Java Course

Florian Cliquet
note taking of online ressources

27 mai 2024

Sommaire

Introduction

This document provides an overview of Java concepts written by Florian Cliquet. It's a set of notes on the Bro Code video "Java Full Course for free," available at the following link : "Java Full Course for free".

1 Source Code

Compiler	.java
Byte Code	.class
JVM	.class
	Object code

---

2 JDK

JDK	Java Development Kit	dev code
JRE	Java Runtime Environment	Libraries & toolkits
JVM	Java Virtual Machine	runs java programs

---

3 Variables

3.1 Primitive

— 8 types
— Stores data
— Can only hold values
— Less memory
— Fast

3.2 Reference

— Unlimited (user-defined)
— Stores an address
— Created to hold more data
— More memory
— Slower

3.3 Data Types

Data Type	**Size**	**Primitive / Reference**	**Values**
boolean	1 bit	primitive	true / false
byte	1 byte	primitive	-128 to 127
short	2 bytes	primitive	-32,768 to 32,767
int	4 bytes	primitive	-2 billion to 2 billion
long	8 bytes	primitive	-9 quintillion to 9 quintillion

float	4 bytes	primitive	up to 6-7 digits
double	8 bytes	primitive	
char	2 bytes	primitive	single char
String	varies	reference	multiple chars

---

4 User Input

4.1 Scanner

- `import java.util.Scanner;`
- Instantiate the object : `Scanner scanner = new Scanner(System.in);`

4.2 How to Use

- `System.out.print("...");`
- `String res = scanner.nextLine();`
- `int res = scanner.nextInt();`
- If you want to use `scanner.nextLine()` after other input, you need an empty `nextLine()`.

---

5 GUI Introduction

- `import javax.swing.JOptionPane;`
- Create a box to get input : `String data = JOptionPane.showInputDialog(...);`
- It returns you a string so you need to parse it if u want to use other Type like :
- `Integer.parseInt(JOptionPane.showInput(...));`
- `Double.parseDouble(JOptionPane.showInput(...));`
- in general `Type.parseType(JOptionPane.showInput(...)`
- `JOptionPane.showMessageDialog(parentComponent,message,title,messageType);`
  - parentComponent :
    - `JFrame`
    - `JDialog`
    - `Jwindow`
    - `null`
  - title :
    - `title of the dialog box`
  - messageType :
    - `JOptionPane.PLAIN_MESSAGE;`
    - `JOptionPane.INFORMATION_MESSAGE;`

  - `JOptionPane.QUESTION_MESSAGE;`
  - `JOptionPane.WARNING_MESSAGE;`
  - `JOptionPane.ERROR_MESSAGE;`

---

6 Random Library

- `import java.util.Random;`
- Instantiate the object : `Random random = new Random();`
- `random.nextType();`

---

7 Switch

- `switch(x) {`
- `case "1": System.out.println("..."); break;`
- `case ...: break;`
- `default: System.out.println("default case");`
- `}`

---

8 String Methods

- `equals(String)` - see if they're equal
- `equalsIgnoreCase(String)` - doesn't care about case
- `length()` - size of string
- `charAt(index)` - char at index x
- `indexOf("B")` - index of B char
- `isEmpty()` - true = empty, false = not empty
- `toUpperCase() / toLowerCase()`
- `trim()` - remove any whitespace before/after
- `replace(oldChar, newChar)` - replaces char

---

9 Wrapper Class

- Provides a way to use primitive data types as reference data types
- Reference data types contain useful methods and can be used with collections
- Primitive : `boolean, char, int, double`
- Wrapper : `Boolean, Character, Integer, Double`
- Autoboxing : the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes
- Unboxing : the reverse of autoboxing

- Exemple : `When writing Boolean a = true, we use Autoboxing and when we compare (a==true) we use Unboxing`

---

10 ArrayList

- `ArrayList<Type> name = new ArrayList<Type>();`
- Methods :
  - `add(data)` - add item into array (last position)
  - `get(index)` - return element at index
  - `set(index, elem)` - change at index the old element by elem
  - `remove(index)`
  - `clear()` - remove everything in the array

- 2D ArrayList
- `ArrayList<ArrayList<Type»name = new ArrayList();`
- Same methods as single dim array

---

11 Overloaded methods

Methods that share the same name but have different parameters.

- `Method name + parameters = method signature`
- static Type name (Type 1 , Type2) {/* Content of the method */}
- static Type name (Type 1, Type2 , Type3){/* Content of the method */}
- if i call name with 2 args it'll take the first and with 3 args the second.

---

12 printf() method

an optional method to control , format and display text to the console. Called by System.out.printf(/**/).

- `two argument : format string + object`
- into format string we place a % to place our object
- %`[flags][precision][width][conversion-character]`
- `[conversion-character] -> associated type`

$\%c$	char
$\%s$	string
$\%d$	int
$\%f$	double / float
$\%b$	boolean

— `[width] -> min number of char to be printed out`
— %xs / %xd ... with x the min number
— `precision -> sets number of digits of precision with floating-point values`
— %.xf with x the precision number
— `[flags] -> add an effect to output based on flag added`

`-`	-> left-justify
`+`	-> output a (+) or (-) based on numerical value
`0`	-> numeric values are zero-padded
`,`	-> comma grouping separator if num > 1000

---

13 keyword

13.1 final

make the variable unchangeable

— `final type name =..;`

13.2 static

static = modifier. A single copy of a variable / method is created and shared. We says that the class "own" the static memeber.
If a class have a "static int numberofelem", with numberofelem++ in the constructor , we can , regardless of the elem, return the number of elem.
— `static type name = ..;`

