remove(1);  
food.clear();  
  
for(int i=0; i<food.size(); i++){  
 System.*out*.println(food.get(i));  
}

Der Vorteil, es ist resizable after compilation. Jedoch nur Reference Data types speichern möglich, keine Primitives.

2d ArrayList:

ArrayList<ArrayList<String>> allLists = new ArrayList<>();  
  
ArrayList<String> bakeryList = new ArrayList<String>();  
bakeryList.add("pasta");  
bakeryList.add("bread");  
bakeryList.add("donuts");  
  
System.*out*.println(bakeryList.get(0));  
  
ArrayList<String> productList = new ArrayList<String>();  
productList.add("tomotos");  
productList.add("peppers");  
productList.add("zucchini");  
  
System.*out*.println(productList.get(0));  
  
ArrayList<String> drinkList = new ArrayList<String>();  
drinkList.add("water");  
drinkList.add("coffee");  
drinkList.add("schnaps");  
  
System.*out*.println(productList.get(0));  
  
allLists.add(bakeryList);  
allLists.add(productList);  
allLists.add(drinkList);  
  
System.*out*.println(allLists);

[[pasta, bread, donuts], [tomotos, peppers, zucchini], [water, coffee, schnaps]]

Um pasta zu bekommen: System.out.println(allLists.get(0).get(0));

For each loop:

String[] animals = {"Tiger", "Elephant", "Frog"};  
  
for(String i : animals){  
 System.*out*.println(i);  
}

Variablen nicht mehr veränderbar:

Final double PI = 3,145;

Common practice: Capital letters in var name;

GUI:

Öffnet ein Windows Fenster mit graphical User Interface.

JFrame jframe = new JFrame();  
jframe.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
jframe.setSize(750,750);  
jframe.setLayout(null);  
jframe.setVisible(true);

JFrame mit JLabel in einem JPanel:

JLabel label = new JLabel();  
label.setText("Hey there!");  
label.setIcon(icon);  
  
  
JPanel redPanel = new JPanel();  
redPanel.setBackground(Color.*red*);  
redPanel.setBounds(0,0, 250,250);  
redPanel.add(label);  
  
JPanel bluePanel = new JPanel();  
bluePanel.setBackground(Color.*blue*);  
bluePanel.setBounds(250,0, 250,250);  
  
JPanel greenPanel = new JPanel();  
greenPanel.setBackground(Color.*green*);  
greenPanel.setBounds(500,0, 250,250);  
  
  
  
  
  
JFrame jframe = new JFrame();  
jframe.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
jframe.setSize(750,750);  
jframe.setLayout(null);  
jframe.setVisible(true);  
jframe.add(redPanel);  
jframe.add(bluePanel);  
jframe.add(greenPanel);

Ein Bild, das Platz enthält.

Automatisch generierte Beschreibung

JButton mit Action:

public class MyFrame extends JFrame implements ActionListener {  
  
 JButton button;  
 MyFrame() {  
  
 button = new JButton();  
 button.setBounds(200,100, 100, 50);  
 button.addActionListener(this);  
  
  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setLayout(null);  
 this.setSize(500,500);  
 this.setVisible(true);  
 this.add(button);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if(e.getSource()==button){  
 System.*out*.println("yeah");  
  
 }  
 }

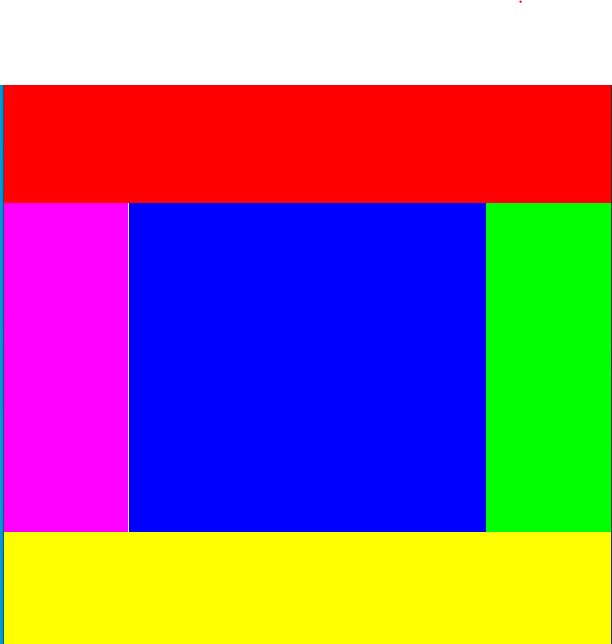
Action in kurz:

button.addActionListener(e -> System.*out*.println("yeah"));

