OpenAI

Type casting in Java

explain about type casting in java ?

give me the proper and deep explaination of type casting in java so that i understand this well

Sure! Let’s learn **Type Casting in Java** in a very clear and deep way 👇

✅ Type Casting in Java — Full Explanation

📌 What is Type Casting?

Type Casting means **converting one data type into another**.
Example: converting `int` to `double`, or `double` to `int`.

Java is a **strictly typed** language, so whenever we convert types, we need to follow certain rules.

🔹 Types of Type Casting in Java

There are **two main categories**:

Type Casting	Name	Direction	Automatic?	Risk of Data Loss?
Widening Casting	Implicit Casting	Smaller → Larger type	Yes	No
Narrowing Casting	Explicit Casting	Larger → Smaller type	No	Yes

---

1️⃣ Widening Type Casting (Implicit)

✔️ Happens automatically by Java
✔️ No data loss
✔️ Smaller data type → Larger data type

Order of data types (Small → Large)

```arduino
byte → short → int → long → float → double
```

Example:

```java
int a = 10;
double b = a;  // int converted to double automatically

System.out.println(b);  // Output: 10.0
```

✔️ Safe conversion because larger type can store smaller type value easily.

2️⃣ Narrowing Type Casting (Explicit)

❌ Not automatic
❌ Possible data loss
➡️ Larger data type → Smaller data type

You MUST write cast manually using `( )`.

Example:

```java
double x = 10.99;
int y = (int) x;  // Narrowing conversion

System.out.println(y);  // Output: 10
```

> Decimal part is **lost** while converting double → int

🔍 Why Data Loss Can Happen?

- Smaller type has limited memory
- Larger type value may not fit completely

Example:

```java
int a = 130;
byte b = (byte) a;
System.out.println(b);
```

Output:

```diff
-126
```

📌 Explanation:

- byte range: -128 to 127
- 130 goes out of range → result is unexpected

This is called **overflow**.

🔸 Type Casting with Characters

Characters (`char`) in Java store **Unicode values**.
So char <-> int conversion is common.

Example:

```java
char ch = 'A';
int val = ch;
System.out.println(val);  // Output: 65
```

Reverse:

```java
int num = 66;
char ch = (char) num;
System.out.println(ch);  // Output: B
```

🔸 Type Casting in Reference Types (Objects)

➡️ Upcasting
➡️ Downcasting

🔹 Upcasting (Implicit)

Child → Parent class reference (safe)

```java
class Animal { }
class Dog extends Animal { }

Animal a = new Dog(); // Upcasting
```

🔹 Downcasting (Explicit)

Parent → Child (Needs extra care)

```java
Animal a = new Dog();
Dog d = (Dog) a;  // Downcasting
```

⚠️ If the actual object is not of type Dog, it will throw:

```nginx
ClassCastException
```

🧠 Important Points to Remember

Type	Example	Cast Required?	Safe?
Widening	int → double	❌ No	✔️ Safe
Narrowing	double → int	✔️ Yes	❌ risky
Upcasting	Dog → Animal	❌ No	✔️ Safe
Downcasting	Animal → Dog	✔️ Yes	❌ risky

✔️ Real Life Example for Better Understanding

Imagine data types as **containers**

Container	Type	Size
Small Glass	int	Smaller
Big Jug	double	Larger

- Pouring water from **small glass to big jug = no problem** (Widening)
- Pouring water from **big jug to small glass = overflow chance** (Narrowing)

That's why narrowing needs caution & force → `(int)`

📌 Summary

Concept	Key Idea
Type Casting	Converting one type to another
Widening Casting	Automatic, No data loss
Narrowing Casting	Manual, Data loss possible
Reference Casting	Upcasting & Downcasting

If you want, I can also give:
✔️ Practice Questions
✔️ Program Examples
✔️ Memory diagrams for deeper understanding

Would you like a few practice programs on Type Casting? 😊