---

14 Objects (OOP)

Object Oriented Programming. An object is an instance of class that may contain attributes and methodes.
We create class like this :
— `public class Example` {
— Type name ;
— ... ;
— Type name_n ;
— `/*Constructor*/ -> we can overload them`
— Example(Type name,..,Type name_)this.name = name , ... , this.name_n=name_n
— `/* Methods*/`

- void method1()/* content of the method*/

And we use them like that :

- Example example1 = new Example(Type name, ..., Type name_n) -> Initialisation of the object
- example1.method1() ; <- Call of the first method for the example1 object

---

15 variable scope

15.1 local

declared inside a method and only visible in the method

15.2 global

declared outside a method , but within a class visible to all parts of a class

---

16 toString() method

Special method that all objects inherit, that returns a string that "textually represents" an object.
This is either explicite or implicite (you don't need to call it).

---

17 Inheritance

The process where one class acquires the attributes and methods of another
For an existing class Class1 :

- `while creating class2`
- public class Class2 `extends` Class1 { /* content */ }

---

18 Method overriding

Declaring a method in sub class which is already present in parent class. Done so that a child class can give it's own implementation of the method.

---

19 Super keyword

Super's keyword refers to the superclass (parent) of an object , very similar to the "this" keyword. In a class that inherit from another, if you want to use constructor from the parent class :

- ChildClass(type name1 , type name2 , type name3){
- super(name1,name2) ; /* Call constructor from the parent class*/
- this.name3 ;
- }

Works aswell with method to call parent's method :

- public type childmethod1(){
- return super.parentmethod() + this.name3 ; /* so that it's change a little bit from the original method without reimplementing it*/
- }

---

20 Abstract keyword

Abstract classes cannot be instantiated, but they can have sub class.
Abstract method are declared without an implementation.
Declared like this :

- public `abstract` class Class1 {/* content */ } <- We can't create Class1 object
- in the Class1 implementation :
- `abstract` void method1() ; <- force sub classes to implement this method
- in Subclass1 :
- @override
- type method1{ /* method1 implementation */ }

---

21 Access modifiers

Access Levels layers :

Modifier	Class	Package (collection of classes)	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

---

22 Encapsulation

Attributes of a class will be hidden or private, can be accesed only through methods (getters & setters).
You should make attributes private if you don't have a reason to make them public or protected.

getters and setters methods are classes methods that you have to implement.
getters method :
Make to return the private variable, example : getname methods,

— public String getname(){
— return name;
— }

setters method :
Make so we can set and edit class variable values.

— public void setname(String name){
— this.name = name;
— }

so we build our constructor like this :

— Classname(type name, ..){
— this.setname = name;
— /* every other variable */
— }

---

23 Copy objects

If you make object1 = object2 will share the same memory address. If you change one it'll change the other one.
In the object class you should create a copy() method like :

— public void copy(Class object) {
— this.setters(object.getobject());
— /* any other */
— }

We can also create overload constructor with :

— Class(Class obj){
— this.copy(obj);
— }

And so we can write : Class object2 = new Class(object1);

---

24 Interface

A template that can be applied to a class, similar to inheritance but specifies what a class has/must do.

Classes can apply more than interface, inheritance is limited to 1 super.
We initialize interface with : `public interface Interface 1{void method1()};`
Link class to interface :
- public class Class1 implements Interface1{
- -> define (override) every methods in Interface 1
- }

2 interface for a single class
- public class Class2 implements Interface1, Interface2 {
- /* override methods */
- }

---

25 Polymorphism

The ability of an object to identify as more than one type.
For n subclasses of Class1, if you want to make an array of object of those subclasses you have to put the Class1 type : "Class1[ ] = {/*every object */}
Dynamic polymorphism
-> dynamic = after compilation (during runtime).
Example how to do that :

- /* Scanner object that ask smthing */
- int choice = scanner.nextInt();
- if (choice ==1){
- /* initialise a subclass */
- }
- else if (choice ==2){
- /* initialise another subclass */
- }

---

26 Exception handling

Exception = an event that occurs during the execution of a program that disrupts the normal flow of instructions.

- try{
- /* content */
- catch("ErrorName" e){
- /* what you do with this error */
- }
- /* you can make others catches */
- } finally {

— /* what you do a the end */
— }

The usage of try / catch / finally make so you handle the error.
Or we can instead use :
`public static void main() throws IOException,..`{/ ∗ *content* ∗ /};

---

27 File Class

Library : java.io.File ;
file = an abstract representation of file and directory pathnames.
we init the object like this : `File file = new File(filepath);`
Some available methods :
— file.exists
— file.getPath() ;
— file.getAbsolutePath() ;
— file.isFile() ;
— file.delete() ;

---

28 FileWriter

We init the object like this : `FileWriter writer = new FileWriter(filename);`
Some available methods :
— writer.write(String) ; <- if you want to write on mulitple file just type '\\n' ;
— writer.close() ;
— writer.append(String) ; <- if you want to add text at the last line

---

29 FileReader

read the contents of a file as a stream of characters. One by one, read() returns an **int** value which contains the byte value. When read returns **-1**, there is no more data to be read.
We init the object like this : `FileReader reader = new FileReader(filepath);`
Some available methods :
— int data = reader.read() ;
— to display an entire file :
— while (data != -1) {
— System.out.println((char)data) ;
— }

Potential error : FileNotFoundException.

