JNI dynamic library example

Write a program to create Dynamic Link Library for any mathematical operation and write an application program to test it. (Java Native Interface / Use VB or VC++).

```
#include <jni.h>
#include <stdio.h>
#include "A3.h"

// NOTE: The contents of this file can be referenced from A3.h which is the generated header file
// Refer explanation for more info: https://git.kska.io/sppu-te-comp-content/SystemsProgrammingAndOperatingSystem/src/branch/main/Codes/Group%20A/Assignment-A3/EXPLANATION.md

JNIEXPORT jint JNICALL Java_A3_add(JNIEnv *env, jobject obj, jint a, jint b) { // Function for addition
    jint result = a + b;
    printf("\n%d + %d = %d\n", a, b, result);
    return result; // Return the result
}

JNIEXPORT jint JNICALL Java_A3_sub(JNIEnv *env, jobject obj, jint a, jint b) { // Function for subtraction
    jint result = a - b;
    printf("\n%d - %d = %d\n", a, b, result);
    return result; // Return the result
}

JNIEXPORT jint JNICALL Java_A3_mul(JNIEnv *env, jobject obj, jint a, jint b) { // Function for multiplication
    jint result = a * b;
    printf("\n%d * %d = %d\n", a, b, result);
    return result; // Return the result
}

JNIEXPORT jint JNICALL Java_A3_div(JNIEnv *env, jobject obj, jint a, jint b) {  // Function for division
    if (b == 0) {
        printf("Error: Division by zero.\n");
        return 0; // Return 0 or handle error appropriately
    }
    jint result = a / b;
    printf("\n%d / %d = %d\n", a, b, result);
    return result; // Return the result
}

----------
```

Write a program to create Dynamic Link Library for any mathematical operation and write an application program to test it. (Java Native Interface / Use VB or VC++).

```
// Importing basic stuff
import java.io.*; // Used for I/O operations
import java.util.*; // Contains basic utilities

class A3 {
  // Class name has to be same as file name for Java

  static {
    // Used for loading the .so (on Linux) or .dll (on Windows) file when running
    // This is the main so called "dynamic library"
    System.loadLibrary("A3");
  }

  // Function declaration
  // private indicates the function is private, duh!
  // Use of native indicates the function body will be written in a language other than Java, such as C/C++
  private native int add(int a, int b); // For addition
  private native int sub(int a, int b); // For subtraction
  private native int mul(int a, int b); // For multiplication
  private native int div(int a, int b); // For division

  public static void main(String[] args) { // the main function
    Scanner sc = new Scanner(System.in); // For taking input

    int a, b;// Declaring variables for calculation
    int choice = 0; // Declaring variable for switch-case

    // Take input for a and b values
    System.out.print("\nValue of a:\t");
    a = sc.nextInt();
    System.out.print("\nValue of b:\t");
    b = sc.nextInt();

    // Main menu
    while (true) {
      System.out.println("----- MAIN MENU -----");
      System.out.println("1 -> Addition");
      System.out.println("2 -> Subtraction");
      System.out.println("3 -> Multiplication");
      System.out.println("4 -> Division");
      System.out.println("5 -> Exit");
      System.out.println("Choose an option:\t");
      choice = sc.nextInt();

      switch (choice) {
        case 1:
```

```
        System.out.println("Result: " + new A3().add(a, b));
        break;
                        case 2:
                        System.out.println("Result: " + new A3().sub(a, b));
                        break;
                        case 3:
                        System.out.println("Result: " + new A3().mul(a, b));
                        break;
                        case 4:
                            System.out.println("Result: " + new A3().div(a, b));
                            break;
                        case 5:
                            System.out.println("## END OF CODE");
                            System.exit(0);
                        default:
                            System.out.println("Please choose a valid option.");
                            break;
    }
   }
  }
}
```