Border Layout Manager:

Man kann z.B. panel dem frame adden und zwar an 5 Positionen: Nord, Ost, West, Süd, Center;

JFrame frame = new JFrame();  
frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
frame.setSize(500,500);  
frame.setVisible(true);  
frame.setLayout(new BorderLayout());  
  
  
JPanel panel1 = new JPanel();  
JPanel panel2 = new JPanel();  
JPanel panel3 = new JPanel();  
JPanel panel4 = new JPanel();  
JPanel panel5 = new JPanel();  
  
panel1.setBackground(Color.*red*);  
panel2.setBackground(Color.*green*);  
panel3.setBackground(Color.*yellow*);  
panel4.setBackground(Color.*magenta*);  
panel5.setBackground(Color.*blue*);  
  
panel1.setPreferredSize(new Dimension(100,100));  
panel2.setPreferredSize(new Dimension(100,100));  
panel3.setPreferredSize(new Dimension(100,100));  
panel4.setPreferredSize(new Dimension(100,100));  
panel5.setPreferredSize(new Dimension(100,100));  
  
frame.add(panel1, BorderLayout.*NORTH*);  
frame.add(panel2, BorderLayout.*EAST*);  
frame.add(panel3, BorderLayout.*SOUTH*);  
frame.add(panel4, BorderLayout.*WEST*);  
frame.add(panel5, BorderLayout.*CENTER*);



Wichtig: Sogar in einem Panel kann wieder ein neues Layout erzeugt werden!

Flow Layout:

Reihenbasiert. Nutzt eine Reihe, wenn die Breite voll ist, wird nächstes Element in nächste Reihe geschoben!

Grid Layout:

Wie Grid aufgebaut mit Rows and Columns, oder zellenbasiert

Ein Bild, das Text, Himmel, Schrank enthält.

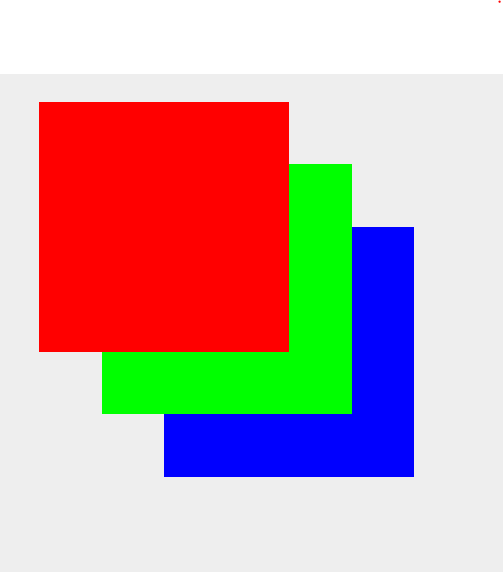
Automatisch generierte Beschreibung

frame.setLayout(new GridLayout(3,3));

JLayeredPane:

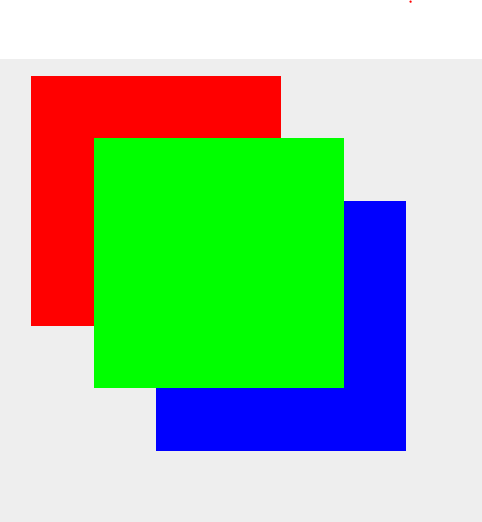
Ist ein Container der 3d Effekte der Komponenten erzeugt, 🡪 z index

JLabel label1 = new JLabel();  
label1.setOpaque(true);  
label1.setBackground(Color.*red*);  
label1.setBounds(50,50,200,200);  
  
JLabel label2 = new JLabel();  
label2.setOpaque(true);  
label2.setBackground(Color.*green*);  
label2.setBounds(100,100,200,200);  
  