---

30 Audio

In order to show on this works, we will create a simple audio file reader with terminal.

```java
import javax.sound.sampled.*;
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class Main{
    public static void main(String[] args) throws
       UnsupportedAudioFileException, IOException,
       LineUnavailableException{
        Scanner scanner = new Scanner(System.in);
        File file = new File("src/test.wav");
        System.out.println("Looking for file at: " + file.
           getAbsolutePath());

        if (!file.exists()) {
            System.out.println("File not found!");
            return;
        }
        AudioInputStream audioStream = AudioSystem.
           getAudioInputStream(file);
        Clip clip = AudioSystem.getClip();
        clip.open(audioStream);

        String response = scanner.next();
        response = response.toUpperCase();

        while(!response.equals("Q")){
            System.out.println("P = play, S = Stop , R = Reset
               , Q = Quit");
            System.out.println("Enter your choice: ");
            response = scanner.next();
            response = response.toUpperCase();

            switch (response){
                case("P"): clip.start(); break;
                case("S"): clip.stop(); break;
                case ("R"): clip.setMicrosecondPosition(0);
                   break;
                default:
                    System.out.println("Not a valid response")
```

```
                        ;
            }
        }
        scanner.close();
    }
}
```

31 GUI

A JFrame is a GUI window to add components to.
library : javax.swing.JFrame

- To create frame instance : `JFrame frame = new JFrame`
- To display the frame : `frame.setVisible(true)`
- To set the size of the frame : `frame.setSizer(x-dim, y-dim)` (x-dim and y-dim are ints)
- To set the title of the frame : `frame.setTitle('your title')`
- To set Icon of the frame :
  - First import javax.swing.ImageIcon
  - Init image object : `ImageIcon image = new ImageIcon(imagepath;`
  - `frame.setIconImage(image.getImage();`
- By default, the close cross just hides the frame. To make it end the script, we need to use : `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`. By default, it's set to `"HIDE_ON_CLOSE,"` but we can even use `"DO_NOTHING_ON_CLOSE"`.
- If you want your frame to be not resizable (be default it is) : `frame.setResizable(true)`
- To change content : `frame.getContentPane()` (document yourself on more methods about it)
- To set background color :
  - First import java.awt.color
  - if you want to create your own color : `new color(rgb value or hexadecima ones)`
  - frame.getContentPane().setBackground(color.color)

It's better to put it in a class so that we can easily create a frame instance.

- public class MyFrame extends JFrame {
- constructor -> we replace frame by this. Example : this.setVisible(true)
- }

And so we just need to call `MyFrame myframe = new MyFrame()` , it'll create a duplicate of the frame.
Or you can just type `new MyFrame()` if you have no changes to make on the frame.

32 Labels

A GUI display area for a string of text, an image or both.
We init it like that : `JLabel label = new JLabel();`. In order to add the label to a frame : `frame.add(label);`.
Style your label : you should type "label." before every methods.

- setText(string) <- set content to your label
- setIcone(image) <- set image
- setPreferredSize(new Dimension(width,height))
- setHorizontalTextPosition(JLabel.CENTER/RIGHT/LEFT) ; <- for content pos
- setVerticalTextPosition(JLabel.CENTER/TOP/BOTTOM) ;
- setVerticalAlignment(JLabel...) ; <- for label pos
- setHorizontalAlignment(JLabel...) ;
- setForeground(color) <- font color
- setFont(fontname,type(italic, bold ...), size)
- setIconTextGap(size) <- set gap between text/img
- setBackground(color) <- By default the label opacity is false, means that we can't see the background, so if you want to display the bg color you should put :
- setOpaque(true) <- The label will take as much spaces as it could
- setBorder(border) <- init border like this : Border border = BorderFactory.createLineBorder(color,size)

Doing only this, it'll position your frame randomly, in order to be precise you can :
- set : `frame.setLayout(null)` (your frame will display nothing until you setBounds of your label)
- labels.setBounds(x,y,width,height)

Or :
- frame.pack() <- it'll adjust label size considering the content.

---

33 Panels

Panels are a GUI component that functions as a container to hold other components.
We init it like that : `JPanel panel = new JPanel();`. It's like a div ngl.
The methods are identic to labels but you can add labels to a panel : `panel.add(label)` and then add multiple panel to a frame : `frame.add(Panel)`.
This way you can position multiple labels into your panel and then position multiple panel in your frame to make a great GUI.

---

34 Buttons

JButton : a button that perform an action when clicked on.
We init it like that : `JButton button = new JButton()`. It shares the same method as Labes for styling.
However in it class you need to make some setup. `public class MyFrame extends JFrame implements ActionListener`.
If you create your object in the constructor you can write : `button.addActionListener(this)`, or `button.addActionListener(e-> /*actiontoperform*/)`.
You can make it unclickable : `setEnabled(false)` or remove annoying border around text : `setFocusable(false)`.
To define action performed you can build a method in your class :

— @Override
— public void actionPerformed(ActionEvent e){
— if(e.getSource() == button)
— /* action performed*/
— }

---

35 LayoutManager

LayoutManager : Defines the natural layour for components within a container.
There is 3 common manager :

35.1 BorderLayout

A borderLayout places components in five areas : NORTH, SOUTH, WEST , EAST , CENTER. All extra space placed in the center area.
Within a frame, you place panel like this : `frame.add(panel,BorderLayout.North)` (you can add panel to panel aswell).
To have margin between panel you can do : `frame.setLayout(new BorderLayout(widthMargin, heightMargin)`.

