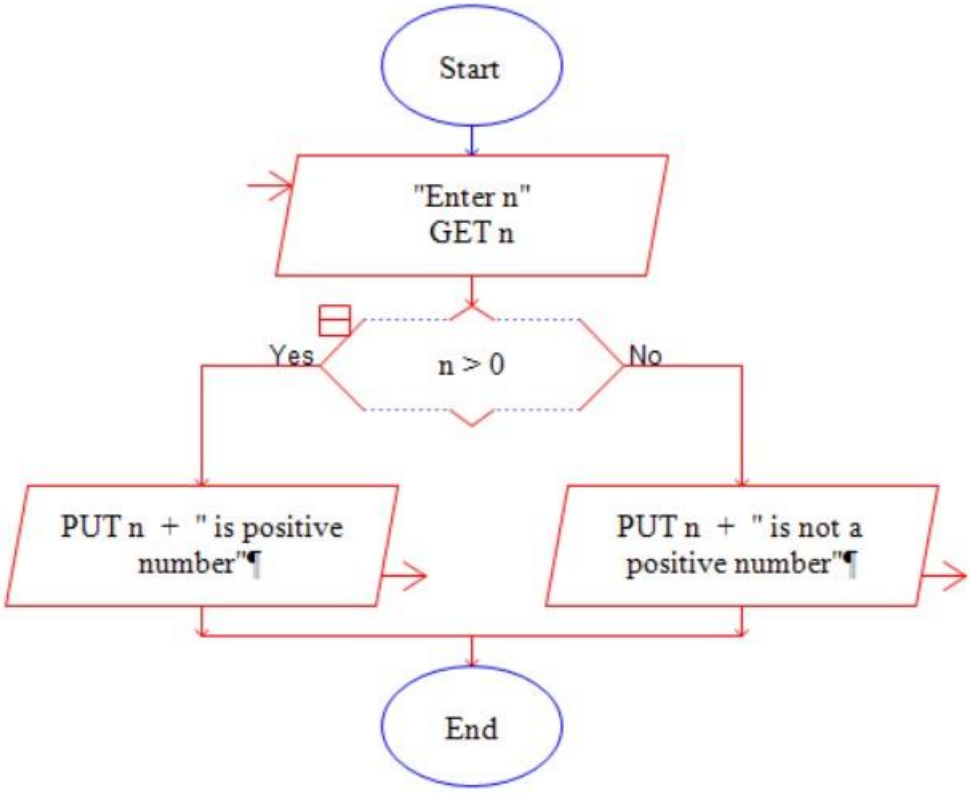
ASSIGNMENT NO: 01

Flowchart + Java Program Questions

1. **Check Positive Number:**

• **Task**: Create a flowchart to check whether a number is positive.

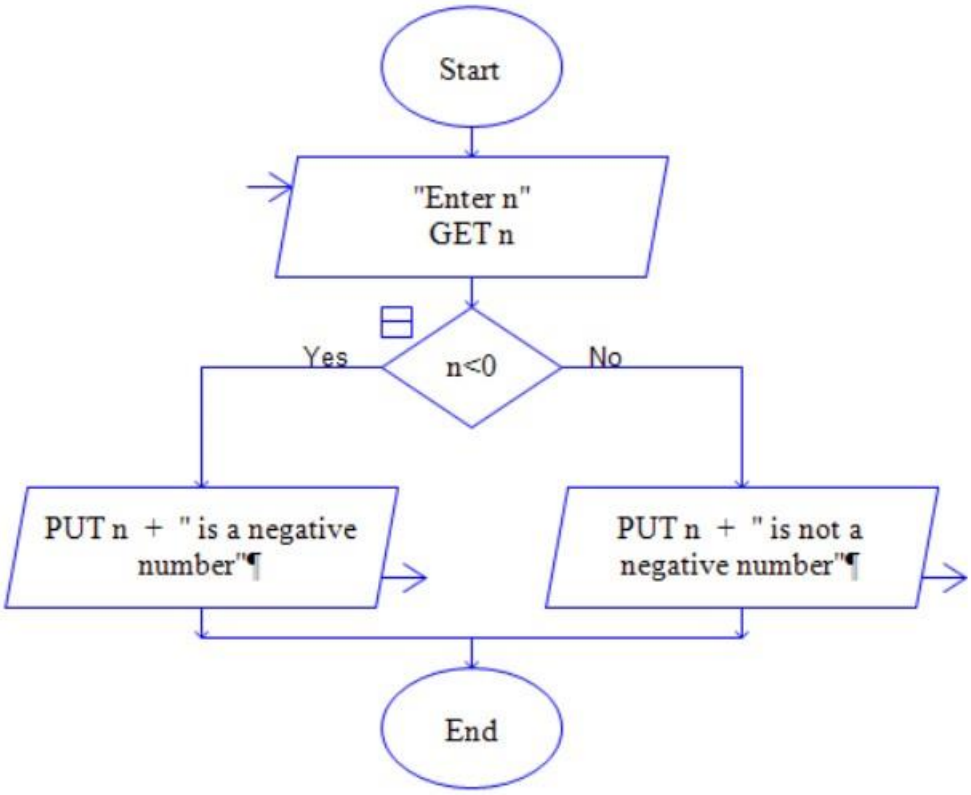
• **Next Step**: Write a Java program that checks if a predefined number is positive using an if-else statement and prints the appropriate message.



