Introduction to Java Conversions

In Java, converting data from one type to another is a common and essential task in software development.

Whether you're working with user input, processing data, or integrating with external systems, understanding how to handle type conversions is critical for building robust and efficient applications.

Java provides a rich set of methods and classes to facilitate seamless data type conversions.

These include primitive-to-string conversions, string-to-numeric conversions, and even more complex operations like working with dates.

Why UNIT 2 - Java Conversions

In programming, data conversion plays a crucial role in ensuring that values of different types can be handled, manipulated, and processed correctly.

Java is a statically-typed language, which means variables must explicitly define their type (like int, double, String, etc.).

However, data often needs to be converted between different types to meet specific requirements, such as performing calculations, comparing values, or formatting data for output.

Why do we need data conversion?

- **Interoperability**: In Java, different data types often need to interact with one another. For instance, combining an integer with a string or converting a date into a string for display.

- **Data Processing**: Data conversion allows for proper manipulation of values, whether they’re simple numbers or complex objects, and ensures that operations like arithmetic, parsing, and formatting are performed accurately.

Why do we need data conversion?

- **Data Consistency:** Converting between data types helps maintain consistency in data usage, particularly when handling diverse forms of input (e.g., converting user input from strings into appropriate numerical or date formats).

- **Memory Efficiency:** Some data types (like int, long, double) consume different amounts of memory, and conversion ensures efficient use of resources by avoiding unnecessary conversions.

Why do we need data conversion?

- **Output Formatting:** When displaying results to users, it’s often necessary to convert data types into more readable formats, such as converting a date object to a string or a number to currency.

The Data Types

Data Type	Purpose	Typical Use Case	Example
String	Used to store and manipulate sequences of characters.	Storing textual data, handling user input, or displaying text.	String name = "John";
int	Represents a whole number used for mathematical operations.	For basic arithmetic, counting, indexing arrays, or loop control.	int age = 25;
long	Used to store larger whole numbers when int isn't sufficient.	Storing large numerical values like population or timestamps.	long distance = 1000000000L;
double	Used for precise decimal number calculations.	For financial calculations, scientific data, or measurements requiring precision.	double pi = 3.14159;
char	Represents a single character used in text-based data.	Storing or manipulating individual characters.	char grade = 'A';
boolean	Represents a true/false value used in conditional logic.	Used for decision making in code (if statements, loops).	boolean isStudent = true;
Object	Serves as the base type for all objects in Java.	To store any Java object, e.g., strings, numbers, or custom classes.	Object obj = "Hello";

Data Type	Purpose	Typical Use Case	Example
Date	Represents a specific point in time (year, month, day, hour, minute, second).	To store and manipulate dates.	Date today = new Date();
LocalDate	Represents only the date (year, month, day).	For date-only operations, like handling birthdays or events.	LocalDate date = LocalDate.now();
LocalTime	Represents only the time (hour, minute, second).	For handling time-only operations.	LocalTime time = LocalTime.now();
LocalDateTime	Represents both date and time.	To store and manipulate date and time together.	LocalDateTime datetime = LocalDateTime.now();
BigDecimal	Used for precise decimal calculations with high accuracy.	For financial applications, scientific computations, and monetary values.	BigDecimal bd = new BigDecimal("123.45");
BigInteger	Stores very large whole numbers beyond the long type.	For managing extremely large integers, like financial contracts or cryptographic keys.	BigInteger bigInt = new BigInteger("12345678901234567890");

Data Type	Purpose	Typical Use Case	Example
Timestamp	Represents a point in time with millisecond precision.	Used in databases and logging to record time-stamped events.	Timestamp ts = new Timestamp(System.currentTimeMillis());
binary	Represents numbers in base 2 (0s and 1s).	Used in computer systems and data manipulation at the bit level.	binary number = 1010;
decimal	Represents base 10 numbers, standard numerical system.	Used in everyday arithmetic, financial transactions, and scientific computations.	decimal num = 123.45;
hex	Represents numbers in base 16 (0-9, A-F).	Used in representing color codes, machine code, or data storage.	hex number = 1A3F;
octal	Represents base 8 numbers (0-7).	Used in low-level data storage and file permissions.	octal number = 755;

Introduction to Java Conversions

This document covers some of the most frequently used conversions in Java, such as:

- String to int and int to String
- String to long and long to String
- String to double and double to String
- String to Date and Date to String

Each of these conversions has specific methods and best practices associated with them, ensuring accurate and error-free transformations.