35.2 FlowLayout

A FlowLayout places components in a row, sized at their preferred size. If the horizontal space in the container is too small, the FlowLayout class uses the next available row.
We set it like that : `frame.setLayout(newFlowLayout(FlowLayout.LEADING/CENTER/TRAILING, horizontalMargin,verticalMargin)`

— LEADING -> stick to left

— CENTER -> stick to center
— TRAILING -> stick to right

35.3 GridLayout

A GridLayout places components in a grid of cells. Each component takes all the available space within its cell, and each cell is the same size.
We set it like that : `frame.setLayout(new GridLayout(numberofCol,numberofRow, HorizontalMargin,VerticalMargin))`

36 LayeredPane

JLayeredPane = Swing container that provide a third dimension for positioning components.
We init it like that : `JLayeredPane layeredPane = new JLayeredPane();`
Then we add label to it : `layeredPane.add(labelX)`
To setup z-index, there is two ways :
— Using JLayeredPane Layer :

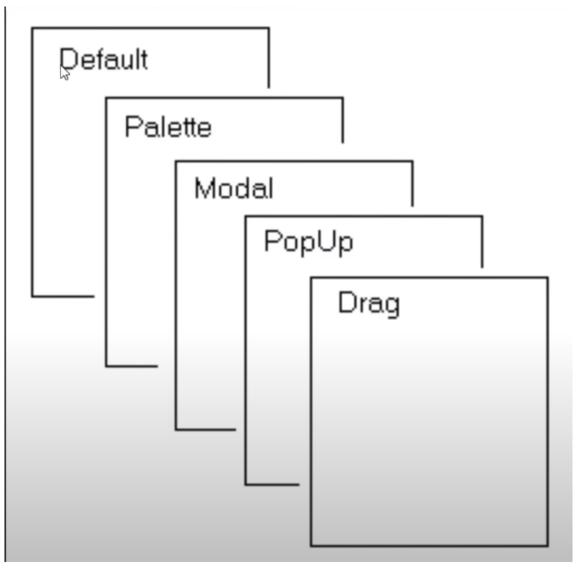


FIGURE 1 – layers

and so you use : `layeredPane.add(labelX,JLayeredPane.DEFAULT_LAYER)`
— Using Integer values : `Integer.valuesOf(0)`

37 frame.dispose()

`frame.dispose()` is a method used in the AWT (Abstract Window Toolkit) framework to handle GUI (Graphical User Interface) components, specifically for `Frame` objects (which are top-level windows). When you call `frame.dispose()`, it performs the following actions :

1. **Destroys the Frame** : It releases all of the native screen resources that are being used by this `Frame` and all of its subcomponents.
2. **Hides the Frame** : It effectively hides the frame, removing it from the screen.
3. **Releases Resources** : It frees up resources like memory and handles associated with the frame, making them available for other applications or processes.
4. **Notifies the Window System** : It informs the window system that the window has been closed, and any associated event listeners will be notified as well.

This method is useful for managing the lifecycle of a window, especially when you want to close a window and ensure that all resources are properly cleaned up.

---

38 JOptionPane

It pop up a standard dialog box that prompts users for a value or informs them of something.
We init it like that : `JOptionPane.showMessageDialog(parentComponent, message,title , messageType)`
messageType : It only change the dialog box icon depending on the nature of your message.
— JOptionPane.PLAIN_MESSAGE
— JOptionPane.INFORMATION_MESSAGE
— JOptionPane.QUESTION_MESSAGE
— JOptionPane.WARNING_MESSAGE
Confirm pop up :
We init it like that : `JOptionPane.showConfirmDialog(parentComponent,message,title, messageType)`

— messageType : `YES_NO_CANCEL_OPTION`
Return an int value.
Input pop up :
We init it like that : `JOptionPane.showInputDialog(String)` Return a string value.
General pop up :

We init it like that : `JOptionPane.OptionDialog(parentComponent,message,title,optionType, messageType,icon,options,initialValue)`
The `optionType` parameter determines the set of options that appear at the bottom of the dialog. It can take one of the following constant values defined in `JOptionPane` :

— `JOptionPane.DEFAULT_OPTION` : No options are displayed, making it an information-only dialog.
— `JOptionPane.YES_NO_OPTION` : Displays `Yes` and `No` buttons.
— `JOptionPane.YES_NO_CANCEL_OPTION` : Displays `Yes`, `No`, and `Cancel` buttons.
— `JOptionPane.OK_CANCEL_OPTION` : Displays `OK` and `Cancel` buttons.

The `options` parameter is an array of `Object` that represents the possible choices that the user can select. This allows you to provide custom text for the buttons. If `options` is `null`, the default buttons defined by `optionType` are used.

```
Object[] options = {"Option 1", "Option 2", "Option 3"};
```

When `options` is provided, the `optionType` parameter is ignored, and only the custom buttons are displayed.

The `initialValue` parameter specifies the option that should be initially selected when the dialog appears. This value should match one of the elements in the `options` array. If `initialValue` is `null`, no option is selected by default.

```
Object initialValue = "Option 2";
```

39 Textfield

JTextField : A GUI textbox component, that can be used to add, set, or get text.
We init it like that : `JTextField textField = new JTextField();`
We can use the same methods as label for styling.
Methods :

— textField.getText() <- return a string
— textField.setText() <- prefilled text
— text.Field.setEditable(false) <- make the textField uneditable

40 Checkbox

JCheckBox = A GUI component that can be selected or deselected.
We init it like that : `JCheckBox checkBox = new JCheckBox()`
We can use the same methods as label for styling.
Methods :

— checkBox.isSelected() <- return boolean value

41 Radio buttons

JRadioButton : One or more buttons in a grouping in which only 1 may be selected per group.
We init it like that : `JRadioButton radiobtn = new JRadioButton(btn_name);`.
We should init a ButtonGroup aswell : `ButtonGroup group = new ButtonGroup();` in which we add every btn by :`group.add(btn_VarName)`.
Add action listener to each btn : btn_Var_Name`.addActionListener(MyFrame)`.
We can use the same methods as label for styling.
The ActionEventListener looks like that :

