Java Backend Roadmap – Notes (Till Today)

Made from our learning + your exact code examples + your doubts & mistakes (topic-wise).

Week 1 – Core Java (Streams, Method References, Date & Time)

1) Method Reference (4 types)

Method reference is a shortcut for lambda. Instead of writing (x) -> method(x), we directly point to the method.

```
// 4 Types
// 1) Static method reference
ClassName::staticMethod

// 2) Instance method reference (object)
obj::instanceMethod

// 3) Instance method reference (parameter)
ClassName::instanceMethod

// 4) Constructor reference
ClassName::new
```

Example you used (Comparator with method reference)

```
Optional<EmployeDetails> top =
        emp.stream()
        .max(Comparator.comparing(EmployeDetails::getSalary));
System.out.println(top.orElse(null));
```

Here: **EmployeDetails::getSalary** is 'instance of parameter' type. Stream internally calls getSalary() for each object.

2) Java Date & Time (java.time)

We used LocalDate, LocalTime, LocalDateTime, Period and DateTimeFormatter.

```
LocalDate d = LocalDate.now();
System.out.println(d);

LocalTime t = LocalTime.now();
System.out.println(t);

LocalDateTime dt = LocalDateTime.now();
System.out.println(dt);

LocalDate setDate = LocalDate.of(2004, 11, 24);
System.out.println(setDate);
```

Compare two dates

```
LocalDate d1 = LocalDate.of(2025, 1, 1);
LocalDate d2 = LocalDate.of(2026, 1, 1);

System.out.println(d1.isAfter(d2));
System.out.println(d2.isAfter(d1));
```

Add / Subtract days

```
LocalDate today = LocalDate.now();

DateTimeFormatter ddd = DateTimeFormatter.ofPattern("dd-MM-yyyy");

System.out.println(ddd.format(today.plusDays(6)));
System.out.println(today.minusDays(3));
```

Find Age (Period)

```
LocalDate mydob = LocalDate.of(2004, 11, 24);
LocalDate tod = LocalDate.now();

Period age = Period.between(mydob, tod);
System.out.println(age.getYears() + "Years " + age.getMonths() + "Months" + age.getDays() +
```

Date formatting pattern

```
LocalDate dd =  LocalDate.now();
DateTimeFormatter df = DateTimeFormatter.ofPattern("dd-MM-yyyy");
System.out.println(dd.format(df));
```

Your mistake + fix

You wrote: **today(plusDays(4))** (wrong). Correct is: **today.plusDays(4)**

Week 2 – Generics (T, Wildcards, extends/super)

1) Why use T?

T is a type placeholder. Same class can work for multiple data types without writing separate classes.

Generic class example you wrote

```
class Box<t>{

    t value;

    public void set(t value) {
        this.value = value;
    }

    public t get() {
        return value;
    }

    public static void main(String[] args) {

        Box<String> b = new Box<>();
        b.set("helloo");
        System.out.println(b.get());

        Box<Integer> n = new Box<>();
        n.set(5);
        System.out.println(n.get());
    }
}
```

If class generic illa na?

You must write separate classes like StringBox, IntegerBox, EmployeeBox → duplicate code waste.

2) Wildcards

Wildcard means: we don't know exact type at compile time. It gives flexibility when passing lists.

Upper bound wildcard

```
public static void printNumbers(List<? extends Number> list){
    list.forEach(System.out::println);
}
```

Important rule

List is read-only inside method for adding. You can read values safely, but you cannot add new values (except null).

Your doubt

You asked: 'extends la String ku epdi? Number ku mattum add pana mudiyadhu, super potta mattum add pannalam?'

Clear explanation

- **? extends** = Producer (you can GET / read) - **? super** = Consumer (you can SET / add)

Your code that ran

```
public class Generics {

    private static void printNumbers(List<? extends String>list) {

        list.forEach(System.out::println);

    }
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("hello");
        list.add("nooo");

        System.out.println("list");
        printNumbers(list);
    }
}
```

Why it runs?

Because you are only printing (reading). If you try list.add("x") inside printNumbers → compile error.

Week 3 – Collections (Set, TreeSet, Map)

HashSet with duplicates

HashSet removes duplicates automatically. It stores unique values only.

```
Set<String> se = new HashSet<>();
se.add("sudha");
se.add("dee");
se.add("sudha");

System.out.println(se);

Set<Integer> i = new HashSet<>();

i.add(5);
i.add(6);
i.add(5);

System.out.println(i);
```

TreeSet (sorted)

```
Set<String> tree = new TreeSet<>();

tree.add("sudha");
tree.add("zebre");
tree.add("apple");

System.out.println(tree);
```

HashMap duplicates

Map key duplicates not allowed. Same key again means value will be replaced.

```
Map<Integer, String> m = new HashMap<>();

m.put(3, "ded");
m.put(3, "bed");
m.put(7, "ded");

System.out.println(m);
```

Week 4 – JDBC + MySQL (Select, Insert, Update, Delete)

1) JDBC Full form

JDBC = Java Database Connectivity.