Whether you're parsing user input, formatting output, or handling numerical calculations, mastering these techniques will make your Java applications more versatile and reliable.

Key Differences Between Implicit Conversion and Explicit Casting:

- **Implicit Conversion:**
  - Automatically performed by the compiler.
  - Happens when converting from smaller data types to larger data types (e.g., int to long).
  - No syntax or manual intervention is required from the programmer.
- **Explicit Casting:**
  - Performed manually by the programmer.
  - Required when converting from larger data types to smaller data types (e.g., double to float).
  - Uses (targetType) syntax to instruct the compiler on how to convert.

When to Use Implicit vs. Explicit Casting:

- Use implicit conversion when you are confident that no data loss will occur.

- Use explicit casting when you need more control and when there might be potential data loss, like converting double to float or float to int.

1. String to int

To convert a String to an int, you can use the Integer.parseInt() or Integer.valueOf() methods.

```
public class StringToInt {
    public static void main(String[] args) {
        String str = "123"; // Input string
        int num1 = Integer.parseInt(str);
        int num2 = Integer.valueOf(str);

        System.out.println("Input String: " + str);
        System.out.println("Using parseInt: " + num1);
        System.out.println("Using valueOf: " + num2);
    }
}
```

OUTPUT:

Input String: 123

Using parseInt: 123

Using valueOf: 123

2. int to String

To convert an int to a String, you can use String.valueOf() or Integer.toString().

```
public class IntToString {
    public static void main(String[] args) {
        int num = 456; // Input integer
        String str1 = String.valueOf(num);
        String str2 = Integer.toString(num);

        System.out.println("Input Integer: " + num);
        System.out.println("Using String.valueOf: " +
str1);
        System.out.println("Using Integer.toString: " +
str2);
    }
}
```

```
Input Integer: 456
Using String.valueOf: 456
Using Integer.toString: 456
```

3. String to long

Parse a string containing a large numeric value into a long.

```
public class StringToLong {
    public static void main(String[] args) {
        String str = "9876543210"; // Input string
        long num = Long.parseLong(str);

        System.out.println("Input String: " + str);
        System.out.println("Converted Long: " + num);
    }
}
```

```
Input String: 9876543210
Converted Long: 9876543210
```

4. long to String

This conversion allows you to represent long numbers as text.

```
public class LongToString {
    public static void main(String[] args) {
        long num = 1234567890L; // Input long
        String str = Long.toString(num);

        System.out.println("Input Long: " + num);
        System.out.println("Converted String: " + str);
    }
}
```

Input Long: 1234567890
Converted String: 1234567890

5. String to double

Useful for parsing numeric strings with decimal values.

```
public class StringToDouble {
    public static void main(String[] args) {
        String str = "45.67"; // Input string
        double num = Double.parseDouble(str);

        System.out.println("Input String: " + str);
        System.out.println("Converted Double: " + num);
    }
}
```

Input String: 45.67
Converted Double: 45.67

6. double to String

Represent a double value as a text for display purposes.

```
public class DoubleToString {
    public static void main(String[] args) {
        double num = 78.9; // Input double
        String str = Double.toString(num);

        System.out.println("Input Double: " + num);
        System.out.println("Converted String: " + str);
    }
}
```

Input Double: 78.9
Converted String: 78.9

7. String to Date

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class StringToDate {
 public static void main(String[] args) {
 String dateStr = "2025-01-17"; // Input string
 try {
 SimpleDateFormat formatter = new
SimpleDateFormat("yyyy-MM-dd");
 Date date = formatter.parse(dateStr);

