

## **SDLC**

Software development life cycle is basically a structured or organized process which allows us to design and develop high quality software with least cost and time. And it is also used for the purpose of testing and maintenance of the software. As the name suggests it is a structured process for entire life cycle of the software

### **Phases of sdlc**

- Requirement specification-> It is used to gather the requirements that the software needs to fulfill or we can say to identify the tasks that the software is supposed to do
- Design-> it involves designing the architecture of the software and creating the design specification document
- Implementation-> implementation stage basically involves designing the actual software around the requirements gathered and the design created
- verification-> verification stage is basically testing phase where the software is tested for bugs, redundancies, errors and to make sure it is doing what it was supposed to do
- Deployment and maintenance-> when software has passed the testing phase, it is deployed in the real world where people can use it, and maintenance involves adding new features and essentially taking care of the software

### **Models**

-Waterfall->In this model the development is divided into multiple phases, where next phase cannot start until previous phase is completed as the output of previous phase is the input of next phase , it is simple and easy to implement, suitable for small projects, but it takes a lot of time and not at all suitable for complex projects

-Iterative-> In iterative model the software is first designed on small scale then gradually developed by adding new features and integrating new components to the software, it is cyclical process and a software product is generated in every cycle  
It is low risk process, easy to detect errors, and add new features  
Complete requirement's understanding is needed to break down the development in smaller phases

-Prototype-> In this model firstly a prototype product is created for better understanding the customer requirement and demonstration purposes, then based on that the final product is generated.  
Low cost , less time , easy to add new features  
Customer involvement may increase the complexity by changing the requirements

-Spiral ->It is a combination of waterfall and incremental model, it pays more attention to the risk involved and the development moves to next step based on the customer evaluation  
Low risk, easy to add new features  
Suited only for large projects, high cost

-V model-> it is an extension of waterfall model but here the development and testing phases are organized in parallel to each other

It is simple , easy to implement and suited for smaller projects

Not suitable for large projects, and requirement change cost is very high

## **AGILE & SCRUM**

**Agile**-> agile is a flexible and iterative approach to project management , which emphasizes collaboration, customer satisfaction and rapid delivery of the product. It is an incremental process where development team directly interacts with the customer, here each phase lasts for one to three weeks (also called sprints)

Phases

-project initiation

-spring planning

-development

-production

-retirement

**Scrum**-> scrum is a framework used to implement agile, where the work is divided into small development cycles called sprints, it allows the developers to develop products in a changing environment. It improves the quality of product, provides better estimates, and allows to be more in control and cope better with change.

Phases of scrum are:

-initiate

-plan and estimate

-implement

-review and retrospect

-release

### **Scrum tools:**

-Release-> it the process of gradually shifting or moving the software from the development environment into the real world where users gets to use the software

-Retrospective-> It is basically a session in which the scrum master and the team gets to communicate with each other.

-Burn up chart-> It is a visual representation of the team's work progress.

## **GIT**

**GITHUB** projects and trello -> above mentioned both are project management softwares where we can specify in detail every phase of the project that we are working on with it's description , it helps to better handle the large projects by dividing it into small sections. We can also define it as more of a to do list for development where we divide the projects into smaller sections and write it down like a to do list.

## **GIT**

-> Git is a version control software which is used to record changes to a file or set of files over a period of time so that we can recall specific versions later, it also allows multiple people to work on same project remotely, in essence it is a collection of software tools that help team manage changes in source code, the advantage of using version control software are improved quality , acceleration and visibility

We can go to github.com and create our account there , we can create repositories where we save files of a project, it can be public so that anyone can access the source code or private. If we need to clone our's or someone else's repositories we can visit the repository and copy the url , then open git bash where we can clone the repo using following commands

Cd to the directory where we want to clone then use the command  
`$git clone 'paste the url here'`

Here we have cloned the repository

We can use the git status command to find out the status of the repository,

In our cloned repo we can make changes as we want to , suppose we need to add new file we can create one using command

`$touch filename`

Then after writing to the file we need to add the file so that it is tracked using

`$git add filename`

Or if we have multiple files then

`$git add -all`

The we can commit the file using

`$git commit -m "commit message"`

After that we can push the commit to github using push command like

`$git push`

But since we have multiple people working on same project we all cannot keep pushing changes to the main branch , for that we use different branches , we can create one as

`$git branch branchName`

We move to a branch using checkout command

`$git checkout branchName`

We delete a branch as

`$git branch -d branchName`

If we want to add file and commit it in single command we can do that as

`$git commit -am "message"`

We can also create a branch and move directly to it in single command as

`$git checkout -b branchName`

And to come back to main branch we have two ways

`$git checkout main`

Or  
\$git pull

In order to merge a give branch with our main branch we firstly need to add needed files ,  
commit it and push it from the branch then checkout to the main branch then we can merge  
previous branch with main as  
\$git merge branchName

Now from main add new files commit it and push it so that the files added from previous branch  
in main branch after merging is visible in github main branch

To list the files we use \$ls -lrt  
In order to check the merge log we use  
\$git log --merge

If multiple people are working on same project and suppose someone made some changes to a  
particular file and pushed it onto the github , and you also want to make some changes to that  
file only so that we need to import the changes made by other person in the file without losing  
the changes made by you in that file we use rebase as

Firstly we fetch the files as  
\$git fetch origin branchName  
Then we rebase it as  
\$git rebase origin/branchName

In order to view a graph of all the commits done in the repo we use  
\$git log --graph --oneline --all  
Or just to see the commits  
\$git log --oneline

We can check the status of the current branch as  
\$git status  
We can echo or write text to file directly as  
\$echo "message" > fileName

In order to get back to previous commit we use  
\$git reset --soft HEAD~1  
But the file changes made remains in the current working tree on index so a git commit  
command will create a commit with exactly same changes as the commit you removed before  
The value 1 can be any n where it goes back to commit with a specified reference

\$git reset --mixed HEAD~1

It is similar to soft here also the changes made to the file remains in the current working tree but not on the index so if we want to redo the commit we will have to add changes (add) before committing

`$git reset --hard HEAD~1`

Doing a hard reset you will lose all the untracked files, and uncommitted changes in addition to previous commit and the changes made to the files will not remain in the working tree, so when we do git status it will show that you don't have any changes in the repository , so be careful when using this one

### **Git Stash**

It temporarily shelves changes you've made to your working copy so that you can work on something else and then come back and apply it later on. Stashing is handy if you need to quickly switch context and work on something else but you are midway through a code change and not yet completed it to commit.

We stash the changes made to a particular file as

`$git stash save`

In order to add it to last commit

`$git add -u`

`$git commit --amend`

And to access the saved file to make changes again we use

`$git stash pop`

```
MINGW64/c/Users/ayush/git/documentation
ayush@Ayush MINGW64 ~/git/documentation (main)
$ echo "local update" > test.txt

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git add .
warning: in the working copy of 'test.txt', LF will be replaced by CRLF the next
time git touches it

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git commit -am "updated test"
[main (root-commit) 17f160d] updated test
1 file changed, 1 insertion(+)
create mode 100644 test.txt

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 228 bytes | 114.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ayush-arya18/documentation.git
 * [new branch]      main -> main

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git checkout -b testBranch
Switched to a new branch 'testBranch'

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git checkout testBranch
Switched to branch 'testBranch'

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ touch testfile.txt

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ echo "test test" > testfile.txt

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ git commit -am "first commit"
On branch testBranch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
MINGW64/c/Users/ayush/git/documentation

nothing added to commit but untracked files present (use "git add" to track)

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ git add .
warning: in the working copy of 'testfile.txt', LF will be replaced by CRLF the
next time git touches it

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ git commit -m "first commit"
[testBranch 5c616c7] first commit
1 file changed, 1 insertion(+)
create mode 100644 testfile.txt

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ git push
fatal: The current branch testBranch has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin testBranch

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetUpRemote' in 'git help config'.

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ git push --set-upstream origin testBranch
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 5 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 284 bytes | 284.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'testBranch' on GitHub by visiting:
remote:   https://github.com/ayush-arya18/documentation/pull/new/testBranch
remote:
To https://github.com/ayush-arya18/documentation.git
 * [new branch]      testBranch -> testBranch
branch 'testBranch' set up to track 'origin/testBranch'.

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$

ayush@Ayush MINGW64 ~/git/documentation (testBranch)
$ git checkout main
Switched to branch 'main'
```

```
MINGW64/c/Users/ayush/git/documentation
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(Use "git push" to publish your local commits)

nothing to commit, working tree clean

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git push
To https://github.com/ayush-arya18/documentation.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/ayush-arya18/documentation.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git fetch origin main
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git fetch origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 917 bytes | 65.00 KiB/s, done.
From https://github.com/ayush-arya18/documentation
 * branch            main       -> FETCH_HEAD
 * 17f160d..57fefef    main       -> origin/main

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git rebase origin/main
Successfully rebased and updated refs/heads/main.

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(Use "git push" to publish your local commits)

nothing to commit, working tree clean
```

```
MINGW64/c/Users/ayush/git/documentation

nothing to commit, working tree clean

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 6 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 286 bytes | 286.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ayush-arya18/documentation.git
   57fefef..bcf694f  main -> main

ayush@Ayush MINGW64 ~/git/documentation (main)
$ code test.txt

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git log --graph --oneline --all
* bcf694f (HEAD -> main, origin/main) first commit
* 57fefef Update test.txt
| * 5c616c7 (origin/testBranch, testBranch) first commit
|/
* 17f160d updated test

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git log --oneline
bcf694f (HEAD -> main, origin/main) first commit
57fefef Update test.txt
17f160d updated test

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git reset --soft HEAD~1

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git log --oneline
57fefef (HEAD -> main) Update test.txt
17f160d updated test

ayush@Ayush MINGW64 ~/git/documentation (main)
```

```
MINGW64/c/Users/ayush/git/documentation
ayush@Ayush MINGW64 ~/git/documentation (main)
$ git reset --soft HEAD~1

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git log --oneline
57fe7ef (HEAD -> main) Update test.txt
17f160d updated test

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git commit -m "redo soft reset"
[main 8119de7] redo soft reset
1 file changed, 1 insertion(+)
create mode 100644 testfile.txt

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git reset --mixed HEAD~1

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git log --oneline
57fe7ef (HEAD -> main) Update test.txt
17f160d updated test

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git add .

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git commit -m "redo mixed reset"
[main 88b77a6] redo mixed reset
1 file changed, 1 insertion(+)
create mode 100644 testfile.txt

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git log --oneline
88b77a6 (HEAD -> main) redo mixed reset
57fe7ef Update test.txt
17f160d updated test

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git reset --hard HEAD~1
HEAD is now at 57fe7ef Update test.txt

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git log --oneline
57fe7ef (HEAD -> main) Update test.txt
17f160d updated test

ayush@Ayush MINGW64 ~/git/documentation (main)
$

MINGW64/c/Users/ayush/git/documentation
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (7af590ded35a03e310bc13b523fd4b12c7ef81ac)

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git stash save
Saved working directory and index state WIP on main: 70e111d Update test.txt

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git add .

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git commit -m "message"
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 2 different commits each, respectively.
(Use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git pull
Merge made by the 'ort' strategy.
 testfile.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 testfile.txt

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git add .

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git commit -m "message"
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
(Use "git push" to publish your local commits)

nothing to commit, working tree clean

ayush@Ayush MINGW64 ~/git/documentation (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 6 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 501 bytes | 501.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ayush-ayal8/documentation.git
   bcf694f..d1aeee2  main -> main
```