Write a program to create Dynamic Link Library for any mathematical operation and write an application program to test it. (Java Native Interface / Use VB or VC++).

```
// WARNING!!!
// THIS FILE IS INCLUDED ONLY FOR REFERENCE
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class A3 */

#ifndef _Included_A3
#define _Included_A3
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     A3
 * Method:    add
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_A3_add
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     A3
 * Method:    sub
 * Signature: (II)I
```

```
 */
JNIEXPORT jint JNICALL Java_A3_sub
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     A3
 * Method:    mul
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_A3_mul
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     A3
 * Method:    div
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_A3_div
  (JNIEnv *, jobject, jint, jint);

#ifdef __cplusplus
}
#endif
#endif


--------------

this are three different files please explain me each as well as tell me how to run them in ubantu terminal
also give input example if required
```

Perfect — this is a **Java Native Interface (JNI)** assignment that integrates **Java** and **C/C++** via a **Dynamic Link Library (DLL on Windows / .so on Linux)**.
Let's go step by step 🔍

🧩 OVERVIEW

You have **three files**, each with a role:

File	Language	Purpose
A3.java	Java	The *application* that loads and calls native functions
A3.h	C Header	The *auto-generated header* describing native method signatures
A3.c	C source file	The *actual native implementation* of the math functions

---

⚙️ FILE 1 — `A3.java`

📘 Purpose

This is your **Java application** that:

- Takes user input (two integers)
- Offers a menu for Add/Sub/Mul/Div
- Calls native (C) methods using `System.loadLibrary("A3")`

💡 Key Points

- `native` keyword means "this function is implemented in C, not Java".
- `System.loadLibrary("A3")` tells the JVM to load a dynamic library named `libA3.so` (on Linux).
- When you call `new A3().add(a,b)`, it jumps into the compiled C code.

🧮 Example Input

```
rust

Value of a: 10
Value of b: 5
----- MAIN MENU -----
1 -> Addition
2 -> Subtraction
3 -> Multiplication
4 -> Division
5 -> Exit
Choose an option: 1
Result: 15
```

⚙️ FILE 2 — `A3.h`

📘 Purpose

This file is **auto-generated by `javac -h`** and tells C what functions Java expects to exist.

Example:

```
c

JNIEXPORT jint JNICALL Java_A3_add(JNIEnv *, jobject, jint, jint);
```

This means Java expects a function named `Java_A3_add` in C that takes two `int`s and returns an `int`.

⚠️ Note

You **don't edit this file manually**. It's created automatically by the Java compiler.

⚙️ FILE 3 — `A3.c`

📘 Purpose

This is your **C implementation** of the functions declared in **`A3.h`**.

Each function corresponds to one declared **`native`** method in Java.

Example:

```
c

JNIEXPORT jint JNICALL Java_A3_add(JNIEnv *env, jobject obj, jint a, jint b) {
    jint result = a + b;
    printf("\n%d + %d = %d\n", a, b, result);
    return result;
}
```

It uses JNI conventions:

- **`JNIEnv *env`** → JNI environment pointer
- **`jobject obj`** → The calling Java object
- **`jint`** → Java int type in C

💻 HOW TO RUN IN UBUNTU TERMINAL

Follow these **exact steps** 🔽

🧩 STEP 1 — Save Files

Create a folder, say **`JNIExample/`**, and place these three files inside:

```
A3.java
A3.c
```

(You don't need **`A3.h`** yet — it will be generated.)

🧩 STEP 2 — Compile the Java File & Generate Header

```
bash

javac -h . A3.java
```

✅ This does two things:

1. Compiles **`A3.java`** into **`A3.class`**
2. Creates **`A3.h`** automatically in the current folder

🧩 STEP 3 — Compile the C File into a Shared Library

```bash
gcc -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -shared -fPIC -o libA3.so A3.c
```

✅ This creates the dynamic link library:

```
libA3.so
```

> **Note:**
> **`$JAVA_HOME`** must point to your Java installation directory.
> For example:
>
> ```bash
> export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
> ```

▯ STEP 4 — Run the Java Program

```bash
java A3
```

🧠 Sample Run

```yaml
Value of a:   8
Value of b:   4
----- MAIN MENU -----
1 -> Addition
2 -> Subtraction
3 -> Multiplication
4 -> Division
5 -> Exit
Choose an option: 1
8 + 4 = 12
Result: 12

Choose an option: 3
8 * 4 = 32
Result: 32

Choose an option: 4
8 / 4 = 2
Result: 2

Choose an option: 5
## END OF CODE
```