— public void actionPerformed(ActionEvent e){
— if(e.getSource()==*btn_name*) {
— /* action */
— }

---

42 ComboBox

JComboBox : A component that combines a button or editable field and a drop-down list.
We should init an array , let's called it Options.
We init it like that : `JComboBox comboBox = new JComboBox(Options);`.
In order to retrieve what's selected we can use : `comboBox.getSelectedItem();` or `comboBox.getSelectedIndex();`

— We can make it editable : `comboBox.setEditable(true)` <- people can search by typing.
— We can return the number of item in the CombBox : `comboBox.getItemCount()`.
— We can add item at the end : `comboBox.addItem(item)`.
— or at a certain index : `comboBox.insertItemAt(item,index)`
— or remove it : `comboBox.removeItem(item)`
— or from his index : `comboBox.removeItemAt(index)`
— or just make the comboBox empty : `comboBox.removeAllItems()`
— Put a default selected value : `comboBox.setSelectedIndex(index)`

---

43 Slider

JSlider : GUI component that lets user enter a value by using an adjustable sliding knob on a track.
We init it like that : `slider = new JSlider(min,max,slider_starting_point)`.
We can use the same methods as label for styling.

- add PaintTicks -> `setPaintTicks(true)`
- Set litte ticks spaces -> `setMinorTickSpacing(ticks_recurrency)`
- Set big ticks spaces -> `setMajorTickSpacing(ticks_recurrency)`
- Set number for major tick -> `setPaintLabels(true)`
- make it vertical : `setOrientation(SwingConstants.VERTICAL)`
- make it horizontal : `setOrientation(SwingConstants.HORIZONTAL)`
- get the value it's pointing to : `slider.getValue()`

---

44 Progress bar

Progress bar : Visual aid to let the user know that an operation is processing.
We init it like that : `JProgressBar bar = new JProgressBar`.
We add the bar to the frame : `frame.add(bar)`.

- Set a pourcentage % to your bar : `setStringPainted(true)`
- value initialised by : `bar.setValue(0)`

Little example of progress bar fill() method :

```java
public void fill() {
    int counter = 0;
    while(counter <= 100) {
        bar.setValue(counter);
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        counter += 1;
    }
    bar.setString("Done");
}
```

---

45 Menu bar

We init it like that : `JMenuBar menuBar = new JMenuBar()`.
We must add content to the MenuBar :
`JMenu fileMenu = new JMenu("File"); menuBar.add(fileMenu)`.
To each menu we should add Item : `JMenuItem item = new JMenuItem(Name); fileMenu.add(item)`.
We should add Action Listener to each Item : `item.addActionListener(MyFrame)`.
You can even make shortcut : `item.setMnemonic(KeyEvent.VK_L)` (you just have to press L to open the item).
It works fro JMenu aswell : `fileMenu.setMnemonic(keyEvent.VK_F)` (press alt+F).

We can use the same methods as label for styling.

---

46 Select a file

JFileChooser : A GUI mechanism that let's a user choose a file.
We init it like that : `JFileChooser fileChooser = new JFileChooser()`.
And init the Current Directory : `fileChooser.setCurrentDirectory(new File(path))`.

— select file to open : `fileChooser.showOpenDialog(null)`
— store it's value as an int
— check if the file is approved : `if(response == JFileChooser.APPROVE_OPTION)`
— if so get the file : `File file = new File(fileChooser.getSelectedFile() .getAbsolutePath())`
— select file to save : `fileChooser.showSaveDialog(null)`

---

47 Color Chooser

JColorChooser : a GUI mechanism that let's a user choose a color.
We init it like that : `JColorChooser colorChooser = new JColorChooser();`.
We open the dialog like this :
`Color color = JColorChooser.showDialog(component,title,initialColor)`

---

48 KeyListener

In the using class, you should add : `implements KeyListener`.
We init it like that : `MyFrame.addKeyListener(MyFrame)`.
We should define 3 methods :

48.1 keyTyped

— public void keyTyped(KeyEvent e){
— switch(e.getKeyChar()){
— case 'a' : /* what you want */ ; break ;
— /* do this for any char */
— }
— }

48.2 keyPressed

— public void keyPressed(KeyEvent e){
— switch (e.getKeyCode()){
— case 'int' : /* what you want */ ; break ;
— }
— }

48.3 keyReleased

— public void keyReleased(KeyEvent e){
— /* switch case to filter each key as u want*/
— }

---

49 Mouse Listener

In the using class, you should add : `implements MouseListener`.
We init it like that : `MyFrame.addMouseListener(MyFrame)`.
We should define 5 methods :

49.1 mouseClicked

— public void mouseClicked(MouseEvent e){
— /* what u want */
— }

49.2 mousePressed

— public void mouseClicked(MouseEvent e){
— /* what u want */
— }

49.3 mouseReleased

— public void mouseReleased(MouseEvent e){
— /* what u want */
— }

49.4 mouseEntered

— public void mouseEntered(MouseEvent e){
— /* what u want */
— }

49.5 mouseExited

— public void mouseExited(MouseEvent e){
— /* what u want */
— }

50 Drag And Drop

We init it like that : `DragPanel dragPanel = new DragPanel()` .
Our DragPanel class is built like this :

```java
public class DragPanel extends JPanel {
    ImageIcon image = new ImageIcon(imageName);
    final int WIDTH = image.getIconWidth();
    final int HEIGHT = image.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);
        image.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();
        }
    }
}
```