## JAVA

Java is an object oriented high level language which is both an interpreted language and compiled language as the java code is firstly converted into byte code (entire file at once ) then that bytecode is fed to jvm (java virtual machine) which executes the code line by line like an interpreter.

The java virtual machine contains multiple components, like it has it's own memory, we can also call it as a virtual machine, it has it's own stack , a heap for dynamic memory allocation, it also contains isa which is instruction set architecture which is a set of instructions or protocol which



defines how the processor is controlled by software and how the processor executes instructions.

Jdk stands for java development toolkit which as the name suggests provides various tools required for developing java programs or applications , jre is java runtime environment which provides a runtime environment to execute java programs.

We can also run the java code from terminal , where we use the following commands , firstly create a java file in our specific directory as

```
$touch fname.java
```

Then we open this file in vs code as

```
$code fname.java
```

After that we will write our code and save it then in terminal we compile the code using

```
$javac fname.java
```

Then execute the program using command

```
$java fname.java
```

The general syntax of java program and a basic program is demonstrated below

```
public class hello_world{  
    public static void main(String[] args) {  
        System.out.println("hello world");  
    }  
}
```

In java we mainly have two kinds of data types , one is primitive and second is non primitive , we can also think of it as primitive data types are those which can only store a single value of it's type , and non primitive data types can store multiple values of it's data type , examples of primitive data types are :

Byte, short, int, double, float, long, char, boolean

Examples of non primitive data types are :

String, array, class, interface

Below is a sample program demonstrating primitive variables and its uses

```
public class variable {  
    public static void details() {  
        int a=11;  
        a=a+9;  
        System.out.println(a);  
    }  
    public static void main(String[] args) {  
        int a=10;  
        int b=20;  
        int c=a+b;  
        System.out.println(c);  
    }  
}
```

```

        double d=3;
        float f= 2;
        char ch= 'c';
        short s=2;
        byte bt = 2;
        long l=2323232;
        System.out.println(d+" "+f+" "+ch+" "+s+" "+bt+" "+l);
        details();
    }
}

```

Value of a Variable of one primitive data type can be converted into value of a variable of another primitive data type which is called typecasting, it is usually of two type

1. **Widening** -> converting a smaller type to larger data type size , it goes like  
byte->short->char->int->long->float->double

We do it as

```
int a =9;
```

```
double d = a;
```

2. **Narrowing**-> converting a larger type into smaller type size , it goes like  
double->float->long->int->char->short->byte

We do it as

```
double d=999999;
```

```
int a = (int)d
```

Below is a program demonstrating the same

```

public class typecastingDemo{
    public static void main(String[] args) {
        char uniCode = '\u00A9';
        System.out.println("Unicode " + uniCode);
        byte bt = 10;
        System.out.println("byte " + bt);
        short byteToShort = bt;
        System.out.println("byte to short " + byteToShort);
        short shortVal = 65;
        char ch = (char) shortVal;
        System.out.println("short to char " + ch);
        int charToInt = ch;
        System.out.println("char to int " + charToInt);
        long intToLong = charToInt;
        System.out.println("int to long " + intToLong);
        float longToFloat = intToLong;
        System.out.println("long to float " + longToFloat);
    }
}

```

```

double floatToDouble = longToFloat;
System.out.println("float to double " + floatToDouble);
int large = 1234567;
double intToDouble = large;
System.out.println("int to double " + intToDouble);
byte sm = 10;
short mid = 100;
int result = sm + mid;
System.out.println("byte+short to int " + result);
int largeInt = 2_000_000_000;
long intMulLong = largeInt * 3L;
System.out.println("longResult " + intMulLong);
long val = 123456789123456789L;
float longToFlt = val;
System.out.println("Long to float " + longToFlt);
}
}

```

when we directly give some no to a long var , the no is read in bits hence it might consider it as it , or while doing some operations , hence we need to specify long by adding L at the end of the number

**Logical operators:** logical operators are used to perform logical and and or operations just like and , or gates in digital electronics.They are used to combine two or more condition constraints or to complement the evaluation of original condition under a particular consideration. One thing to keep in mind while using and is that if the first condition is false it will not evaluate the second condition and in OR when the first condition is true it will not evaluate the second condition. Logical operators are : AND(&&) OR(||) NOT(!)

**Ternary operators:** It is one liner replacement of if then else statement and used very commonly in java, in place of if else or even switch, and it is the shortest way to write if else statements, the syntax is as :

Variable = condition ? if true execute this : if false execute this

Below is a program consisting of logical operators and ternary operators.

```

public class logicalOperators {
    public static void main(String[] args){
        boolean tr = true;
        boolean fl = false;
        System.out.println("logical operations ");
        System.out.println("AND operator: true AND false "+(tr && fl));
        System.out.println("AND operator: true AND true "+(tr && tr));
        System.out.println("AND operator: false AND false "+(fl && fl));
    }
}

```

```

System.out.println("AND operator: flase AND true "+(fl && tr));

System.out.println("OR operator: true or false "+(tr || fl));
System.out.println("OR operator: true or true "+(tr || tr));
System.out.println("OR operator: false or true "+(fl ||tr));
System.out.println("OR operator: false or false "+(fl || fl));

System.out.println("Not true "+(!tr));
System.out.println("Not false "+(!fl));

System.out.println("Short circuit");
System.out.println("false && calc "+(fl && (1/0>0)));
System.out.println("true || calc "+(tr || (1/0>0)));

System.out.println("operator precedence");
System.out.println("(true && true) || false "+((tr && tr) || fl));
System.out.println("false || true && true "+(fl || tr && tr));

System.out.println("combining operators with comparison
operator");
int a=1,b=2;
System.out.println("(a<b) && (b>0) "+((a<b) && (b>0)));
System.out.println("(a>b) && (b>0) "+((a>b) && (b>0)));

System.out.println("Complex conditions ");
boolean x = true, y = false, z = true;
System.out.println("(x && y) || (x && z) " + ((x && y) || (x &&
z)));

System.out.println(" x && (y || z) " + (x && (y || z)));
System.out.println(" !x || (y && !z) " + (!x || (y && !z)));

System.out.println("Bitwise vs. logical ");
System.out.println("true & false = " + (tr & fl));
System.out.println("true | false = " + (tr | fl));
System.out.println("true ^ false = " + (tr ^ fl));

System.out.println("Short circuit vs Non ");
int i = 0;
boolean r1 = (fl && (++i > 0));
boolean r2 = (fl & (++i > 0));

```

```

System.out.println("Short circuit " + r1 + ", i = " + i);
System.out.println("Non " + r2 + ", i = " + i);

System.out.println("Logical operators with Non-boolean Operands
");

System.out.println("(1 < 2) && (3 < 4) " + ((1 < 2) && (3 < 4)));
System.out.println(" ('a' < 'b') || ('c' > 'd') " + (('a' < 'b') ||
('c' > 'd')));

System.out.println("Logical operators with Control statements ");
if (tr && !fl) {
    System.out.println("print");
}
int itr = 0;
while (itr < 5 && tr) {
    System.out.println("itr = " + itr);
    itr++;
}

System.out.println("logical operators with methods ");
System.out.println("Positive(10) && Even(12) = " + (Positive(10)
&& Even(12)));
System.out.println("Positive(-7) || Even(9) = " + (Positive(-7) ||
Even(9)));

System.out.println("Logical operators with null ");
String s = null;
System.out.println("(string != null) && (string.length() > 0) = "
+ ((s != null) && (s.length() > 0)));

System.out.println("Using logical operator for conditional
assignments and demonstrating ternary operators ");
int res = tr ? 1 : 0;
System.out.println("result = " + res);

System.out.println("Logical operators in lambda");
java.util.function.Predicate<Integer> posAndEven = no -> no > 0 &&
no % 2 == 0;
System.out.println("Is 12 positive and even " +
posAndEven.test(12));

```

```

        System.out.println("Is -6 positive and even " +
posAndEven.test(-6));

        System.out.println("ternary ");
        int n1=10,n2=10;
        int ans = n1<n2 ? 1: 0;
        System.out.println(ans);
    }
    private static boolean Positive(int n) {
        return n > 0;
    }

    private static boolean Even(int n) {
        return n % 2 == 0;
    }
}

```

**Methods:** methods are a block of code that gets executed when it's called , we can pass data known as parameters to the methods , these are used to perform certain actions and are also called functions, the main purpose of methods is to avoid writing the same code over and over again.