🧩 FILE RELATIONSHIP DIAGRAM

```
pgsql


   +----------------+
   |   A3.java      |
   |  (Java code)   |
   +-------+--------+
           |
           |  javac -h .
           v
   +----------------+
   |   A3.h         |
   | (Header file)  |
   +-------+--------+
           |
           |  Implement functions in C
           v
   +----------------+
   |   A3.c         |
   | (C code)       |
   +-------+--------+
           |
           |  gcc -shared -fPIC -o libA3.so
           v
   +----------------+
   |   libA3.so     |
   | (Dynamic Lib)  |
   +----------------+
           |
           |  System.loadLibrary("A3")
           v
   +----------------+
   |   Java Runtime |
   | (Executes code)|
   +----------------+
```

🧠 Common Errors and Fixes

Error	Cause	Fix
`UnsatisfiedLinkError`	libA3.so not found	Ensure it's in the current directory or in LD_LIBRARY_PATH
`cannot find -ljvm`	Java not installed or JAVA_HOME wrong	Export correct JAVA_HOME path
`error: jni.h: No such file`	Missing include path	Use **`-I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux"`**
Division by zero	b = 0	The code already handles this gracefully

---

✅ Summary

Step	Command	Description
1	**`javac -h . A3.java`**	Compile Java + Generate header
2	**`gcc -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -shared -fPIC -o libA3.so A3.c`**	Compile C file into shared library
3	**`java A3`**	Run the Java program
4	*(Input values)*	e.g., a = 10, b = 5, choice = 1 for addition

Would you like me to give a **short Viva Q&A (20–25 questions)** for this JNI program (like in your other OS assignments)?

now give viva 25 question along with solution for the same code

Excellent ✅ — below are **25 viva voce (oral exam) questions and answers** for your **JNI (Java Native Interface) Dynamic Link Library** program that integrates **Java and C** for mathematical operations.
These are practical + theoretical, just like your university viva would ask.

🧠 VIVA QUESTIONS AND ANSWERS (25)

Topic: *JNI – Creating and Using Dynamic Link Library in Java and C*

---

1. What is JNI?

Answer:
JNI stands for **Java Native Interface**. It allows Java code running in the JVM to **call functions written in other languages** like C or C++ and vice versa.

---

2. Why do we need JNI?

Answer:
We use JNI when:

- We need **faster execution** (C/C++ is faster than Java for certain operations).
- We need to **use existing C/C++ libraries**.
- We want to **access low-level system resources** not directly available in Java.

---

3. What is a Dynamic Link Library (DLL / SO file)?

Answer:
A Dynamic Link Library (DLL on Windows or `.so` on Linux) is a **compiled module** containing code that can be **loaded at runtime** by applications.
It helps in **modular programming** and **code reuse**.

---

4. What is the function of `System.loadLibrary()` in Java?

Answer:
It loads the native shared library (`.dll` or `.so`) into the JVM memory so that Java can call its native functions.

Example:

```
java

System.loadLibrary("A3");
```

loads `libA3.so` on Linux or `A3.dll` on Windows.

---

5. What does the keyword `native` mean in Java?

Answer:
`native` indicates that the **method implementation is written in another language** (like C or C++), not in Java.

---

6. How are native functions connected to Java methods?

Answer:
Through the **header file (`A3.h`)** generated by the command:

```
bash

javac -h .
```

It ensures that Java method signatures match the C implementations.

---

7. What does the `-h` option in `javac` do?

Answer:
It tells the Java compiler to **generate C header files (`.h`)** for all native methods defined in the Java file.

---

8. What command compiles the C code into a shared library in Ubuntu?

Answer:

```
bash

gcc -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -shared -fPIC -o libA3.so A3.c
```

---

9. Why do we use `-I"$JAVA_HOME/include"` in the gcc command?

Answer:
It adds the **include path** where the `jni.h` file is located. This file defines JNI data types and functions.

---

10. What is the purpose of the `JNIEnv *env` parameter in C functions?

Answer:
It's a **pointer to the JNI environment**.
It allows C code to **interact with the Java environment**, like calling Java methods, accessing fields, etc.