51 Key Bindings

Key bindings is bind an Action to a KeyStroke, don't require you to click a component to give it focus. All Swing components use key bindings. You can increase flexibility compared to KeyListeners and you can assign key strokes to individual Swing components but more difficult to utilize and set up :(.
Let's take the example of a game where you would like to bind Up , Down , Left and Right.
Game class :

```java
public class Game{
    JFrame frame;
    JLabel label;
    Action upAction;
    Action downAction;
    Action leftAction;
    Action rightAction;
    Game(){
        frame = new JFrame("KeyBinding Demo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(420,420);
        frame.setLayout(null);

        label = new JLabel();
        label.setBackground(Color.red);
        label.setBounds(100,100,100,100);
        label.setOpaque(true);

        upAction = new UpAction();
        downAction = new DownAction();
        leftAction = new LeftAction();
        rightAction = new RightAction();

        label.getInputMap().put(keyStroke.getKeyStroke("UP"),
           "upAction");
        label.getActionMap().put("upAction",upAction);
        label.getInputMap().put(keyStroke.getKeyStroke("DOWN")
           , "downAction");
        label.getActionMap().put("downAction",downAction);
        label.getInputMap().put(keyStroke.getKeyStroke("RIGHT"
           ), "rightAction");
        label.getActionMap().put("rightAction",rightAction);
        label.getInputMap().put(keyStroke.getKeyStroke("LEFT")
           , "leftAction");
        label.getActionMap().put("leftAction",leftAction);

        frame.add(label);
        frame.setVisible(false);
    }
    public class UpAction extends AbstractAction{

        @Override
        public void actionPerformed(ActionEvent e){
            label.setLocation(label.getX(), label.getY()-10);
        }
    }
    public class DownAction extends AbstractAction{
        @Override
        public void actionPerformed(ActionEvent e){
            label.setLocation(label.getX(), label.getY()+10);
        }
    }
    public class LeftAction extends AbstractAction{
        @Override
        public void actionPerformed(ActionEvent e){
            label.setLocation(label.getX()-10, label.getY());
        }
    }
    public class UpAction extends AbstractAction{
        @Override
        public void actionPerformed(ActionEvent e){
                label.setLocation(label.getX()+10, label.getY
                   ());
        }
    }
```

```
}
```

52 2D Graphics

2D Graphics is a lib made to display drawing. In order to use it we will create a paint class.
Here's a quick example of how can we use this lib.

```
public void paint(Graphics g){
    Graphics2D g2D = (Graphics2D) g;

    g2D.setStroke(new BasicStroke(size)); \\ Set up the width of the line.
    g2D.setPaint(Color.*color*); \\ Set up the color of the line.
    g2D.drawline(x1,y1,x2,y2); \\ x1 , y1 are the inital point and x2 , y2 the final one and it draw line to the inital to the end point.
    g2D.drawRect(x,y,width,height); \\ x,y are the top-left point position;
    g2D.fillRect(x,y,width,height); \\ same as drawRect but it willl be filled up.
    g2D.drawOval(x,y,width,height); \\ fillOval for filled one.
    g2D.drawArc(x,y,width,height,startAngle, arcAngle); \\ fillArc ...
    g2D.drawPolygon(xPoints,yPoints,nPoints); \\xPoints and yPoints are both arrays of integers. We have aswell fillPolygon.
    g2D.setFont(new Font("fontname"));
    g2D.drawString(String,x,y);
    g2D.drawImage(img,x,y,observer); \\ The ImageObserver interface is used to receive notifications about the status of an image as it is being loaded. This is particularly useful for images that are loaded asynchronously, such as images loaded over a network or from a disk. When the image is being drawn, the ImageObserver can be notified about the progress of the loading process, including when the image is completely loaded, when an error occurs, or when more information becomes available.
}
```

53 2D animation

53.1 MyFrame Class

The `MyFrame` class extends `JFrame` to set up the main window and add the custom panel.

```java
import java.awt.*;
import javax.swing.*;

public class MyFrame extends JFrame {
    MyPanel panel;

    MyFrame() {
        panel = new MyPanel();
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.add(panel);
        this.pack();
        this.setLocationRelativeTo(null);
        this.setVisible(true);
    }
}
```

53.2 MyPanel Class

The `MyPanel` class extends `JPanel` and implements `ActionListener` to handle animation logic.

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MyPanel extends JPanel implements ActionListener
   {
    final int PANEL_WIDTH = 500;
    final int PANEL_HEIGHT = 500;
    Image enemy;
    Timer timer;
    int xVelocity = 1;
    int yVelocity = 1;
    int x = 0;
    int y = 0;

    MyPanel() {
        this.setPreferredSize(new Dimension(PANEL_WIDTH,
           PANEL_HEIGHT));
        this.setBackground(Color.black);
        enemy = new ImageIcon("enemy.png").getImage();
        timer = new Timer(10, this);
        timer.start();
    }

    public void paint(Graphics g) {
        super.paint(g); // Paint background
        Graphics2D g2D = (Graphics2D) g;
        g2D.drawImage(enemy, x, y, null);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (x >= PANEL_WIDTH - enemy.getWidth(null) || x < 0)
           {
            xVelocity *= -1;
        }
        x += xVelocity;

        if (y >= PANEL_HEIGHT - enemy.getHeight(null) || y <
           0) {
            yVelocity *= -1;
        }
        y += yVelocity;
        repaint();
    }
}
```

53.3 Key Features

- **Animation Loop :** Uses `javax.swing.Timer` to create an animation loop that updates the position of the image at regular intervals.
- **Collision Detection :** Checks for collisions with the panel edges and reverses the direction of movement accordingly.
- **Image Rendering :** Utilizes `Graphics2D` to draw the image on the panel.

53.4

Code Explanation The `MyPanel` class is the core component of the animation :