JLabel label3 = new JLabel();  
label3.setOpaque(true);  
label3.setBackground(Color.*blue*);  
label3.setBounds(150,150,200,200);  
  
JLayeredPane pane = new JLayeredPane();  
pane.setBounds(0,0, 500,500);  
pane.add(label1);  
pane.add(label2);  
pane.add(label3);  
  
  
  
  
JFrame frame = new JFrame();  
frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
frame.setSize(500,500);  
  
frame.setLayout(null);  
frame.add(pane);  
  
frame.setVisible(true);



Verschiedene Layer:

pane.add(label1, Integer.*valueOf*(0));  
pane.add(label2, Integer.*valueOf*(2));  
pane.add(label3, Integer.*valueOf*(1));



JOptionPane = DialogBox

JOptionPane.*showMessageDialog*(null, "hi", "hey", JOptionPane.*PLAIN\_MESSAGE*);

Mit Optionen:

JOptionPane.*showConfirmDialog*(null,"Do you code?", "Coding", JOptionPane.*YES\_NO\_CANCEL\_OPTION*);

Mit Input Feld und Ausgabe:  
  
System.*out*.println(JOptionPane.*showInputDialog*(null,"Do you code?", "Coding", JOptionPane.*QUESTION\_MESSAGE*));

Mit Optionen und personal Icons etc.

System.*out*.println(JOptionPane.*showOptionDialog*(null, "You are awesome","secret", JOptionPane.*YES\_NO\_CANCEL\_OPTION*, JOptionPane.*INFORMATION\_MESSAGE*,null,null,0));

JTextField: TextBox!

this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
this.setLayout(new FlowLayout());  
  
JTextField textfield = new JTextField();  
textfield.setPreferredSize(new Dimension(250,40));  
this.add(textfield);  
  
  
  
this.pack();  
this.setVisible(true);

Mit Submit-Button:

JButton button = new JButton("Submit");  
JTextField textfield = new JTextField();  
MyFrame() {  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setLayout(new FlowLayout());  
  
  
 button.setPreferredSize(new Dimension(100,50));  
 button.setFocusable(false);  
 button.addActionListener(this);  
  
  
 textfield.setPreferredSize(new Dimension(250,40));  
  
 this.add(textfield);  
 this.add(button);  
  
  
  
 this.pack();  
 this.setVisible(true);  
}  
  
@Override  
public void actionPerformed(ActionEvent e) {  
 if(e.getSource() == button){  
 System.*out*.println(textfield.getText());  
 }  
}

JCheckbox:

JCheckBox checkBox = new JCheckBox();  
checkBox.setText("I am not a robot");  
checkBox.setFocusable(false);  
checkBox.setFont(new Font("Consolas", Font.*PLAIN*, 35));  
this.add(checkBox);

JButton button = new JButton();  
JCheckBox checkBox = new JCheckBox();  
MyFrame() {  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setLayout(new FlowLayout());  
  
  
 button.setText("Submit");  
 button.addActionListener(this);  
  
  
  
 checkBox.setText("I am not a robot");  
 checkBox.setFocusable(false);  
 checkBox.setFont(new Font("Consolas", Font.*PLAIN*, 35));  
 this.add(checkBox);  
 this.add(button);  
  
  
  
 this.pack();  
 this.setVisible(true);  
}  
  
@Override  
public void actionPerformed(ActionEvent e) {  
 if(e.getSource()==button){  
 System.*out*.println(checkBox.isSelected());  
 }  
}

JRadioButton

this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
this.setLayout(new FlowLayout());  
  
JRadioButton pizzaButton = new JRadioButton("pizza");  
JRadioButton hamburgerButton = new JRadioButton("hamburger");  
JRadioButton hotdogButton = new JRadioButton("hotdog");  
  
ButtonGroup group = new ButtonGroup();  
group.add(pizzaButton);  
group.add(hamburgerButton);  
group.add(hotdogButton);  
  
button.setText("Submit");  
  
  
this.add(pizzaButton);  
this.add(hamburgerButton);  
this.add(hotdogButton);  
  
  
this.add(button);  
  
  
  
this.pack();  
this.setVisible(true);

Jeder der buttons hat automatisch ein Event!

