)**What is type casting ? Explain different types of type casting with example** .

Type casting in Java is the process of converting a value from one data type to another

**. implicit type Casting:**

- Occurs when converting a smaller data type to a larger data type.
- No data loss happens because the larger data type can hold the value of the smaller type without any truncation.
- Happens automatically without any explicit casting required by the programmer.

**Explicit Casting:**

- Occurs when converting a larger data type to a smaller data type.
- Might lead to data loss if the value in the larger type cannot be fully represented in the smaller type.
- Requires **explicit casting** using the target data type in parentheses before the value being cast.

# Briefly explain various features of java

- **Object-Oriented (OO):** Java revolves around objects, promoting code organization and reusability through concepts like classes, inheritance, and polymorphism. Think building blocks with specific functionalities.

- **Platform Independent (WORA):** Java code gets compiled into bytecode that runs on any platform with a Java Virtual Machine (JVM). Write once, run almost anywhere!

- **Robust & Secure:** Automatic memory management (garbage collection) and type safety help prevent runtime errors and security vulnerabilities. Java prioritizes stability and avoiding crashes.

- **Rich Standard Library:** Java offers a vast collection of built-in functionalities like networking, file I/O, and graphical user interface (GUI) development tools. Lots of tools at your disposal!

- **Multithreaded:** Java programs can handle multiple tasks concurrently, improving responsiveness for applications that can benefit from parallelism. Do more than one thing at a time (efficiently).

**UNIT-III**

# Difference between Abstract class and Interface

- **Abstract Class:** Can have both abstract (without implementation) and concrete (implemented) methods. Provides a partial implementation blueprint for subclasses.
- **Interface:** Only contains abstract methods (no implementation). Defines a contract that subclasses must fulfill.

- **Abstract Class:** A class can extend only one abstract class but can implement multiple interfaces.
- **Interface:** A class can implement multiple interfaces but cannot directly inherit from multiple abstract classes (Java doesn't support multiple inheritance of classes).

- **Abstract Class:** You cannot directly create objects from an abstract class because it might have unimplemented methods.
- **Interface:** You cannot create objects from an interface either. Interfaces define what a class must do, not how to do it.

- **Abstract Class:** Can have static, non-static, final, and non-final member variables.
- **Interface:** Can only have static and final member variables (constants) by default.

- **Abstract Class:** Serves as a base class for inheritance, providing a common structure and potentially some default behavior for subclasses.
- **Interface:** Focuses on defining a contract (what methods a class must implement) to promote loose coupling and encourage polymorphism.

# What is the final keyword in Java

The `final` keyword in Java is a modifier used to restrict the behavior of variables, methods, and classes

**1. Final Variables:**

- Once a final variable is initialized, its value cannot be changed (becomes constant).
- Useful for declaring constants that shouldn't be modified during program execution (e.g., `final double PI = 3.14159;`).
- Final variables can be initialized during declaration or within a constructor block.

**2. Final Methods:**

- Cannot be overridden by subclasses in inheritance.
- Ensures a method's behavior remains consistent across all subclasses.
- Often used for core functionalities within a class hierarchy that shouldn't be altered by subclasses.

**3. Final Classes:**

- Cannot be extended by other classes.
- Prevents inheritance and enforces the class as the final implementation in the inheritance hierarchy.
- Useful for utility classes or core components where you don't want subclasses to modify behavior.

# Difference between final, finally..

### final:

- Keyword used to restrict modifications.

- **Variables:** Declares a constant value (unchangeable after initialization).
- **Methods:** Prevents overriding in subclasses (enforces specific behavior).
- **Classes:** Disallows inheritance (seals the class implementation).

**finally:**

- Keyword used for guaranteed code execution.
- Placed in a `try-catch` block.
- Ensures the code within the `finally` block always executes, regardless of exceptions or normal program flow.

# Explain about Exception Propagation

**Exception propagation in Java refers to the mechanism by which unchecked or checked exceptions travel up the call stack when they are not handled within the method where they occur.**

- **Exception Occurs:** An exception is thrown within a method due to an error condition (e.g., `NullPointerException`, `IndexOutOfBoundsException`).

- **Unhandled Exception:** If the method doesn't handle the exception using a `try-catch` block, it's propagated upwards.

- **Call Stack Search:** The exception travels up the call stack, searching for a method that has a `try-catch` block that can handle the specific exception type.

- **Handling or Propagation:**

  - **Handled Exception:** If a suitable `try-catch` block is found, the exception is handled within that block, and program execution might continue.
  - **Unhandled Exception:** If no matching `try-catch` block is found all the way up to the `main` method, the program terminates abnormally, and an error message related to the unhandled exception is displayed.

## What are the Exception handling keywords in Java?

- **try:** This keyword identifies a block of code where an exception might potentially occur. The code within the `try` block is executed, and if no exception is thrown, the program continues as usual.

- **catch:** This keyword is used with a following parenthesis containing the exception type. One or more `catch` blocks can follow a `try` block. If an exception is thrown within the `try` block, the JVM searches for a matching `catch` block that can handle the specific exception type. The code within the matching `catch` block is then executed to handle the exception.

- **throw:** This keyword is used to explicitly throw an exception from a method or any block of code. It's often used to indicate that an error condition has occurred and needs to be handled by a higher level in the call stack.

- **throws:** This keyword is used in method signatures to declare the exceptions that a method might throw. It informs the calling code about the potential exceptions that could arise when calling the method. This allows the calling code to handle these exceptions appropriately using `try-catch` blocks.

**finally:** This keyword is used with a block of code that executes regardless of whether an exception is thrown within the `try` block or not, or how the `try` block exits (normally or through a `return` statement). The `finally` block is typically used for cleanup actions such as closing files, releasing resources, or resetting flags.