```java
public void paint(Graphics g) {
    super.paint(g); // Paint background
    Graphics2D g2D = (Graphics2D) g;
    g2D.drawImage(enemy, x, y, null);
}
```

The `paint` method is overridden to draw the image at the current coordinates. The `actionPerformed` method updates the position of the image and checks for collisions :

```java
@Override
public void actionPerformed(ActionEvent e) {
    if (x >= PANEL_WIDTH - enemy.getWidth(null) || x < 0) {
        xVelocity *= -1;
    }
    x += xVelocity;

    if (y >= PANEL_HEIGHT - enemy.getHeight(null) || y < 0) {
        yVelocity *= -1;
    }
    y += yVelocity;
    repaint();
}
```

53.5 Conclusion

This project demonstrates the basics of 2D animation in Java using Swing and AWT. It covers setting up a JFrame and JPanel, loading and rendering images, and creating an animation loop with collision detection.

54 Generics

Enable types (classes and interfaces) to be parameters when defining : classes, interfaces and methods.
A benefits is to eliminate the need to create multiple versions of methods or classes for various data types.
Use 1 version for all reference data types.

54.1 Generics methods

```java
public static <Name> void displayArray(Name[] array){ \\ Name
   can be changed by everything.
    for(Name x: array){
        System.out.println(x+"");
    }
    System.out.println()
}
```

By using this syntaxe , displayArray can handle Integer array , double and even char array.

54.2 Generics classes

```
public class MyGenericClass <Name> { \\ Works aswell if we
   take in count multiple types.
    Name x;

    MyGenericClass(Name x){
        This.x = x;
    }

    public Name getValue(){
        return x;
    }
}
public class MyGenericClass2 <Name,Name1>{}
MyGenericClass<Integer> myInt = new MyGenericClass<>(1);
MyGenericClass<Double> myDouble = new MyGenericClass<>(3.14);
MyGenericClass<Character> myChar = new MyGenericClass<>('@');
MyGenericClass2<Integer,Integer> twoInt = new MyGenericClass2
   <1,2>;
```

54.3 Bounded Types

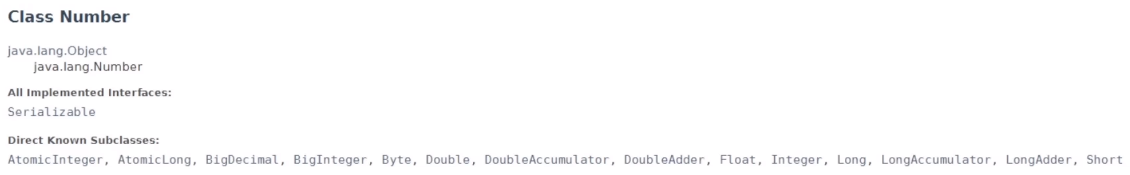
Bounded Types : you can create the objects of a generic class to have data of specific derived types ex.Number.

```
public class MyGenericClass3 <Thing extends Number, Thing2
   extends Number>{
    Thing x;
    Thing2 y;

    MyGenericClass3(Thing x, Thing2 y){
        this.x = x;
        this.y = y;
    }

    public Thing2 getValue(){
        return y;
    }
}
```

The extends Number is a reference to the class Number.



Class Number

java.lang.Object
    java.lang.Number

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
AtomicInteger, AtomicLong, BigDecimal, BigInteger, Byte, Double, DoubleAccumulator, DoubleAdder, Float, Integer, Long, LongAccumulator, LongAdder, Short

FIGURE 2 – layers

By using it, we reduced the scope of type by only the Direct Known Subclasses.

55 Serialization

Serialization : The process of converting an object into a byte stream. Persists (save the state) the object after program exits. This byte stream can be saved as a file or sent over a network and to a different machine. Byte stream can be saved as a file (.ser) which is platform independant (Think of this as if you're saving a file with the object's information).

Steps to Serialize :

— Your object class should implement Serializable interface
— add import java.io.Serializable ;
— FileOutputStream fileOut = new FileOutputStream(fileName.ser) ;
— ObjectOutputStream out = new ObjectOutputStream(fileOut) ;
— out.writeObject(objectName) ;
— out.close() ; fileOut.close() ;

Deserialization : The reverse process of converting a byte stream into an ojbect. (Think of this as if you're loading a saved file).

Steps to Deserialize :

— Declare your object (don't instantiate)
— Your class should implement Serializable interface
— add import java.io.Serializable ;
— FileInputStream fileIn = new FileInputStream(file path) ;
— ObjectInputStream in = new ObjectInputStream(fileIn) ;
— objectNam = (Class) in.readObject() ;
— in.close() ; fileIn.close() ;

55.1 Advanced

Important notes :
- children classes of a parent class that implements Serializable will do so as well.
- static fields are not serialized (they belong to the class, not an individual object)
- the class's definition ('class file') itself is not recorded, cast it as the object type
- Fields declared as "transient" aren't serialized , they're ignored
- serialVersionUID is a unique version ID

serialVersionUID : is a unique ID that functions like a version , verifies that the sender and receiver of a serialized object, have loaded classes for that object that match. Ensures object will be compatible between machines. Number must match, otherwise this will cause a InvalidClassException. A SerialVersionUID will be calculated based on class properties, members , etc. A serializable class can decalre its own serialVersionUID explicitly (recommended).
`long serialVersionUID = ObjectStreamClass.lookup(user.getClass()).getSerialVersionUID();`

---

56 TimerTask

Timer : A facility for threads to schedule tasks for future execution in a background thread.
TimerTask : A task that can be scheduled for one-time or repeated execution by a Timer.
We init it like that : `Timer time = new Timer();`

