6. Graphical API

6.1. Standard Java library API

The Java ***application program interface (API)*** also known as ***library*** provides classes for developing predefined classes and interfaces for developing Java applications, it consists of a set of separate programs. Java is a powerful language that can be used in many domains: you can use it to develop applications for servers, desktop computers, and small devices. It comes in three editions:

- **(Java SE) Java Standard Edition** to develop client-side applications. The applications can run as a standalone program.
- **(Java EE) Java Enterprise Edition** used for server-side applications, such as Java servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF).
- **(Java ME) Java Micro Edition** designed for mobile devices applications, such as sensors, cell phones.

Java SE also known as ***Java Standard Library*** is a highly utilized platform for creating diverse applications relating to diverse domains like desktop applications, web applications and enterprise systems. It remains a popular choice among developers due to its portability, reliability, and wide community support.

Java SE includes many features such as:

- **Java Virtual Machine (JVM):** provides a robust executing environment for running Java code.
- **Versatile Libraries:** Java SE offers in a comprehensive way a set of standard libraries, known as the Java API (Application Programming Interface). These libraries provide essential functionality for tasks such as input/output, networking, data structures, utilities, and more.
- **Security:** Java SE provides a solid security model to ensure the safe execution of java code.
- **Development Tools:** Java SE delivers a set of development tools such as Java Development Kit (JDK) and Integrated Development Environment (IDE).
- **Updates and Versions:** regular updates is a key feature that offered by Java SE which receives updates and new versions, enhancements, bug fixes, and more.

The libraries in Java (standard API) encompass a comprehensive set of predefined classes and packages for a large variety of programming tasks. Here are some key features of the Core libraries:

java.lang: includes classes for basic data types, exceptions, threads, and the Object class. This package contains classes that are fundamental to the design of the Java programs. This package is automatically imported into any Java program.

java.util: (as seen in previous chapter) The **java.util** package provides classes, packages and data structures such as lists, sets, maps, queues, and more data collection. Developers can also working with dates and times by using the classes ***Date*** and ***Calendar*** provided in this package.

java.security: includes classes for cryptography and more security model like secure random number. The ***java.security*** package contains classes related to Java's security model.

java.text: includes classes for parsing and formatting text, dates, and numbers. This package is suitable for internationalization of Java programs.

java.io: basics input and output operations are included in this package, which allow developers to read and write files, streams, and other input/output sources. It includes file I/O, stream I/O, and object serialization.

6.2. Java Natives Interface (JNI)

Native code means that we can use other language code like C in our java programs. The need for using native code is capital essentially in:

- An existing library in other language that we want to reuse without to rewrite it in java
- The need to manage some hardware
- When performance optimization is required …etc.

To use the native code, JDK offers a bridge between the bytecode and the native code (usually written in C or C++).

The bridge provided by java has three steps that must be respected to achieve the correctness of running code:

Step 1: Write a Java Class that include a Native Method

Step 2: Generate Header File (*.h)

Step 3: Compile the C Code

Example:

Let us take an example that use the *add library* written in c language

```
// AddNum.c
#include <stdio.h>
int add(int a, int b) {
    return a + b;  }
```

Step1: Write a Java Class:

```
public class AddJNI {
    static {
        // Load the native library
        System.loadLibrary("Addnum"); }
    // Declare the native method
```

```
    public native void add();
    public static void main(String[] args) {
        // Create an instance of the class
        AddJNI addnumbers = new AddJNI();
        // Call the native method
        Float result = addnumbers.add(2,3);}}
```

Step2: Generate Header File

Use the ***javac*** to compile the Java class and the ***javah*** tool to generate the header file.

```
javac AddJNI.java
javah -jni AddNum
```

This will generate a header file named **AddNum*.h***

Step 3: Compile the C Code

Use the gcc compiler to compile the addNum.c. This will result a file named addNum.dll

Note: we use the path **C:\jniexample** as example :

```
C:\ gcc -shared -o C:\jniexample\addnum.dll
```

Running the Java Program

Use the java application to run our example

```
C:\ java addJni
```

This command will load the native library (addNum.dll) and call the native method.

6.3. AWT and SWINGx library

In java, the Abstract Window Toolkit (AWT) and Swing are two important package that provide the main API for graphics programming. These libraries offer classes and interfaces for creating graphical user interfaces (GUIs), drawing graphics, and handling events.

6.3.1. AWT (Abstract Window Toolkit): in this library, we found the most used classes for working in graphics mode like color, font …etc.

java.awt.Graphics: provides methods for drawing shapes, text, and images on the screen. It is used when calling the paint() method of components.

java.awt.Color: utilized for working with colors in the RGB model (red, green, blue).

java.awt.Font: Represents fonts for rendering text.

java.awt.Canvas: A space that can be drawn.

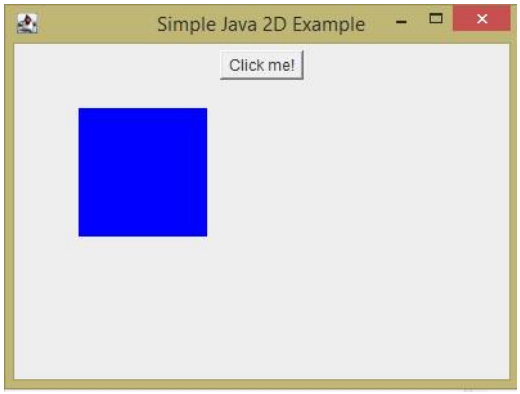
java.awt.Frame and **java.awt.Panel**: Basic containers for organizing components in a GUI.

Example

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Button;
```

```
import javax.swing.JFrame;
import javax.swing.JPanel;

public class cl1 extends JFrame {

    public cl1() {
        super("Simple Java 2D Example");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(new MyPanel());  }

    public static void main(String[] args) {
        cl1 example = new cl1();
        example.setVisible(true); }

    class MyPanel extends JPanel {
        public void paintComponent(Graphics g) {
            super.paintComponent(g);
             Button button = new Button("Click me!");
             add(button);
             Graphics2D g2d = (Graphics2D) g;
             g2d.setColor(Color.BLUE);
             g2d.fillRect(50, 50, 100, 100);}}}
```

This example result:



6.3.2. Swingx:

SwingX offers several components, utilities, and enhancements to the core Swing library, which developers can create rich and complex user interfaces (GUI).

javax.swing.JFrame: A top-level container for Swing applications.

javax.swing.JPanel: A container that can hold other components and can be used for grouping components.

javax.swing.JButton, javax.swing.JLabel, etc.: Several components for creating GUI elements like buttons, labels, and other.

javax.swing.JComponent: The parent class for all Swing components.