**Constructor:** constructor is basically a block of code that gets automatically executed whenever a new instance of that particular class is created, it is usually used to initialize the objects, we can define constructors just like we define methods, but the constructor should have the same name as that of the class and should not have any return type not even void. We can also have multiple constructors in a class, they all will have the same name ,no return type and should differ in the number and types of parameters,this is called constructor overloading and is very useful when a program gets more n more complicated.

Parameterized constructor:these are the type of constructor that takes parameters, when a new object is created allowing the developer to initialize the object's attributes with values of their own choice

Below is the code demonstrating the same

```

public class employee {
    public String name,age,eid,dept,city,salary,company;
    public void printDetails(){
        System.out.println("name "+name);
        System.out.println("age "+age);
        System.out.println("eid "+eid);
        System.out.println("dept "+dept);
    }
}

```

```

        System.out.println("city "+city);
        System.out.println("salary "+salary);
        System.out.println("company "+company);
    }
    public void setDetails(String name, String age, String eid, String
dept, String city, String salary, String company){
        this.name = name;
        this.age=age;
        this.eid=eid;
        this.dept=dept;
        this.city=city;
        this.salary=salary;
        this.company=company;
        printDetails();
    }
    public employee(){
        name="Ayush";
        age="20";
        eid="lva20";
        dept="dev";
        city="blr";
        salary="100";
        company="mthree";
    }
    public static void main (String [] args){
        employee emp = new employee();
        System.out.println("printing default details");
        emp.printDetails();
        System.out.println("printing custom details");

emp.setDetails("rahul","25","lsjc10","test","pune","200","mthree");
    }
}

```

**Static:** static is a keyword that is used with variables and methods to make them associated with the class and not any particular instance of the class. if we have a method in the same class we can call it using an object or if the method is static we can call it directly also .

By definition static variable is a variable whose memory has been allocated during the compile time itself and is available throughout the entire run of the program and these have higher performance than application variables, these are the variables that are declared in a class outside of any method , constructor or block using the keyword static , but unlike instance

variables we can only have one copy of these variables irrespective of the no of objects that we create as these are associated with the class and not any particular instance of the class. Below is a program demonstrating operations on static variable

```
public class staticExample{
    private static int count =0;
    private int num;
    public staticExample(){
        count++;
        num = count;
    }
    public void printDetails(){
        System.out.println("Instance Number: " + num+ "\n" +"Count: "+count);
    }
    public static void main(String[] args) {
        staticExample obj1 = new staticExample();
        obj1.printDetails();
        staticExample obj2 = new staticExample();
        obj2.printDetails();
        staticExample obj3 = new staticExample();
        obj3.printDetails();
    }
}
```

**Objects:** object is a real world entity , or we can say that an object is an instance of it's class using which we can access the methods and attribute of the class that it belongs to

**Inheritance:** inheritance is basically a mechanism for inheriting the methods and attributes of one class into another class , the class that gets inherited is called super class or parent class and the class the inherits is called child class or subclass. We inherit a class into another using the keyword extends like

Class abc extends def

So here class def is being inherited by the class abc so class abc will be called the subclass or child class and will inherit the methods and attributes of class def which will be called parent class or super class.

```
class vehicle {
}
class car extends vehicle{
    public static void main(String[] args){
        vehicle obj = new car();
        boolean res = obj instanceof car;
        System.out.println(res);
    }
}
```



```
}  
}
```

**Instanceof:** The instanceof keyword checks whether an object is an instance of a class or interface, it compares the instance with type and returns either true and false, in above example object obj is in instance of class car hence it will return true.

**Date:** Using the class module we can access the current date, all we need to do is to import it and then create an object like

```
Date date = new Date();
```

Now the object date will contain the today's day and date, we can also format the date in the format we want using SimpleDateFormat class, we also need to import it then we create an object by passing the format in which we want the date to be like

```
SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
```

Then we can use this object to format the date object like

```
String currentDate=formatter.format(date);
```

Below is a program demonstrating the same

```
import java.util.Date;  
import java.text.SimpleDateFormat;  
public class date {  
    public static void main(String[] args) {  
        Date date = new Date();  
        System.out.println(date);  
        SimpleDateFormat myFormat = new SimpleDateFormat("dd/MM/yyyy");  
        String dt = myFormat.format(date);  
        System.out.println(dt);  
    }  
}
```

**Scanner:** Scanner is class which is used to get input from user and is found in java.util package. To use the class we create an object of the class and use any of the available methods found in scanner class like next(), nextInt() and so on.

We create the object as

```
Scanner scan = new Scanner(System.in);
```

Then we take input as

```
String name = scan.next();
```

**Switch:** Instead of writing many if else statements we use switch statements, where it selects one among the many code blocks to be executed. The switch expression is evaluated once, the value of expression is compared with value of each case if there is a match the associated block

is executed and if not then default block is executed and break statement is used to get out of switch once a block has been properly executed.

Below is a simple program of a calculator demonstrating the use of switch and scanner.

```
import java.util.Scanner;
public class calculator {
    public static void main(String [] args){
        Scanner scan = new Scanner(System.in);
        System.out.println("enter your name");
        String userName=scan.next();
        System.out.println("hello "+userName);
        System.out.println("enter first number");
        int num1=scan.nextInt();
        System.out.println("enter second number");
        int num2=scan.nextInt();
        System.out.println("enter the operator");
        char operator=scan.next().charAt(0);
        int result=0;
        switch(operator){
            case '+':
                result=num1+num2;
                System.out.println(result);
                break;
            case '-':
                result =num1-num2;
                System.out.println(result);
                break;
            case '*':
                result = num1*num2;
                System.out.println(result);
                break;
            case '/':
                result=num1/num2;
                System.out.println(result);
                break;
            case '%':
                result=num1%num2;
                System.out.println(result);
                break;
            default:
                System.out.println("invalid operator selected");
        }
    }
}
```

```

        break;
    }
    scan.close();
}
}

```

Below is a programme designed to simulate a banking application where user can deposit, withdraw , transfer money and check balance.

```

import java.util.*;
import java.text.SimpleDateFormat;
public class bank {
    //method creation to perform various banking operations
    public static void bankOperations(String type){
        //creating an object of Date class to get the date
        Date date = new Date();
        //creating object of simple date format with specified format
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        String currentDate = sdf.format(date); //formatting the date object
        to get date in our specified format
        Scanner sc = new Scanner(System.in); //creating object of scanner
class
        System.out.println("enter the name of account holder");
        String name=sc.next(); //taking name input
        System.out.println("enter the account number");
        String acc=sc.next(); //taking acc no as input
        System.out.println("enter the ifsc");
        String ifsc=sc.next(); //taking ifsc code as input
        int i=0,balance=0;
        while(i==0){ //starting loop
            System.out.println("select the operation");
            System.out.println("1.deposit");
            System.out.println("2.withdraw");
            System.out.println("3.check balance");
            System.out.println("4.transfer funds");
            System.out.println("5.exit");
            int choice=sc.nextInt(); //taking use choice as input
            switch (choice){
                case 1:

```

```

        System.out.println("Enter the amount you want to
deposit");
        int deposit=sc.nextInt();//taking input of deposit
amount
        balance=balance+deposit;//calculating account balance
        System.out.println("amount deposited successfully");
        //printing the balance with account holder detils and
date
        System.out.println("current balance of your "+type+"
account is "+balance+"\n for
        account no "+acc+"\n name: "+name+"\n ifsc: "+ifsc+"\n
date: "+currentDate);
        break;
    case 2:
        System.out.println("Enter the amount you want to
withdraw");
        int withdraw=sc.nextInt();//taking input of withdraw
amount
        if(withdraw>balance){//checking if account has enough
money to withdraw
            System.out.println("insufficient balance");
            System.out.println("current balance of your
"+type+" account is "+balance+"\n for
            account no "+acc+"\n name: "+name+"\n ifsc:
"+ifsc+"\n date: "+currentDate);
            break;
        }
        balance=balance-withdraw;//calculating the remaining
balance
        System.out.println("amount withdrawn successfully");
        //printing the balance with details and date
        System.out.println("current balance of your "+type+"
account is "+balance+"\n for
        account no "+acc+"\n name: "+name+"\n ifsc: "+ifsc+"\n
date: "+currentDate);
        break;
    case 3:
        //printing balance with details and date
        System.out.println("current balance of your "+type+"
account is "+balance+"\n for

```

```

        account no "+acc+"\n name: "+name+"\n ifsc: "+ifsc+"\n
date: "+currentDate);
        break;
    case 4:
        System.out.println("enter the name of person you want
to transfer funds to ");
        String name1=sc.next();//taking input of name of user
to whom funds has to be sent
        System.out.println("enter their account number");
        String acc1=sc.next();//taking input of acc no of user
to whom funds has to be sent
        System.out.println("enter the amount");
        int transfer=sc.nextInt();//taking input of transfer
amount
        if(transfer>balance){//checking if account has enough
balance to complete transfer
            System.out.println("insufficient balance");
            System.out.println("current balance of your
"+type+" account is "+balance+"\n for
        account no "+acc+"\n name: "+name+"\n ifsc:
"+ifsc+"\n date: "+currentDate);
            break;
        }
        balance=balance-transfer;//calculating remaining
balance
        System.out.println("amount transferred successfully
to: "+name1+"\n acc no: "+acc1);
        System.out.println("current balance of your "+type+"
account is "+balance+"\n for
        account no "+acc+"\n name: "+name+"\n ifsc: "+ifsc+"\n
date: "+currentDate);
        break;
    case 5:
        //increasing value of i to break the loop
        i=5;
        break;
    }
}
sc.close();
}