JComboBox Button oder Feld mit Drop-Down-List

MyFrame() {  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setLayout(new FlowLayout());  
  
 String[] animals = {"Rabbit", "Dog", "Cat"};  
  
 comboBox = new JComboBox(animals);  
 comboBox.addActionListener(this);  
  
  
  
 button.setText("Submit");  
  
  
 this.add(comboBox);  
 this.add(button);  
  
  
 this.pack();  
 this.setVisible(true);  
}  
  
@Override  
public void actionPerformed(ActionEvent e) {  
 if(e.getSource()==comboBox){  
 System.*out*.println(comboBox.getSelectedItem());  
 }  
}

JSlider:

Achtung hier: addChangeListener!

MyFrame() {  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setLayout(new FlowLayout());  
  
 panel = new JPanel();  
 label = new JLabel();  
 slider = new JSlider(0, 100, 50);  
  
 slider.setPreferredSize(new Dimension(400,200));  
 slider.setPaintTicks(true);  
 slider.setMinorTickSpacing(10);  
 slider.setPaintLabels(true);  
 slider.setFont(new Font("MV Boli", Font.*PLAIN*, 15));  
 slider.setOrientation(SwingConstants.*VERTICAL*);  
  
 slider.setPaintTrack(true);  
 slider.setMajorTickSpacing(25);  
 slider.addChangeListener((e) -> label.setText("C = " + slider.getValue()));  
  
 label.setText("C = " + slider.getValue());  
  
 panel.add(slider);  
 panel.add(label);  
  
 button.setText("Submit");  
 this.setSize(420,420);  
 this.add(panel);  
 this.add(button);  
  
  
 this.pack();  
 this.setVisible(true);  
}  
  
public void actionPerformed(ActionEvent e) {  
 label.setText("C = " + slider.getValue());  
}  
  
@Override  
public void stateChanged(ChangeEvent e) {  
  
}

JProgressBar:

public class ProgressBar {  
  
 JFrame frame = new JFrame();  
  
 JProgressBar bar = new JProgressBar();  
 ProgressBar() {  
  
 bar.setValue(10);  
 bar.setBounds(0,0,420,50);  
 bar.setStringPainted(true);  
  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setSize(420,420);  
 frame.setLayout(null);  
  
  
  
 frame.add(bar);  
  
 frame.setVisible(true);  
  
 fill();  
 }  
  
 public void fill() {  
 int counter = 0;  
  
 while (counter <=100) {  
 bar.setValue(counter);  
 try {  
 Thread.*sleep*(50);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 counter += 1;  
 }  
 }  
}

JMenuBar:

Menü!



public class MyFrame extends JFrame implements ActionListener {  
 JMenuItem loadItem;  
 JMenuItem saveItem;  
 JMenuItem exitItem;  
  
 MyFrame() {  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setSize(500,500);  
 this.setLayout(new FlowLayout());  
  
 JMenuBar menuBar = new JMenuBar();  
  
 this.setJMenuBar(menuBar);  
 JMenu fileMenu = new JMenu("File");  
 JMenu editMenu = new JMenu("Edit");  
 JMenu helpMenu = new JMenu("Help");  
  
 loadItem = new JMenuItem("Load");  
 saveItem = new JMenuItem("Save");  
 exitItem = new JMenuItem("Exit");  
  
  
 loadItem.addActionListener(this);  
 saveItem.addActionListener(this);  
 exitItem.addActionListener(this);  
  
 fileMenu.add(loadItem);  
 fileMenu.add(saveItem);  
 fileMenu.add(exitItem);  
  
  
  
 menuBar.add(fileMenu);  
 menuBar.add(editMenu);  
 menuBar.add(helpMenu);  
  
  
  
  
 this.setVisible(true);  
 }  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if(e.getSource()==loadItem){  
 System.*out*.println("Load-item");  
 }  
 if(e.getSource()==saveItem){  
 System.*out*.println("save-item");  
 }  
 if(e.getSource()==exitItem){  
 System.*out*.println("ext-item");  
 }  
  
 }

Keyboard-Shurtcut für Menü:

loadItem.setMnemonic(KeyEvent.*VK\_L*);  
saveItem.setMnemonic(KeyEvent.*VK\_S*);  
exitItem.setMnemonic(KeyEvent.*VK\_E*);

JFileChooser:

public class MyFrame extends JFrame implements ActionListener {  
 JButton button;  
  
 MyFrame() {  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setSize(500,500);  
 this.setLayout(new FlowLayout());  
  
 button = new JButton("select file");  
 button.addActionListener(this);  
  
 this.add(button);  
  