11. What does `JNIEXPORT` and `JNICALL` mean in C?

Answer:

- **`JNIEXPORT`** → Makes the function visible for Java linking.
- **`JNICALL`** → Specifies the **calling convention** used by the JNI functions.

---

12. Why do we prefix native functions with `Java_ClassName_methodName`?

Answer:
This is **JNI naming convention**.
It links Java's native method (in **`A3.java`**) with its implementation in C.

Example:

```
c

Java_A3_add
```

matches **`native int add()`** in **`A3.java`**.

---

13. Why is `libA3.so` used instead of `A3.so` on Linux?

Answer:
Java automatically prefixes **"lib"** and suffixes **".so"** when searching for native libraries on Linux systems.

---

14. What is the difference between `dll` and `so`?

Answer:

- **`.dll`** = Dynamic Link Library (Windows)
- **`.so`** = Shared Object (Linux/Unix)
  Both serve the same purpose but differ by OS.

---

15. How do you handle division by zero in your program?

Answer:
In the C code:

```
c

if (b == 0) {
    printf("Error: Division by zero.\n");
    return 0;
}
```

This prevents runtime errors and prints an error message.

---

16. What is the output of `Java_A3_add` when `a=5` and `b=7`?

Answer:
The output printed on terminal:

```
makefile


5 + 7 = 12
Result: 12
```

---

17. How does Java know which C function to call?

Answer:
Through **JNI naming conventions** and **the header file**.
When `A3.add()` is called in Java, JVM searches for `Java_A3_add()` in the loaded native library.

---

18. What are `jint`, `jobject`, and `JNIEnv` in C?

Answer:
They are **JNI-specific data types** used to ensure compatibility between Java and C types:

- `jint` → Java `int`
- `jobject` → Reference to a Java object
- `JNIEnv` → Interface pointer for JNI functions

---

19. What is the role of `-shared` and `-fPIC` in gcc command?

Answer:

- `-shared` → Create a shared (dynamic) library.
- `-fPIC` → Generate **position-independent code**, required for shared libraries.

---

20. How can you call C++ functions using JNI?

Answer:
By using `extern "C"` in C++ source to prevent name mangling:

```
cpp


extern "C" {
JNIEXPORT jint JNICALL Java_A3_add(JNIEnv *env, jobject obj, jint a, jint b) {
    return a + b;
}
}
```

---

21. What happens if the `.so` library is missing when running Java program?

Answer:
The JVM throws:

```
yaml

java.lang.UnsatisfiedLinkError: no A3 in java.library.path
```

because it cannot find or load `libA3.so`.

---

22. What are some common uses of JNI in real-world applications?

Answer:

- Connecting Java with legacy C/C++ libraries.
- Game engines (e.g., OpenGL bindings).
- Device drivers or hardware communication.
- Performance-critical computation modules.

---

23. What is the importance of using `System.exit(0);` in the code?

Answer:
It **terminates the program safely** when the user selects the Exit option.

---

24. How do you test if the shared library was compiled successfully?

Answer:
Run:

```
bash

ls libA3.so
```

If the file exists, compilation succeeded.

---

25. What is the sequence of execution from Java to C and back?

Answer:

1. Java calls a native method (e.g., `add(a, b)`).
2. JVM looks up the function in the loaded library (`libA3.so`).
3. C function executes and returns the result.
4. JVM receives the result and continues Java execution.

🧩 BONUS – Execution Summary

Step	Command	Description
1	`javac -h . A3.java`	Compiles Java file & generates **`A3.h`**
2	`gcc -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -shared -fPIC -o libA3.so A3.c`	Compiles C file
3	`java A3`	Runs the program
4	*(Input)*	**a=8**, **b=4**, choose 1 for addition

🧮 Sample Output

```
rust


Value of a: 8
Value of b: 4
----- MAIN MENU -----
1 -> Addition
2 -> Subtraction
3 -> Multiplication
4 -> Division
5 -> Exit
Choose an option: 1
8 + 4 = 12
Result: 12
```

Would you like me to make a **printable viva sheet (PDF)** for this (with Q&A formatted neatly for submission or practice)?