 System.out.println("Input String: " + dateStr);
 System.out.println("Converted Date: " + date);
 } catch (Exception e) {
 System.out.println("Invalid date format!");
 }
 }
}
```

7. String to Date

Input String: 2025-01-17

Converted Date: Fri Jan 17 00:00:00 IST 2025

8. Date to String

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class DateToString {
    public static void main(String[] args) {
        Date date = new Date(); // Current date
        SimpleDateFormat formatter = new
SimpleDateFormat("yyyy-MM-dd");

        String dateStr = formatter.format(date);
        System.out.println("Input Date: " + date);
        System.out.println("Formatted Date String: " +
dateStr);
    }
}
```

Input Date: Fri Jan 17 14:30:00 IST 2025
Formatted Date String: 2025-01-17

9. String to char

```
public class StringToChar {
    public static void main(String[] args) {
        String str = "A"; // Input string
        if (str.length() == 1) {
            char ch = str.charAt(0); // Extract the first (and only) character
            System.out.println("Input String: " + str);
            System.out.println("Converted Character: " + ch);
        } else {
            System.out.println("Input is not a single-character string!");
        }
    }
}
```

Input String: A
Converted Character: A

10. String to Object

Convert a string to an object, often using string assignment to an Object type variable.

```
public class StringToObject {
    public static void main(String[] args) {
        String str = "Hello World"; // Input string
        Object obj = str; // Assign the string to an Object

        System.out.println("Input String: " + str);
        System.out.println("Converted Object: " + obj);
    }
}
```

Input String: Hello World
Converted Object: Hello World

11. Object to String

Convert an object to a string by calling the toString() method.

```
public class ObjectToString {
    public static void main(String[] args) {
        Object obj = "Welcome to Java"; // Input object
        String str = obj.toString();

        System.out.println("Input Object: " + obj);
        System.out.println("Converted String: " + str);
    }
}
```

```
Input Object: Welcome to Java
Converted String: Welcome to Java
```

12. int to long

Convert an integer to a long. This is an implicit conversion as long has a larger capacity.

```
public class IntToLong {
    public static void main(String[] args) {
        int num = 42; // Input integer
        long lng = num; // Implicit conversion

        System.out.println("Input Integer: " + num);
        System.out.println("Converted Long: " + lng);
    }
}
```

```
Input Integer: 42
Converted Long: 42
```

13. long to int

Convert a long to an int. This requires explicit casting, as a long might not fit into an int.

```
public class LongToInt {
    public static void main(String[] args) {
        long lng = 123456L; // Input long
        int num = (int) lng; // Explicit casting

        System.out.println("Input Long: " + lng);
        System.out.println("Converted Integer: " + num);
    }
}
```

```
Input Long: 123456
Converted Integer: 123456
```

14. int to double

Convert an integer to a double. This is an implicit conversion as double has greater precision.

```
public class IntToDouble {
    public static void main(String[] args) {
        int num = 25; // Input integer
        double dbl = num; // Implicit conversion

        System.out.println("Input Integer: " + num);
        System.out.println("Converted Double: " + dbl);
    }
}
```

```
Input Integer: 25
Converted Double: 25.0
```

15. double to int

Convert a double to an integer. This requires explicit casting, and the fractional part will be truncated.

```
public class DoubleToInt {
    public static void main(String[] args) {
        double dbl = 45.67; // Input double
        int num = (int) dbl; // Explicit casting

        System.out.println("Input Double: " + dbl);
        System.out.println("Converted Integer: " + num);
    }
}
```

```
Input Double: 45.67
Converted Integer: 45
```

16. Char to Int

Converts a character to its ASCII integer value.

```
public class CharToInt {
    public static void main(String[] args) {
        char ch = 'A';
        int charToInt = (int) ch;
        System.out.println("Char to Int: " + charToInt); //
Output: 65
    }
}
```

```
Char to Int: 65
```

17. Int to Char

Converts an integer to its corresponding ASCII character.

```
public class IntToChar {
    public static void main(String[] args) {
        int num = 65;
        char intToChar = (char) num;
        System.out.println("Int to Char: " + intToChar);
    }
}
```