  
  
 this.pack();  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if(e.getSource()==button) {  
 JFileChooser fileChooser = new JFileChooser();  
 fileChooser.setCurrentDirectory(new File("."));  
  
 int response = fileChooser.showOpenDialog(null);  
 //int response = fileChooser.showSaveDialog(null);  
  
 if(response == JFileChooser.*APPROVE\_OPTION*){  
 File file = new File(fileChooser.getSelectedFile().getAbsolutePath());  
 System.*out*.println(file);  
  
 }  
 }  
 }

JColorChooser:

public class MyFrame extends JFrame implements ActionListener {  
  
 JButton button;  
 JLabel label;  
  
 MyFrame() {  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setSize(500,500);  
 this.setLayout(new FlowLayout());  
  
 button = new JButton("Pick a Color!");  
 label = new JLabel();  
 label.setBackground(Color.*white*);  
 label.setOpaque(true);  
 label.setText("Hey");  
 label.setFont(new Font("MV Boli", Font.*BOLD*, 100));  
 button.addActionListener(this);  
  
 this.add(button);  
 this.add(label);  
  
  
 this.pack();  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if(e.getSource()==button){  
 JColorChooser colorChooser = new JColorChooser();  
 Color color = JColorChooser.*showDialog*(null, "Pick a color", Color.*black*);  
 label.setBackground(color);  
 }  
 }

KeyListener:

public class MyFrame extends JFrame implements KeyListener {  
  
 ImageIcon image = new ImageIcon("logo.png");  
 JLabel label;  
 Integer x = 0;  
 Integer y = 0;  
  
 MyFrame() {  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setSize(500,500);  
 this.setLayout(null);  
 this.addKeyListener(this);  
  
 label = new JLabel();  
 label.setIcon(image);  
 label.setBounds(0,0, 100,100);  
   
 this.add(label);  
  
 this.setVisible(true);  
 }  
  
  
 @Override  
 public void keyTyped(KeyEvent e) {  
  
 }  
  
 @Override  
 public void keyPressed(KeyEvent e) {  
 if(e.getKeyCode() == 39) {  
 x += 10;  
  
 }  
 if(e.getKeyCode() == 37) {  
 x-=10;  
 }  
 setCoordinates();  
 }  
  
 @Override  
 public void keyReleased(KeyEvent e) {  
 System.*out*.println("You released key character: " + e.getKeyChar());  
 System.*out*.println("You released key character: " + e.getKeyCode());  
 }  
  
 public void setCoordinates() {  
 label.setLocation(x, y);  
 }

MouseListener: implements MouseListener!

Drag and Drop:

DragPanel:

public class DragPanel extends JPanel {  
 ImageIcon icon = new ImageIcon("logo.png");  
 final int WIDTH = icon.getIconWidth();  
 final int HEIGHT = icon.getIconHeight();  
 Point imageCorner;  
 Point prevPt;  
 DragPanel() {  
  
 imageCorner = new Point(0,0);  
 ClickListener clickListener = new ClickListener();  
 DragListener dragListener = new DragListener();  
 this.addMouseListener(clickListener);  
 this.addMouseMotionListener(dragListener);  
  
 }  
  
 public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 icon.paintIcon(this, g, (int)imageCorner.getX(), (int)imageCorner.getY());  
  
 }  
  
 private class ClickListener extends MouseAdapter {  
 public void mousePressed(MouseEvent e) {  
 prevPt = e.getPoint();  
 }  
  
 }  
  
 private class DragListener extends MouseMotionAdapter {  
 public void mouseDragged(MouseEvent e) {  
 Point currentPt = e.getPoint();  
 imageCorner.translate(  
 (int)(currentPt.getX() - prevPt.getX()),  
 (int)(currentPt.getY() - prevPt.getY())  
 );  
 prevPt = currentPt;  
 repaint();  
 }  
 }

MyFrame:

public class MyFrame extends JFrame {  
  
 DragPanel dragPanel = new DragPanel();  
 MyFrame() {  
 this.add(dragPanel);  
 this.setTitle("Drag & Drop demo");  
 this.setSize(600,600);  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
  
  
  
 this.setVisible(true);  
 }

KeyBindings:  
  
Eine Aktion an eine Taste binden.

Mehr Flexibilität als KeyListeners.

Tasten können an Komponente gebunden werden. Ist jedoch nicht ganz so einfach.

Dazu z.B. Game Class mit individuellen Actions z.B. UpAction extends AbstractAction{}

Und dann Action upAction;