2) Why JDBC?

Java program la irundhu MySQL database connect panni data fetch / insert / update / delete panna use pannuvom.

3) MySQL Workbench output doubt

You asked: 'yen onnu mattum output varudhu?'

Reason: You inserted multiple rows but only one row was actually inserted successfully (others might have failed due to primary key duplicate / query execution not selected properly).

4) JDBC URL meaning

- **localhost** → your laptop DB - **3306** → default MySQL port - **stu** → database name

How to find your port?

In MySQL Workbench → Server → Server Status (or in MySQL Installer) you can see port. Usually 3306 for most laptops.

5) Maven dependency (MySQL connector)

You asked: 'where to put dependency every time?'

Answer: In pom.xml inside tag.

```
<dependencies>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.3.0</version>
    </dependency>
</dependencies>
```

6) SELECT (Fetch) – Your final working code

```
package demo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class Jdbc1 {
  private static  String url ="jdbc:mysql://localhost:3306/stu";
    private static String user = "root";
    private static String pass = "root123";
    public static void main(String[] args) {

try {
    Connection co = DriverManager.getConnection(url, user, pass);

    String sql = "select * from student";
```

```
        PreparedStatement ps = co.prepareStatement(sql);

        ResultSet rs = ps.executeQuery();

        while (rs.next()){
            System.out.println(
                    rs.getInt("id") + " " +
                     rs.getString("name") + " " +
                           rs.getString("registernumber") + " " +
                           rs.getInt("phoneno") + " " +
                           rs.getString("dept")
            );
        }

    }catch (Exception e) {
        throw new RuntimeException("Not cretaed");
    }
        }
    }
```

ResultSet part explained (your doubt)

- rs.next() → row by row move pannum - getInt("id") → id column value eduthu varum - getString("name") → name column value eduthu varum

You asked: 'y used get keyword?' → Because ResultSet is read-only data, so we use getXXX() to fetch.

7) INSERT – Your code

```
package demo;

import java.sql.*;

public class Jdbc {

    private static String url = "jdbc:mysql://localhost:3306/stu";
    private static String user = "root";
    private static String pass = "root123";

    public static void main(String[] args) {

        String sql = "insert into student (id,name,registernumber,phoneno,dept) values(?, ?,

        try
                (Connection co = DriverManager.getConnection(url, user, pass);
                 PreparedStatement ps = co.prepareStatement(sql)) {

            ps.setInt(1, 5);
            ps.setString(2,"amsu");
            ps.setString(3,"c2s25564");
            ps.setInt(4,45546546);
            ps.setString(5,"cs");

            ps.setInt(1, 15);
            ps.setString(2,"masss");
            ps.setString(3,"c2s25564");
            ps.setInt(4,478786546);
            ps.setString(5,"bcom");

            int rows = ps.executeUpdate();

            System.out.println(rows + "Added Sucessfully");

        } catch (SQLException e) {
```

```
                throw new RuntimeException(e);
            }
        }
    }
```

Your doubt

You asked: 'indha mari neriya type pannama short method irukuma? like loop?'

Yes: Batch Insert (addBatch + executeBatch) or loop. We will do next.

8) UPDATE – Your code

```
package demo.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class jdbc_update {

    private static String url = "jdbc:mysql://localhost:3306/stu";
    private static  String user = "root";
    private static  String pass ="root123";

    public static void main(String[] args) {

        String sql = "update student set dept = ? where id = ?";

        try {
            Connection co = DriverManager.getConnection(url, user, pass);
            PreparedStatement ps = co.prepareStatement(sql);{

                ps.setString(1,"tamil");
                ps.setInt(2,3);

                int rows = ps.executeUpdate();

                System.out.println(rows + "Updated");
            }

        }catch (Exception e){
            throw new RuntimeException(e);
        }
    }
}
```

Your doubt

ps.setInt(2,3) means: 2nd question mark (?) value is 3. So WHERE id = 3.

9) JDBC Concept summary (you said)

You understood perfectly:

• SELECT = getXXX() because we fetch data

• INSERT/UPDATE = setXXX() because we set values into query

10) JDBC URL every time type panna venduma?

In real projects: No. We keep it in properties file (application.properties) or env variables.

Roadmap Status – What we finished & what next

Topic	Status	Notes
Core Java basics	Done	You are comfortable
Streams + Method References	Done	max(), Comparator, ::
Date & Time (java.time)	Done	LocalDate/Time/Period/Formatter
Generics + Wildcards	Done	extends/super doubts cleared
Collections (Set/Map)	Done	HashSet duplicates, TreeSet sorting, HashMap replace
MySQL setup + Workbench	Done	DB created, table created, rows inserted
JDBC (Select/Insert/Update/Delete)	Done	Working code executed successfully
SQL (Deep)	Next	Need strong: joins, group by, constraints, indexes
Spring Boot	Next	After SQL strong
Spring Data JPA	Next	After Spring Boot basics

Next Plan (as you requested)

1) SQL strong (must) → 2) Spring Boot → 3) Spring Data JPA → 4) Build backend project.