Output

```
Int to Char: A
```

18. String to Boolean

Converts a string to a boolean value (true for "true", otherwise false).

```
public class StringToBoolean {
    public static void main(String[] args) {
        String str = "true";
        boolean stringToBoolean =
Boolean.parseBoolean(str);
        System.out.println("String to Boolean: " +
stringToBoolean);
    }
}
```

Output

String to Boolean: true

19. Boolean to String

Converts a boolean value to a string representation.

```
public class BooleanToString {
    public static void main(String[] args) {
        boolean boolValue = true;
        String booleanToString =
String.valueOf(boolValue);
        System.out.println("Boolean to String: " +
booleanToString);
    }
}
```

Output

Boolean to String: true

20. Date to Timestamp

Converts a Date object to a Unix timestamp (milliseconds since 1970).

```
import java.util.Date;

public class DateToTimestamp {
    public static void main(String[] args) {
        Date date = new Date();
        long timestamp = date.getTime();
        System.out.println("Date to Timestamp: " +
timestamp);
    }
}
```

Output

```
Date to Timestamp: 1704068492345
```

21. Timestamp to Date

Converts a timestamp (milliseconds) to a human-readable date.

```
import java.util.Date;

public class TimestampToDate {
    public static void main(String[] args) {
        long timestamp = 1704068492345L;
        Date date = new Date(timestamp);
        System.out.println("Timestamp to Date: " + date);
    }
}
```

Output

```
Timestamp to Date: Fri Jan 05 10:08:12 IST 2024
```

22. Binary to Decimal

Converts a binary number (string) to decimal integer.

```
public class BinaryToDecimal {
    public static void main(String[] args) {
        String binary = "1010";
        int decimal = Integer.parseInt(binary, 2);
        System.out.println("Binary to Decimal: " +
decimal);
    }
}
```

Output

```
Binary to Decimal: 10
```

23. Decimal to Binary

Explanation: Converts a decimal number to its binary representation.

```
public class DecimalToBinary {
    public static void main(String[] args) {
        int decimal = 10;
        String binary = Integer.toBinaryString(decimal);
        System.out.println("Decimal to Binary: " + binary);
    }
}
```

Output

```
Decimal to Binary: 1010
```

24. Hex to Decimal

Converts a hexadecimal string to a decimal integer.

```
public class HexToDecimal {
    public static void main(String[] args) {
        String hex = "A";
        int decimal = Integer.parseInt(hex, 16);
        System.out.println("Hex to Decimal: " + decimal);
    }
}
```

Output

```
Hex to Decimal: 10
```

25. Decimal to Hex

Converts a decimal number to its hexadecimal representation.

```
public class DecimalToHex {
    public static void main(String[] args) {
        int decimal = 10;
        String hex = Integer.toHexString(decimal);
        System.out.println("Decimal to Hex: " + hex);
    }
}
```

Output

```
Decimal to Hex: a
```

26. Octal to Decimal

Converts a decimal number to its hexadecimal representation.

```
public class OctalToDecimal {
    public static void main(String[] args) {
        String octal = "12";
        int decimal = Integer.parseInt(octal, 8);
        System.out.println("Octal to Decimal: " +
decimal);
    }
}
```

Output

```
Octal to Decimal: 10
```

27. Decimal to Octal

Converts a decimal number to its octal representation.

```
public class DecimalToOctal {
    public static void main(String[] args) {
        int decimal = 10;
        String octal = Integer.toOctalString(decimal);
        System.out.println("Decimal to Octal: " + octal);
    }
}
```

Output

```
Decimal to Octal: 12
```

- Base-2 (Binary): Used in computers, 0 and 1 only.
- Base-8 (Octal): Used in some computing systems, digits 0-7.
- Base-10 (Decimal): Standard number system. digits, 0,1,2,3,4,5,6,7,8,9
- Base-16 (Hexadecimal): Used in memory addresses and colors (#FF5733).