2. **Check Negative Number:**

• **Task**: Create a flowchart to check whether a number is negative.

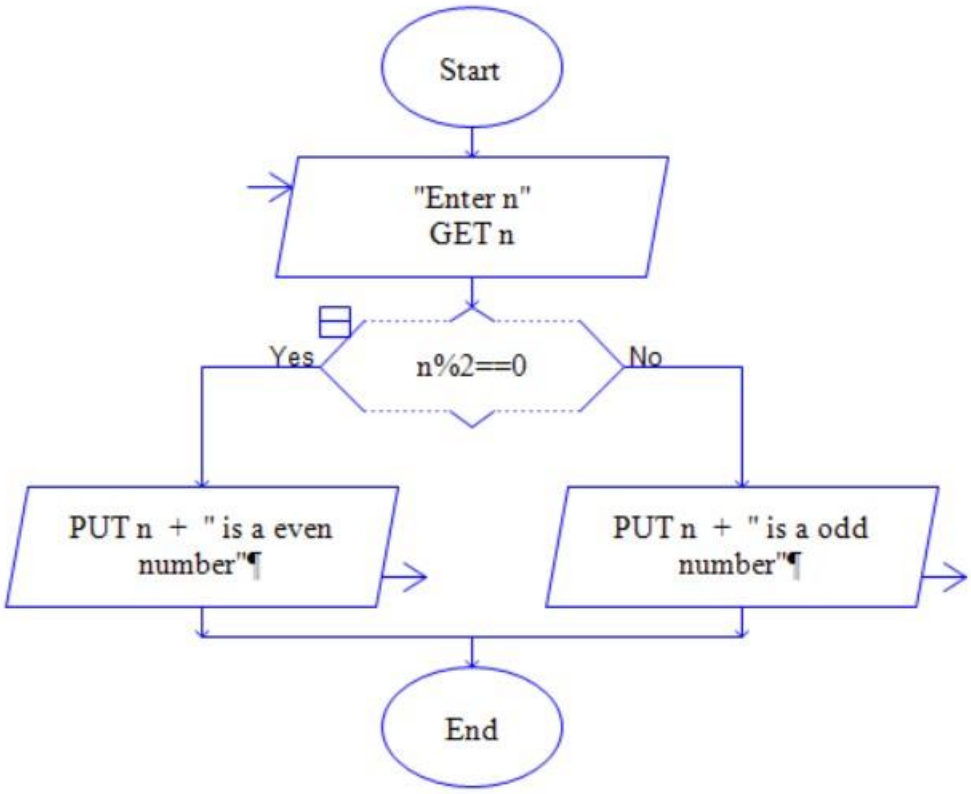
• **Next Step**: Write a Java program that checks if a predefined number is negative using an if-else statement and displays the result.



3. **Check Odd or Even Number:**

• **Task**: Create a flowchart to determine whether a number is odd or even.

• **Next Step**: Write a Java program that checks if a predefined number is odd or even. Use an if-else statement and the modulus operator (%) to determine whether the number is divisible by 2 or not.