# f. What are the advantages of Exception handling?

- **Program Stability:** Exception handling prevents unexpected program termination due to errors. By catching and handling exceptions gracefully, you can ensure the program continues to run even if errors occur.

- **Improved Code Readability and Maintainability:** Exception handling makes code easier to understand and maintain. By separating error handling logic from the main program flow, you can keep code cleaner and more focused on core functionality.

- **Error Isolation and Debugging:** Exception handling helps isolate the source of errors. Exception messages and stack traces provide valuable information for debugging, making it easier to identify and fix problems.

- **Predictable Behavior:** Exception handling allows you to define how the program should behave in case of specific errors. This predictability makes code more reliable and robust.

- **Resource Management:** The `finally` block in exception handling can be used for critical resource management tasks like closing files or releasing database connections. This ensures resources are freed up properly, even if exceptions occur.

# What is a Java package and how is it used?

**In Java, a package is a way to organize related classes, interfaces, and sub-packages into a hierarchical structure. It provides several benefits for code maintainability and reusability**

**Purpose:**

- **Organization:** Group related functionalities together, making code easier to navigate and understand.
- **Name Space Management:** Prevent naming conflicts between classes from different parts of your codebase.
- **Access Control:** Control access to classes and interfaces within a package using access modifiers (public, private, protected).
- **Reusability:** Packages can be imported into other projects, promoting code sharing and reducing duplication.

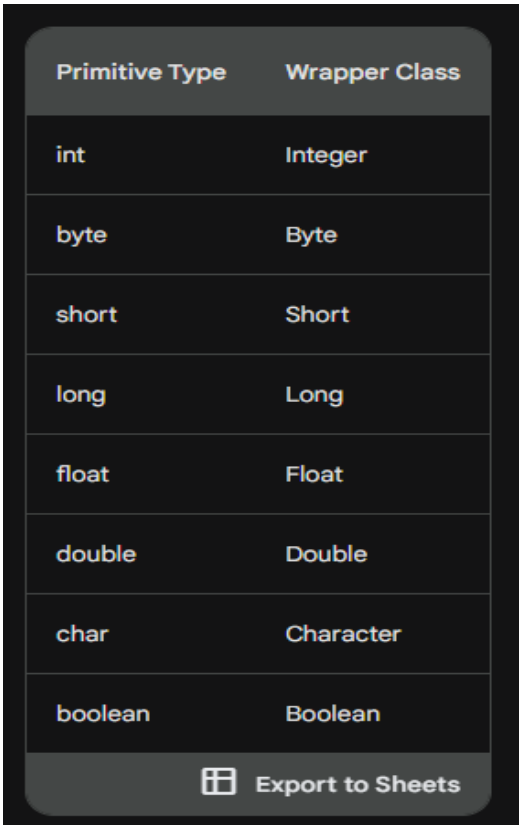
**Using Packages:**

1. **Package Declaration:** At the beginning of your source file, you can declare the package your class belongs to using the `package` keyword followed by the package name.
2. **Importing Packages:**
    - To use classes or interfaces from another package in your code, you need to import them using the `import` keyword.
    - You can import an entire package (importing all its public members) or specific classes/interfaces from a package.

# UNIT-IV

## What do you know by wrapper class in java? Write the name of all wrapper classes corresponding to all primitive types.

- In Java, a wrapper class is a class that encapsulates, or "wraps," a primitive data type into an object
- This allows primitive data types to be used as objects in Java.
- Wrapper classes provide a way to convert primitive data types into objects, which is useful when working with collections, generics, and other situations where objects are required



| Primitive Type | Wrapper Class |
|---|---|
| int | Integer |
| byte | Byte |
| short | Short |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

Export to Sheets

# UNIT – V

## Write differences between applet and application programs.

- **Applet:** Runs within a web browser that supports Java. Requires Java Runtime Environment (JRE) to be installed on the user's machine.
- **Application:** Runs directly on the user's computer's operating system. Does not require a web browser or JRE installation on the user's machine (as it comes bundled with the application itself).
- **Applet:** Developed using Java and HTML. Deployed by embedding the applet code within an HTML document and placing it on a web server. Users access the applet by visiting the web page containing the HTML code.
- **Application:** Developed solely using Java. Compiled into an executable file (`.jar` or native executables depending on platform). Distributed by providing the executable file to users, who can then run it on their machines.
- **Applet:** Restricted access to system resources (files, network) due to security concerns within a web browser environment. Relies on a security manager to control applet behavior.
- **Application:** Has more access to system resources compared to applets (depending on user permissions). Needs careful design and security considerations to prevent potential security vulnerabilities.

- **Applet:** Generally designed for smaller, interactive tasks that can be embedded within a web page. May have limited functionality due to security restrictions.
- **Application:** Can be designed for a wider range of functionalities, from simple utilities to complex software with extensive features.

## What are differences between AWT and SWING components?

Look & Feel: AWT - Platform dependent, Swing - Pluggable themes
Components: AWT - Basic (buttons, labels), Swing - Richer (progress bars, tables)
Event Handling: AWT - Simpler model, Swing - More flexible with listeners
Performance: AWT - Heavyweight (slower), Swing - Lightweight (faster)

## Write an applet program to display your name, branch and roll number in applet window

```
import java.applet.*;
import java.awt.*;

public class MyInfo extends Applet {
  public void paint(Graphics g) {
    g.drawString("Name: Rudra", 20, 20);
    g.drawString("Branch: Computer Science", 20, 40);
    g.drawString("Roll Number: 12345", 20, 60);
  }
}
```

```
<html>
<head>
  <title>My Information Applet</title>
</head>
<body>
  <applet code="MyInfo.class" width="300" height="100">
  </applet>
</body>
</html>
```