```
TimerTask Task = new TimerTask();{
    @Override
    public void run(){
        /* action */
    }
};
//timer.schedule(task,time(ms))
Calendar date = Calendar.getInstance();
date.set(Calendar.YEAR,2024);
date.set(Calendar.MONTH ,Calendar.MAY);
date.set(Calendar.DAY_OF_MONTH,27);
date.set(Calendar.HOUR_OF_DAY,0);
date.set(Calendar.MINUTE,0);
date.set(Calendar.SECOND,0);
date.set(Calendar.MILLISECOND);
timer.schedule(task,date.getTime()); <- when your pc hit
   27/05/2024 at midnight the task will run.
timer.scheduleAtFixedRate(task,firstTime,period(ms));
```

57 Threads

threads : A thread of execution in a program (kind of like a virtual CPU) The JVM allows an application to have multiple threads running concurrently Each thread can execute parts of you code in parallel with the main thread Each thread has a priority. Threads with higher priority are executed in preference compared to threads with a lower priority

The Java Virtual Machine continues to execute threads until either of the following occurs 1. The exit method of class Runtime has been called 2. All user threads have died

When a JVM starts up, there is a thread which calls the main method This thread is called “main”

Daemon thread is a low priority thread that runs in background to perform tasks such as garbage collection JVM terminates itself when all user threads (non-daemon threads) finish their execution.
To start a thread : `thread.start();` In order to check the actual running thread : `Thread.activeCount();`.
And to get their name : `Thread.currentThread().getName());`.
You can set a name for the currentThread : `Thread.currentThread().setName(String)`.
In order to check his prority : `Thread.currentThread().getPriority()`.
To set the priority : `Thread.currentThread().setPriority(int)`.
Check if the current thread is alive or not : `Thread.currentThread().isAlive();`
<- Return a boolean value.
You can pause a thread by :
Thread.sleep(ms) ;
To make your thread a Daemon thread : `thread.setDaemon(false/true);`.

58 Multithreading

Multithreading : Process of executing multiple threads simultaneously. Helps maximum utilization of CPU. Threads are independant , they don’t affect the execution of other threads. An exception in one thread will not interrupt other threads. It’s useful for serving multiple clients, multiplayer games, or other mutually independent tasks.
Create a class for each thread :

```java
public class MyThread extends Thread{
    /* Any kind of method you want*/
}
```

Extends your classs with Thread to it shares the methods.
Either you can create thread instance by creating runnable class :

```java
public class MyRunnable implements Runnable{
    @Override
    public void run(){
    /* action */
    }
}
// in your main class
MyRunnable runnable1 = new MyRunnable();
Thread thread2 = new Thread(runnable1);
```

I think it's better to use the Runnable methods because you can still extends your "MyRunnable" class to other class but not MyThread because it's already extended to Thread.
join method : given 2 threads, thread1 and thread2 if you type `thread1.start(); thread1.join();thread2.start()`, it will wait the end of thread1 before starting thread2. You can aswell use `thread1.join(ms)`. It will wait X millisecond before the start of thread2.

---

59 Packages

Package : is a collection of multiple class.
You can import class from a package by : `PackageName.ClassName`

---

60 Compile

Compile and run Java with Command Prompt :

— Make sure you have a Java JDK installed
— use a text editor and save a file as .java
— Open Command Prompt (windows / Linux) or Terminal (Mac)
— set path=C :Files-13.0.1(where JDK is located)
— cd path to your java file
— javac FileName.java
— java FileName (to run a .class file ,it's portable)

---

61 Executable (.jar)

Create an executable jar with Eclipse IDE
— Right-click on the Java project
— Export
— Java > Runnable JAR file
— At launch configuration, select your project
— At export destination, select where you want the JAR file to be exported
— Click Finish
Create an executable jar with IntelliJ
— File > Project Structure > Artifacts > (+) > JAR > From modules with dependencies
— Main class : select the class containing your main method
— Build > Build Artifacts > Build
Create an executable jar with CMD (Windows), Terminal (Linux/macOS)
— Open CMD (Command Prompt) on Windows, or Terminal on Linux/macOS
— Navigate to your project directory :

```
cd path/to/your/project
```

— Compile your Java files into the ‘bin‘ directory :

```
javac -d bin src/*.java
```

— Navigate to the ‘bin‘ directory :

```
cd bin
```

— Create a manifest file (e.g., ‘manifest.txt‘) with the following content :

```
Main-Class: com.example.MainClass
```

— Create the JAR file using the ‘jar‘ command :

```
jar cfm MyExecutable.jar manifest.txt com/example/*.class
```

— Your executable JAR file (‘MyExecutable.jar‘) is now created in the ‘bin‘ directory.

61.1 Detailed Steps for Each Operating System

61.1.1 Windows

1. Open Command Prompt :

```
cmd
```

2. Navigate to your project directory and follow the steps listed above.

61.1.2 Linux

1. Open Terminal.
2. Navigate to your project directory and follow the steps listed above.

61.1.3 macOS

1. Open Terminal.
2. Navigate to your project directory and follow the steps listed above.

Note : The commands are the same across Windows, Linux, and macOS when using the ‘javac‘ and ‘jar‘ tools. Ensure you have the JDK installed and properly configured in your system’s PATH environment variable.

62 Conclusion

Thanks for reading the entire pdf, i hope it's usefull to you. If you need any help contact me on Github