**Sanity Testing** is a type of software testing performed to quickly evaluate whether a particular function or feature of an application is working correctly after changes or fixes have been made. It is often a narrow and focused form of testing that ensures that the new changes have not broken the basic functionality of the software, and the software is stable enough to proceed with further detailed testing.

**Key Characteristics of Sanity Testing:**

1. **Focused Testing**: Unlike comprehensive regression testing, sanity testing focuses only on the areas that were changed or impacted by the recent updates. It is not exhaustive but aims to ensure that the specific functionality works as expected.
2. **Quick Assessment**: Sanity testing is typically fast and involves checking the main functions of an application after code changes to verify if the software can proceed with further in-depth testing.
3. **Performed After Bug Fixes**: Sanity testing is commonly done after a new build, especially when bug fixes or small feature updates are made, to verify that the issue has been resolved and the related functionality works as intended.
4. **Non-Exhaustive**: It is not as thorough as smoke testing or regression testing. The goal is not to uncover all defects but to confirm that critical functionalities are working.
5. **Ensures Stability**: It ensures that the basic functionality is stable enough to continue testing, preventing wasted effort on detailed testing if major issues exist.

**When is Sanity Testing Performed?**

* **After receiving a new build**: When a new build or patch is deployed, sanity testing checks whether the changes have broken any important features.
* **After bug fixes or patches**: When developers fix a bug, sanity testing is used to verify that the fix works and hasn't introduced new issues.
* **In Agile Development**: Sanity testing is often performed after each sprint or update to ensure that new changes are functional before moving to more comprehensive testing.

**Difference Between Sanity and Smoke Testing:**

* **Sanity Testing**: Focuses on specific areas after bug fixes or small changes, ensuring that the functionality related to the changes works properly.
* **Smoke Testing**: Involves a broad check of the application, ensuring that the most critical features work as a whole, often performed at the start of the testing cycle before any in-depth testing.

**Example:**

Imagine a development team has fixed a bug where the login button wasn’t working. After receiving the updated build with the bug fix, the QA team will perform **sanity testing** by checking the login functionality to ensure that the issue has been resolved and the basic login flow is functioning. They might not test the entire application at this point, as the focus is on validating the fix rather than checking all features.

**Conclusion:**

Sanity testing is an essential step in ensuring that the software is stable and functional enough for further testing. It provides a quick check of whether critical features work as expected after small changes or fixes, helping avoid wasting time on detailed testing if the build is fundamentally unstable.

Setting environment variables for Java installation is crucial for ensuring that Java programs and tools can be accessed and executed from anywhere on your system, without needing to specify their full path each time. The environment variables also enable various applications and development tools to interact with Java in a seamless manner.

Here’s why setting environment variables like JAVA\_HOME and updating the PATH is important:

**1. JAVA\_HOME Variable:**

* **Defines Java Installation Directory**: The JAVA\_HOME environment variable points to the root directory of your Java installation (e.g., C:\Program Files\Java\jdk-14 on Windows or /usr/lib/jvm/java-14 on Linux).
* **Required by Many Java-based Applications**: Many development tools, like **Maven**, **Gradle**, **Eclipse**, and others, rely on the JAVA\_HOME environment variable to locate the installed Java Development Kit (JDK). Without JAVA\_HOME, these tools might not function correctly or fail to find the right Java version.
* **Centralized Reference**: By setting JAVA\_HOME, you avoid having to specify the Java path every time you run a Java program or command.

**2. PATH Variable:**

* **Allows Running Java Commands from Anywhere**: By adding the bin directory of the JDK to your system’s PATH (e.g., C:\Program Files\Java\jdk-14\bin on Windows or /usr/lib/jvm/java-14/bin on Linux), it allows you to run Java commands like java, javac, java -version, etc., directly from the terminal or command prompt without having to specify the full path to the executable.
* **Global Access to Java Tools**: The PATH variable ensures that when you type java or javac, your operating system knows where to look for the relevant executable, making it easier to run Java commands.

**3. Why Are Environment Variables Necessary?**

* **Consistency Across Applications**: When different tools and applications require access to Java, the environment variables provide a standard way to reference Java regardless of where it's installed on the system.
* **Seamless Development Workflow**: For developers, setting these variables ensures that you don't have to specify the full path to the Java executable or libraries every time you compile or run a program. It simplifies working with command-line tools, build tools, and IDEs.
* **Multiple Versions of Java**: By setting JAVA\_HOME, you can easily switch between different versions of Java on the same system by modifying the JAVA\_HOME value. This is particularly useful in environments where backward compatibility or version-specific behavior is important.

**4. How Do You Set Environment Variables for Java?**

* **On Windows**:
  1. Install Java (JDK).
  2. Set the JAVA\_HOME environment variable to the JDK directory.
     + Go to **Control Panel** → **System** → **Advanced system settings** → **Environment Variables**.
     + Click on **New** under **System Variables** and set the variable name as JAVA\_HOME and the value as the path to your JDK (e.g., C:\Program Files\Java\jdk-14).
  3. Update the PATH variable:
     + Edit the Path variable under **System Variables** and add the bin directory inside the JDK folder (e.g., C:\Program Files\Java\jdk-14\bin).
  4. Verify by opening a new command prompt and typing java -version and javac -version to check if Java is properly installed.