4. **Display Good Morning Message Based on Time:**

• **Task**: Create a flowchart to display a "Good Morning" message based on a given time.

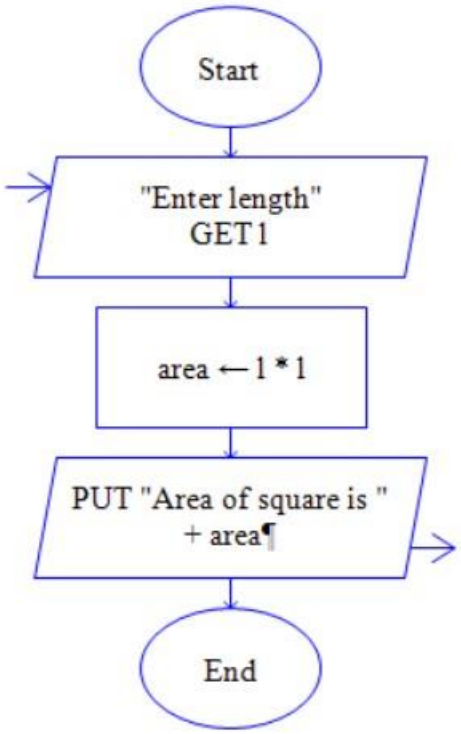
• **Next Step**: Write a Java program that displays a "Good Morning" message if the

predefined time is between 5 AM and 12 PM. Use an if statement to implement the logic.

5. **Print Area of a Square:**

• **Task**: Create a flowchart to calculate and print the area of a square. • **Next Step**: Write a Java program that calculates the area of a square using the formula

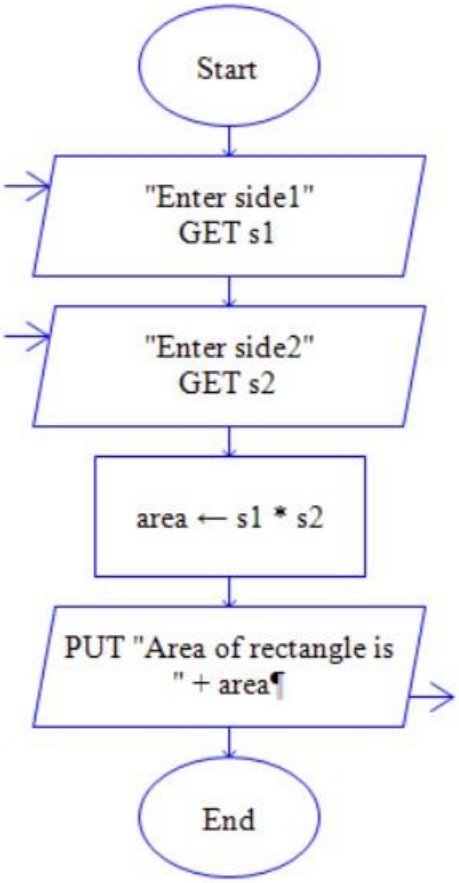
area = side * side. Use a predefined side length.



6. **Print Area of a Rectangle:**

• **Task**: Create a flowchart to calculate and print the area of a rectangle.