```

```

public static void main(String[] args){
    Scanner sc = new Scanner(System.in); //creating object of scanner
class
    System.out.println("Welcome to ABC bank");
    System.out.println("Select the type of account (1 or 2) ");
    System.out.println("1. savings");
    System.out.println("2. Current");
    int type=sc.nextInt(); //taking input of user choice
    switch(type){
        case 1:
            bankOperations("Savings"); //calling bank operations method
with savings as argument
            break;
        case 2:
            bankOperations("current"); //calling bank operations method
with current as argument
            break;
        default :
            System.out.println("enter valid choice");
            break;
    }
    sc.close(); //closing the sc object of scanner class
}
}

```

Below is a code showing the implementation of control statements ie, if else, if else-if else , nested if , switch, looping statements i.e, for, foreach, while , do while, jumps i.e, continue, break, return, basic exception handling i.e try catch, try catch finally, try catch with resources, and assertion

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileReader;
public class cond_loops_exception {
    public static void conditional(){
        // if else conditional statements
        System.out.println("if else conditional statements");
        int num=-1;
        if(num>0)
            System.out.println("positive");
    }
}

```

```
else
    System.out.println("negative");

//if else if conditional statements
System.out.println("if else if conditional statements");
if(num>0)
    System.out.println("positive");
else if(num==0)
    System.out.println("zero");
else
    System.out.println("negative");

//Nested if

System.out.println("nested if conditional statements");
int n=13;
if(n>0){
    if(n%2==0)
        System.out.println("even");
    else
        System.out.println("odd");
}else{
    System.out.println("negative");
}

//switch case
System.out.println("switch case");
int i=4;
switch(i){
    case 1:
        System.out.println("one");
        break;
    case 2:
        System.out.println("Two");
        break;
    case 3:
        System.out.println("three");
        break;
    case 4:
        System.out.println("Four");
```

```

        break;
    default:
        System.out.println("no too large or too small");
    }
}

public static void looping() {
    //for loop
    System.out.println("for loop");
    for(int i=0;i<5;i++){
        System.out.println(i);
    }

    //for each;
    System.out.println("for each");
    int [] arr ={1,2,3,4,5};
    for(int num: arr){
        System.out.println(num);
    }

    //while loop
    System.out.println("while loop");
    int j=0;
    while(j<5){
        System.out.println(j);
        j++;
    }

    //do while loop
    System.out.println("do while loop");
    int k=0;
    do{
        System.out.println(k);
        k++;
    }while(k<5);
}

public static void jumps() {
    //break
    System.out.println("break demonstration");
    for(int i=0;i<5;i++){
        if(i==3)

```



```

        break;
        System.out.println(i);
    }

    //continue
    System.out.println("continue demonstration");
    for(int i=0; i<5; i++){
        if(i==3)
            continue;
        System.out.println(i);
    }

    //return
    System.out.println("return demonstration");
    System.out.println("sum: "+sum(8,9));
}
public static int sum(int a , int b){
    return a+b;
}
public static void exceptions(){
    //try catch block
    System.out.println("try catch block");
    try{
        int x = 10/0;
        System.out.println(x);
    }catch(Exception e){
        System.out.println(e.getMessage());
    }

    //try catch finally block
    System.out.println("try catch finally block");
    try{
        int x=10/0;
        System.out.println(x);
    }catch(Exception e){
        System.out.println(e.getMessage());
    }finally{
        System.out.println("the is the finally block being executed");
    }
}

```

```

        //try catch with resources
        System.out.println("try catch with resources");
        try{
            BufferedReader br = new BufferedReader(new
FileReader("abc.txt"));
            String str;
            while((str=br.readLine())!=null){
                System.out.println(str);
            }
            br.close();
        }catch(IOException e){
            System.out.println(e.getMessage());
        }
    }
    public static void assertion(){
        //assertion
        System.out.println("assertion");
        int a=-1;
        assert a>0 : "entry cannot be negative";
        System.out.println(a);
    }
    public static void main(String [] args){
        conditional();
        looping();
        jumps();
        exceptions();
        assertion();
    }
}

```

## OOPS

**class** -> class is a collection or set of objects, which shares same properties, characteristics, behavior and attributes

**object** -> object essentially represents the real world entity , an object is an instance of its class which is used to access the attributes and methods of the class

**inheritance** -> it is a mechanism that allows a class to inherit the attributes and methods of another class, the class which inherits is called sub class or child class and the method that gets inherited is called super class or parent class

**encapsulation** -> it is a technique to bind the variables and methods together in one unit , in encapsulation the variables are hidden from other class and can only be accessed using the method inside of its own class

**abstraction** -> it is the process of hiding the implementation detail and showing only the functionality to the user it can be achieved through abstract classes and interfaces

Abstract methods are those methods which does not have a body

Interfaces are those classes which only contains abstract methods

**polymorphism** -> polymorphism is basically ability to take multiple forms , in java using polymorphism an object can respond in different ways for the same message or same method call . In java we implement polymorphism using method overloading and method overriding.

: method overloading -> in java we can define multiple methods with same name and they may or may not have different return types, but it should have different parameters

: method overriding -> in java we can define a different implementation of a method in subclass which is already defined in super class , so when the method is called the method in subclass overrides the method in super class

Below is a program showcasing the above mentions oops concepts

```
//oops concepts
//class - class is essentially a collection of objects with same behavior,
attributes, properties, or we can say
//that it is a blueprint to create a class
//object- object is a real world entity or we can say that it is an
instance of it's class using which we can access
//the attributes and methods of the class to which it belongs to

class car{
    //attributes declaration
    String name;
    String model;
    int year;
    //parameterised constructor
    car(String name, String model, int year){
        this.name=name;
        this.model=model;
        this.year=year;
    }
    //method to display the details
    void display(){
        System.out.println(name+" "+model+" "+year);
    }
}
```

```

/* Encapsulation
 * it is a technique to bind the variables and methods together in one
unit , in encapsulation the variables
 * are hidden from other class and can only be accessed using the method
inside of it's own class
 */
class person{
    //private attribute declaration
    private String name,city;
    private int age;
    //parameterized constructor
    person(String name, String city, int age){
        this.name=name;
        this.city=city;
        this.age=age;
    }
    //method to display the details
    void display(){
        System.out.println(name+" "+city+" "+age);
    }
}

/* Inheritance
 * it is a mechanism that allows a class to inherit the attributes and
methods of another class, the class which
 * inherits is called sub class or child class and the method that gets
inherited is called super class or
 * parent class
 */
class bike{
    //attributes declaration
    String name;
    //constructor
    bike(String name){
        this.name=name;
    }
    //display
    void displaySuper(){
        System.out.println(name);
    }
}

```

```

class model extends bike{
    //constructor
    model(String name){
        super(name);
    }
    void display(){
        System.out.println("model name is printed by super class");
    }
}

/* Polymorphism
 * polymorphism is basically ability to take multiple forms , in java
using polymorphism an object can respond
 * in different ways for the same message or same method call . In java we
implement polymorphism using method
 * overloading and method overriding.
    : method overloading -> in java we can define multiple methods with
same name and they may or may not have different return types, but it
should have
    different parameters
    : method overriding -> in java we can define a different
implementation of a method in sub class which is already defined in super
class , so when
    the method is called the method in sub class over rides the method in
super class
 */
class horsepower extends bike{
    //constructor
    String value;
    horsepower(String value){
        super(value);
    }
    @Override
    //displaySuper method overrides the method in bike class
    void displaySuper(){
        System.out.println("horsepower is "+value);
    }
}

/* abstraction
 * Abstraction is the process of hiding the implementation details and
showing only the functionality to the user.

```

```

* It can be achieved through abstract classes and interfaces.
*/
abstract class calculate{
    //abstract method
    abstract void calculateArea();
    void display(){
        System.out.println("calculate area");
    }
}

class square extends calculate{
    //method overriding
    float side;
    square(float side){
        this.side=side;
    }
    @Override
    void calculateArea(){
        System.out.println(side*side);
    }
}

// Interface
// interface is an abstract class that only contains abstract methods
interface drawable{
    //abstract method
    abstract void draw();
}

class rectangle extends calculate implements drawable{
    int length,breadth;
    public rectangle(int length, int breadth){
        this.lenght = length;
        this.breadth=breadth;
    }
    @Override
    void calculateArea(){
        System.out.println(length*breadth);
    }
    @Override
    public void draw(){
        System.out.println("drawing");
    }
}

```

```

}

public class oops_concepts {
    public static void main(String [] args){
        System.out.println("car and objects");
        car mycar = new car("mahindra","scorpio",2021);
        mycar.display();
        //encapsulation
        System.out.println("encapsulation");
        person p1 = new person("ayush","blr",22);
        p1.display();
        //inheritance
        System.out.println("inheritance");
        model m1 = new model("yamaha R1");
        m1.displaySuper();
        m1.display();
        //polymorphism
        System.out.println("polymorphism");
        bike b1 = new horsepower("220hp");
        b1.displaySuper();
        //abstraction
        System.out.println("abstraction");
        calculate ar = new square(20);
        ar.calculateArea();
        //interface
        System.out.println("interface");
        rectangle rc = new rectangle(10,5);
        rc.calculateArea();
        rc.draw();
    }
}

```

Below is the code for a banking application that uses various conditional statements and demonstrates the concepts of object oriented programming

```

//abstract class that represents a bank account
abstract class acc{
    //private attributes (encapsulation)
    private String name,accNo;

```

```

private double balance;
//parameterised constructor to initialize the account with name ,
//account number and initial balance
public acc(String name, String accNo, double balance){
    this.name=name;
    this.accNo=accNo;
    this.balance=balance;
}
//method that will return the balance
public double getBalance(){
    return balance;
}
//method to deposit money into account
public void deposit(double amount){
    if(amount>0){//check if deposit amount is positive
        balance+=amount;//calculate new balance
        System.out.println("Deposited amount: "+amount+"\n for user
"+name+"\n account no "+accNo);//encapsulation
        System.out.println("Account balance is: "+balance);
    }else{
        System.out.println("invalid amount");
    }
}
//method to withdraw money
public void withdraw(double amount){
    if(amount>0 && amount<=balance){//check for sufficient balance and
valid amount
        balance-=amount;//calculate remaining balance
        System.out.println("amount withdrawn "+amount+"\n for user
"+name+"\n account no "+accNo);
        System.out.println("Account balance is: "+balance);
    }else{
        System.out.println("insufficient balance");
    }
}
//abstract method to calculate interest which will be implemented
differently by subclasses(abstraction)
public abstract void interest();
}
//class savings inheriting acc(inheritance)

```