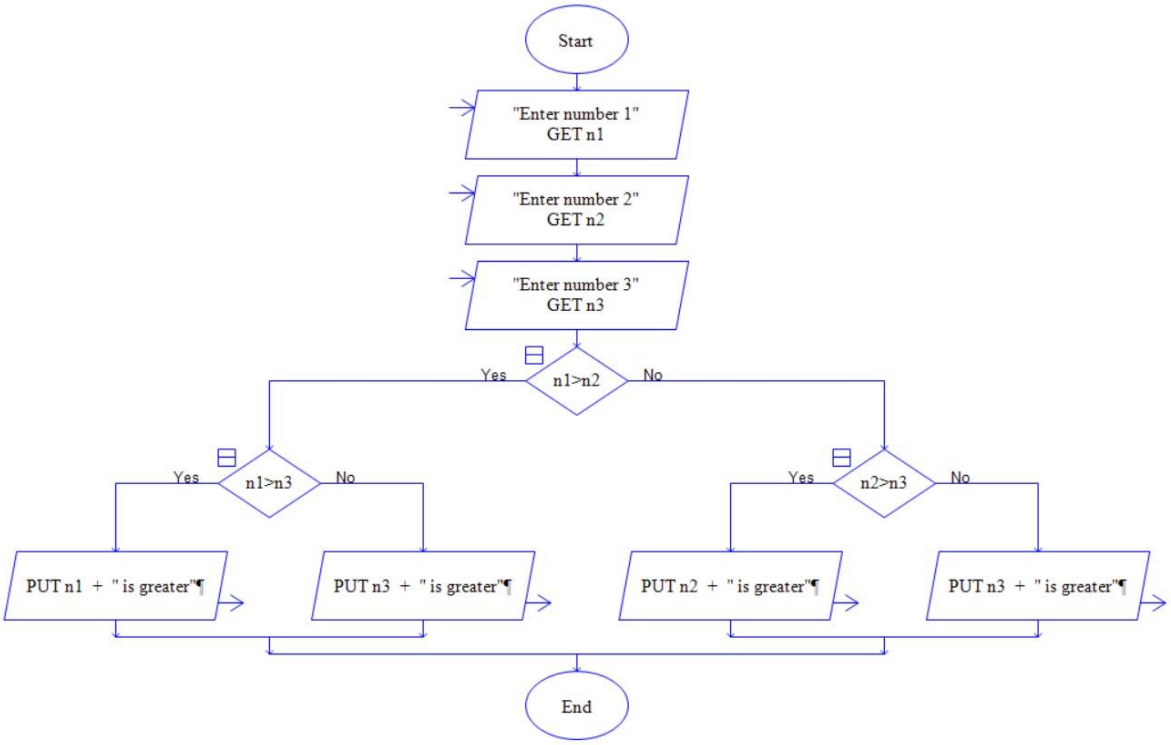
• **Next Step**: Write a Java program that calculates the area of a rectangle using the formula area = length * width. Use predefined values for length and width.



7. **Find the Largest of Three Numbers:**

• **Task**: Create a flowchart to find the largest of three numbers.

• **Next Step**: Write a Java program that finds and prints the largest of three predefined numbers using if-else statements.



Food for Thought: Research and Read More About

- **History of Java:**

Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

The first public version of java is JDK 1.0 released on January 23, 1996.

The current stable version is Java SE 22 released in March 2024.

- **How Java is Useful & Problems It Solves:**

Java is the most popular, widely used object-oriented programming language. The security feature of Java makes it popular and widely used. It used by many Java enthusiasts for different purposes. By using Java, we can develop a variety of applications such as enterprise applications, network applications, desktop applications, web applications, games, android app, and many more. In this section, we will focus on what is Java used for, the applications of Java, and why we use Java.

In comparison to other programming languages, Java stands alone for its security and functionality. Java isolates itself from other programming languages because of functionality and security and it is relevant too. There are some other reasons to use Java are as follows:

- Scalability: Scalability adds capacity to our system. It improves the capacity of the system by adding the system resources without affecting the deployment architecture. We can achieve scalability by increasing the resources such as RAM and CPU in a single system. It is important because it handles the workload, increases the system performance, and maximizes productivity.
- Cross-Platform: Cross-platform means, a compiled Java program can be run on all the platforms. Remember that the system must have JVM. After compiling a Java program, the Java code gets converted into the bytecode which is platform-independent. This bytecode is understood by the JVM. We can run this bytecode on any platform.
- Memory-Management: Java provides its own mechanism for managing the memory is known as garbage collection. We need not to care about memory and do not required to implement it to manage the memory. It automatically deletes the objects when they no longer used by the application. It improves the speed of the application.
- Multi-threading: Thread is a light-weight subprocess. Multi-threading in Java allows concurrent execution of two or more threads simultaneously. It maximizes the utilization of the CPU.

- **Role of the Java Virtual Machine (JVM):**

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

It is a specification where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.

Its implementation is known as JRE (Java Runtime Environment).

Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

JVM Role:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

• **Java Runtime Environment (JRE):**

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The components of JRE are :

1. Deployment technologies, including deployment, Java Web Start, and Java Plug-in.
2. User interface toolkits, including Abstract Window Toolkit (AWT), Swing, Java 2D, Accessibility, Image I/O, Print Service, Sound, drag, and drop (DnD), and input methods.
3. Integration libraries, including Interface Definition Language (IDL), Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), Remote Method Invocation (RMI), Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP), and scripting.
4. Other base libraries, including international support, input/output (I/O), extension mechanism, Beans, Java Management Extensions (JMX), Java Native Interface (JNI), Math, Networking, Override Mechanism, Security, Serialization, and Java for XML Processing (XML JAXP).
5. Lang and util base libraries, including lang and util, management, versioning, zip, instrument, reflection, Collections, Concurrency Utilities, Java Archive (JAR), Logging, Preferences API, Ref Objects, and Regular Expressions.
6. Java Virtual Machine (JVM), including Java HotSpot Client and Server Virtual Machines.

• **Difference Between JDK, JRE, and JVM:**

The differences between JDK, JRE, and JVM:

1. JVM (Java Virtual Machine):
    - Abstract Machine: JVM is an abstract machine, meaning it doesn't physically exist. It provides a runtime environment for executing Java bytecode.
    - Platform Independence: JVM ensures that Java programs can run on any platform without modification.
    - Tasks:
        - Loads code.
        - Verifies code.
        - Executes code.

        - Provides a runtime environment.
    - Availability: JVMs are available for various hardware and software platforms.
2. JRE (Java Runtime Environment):
    - Implementation of JVM: JRE is the implementation of JVM. It physically exists.
    - Components:
        - Contains libraries and other files used by JVM at runtime.
        - Enables execution of Java applications.
    - Platform Dependent: JRE is platform-dependent due to OS configurations.
3. JDK (Java Development Kit):
    - Software Development Environment: JDK is used for developing Java applications and applets.
    - Components:
        - Includes JRE.
        - Contains development tools (e.g., javac, jar, Javadoc).
        - Provides a private JVM.
    - Java Platforms: JDK corresponds to one of the Java Platforms (Standard Edition, Enterprise Edition, or Micro Edition).

• **Memory Areas in JVM:**

Memory Areas Allocated By the JVM:

1. Class (Method) Area

The class method area is the memory block that stores the class code, variable code(static variable, runtime constant), method code, and the constructor of a Java program. (Here method means the function which is written inside the class). It stores class-level data of every class such as the runtime constant pool, field and method data, the code for methods.

2. Heap

The Heap area is the memory block where objects are created or objects are stored. Heap memory allocates memory for class interfaces and arrays (an array is an object). It is used to allocate memory to objects at run time

3. Stack

Each thread has a private JVM stack, created at the same time as the thread. It is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.

Java Stack stores frames and a new frame is created each time at every invocation of the method. A frame is destroyed when its method invocation completes

4. Program Counter Register:

Each JVM thread that carries out the task of a specific method has a program counter register associated with it. The non-native method has a PC that stores the address of the available JVM instruction whereas, in a native method, the value of the program counter is undefined. PC register is capable of storing the return address or a native pointer on some specific platform.

5. Native method Stacks:

Also called C stacks, native method stacks are not written in Java language. This memory is allocated for each thread when it’s created And it can be of a fixed or dynamic nature.

• **Primitive Data Types in Java:**

Java has eight primitive data types, which are predefined by the programming language and used to store simple values:

- byte: This is an 8-bit signed two's complement integer with a minimum value of -128 and a maximum value of 127. Its default value is 0.
- short: A 16-bit signed two's complement integer with a minimum value of -32,768 and a maximum value of 32,767. Its default value is also 0.
- int: A 32-bit signed two's complement integer with a larger range, from -2,147,483,648 to 2,147,483,647. The default value for an int is 0.
- long: An even larger 64-bit two's complement integer ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Its default value is 0L.
- float: A single-precision 32-bit IEEE 754 floating-point number. It's used for decimal values and has a default value of 0.0f.
- double: A double-precision 64-bit IEEE 754 floating-point number, used for decimal values with more precision. The default value is 0.0d.
- boolean: Represents one bit of information, but its "size" isn't precisely defined. It can only take the values true or false, with a default value of false.
- char: A single 16-bit Unicode character with a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535). The default value is '\u0000'.