```

class savings extends acc{
    private double ir=0.4;//fixed interest rate
    //constructor to call superclass constructor
    public savings(String name,String accNo, double balance){
        super(name, accNo, balance);
    }
    //override the interest method (polymorphism)
    @Override
    public void interest(){
        double interestValue=getBalance()*ir;//calculate interest amount
        System.out.println("interest is "+interestValue);
    }
}
//class current inherits class acc (inheritance)
class current extends acc{
    private double overDraft=1000;//overdraft limit
    //constructor to call superclass constructor
    public current(String name, String accNo, double balance){
        super(name,accNo,balance);
    }
    //override the interest method
    @Override
    public void interest(){
        System.out.println("no interest on current account");
    }
    //override the withdraw method to support overdraft
    @Override
    public void withdraw(double amount){
        if(amount >0 && (getBalance() + overDraft >= amount)){
            double remains = getBalance()-amount;
            System.out.println("amount withdrawn "+amount);
            if(remains<0){
                System.out.println("overdraft used :"+Math.abs(remains));
            }
        }else{
            System.out.println("insufficient balance or invalid amount");
        }
    }
}
public class bankAppl {

```

```

public static void main(String[] args) {
    //create saving account obj
    acc saving=new savings("ayush","7687687",2000);
    saving.deposit(800);
    saving.withdraw(200);
    saving.interest();

    //create current account obj
    acc current=new current("rahul","6767676",3500);
    current.deposit(120);
    current.withdraw(400);
    current.interest();
}
}

```

The below specified program demonstrates the usage of **composition** in java

```

//composition example

//component classes
class engine{
    private int cc, hp, torque;
    public engine(int cc, int hp, int torque){
        this.cc = cc;
        this.hp = hp;
        this.torque=torque;
    }
    public void start(){
        System.out.println("capacity: "+cc+" horsepower: "+hp+" torque: "+torque);
    }
}

class wheels{
    private String brand;
    private int size;
    public wheels(String brand, int size){
        this.brand = brand;
        this.size=size;
    }
}

```

```

        public void rotate(){
            System.out.println("brand: "+brand+" size: "+size);
        }
    }
}
// Main class using composition
class cars{
    private engine ENGINE;
    private wheels WHEELS;
    public cars (engine ENGINE, wheels WHEELS){
        this.ENGINE = ENGINE;
        this.WHEELS=WHEELS;
    }
    public void Specification(){
        System.out.println("Specification");
        ENGINE.start();
        WHEELS.rotate();
    }
    // Getter methods for components
    public engine getEngine() {return ENGINE;}
    public wheels getWheels() {return WHEELS;}
}
public class composition {
    public static void main(String [] args){
        engine ENGINE = new engine(1000,200,190);
        wheels WHEELS=new wheels("perelli",180);
        cars c1 = new cars(ENGINE,WHEELS);
        c1.Specification();
        c1.getEngine().start();
        c1.getWheels().rotate();
    }
}

```

The below given code showcases the parameterized constructor use cases based on various different scenarios

```

class Book {
    private String name;
    private String writer;
}

```

```

private int totalPages;
private String isbnCode;
private boolean isBound;

// Parameterized constructor with all attributes
public Book(String name, String writer, int totalPages, String
isbnCode, boolean isBound) {
    this.name = name;
    this.writer = writer;
    this.totalPages = totalPages;
    this.isbnCode = isbnCode;
    this.isBound = isBound;
}

// Parameterized constructor with essential attributes
public Book(String name, String writer, String isbnCode) {
    this(name, writer, 0, isbnCode, false); // Calls the full
constructor with default values
}

// Parameterized constructor for a book without known ISBN
public Book(String name, String writer, int totalPages) {
    this(name, writer, totalPages, "Unknown", false);
}

// Copy constructor
public Book(Book anotherBook) {
    this(anotherBook.name, anotherBook.writer, anotherBook.totalPages,
anotherBook.isbnCode, anotherBook.isBound);
}

@Override
public String toString() {
    return "Book{" +
        "name='" + name + '\'' +
        ", writer='" + writer + '\'' +
        ", totalPages=" + totalPages +
        ", isbnCode='" + isbnCode + '\'' +
        ", isBound=" + isBound +
        '}';
}

```

```

    }
}

class BankAccount {
    private String accNumber;
    private String accHolder;
    private double accBalance;
    private String accType;

    // Parameterized constructor for creating a new account
    public BankAccount(String accHolder, String accType, double
initialBalance) {
        this.accNumber = generateAccNumber();
        this.accHolder = accHolder;
        this.accType = accType;
        this.accBalance = initialBalance;
    }

    // Parameterized constructor for loading an existing account
    public BankAccount(String accNumber, String accHolder, double
accBalance, String accType) {
        this.accNumber = accNumber;
        this.accHolder = accHolder;
        this.accBalance = accBalance;
        this.accType = accType;
    }

    private String generateAccNumber() {
        // Simulating account number generation
        return "BNK" + Math.round(Math.random() * 1000000);
    }

    @Override
    public String toString() {
        return "BankAccount{" +
            "accNumber='" + accNumber + '\'' +
            ", accHolder='" + accHolder + '\'' +
            ", accBalance=" + accBalance +
            ", accType='" + accType + '\'' +
            '}';
    }
}

```

```

    }
}

public class ParameterizedConstructorExamples {
    public static void main(String[] args) {
        // Book examples
        Book novel1 = new Book("Moby Dick", "Herman Melville", 635,
"9780142437247", true);
        Book novel2 = new Book("Pride and Prejudice", "Jane Austen",
"9780679783268");
        Book novel3 = new Book("The Hobbit", "J.R.R. Tolkien", 310);
        Book novel4 = new Book(novel1); // Using copy constructor

        System.out.println("Book Examples:");
        System.out.println(novel1);
        System.out.println(novel2);
        System.out.println(novel3);
        System.out.println(novel4);

        // BankAccount examples
        BankAccount savingsAcc = new BankAccount("Alice Brown", "Savings",
1500.0);
        BankAccount checkingAcc = new BankAccount("BNK654321", "Bob
Green", 3000.0, "Checking");

        System.out.println("\nBank Account Examples:");
        System.out.println(savingsAcc);
        System.out.println(checkingAcc);
    }
}

```

Below code demonstrates how to handle custom exceptions in java for different use cases

```

import java.util.Scanner;

// Custom checked exception for insufficient balance
class LowBalanceException extends Exception {
    private double deficit;

```

```

    public LowBalanceException(double deficit) {
        super("Insufficient balance. You are short by: " + deficit + " to
complete this transaction.");
        this.deficit = deficit;
    }

    public double getDeficit() {
        return deficit;
    }
}

// Custom unchecked exception for invalid account
class AccountNotFoundException extends RuntimeException {
    private String invalidAccountId;

    public AccountNotFoundException(String invalidAccountId) {
        super("Account ID not found: " + invalidAccountId);
        this.invalidAccountId = invalidAccountId;
    }

    public String getInvalidAccountId() {
        return invalidAccountId;
    }
}

// Main class demonstrating the custom exception handling
public class TransactionDemo {

    public static void main(String[] args) {
        UserAccount userAccount = new UserAccount("ACC98765", 1500.0);
        Scanner inputScanner = new Scanner(System.in);

        while (true) {
            try {
                // Prompting user for account ID and withdrawal amount
                System.out.print("Enter your Account ID: ");
                String enteredAccountId = inputScanner.nextLine();
                System.out.print("Enter the amount to withdraw: ");

```

```

        double withdrawalAmount =
Double.parseDouble(inputScanner.nextLine());

        // Withdraw operation may throw AccountNotFoundException
or LowBalanceException
        userAccount.processWithdrawal(enteredAccountId,
withdrawalAmount);

        System.out.println("Withdrawal successful. Remaining
balance: " + userAccount.getCurrentBalance());
        break;
    } catch (LowBalanceException ex) {
        // Handling the custom checked exception
        System.out.println("Transaction failed: " +
ex.getMessage());
        System.out.println("You are short by $" + ex.getDeficit()
+ ". Retry? (yes/no)");
        if (!inputScanner.nextLine().equalsIgnoreCase("yes")) {
            break;
        }
    } catch (AccountNotFoundException ex) {
        // Handling the custom unchecked exception
        System.out.println("Error: " + ex.getMessage());
        System.out.println("Please check and enter a valid account
ID.");
    } catch (NumberFormatException ex) {
        // Handling invalid numeric input for withdrawal amount
        System.out.println("Error: Invalid input. Please enter a
numeric value for the amount.");
    } catch (Exception ex) {
        // Catching any unexpected exceptions
        System.out.println("Unexpected error occurred: " +
ex.getMessage());
        ex.printStackTrace();
        break;
    } finally {
        // Always executes, indicating completion of an attempt
        System.out.println("Transaction attempt finished.");
    }
}

```



```

        inputScanner.close();
    }
}

// Class representing a user bank account
class UserAccount {
    private String userAccountId;
    private double currentBalance;

    public UserAccount(String userAccountId, double startingBalance) {
        this.userAccountId = userAccountId;
        this.currentBalance = startingBalance;
    }

    public double getCurrentBalance() {
        return currentBalance;
    }

    // Method to handle withdrawal, throws custom exceptions
    public void processWithdrawal(String providedAccountId, double amount)
throws LowBalanceException {
        // Verify the account ID (may throw unchecked exception)
        if (!this.userAccountId.equals(providedAccountId)) {
            throw new AccountNotFoundException(providedAccountId);
        }

        // Check if the balance is sufficient (may throw checked
exception)
        if (amount > currentBalance) {
            double deficitAmount = amount - currentBalance;
            throw new LowBalanceException(deficitAmount);
        }

        // Deduct the amount if all checks are passed
        currentBalance -= amount;
    }
}

```

Below is the code that demonstrates various ways in which we can write content to the file, read content from the file, demonstrating file operations, checking details, accessing absolute path, renaming file, deleting files using bufferedReader, inputStream, scanner etc

```
import java.io.*;
import java.nio.file.*;
import java.util.Scanner;
import java.util.stream.Stream;

public class FileIOOperations {

    public static void main(String[] args) {

        String filePath = "abc.txt";
        String data = "Welcome to Java I/O operations!";

        // Writing content to a file
        writeToFile(filePath, data);

        // Reading from the file using different methods
        System.out.println("1. Reading with BufferedReader:");
        readUsingBufferedReader(filePath);

        System.out.println("\n2. Reading with FileInputStream:");
        readUsingFileInputStream(filePath);

        System.out.println("\n3. Reading with Scanner:");
        readUsingScanner(filePath);

        System.out.println("\n4. Reading with Files.readAllLines (NIO):");
        readUsingFilesReadAllLines(filePath);

        System.out.println("\n5. Reading with Files.lines (NIO) and Stream
API:");
        readUsingFilesLines(filePath);

        // Demonstrating various file operations
        performFileOperations(filePath);
    }

    // Method to write data to a file
```

```
public static void writeToFile(String filePath, String data) {
    try (BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(filePath))) {
        bufferedWriter.write(data);
        System.out.println("Data successfully written to file.");
    } catch (IOException e) {
        System.err.println("Error writing to file: " +
e.getMessage());
    }
}

// Method to read file using BufferedReader
public static void readUsingBufferedReader(String filePath) {
    try (BufferedReader bufferedReader = new BufferedReader(new
FileReader(filePath))) {
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        System.err.println("Error reading the file: " +
e.getMessage());
    }
}

// Method to read file using FileInputStream
public static void readUsingFileInputStream(String filePath) {
    try (FileInputStream fileInputStream = new
FileInputStream(filePath)) {
        int character;
        while ((character = fileInputStream.read()) != -1) {
            System.out.print((char) character);
        }
        System.out.println();
    } catch (IOException e) {
        System.err.println("Error reading the file: " +
e.getMessage());
    }
}
```

```

// Method to read file using Scanner
public static void readUsingScanner(String filePath) {
    try (Scanner fileScanner = new Scanner(new File(filePath))) {
        while (fileScanner.hasNextLine()) {
            System.out.println(fileScanner.nextLine());
        }
    } catch (FileNotFoundException e) {
        System.err.println("File not found: " + e.getMessage());
    }
}

// Method to read file using Files.readAllLines (NIO)
public static void readUsingFilesReadAllLines(String filePath) {
    try {
Files.readAllLines(Paths.get(filePath)).forEach(System.out::println);
    } catch (IOException e) {
        System.err.println("Error reading the file: " +
e.getMessage());
    }
}

// Method to read file using Files.lines (NIO) and Stream API
public static void readUsingFilesLines(String filePath) {
    try (Stream<String> lines = Files.lines(Paths.get(filePath))) {
        lines.forEach(System.out::println);
    } catch (IOException e) {
        System.err.println("Error reading the file: " +
e.getMessage());
    }
}

// Method to demonstrate file operations like renaming, deleting, etc.
public static void performFileOperations(String filePath) {
    File originalFile = new File(filePath);

    System.out.println("\nFile Operations:");
    System.out.println("File exists: " + originalFile.exists());
    System.out.println("Is it a file: " + originalFile.isFile());

```

```

        System.out.println("Is it a directory: " +
originalFile.isDirectory());
        System.out.println("Readable: " + originalFile.canRead());
        System.out.println("Writable: " + originalFile.canWrite());
        System.out.println("Absolute Path: " +
originalFile.getAbsolutePath());

        // Renaming the file
        File renamedFile = new File("renamed_" + filePath);
        if (originalFile.renameTo(renamedFile)) {
            System.out.println("File renamed successfully.");
        } else {
            System.out.println("File rename failed.");
        }

        // Deleting the file
        if (renamedFile.delete()) {
            System.out.println("File deleted successfully.");
        } else {
            System.out.println("File deletion failed.");
        }
    }
}

```

Below written code demonstrates the use of access modifiers in different cases and how variables declared with different access modifiers can be accessed from different classes and packages of java project

```

public class modifierExampleDemo {

    // Public member variable
    // Can be accessed from any other class
    public int globalVar = 100;

    // Protected member variable
    // Can be accessed within the same package and by subclasses (even if
in a different package)
    protected int inheritedVar = 200;
}

```

```

// Default (package-private) member variable
// Can be accessed only within the same package
int packageVar = 300;

// Private member variable
// Can be accessed only within this class
private int restrictedVar = 400;

// Public method
// Can be called from any other class
public void globalMethod() {
    System.out.println("This is a public method");
    // Public methods can access all members of the class
    System.out.println("Accessing all variables:");
    System.out.println("globalVar: " + globalVar);
    System.out.println("inheritedVar: " + inheritedVar);
    System.out.println("packageVar: " + packageVar);
    System.out.println("restrictedVar: " + restrictedVar);
}

// Protected method
// Can be called within the same package and by subclasses
protected void inheritedMethod() {
    System.out.println("This is a protected method");
}

// Default (package-private) method
// Can be called only within the same package
void packageMethod() {
    System.out.println("This is a default (package-private) method");
}

// Private method
// Can be called only within this class
private void restrictedMethod() {
    System.out.println("This is a private method");
}

// Inner class to demonstrate access modifiers within the same class

```

```

    public class InternalClass {
        public void accessAllMembers() {
            // Inner classes can access all members of the enclosing
class,
            // including private members
            System.out.println("InternalClass accessing restrictedVar: " +
restrictedVar);
            restrictedMethod();
        }
    }
}

// This class is in the same file to demonstrate default access
class PackageSameFileClass {
    public void accessDemoMembers() {
        modifierExampleDemo demo = new modifierExampleDemo();
        // Can access public, protected, and default members
        System.out.println("PackageSameFileClass accessing globalVar: " +
demo.globalVar);
        System.out.println("PackageSameFileClass accessing inheritedVar: "
+ demo.inheritedVar);
        System.out.println("PackageSameFileClass accessing packageVar: " +
demo.packageVar);
        // Cannot access private members
        // System.out.println(demo.restrictedVar); // This would cause a
compilation error

        demo.globalMethod();
        demo.inheritedMethod();
        demo.packageMethod();
        // demo.restrictedMethod(); // This would cause a compilation
error
    }
}

// File: PackageSubClass.java (Assume this is in the same package)

// Subclass in the same package
class PackageSubClass extends modifierExampleDemo {
    public void accessDemoMembers() {

```

```

        // Can access public, protected, and default members of the
superclass
        System.out.println("PackageSubClass accessing globalVar: " +
globalVar);
        System.out.println("PackageSubClass accessing inheritedVar: " +
inheritedVar);
        System.out.println("PackageSubClass accessing packageVar: " +
packageVar);
        // Cannot access private members of the superclass
        // System.out.println(restrictedVar); // This would cause a
compilation error

        globalMethod();
        inheritedMethod();
        packageMethod();
        // restrictedMethod(); // This would cause a compilation error
    }
}

// File: DifferentPackageClass.java (Assume this is in a different
package)

// Uncomment the following line when actually placing this class in
another package
// package com.example.differentpackage;

// import com.example.ModifierExampleDemo;

// Class in a different package
public class DifferentPackageClass {
    public void accessDemoMembers() {
        modifierExampleDemo demo = new modifierExampleDemo();
        // Can only access public members
        System.out.println("DifferentPackageClass accessing globalVar: " +
demo.globalVar);
        // Cannot access protected, default, or private members
        // System.out.println(demo.inheritedVar); // Compilation error
        // System.out.println(demo.packageVar); // Compilation error
        // System.out.println(demo.restrictedVar); // Compilation error
    }
}

```



```

        demo.globalMethod();
        // demo.inheritedMethod(); // Compilation error
        // demo.packageMethod();   // Compilation error
        // demo.restrictedMethod(); // Compilation error
    }
}

// File: DifferentPackageSubClass.java (Assume this is in a different
package)

// import com.example.ModifierExampleDemo;

// Subclass in a different package
public class DifferentPackageSubClass extends modifierExampleDemo {
    public void accessDemoMembers() {
        // Can access public and protected members of the superclass
        System.out.println("DifferentPackageSubClass accessing globalVar:
" + globalVar);
        System.out.println("DifferentPackageSubClass accessing
inheritedVar: " + inheritedVar);
        // Cannot access default or private members of the superclass
        // System.out.println(packageVar); // Compilation error
        // System.out.println(restrictedVar); // Compilation error

        globalMethod();
        inheritedMethod();
        // packageMethod(); // Compilation error
        // restrictedMethod(); // Compilation error
    }
}

```

ENUM- enum is a special data type that consists of set of predefined named values separated by commas, Below is a simple program that demonstrates how enums are created and operated upon

```

import java.util.*;
public class book {
    public enum books{
        b1(100,50),

```

```

        b2(200,75),
        b3(300,100);
        private final int pages,price;

        private books(int pages, int price){
            this.pages = pages;
            this.price=price;
        }
        public int getpage(){
            return pages;
        }
        public int getprice(){
            return price;
        }
    }

    public static void main(String[] args){
        System.out.println("welcome to the book store \nThese are the
available books");
        for (books b : books.values()) {
            System.out.println("Book Name: " + b.name() + ", Price: " +
b.getprice()+"", Pages: "+b.getpage());
        }
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the choice ");
        String Choice = scan.next();
        try{
            books selected = books.valueOf(Choice);
            System.out.println("Selected book is "+selected.name());
            System.out.println("price is: "+selected.getprice());
            System.out.println("No of pages: "+selected.getpage());
        }catch(Exception e){
            System.out.println("Invalid choice");
        }
        scan.close();
    }
}

```

**Apache Maven** -> It is a software project management and build tool that helps developers build, publish and deploy projects.

Firstly we need to download the maven from it's website then extract it and set up the environment variables, then go to bash and run `$mvn -version` to check if it has been properly set up or not

Then to set up a maven project we run the following command

```
$mvn archetype:generate -DgroupId=com.cache -DartifactId=my-caching-project
```

```
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

After that we initialize git

```
$git init
```

Then create the gitignore file and write name of all the files that we do not want to be pushed

```
$touch .gitignore
```

Then follow the general git commands after writing the code

```
$git add -all
```

```
$git commit -m "caching"
```

go to github create new repo with same name as here

```
$git remote add origin URL
```

```
$git push -u origin master
```

```
cd my-caching-project
```

```
$mvn clean install -> to install all dependencies
```

**Caching** -> Is a technique to access frequently accessed data in a fast and easily retrievable location so that reducing the need to fetch the same data repeatedly from a slower data source like a database

```
package com.cache;
import java.io.*;
import java.util.*;
public class MovieRecommendation {
    /**
     * Cache for storing recent movie recommendations.
     * This cache uses a LinkedHashMap with the following characteristics:
     *
     * @param initialCapacity 100 - The initial capacity of the map.
     * @param loadFactor 0.75f - The load factor used to determine when to
     * resize the map.
     * @param accessOrder true - Enables access-order iteration (least
     * recently used order).
     *
     * How it works:
     * 1. The cache stores movie recommendations as key-value pairs
     * (String, String).
     * 2. It has a fixed capacity of 100 entries.
     */
}
```

```

    * 3. When the cache reaches capacity and a new entry is added, the
least recently accessed
    *    entry is automatically removed.
    * 4. The load factor of 0.75 means the map will resize when it's 75%
full, improving performance.
    * 5. Access-order iteration ensures that entries are ordered based on
their access time,
    *    with the least recently used entry at the beginning and the most
recently used at the end.
    */
    //L1 Cache Implementation
    private static final int MAX_ENTRIES = 100;
    private static final Map<String, String> recentMovieCache =
        new LinkedHashMap<String, String>(100, 0.75f, true){
            protected boolean removeEldestEntry(Map.Entry<String, String>
eldest){
                return size() > MAX_ENTRIES;
            }
        };
    //L2 Cache Implementation
    private static final int MAX_ENTRIES_L2 = 1000;
    private final Map<String, String> popularMovieCache =
        new LinkedHashMap<String, String>(MAX_ENTRIES_L2, 0.75f, true){
            protected boolean removeEldestEntry(Map.Entry<String, String>
eldest){
                return size() > MAX_ENTRIES_L2;
            }
        };
    public static void main(String[] args) {

        Map<String, String> recentMovieCache =
            new LinkedHashMap<String, String>(100, 0.75f, true);

        //L2 Cache Implementation

        Map<String, String> popularMovieCache =
            new LinkedHashMap<String, String>(1000, 0.75f, true);

        String[] genres= {"Action", "Comedy", "Drama", "Horror",
"Romance", "Sci-Fi", "Thriller"};

```

```

        //Write code to generate 1000 movies with random genres and
push in popularMovieCache
        for(int i=0; i<2000; i++){
            String movie = "Movie" + i;
            String genre = genres[new
Random().nextInt(genres.length)];
            popularMovieCache.put(movie, genre);
        }
        //Write code to generate 100 recently viewed movies and
push in recentMovieCache
        for(int i=0; i<100; i++){
            String movie = "Movie" + i;
            recentMovieCache.put(movie, "Recently Viewed");
        }
        //Write code to print it with time it takes to fetch the
movie form map
        long startTime = System.nanoTime();
        String movie = popularMovieCache.get("Movie100");
        long endTime = System.nanoTime();
        System.out.println("Time taken to fetch the movie: " +
(endTime - startTime) + " nanoseconds");
    }
}

```

## CollectionsConcepts

### LISTS

**List** -> list interface in java provides a way to store the ordered collections, it is the child interface of collections.

**ArrayList** -> it is a part of the java.util package and is a resizable array implementation of the List interface. Unlike regular arrays in Java, which have a fixed size, an ArrayList can grow and shrink dynamically as elements are added or removed.

**LinkedList** -> In Java, a LinkedList is part of the java.util package and implements the List, Deque, and Queue interfaces. Unlike an ArrayList, which uses a dynamic array to store elements, a LinkedList stores elements in a doubly linked list, where each element (node) contains a reference to the next and previous node

**Stack** -> In Java, a Stack is a data structure that operates on the Last In, First Out (LIFO) principle, meaning the last element added is the first one to be removed. Java provides a Stack class in the java.util package, which is a subclass of Vector.

**Vector** -> In Java, a Vector is a part of the `java.util` package and implements a dynamic array that can grow as needed to accommodate new elements. It is similar to `ArrayList` but with two key differences: synchronization and legacy use.

## SETS

**HashSet** -> In Java, a HashSet is a part of the `java.util` package and is a widely used implementation of the Set interface. It is backed by a Hashtable and does not allow duplicate elements. Unlike lists, HashSet does not maintain the order of its elements.

**LinkedHashSet** -> In Java, a LinkedHashSet is a subclass of HashSet and implements the Set interface. It combines the functionality of a HashSet with a linked list to maintain insertion order. This means that elements in a LinkedHashSet are stored in the order in which they were inserted, unlike HashSet which does not maintain any specific order.

**TreeSet** -> In Java, a TreeSet is a part of the `java.util` package and implements the Set interface. It is a sorted set that stores elements in a natural ordering (ascending by default) or based on a custom comparator provided at the time of creation. Internally, TreeSet is backed by a Red-Black Tree, a self-balancing binary search tree.

## QUEUE

**Queue** -> In Java, a Queue is a part of the `java.util` package and is an interface that extends the Collection interface. It represents a FIFO (First-In-First-Out) data structure where elements are added at the tail (end) and removed from the head (front). The most common use of a Queue is in scenarios where you need to maintain the order of elements while processing them.

**Deque** -> In Java, a Deque (short for Double-Ended Queue) is a part of the `java.util` package and is an interface that extends Queue. Unlike a regular queue (which operates in FIFO order), a Deque allows you to insert and remove elements from both the front and the back, offering FIFO (First-In-First-Out) or LIFO (Last-In-First-Out) behavior depending on how you use it.

**PriorityQueue** -> In Java, a PriorityQueue is a part of the `java.util` package and implements the Queue interface. Unlike a regular queue, which follows a FIFO (First-In-First-Out) order, a PriorityQueue orders its elements based on their natural ordering (for example, for numbers it orders them in ascending order) or according to a custom comparator that you can provide during its construction.

**BlockingQueue** -> In Java, a BlockingQueue is an interface in the `java.util.concurrent` package that extends the Queue interface. It is designed for use in concurrent programming and allows threads to wait for the queue to become non-empty when attempting to retrieve elements, or wait for space to become available when attempting to insert elements. This makes it particularly useful in producer-consumer scenarios.

## MAP

**HashMap** -> In Java, a HashMap is a part of the `java.util` package and is an implementation of the Map interface. It stores key-value pairs and allows for efficient retrieval, insertion, and deletion operations based on the keys. It is one of the most commonly used data structures for storing and manipulating collections of data.

**LinkedHashMap** -> In Java, a LinkedHashMap is a part of the `java.util` package and extends the HashMap class. It maintains a linked list of the entries in the map, which allows it to preserve the order of insertion. This means that when you iterate over a LinkedHashMap, you get the entries in the order they were added.

**TreeMap** -> In Java, a TreeMap is a part of the `java.util` package and implements the NavigableMap interface, which in turn extends the SortedMap interface. It is a map that stores its entries in a sorted order based on the natural ordering of its keys, or by a specified comparator provided at the time of creation.

**HashTable** -> In Java, a Hashtable is part of the `java.util` package and is a collection that implements a hash table data structure. It is similar to HashMap, but there are some key differences. Hashtable was part of the original version of Java, while HashMap was introduced later with the Java Collections Framework.

**ConcurrentMap** -> In Java, a ConcurrentMap is an interface that extends the Map interface and provides additional methods for thread-safe operations. It is part of the `java.util.concurrent` package and is designed for concurrent access by multiple threads, offering better performance than synchronized collections.

## UTILITY CLASSES

**Arrays** -> In Java, arrays are data structures that store multiple values of the same data type. Arrays are a fixed-size structure, meaning that once an array is created, its size cannot be changed. Arrays in Java can store both primitive types (e.g., `int`, `char`, `double`) and reference types (e.g., objects like `String`).

**List** -> In Java, a List is an ordered collection (also known as a sequence) that allows duplicate elements. It is part of the `java.util` package and is an interface that extends the Collection interface. Lists provide positional access, meaning that elements can be accessed based on their position in the list.

**UnmodifiableList** -> In Java, an unmodifiable list is a list that cannot be modified after it has been created. This means that attempts to add, remove, or modify elements in such a list will result in an `UnsupportedOperationException`. The unmodifiable list is a wrapper around an existing list, providing a read-only view of that list.

**SynchronizedList** -> In Java, a synchronized list is a thread-safe list that ensures that all access to the list (both read and write operations) is synchronized, meaning only one thread can access the list at a time. This prevents concurrent modification issues when multiple threads are interacting with the list simultaneously.

Below is a program demonstrating all of the above concepts, on how to create them modify them and basically make use of it

```

import java.util.*;
import java.util.concurrent.*;
import java.util.stream.Collectors;
public class CollectionsImplementation {
    public static void main(String[] args){
        //List interface
        //ArrayList
        ArrayList<String> AL = new ArrayList<>();
        AL.add("mango");
        AL.add("banana");
        AL.add("grapes");
        System.out.println("ArrayList: "+AL);

        //Linked List
        LinkedList<String> ll = new LinkedList<>(AL);
        ll.add(1,"hello");
        ll.add(2,"world");
        ll.add(3,"how are you");
        System.out.println("LinkedList: "+ll);

        //vector
        Vector<String> v = new Vector<>(AL);
        v.add("orange");
        System.out.println("Vector: "+v);

        //Stack
        Stack<String> st = new Stack<>();
        st.push("apple");
        st.push("cherry");
        System.out.println("Stack: "+st);
        System.out.println("element popped from stack: "+st.pop());

        //operations
        Collections.sort(AL);
        System.out.println("Sorted arrayList "+AL);
        System.out.println("Index of grapes in arraylist(Using binary
search): "+Collections.binarySearch(AL,"grapes"));

        //SET
        //HashSet implementation (no specific order)

```



```

HashSet<String> hs = new HashSet<>();
hs.add("yamaha");
hs.add("suzuki");
hs.add("honda");
hs.add("kawasaki");
System.out.println("HashSet (no specific order): "+hs);

//linked HashSet (maintains insertion order)
LinkedHashSet<String> lhs = new LinkedHashSet<>();
lhs.add("yamaha");
lhs.add("suzuki");
lhs.add("honda");
lhs.add("kawasaki");
System.out.println("linkedHashSet (maintains insertion order):
"+lhs);

//TreeSet (red Black Tree Implementaion)
TreeSet<String> ts = new TreeSet<>();
ts.add("yamaha");
ts.add("suzuki");
ts.add("honda");
ts.add("kawasaki");
System.out.println("TreeSet (Red Black Tree Implementation):
"+ts);

//operations
HashSet<String> hs1=new HashSet<>(Arrays.asList("A","B","C"));
HashSet<String>hs2=new HashSet<>(Arrays.asList("B","C","D"));
hs1.retainAll(hs2);
System.out.println("Intersection: "+hs1);

//Queue
//priority queue
PriorityQueue<String> pq = new PriorityQueue<>();
pq.offer("bmw");
pq.offer("audi");
pq.offer("mercedes");
System.out.println("PriorityQueue: "+pq);
//access the highest priority element

```

```

        System.out.println("element polled from priority queue:
"+pq.poll());

//queue
Queue <String>q = new LinkedList<>();
q.offer("bmw");
q.offer("audi");
q.offer("mercedes");
System.out.println("Queue: "+q);
System.out.println("element polled from queue: "+q.poll());

//deque
Deque<String> dq = new ArrayDeque<>();
dq.offerFirst("bmw");
dq.offerLast("audi");
dq.offerLast("mercedes");
System.out.println("Deque: "+dq);
System.out.println("element polled from deque: "+dq.pollFirst());
System.out.println("element polled from deque: "+dq.pollLast());

//blocking queue (thread safe queueing with blocking operations)
BlockingQueue <String> bq = new LinkedBlockingQueue<String>();
bq.offer("hello");
bq.offer("audi");
System.out.println("blocking queue: "+bq);

//Map Interfaces
//HashMap
Map<String, Integer> hm = new HashMap<>();
hm.put("bmw", 5);
hm.put("audi", 4);
hm.put("mercedes", 3);
System.out.println("HashMap: "+hm);

//LinkedHashMap (Maintain insertion order)
LinkedHashMap<String,Integer>lhm=new LinkedHashMap<>();
lhm.put("bmw", 5);
lhm.put("audi", 4);
lhm.put("mercedes",3);
System.out.println("LinkedHashMap: "+lhm);

```

```

//TreeMap (red black tree)
TreeMap<String, Integer> tm = new TreeMap<>();
tm.put("bmw", 5);
tm.put("audi", 4);
System.out.println("tree map: "+tm);

//HashTable: Thread safe hash table
Hashtable<String, Integer> ht = new Hashtable<>();
ht.put("bmw", 5);
ht.put("audi", 4);
System.out.println("hashtable "+ht);

//Concurrent hash map (Thread safe hash table with better
performance than hashtable)
ConcurrentHashMap<String,Integer> cm = new ConcurrentHashMap<>();
cm.put("bmw", 5);
cm.put("audi", 4);
System.out.println("Concurrent hash map "+cm);

//operations
System.out.println("keys in hashmap "+hm.keySet());
System.out.println("Values in hashmap: "+hm.values());
System.out.println("Entries in hashmap: "+hm.entrySet());

//Utility classes
//Arrays
int [] array={1,2,3,4,5,6};
Arrays.sort(array);
System.out.println("array: "+Arrays.toString(array));
System.out.println("binary search for 5:
"+Arrays.binarySearch(array,5));

//Collections utility class
List<String> lt = Arrays.asList("apple", "banana", "mango");
Collections.reverse(lt);
System.out.println("Reversed: " + lt);
Collections.shuffle(lt);
System.out.println("Shuffled: " + lt);
System.out.println("Max element: " + Collections.max(lt));

```

```

        System.out.println("Min element: " + Collections.min(lt));

        //unmodifiable collection
        List<String> ult = Collections.unmodifiableList(new
ArrayList<>(lt));
        System.out.println("unmodifiable collection "+ult);

        //synchronized list
        List <String> slt = Collections.synchronizedList(new
ArrayList<>(lt));
        System.out.println("Synchronized list: "+slt);

        //Streams
        List<Integer> strm = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        List<Integer> evenSquares = strm.stream()
                                .filter(n -> n % 2 == 0)
                                .map(n -> n * n)
                                .collect(Collectors.toList());
        System.out.println("Even squares: " + evenSquares);
    }
}

```

**MultiThreading** ->multithreading is used for concurrent execution of two or more parts of the program at the same time, each of such part is called thread, we can also say that thread is a lightweight process within a process, we implement multithreading in two different ways

- class abc **extends** Thread (extending class thread)
- class abc **implements** Runnable (implementing Runnable interface)

the execution of each thread starts from the public void **run()** method

**Thread.sleep()**: thread.sleep() method is used to temporarily pause the execution of a particular thread for a specified period of time, suppose 2 sec is the specified time , then the thread execution stops and then after 2 seconds the execution begins from the same point where it was paused

Below program demonstrate thread usage and various different functions using which we operate on thread

```

public class multThreading {
    // volatile keyword is used to make sure that the variable is always
read from main memory
    private volatile boolean flag = false;
    public synchronized void syncMethod() {

```

```

        System.out.println(Thread.currentThread().getName()+" entered sync
method");

        try{
            Thread.sleep(1000);
        }catch(Exception e){
            System.out.println(e.getMessage());
        }
        System.out.println("exited synchronized method");
    }

    class MyRunnable implements Runnable{
        public void run(){
            System.out.println(Thread.currentThread().getName()+" is
running ");
        }
    }

    class myThread extends Thread{
        public void run(){
            System.out.println(Thread.currentThread().getName()+" is
running");
        }
    }

    public void demonstrateThreading(){
        Thread t1 = new Thread(new MyRunnable(),"Runnable thread");
        t1.start();

        Thread t2 = new myThread();
        t2.setName("extened Thread");
        t2.start();

        Thread t3 = new
Thread()->System.out.println(Thread.currentThread().getName()+" is
running"), "lambdaThread");
        t3.start();
        //thread set demonstration
        System.out.println("Thread t3 state "+t3.getState());
        //thread priority
        t3.setPriority(Thread.MAX_PRIORITY);
    }

```

```

        System.out.println("Thread t3 priority  "+t3.getPriority());
        //joining thread
        try{
            t1.join();
            t2.join();
            t3.join();
        }catch(InterruptedException e){
            e.printStackTrace();
        }
        synchronized(this){
            System.out.println(Thread.currentThread().getName()+" in
synchronized block");
        }final Object lock = new Object();

        Thread waiter = new Thread(() -> {
            synchronized(lock) {
                try {
                    System.out.println("Waiter is waiting");
                    lock.wait();
                    System.out.println("Waiter is notified");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread notifier = new Thread(() -> {
            synchronized(lock) {
                System.out.println("Notifier is notifying");
                lock.notify();
            }
        });

        waiter.start();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        notifier.start();

```

```

Thread sleeper = new Thread(() -> {
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        System.out.println("Sleeper was interrupted");
    }
});
sleeper.start();
sleeper.interrupt();
ThreadLocal<Integer> threadL = new ThreadLocal<>();
threadL.set(60);
System.out.println("ThreadLocal value: " + threadL.get());

final Object r1 = new Object();
final Object r2 = new Object();
Thread deadlockT1 = new Thread(() -> {
    synchronized(r1) {
        System.out.println("Thread 1: Holding resource 1...");
        try { Thread.sleep(100);} catch (InterruptedException e)
{{
            System.out.println("Thread 1: Waiting for resource 2...");
            synchronized(r2) {
                System.out.println("Thread 1: Holding resource 1 and
resource 2");
            }
        }
    }
});
Thread deadlockT2 = new Thread(() -> {
    synchronized(r2) {
        System.out.println("Thread 2: Holding resource 2...");
        try { Thread.sleep(100);} catch (InterruptedException e)
{{
            System.out.println("Thread 2: Waiting for resource 1...");
            synchronized(r1) {
                System.out.println("Thread 2: Holding resource 2 and
resource 1");
            }
        }
    }
});
deadlockT1.start();

```

```

        deadlockT2.start();
    }
    public static void main(String [] args){
        new multThreading().demonstrateThreading();
    }
}

```

Below program demonstrates a simple caching

```

import java.util.*;
public class SimpleCache {
    private final Map<K,V> cache ;
    public SimpleCache() {
        this.cache = new HashMap<K,V>();
    }
    public void putItem(K key, V value){
        cache.put(key, value);
    }
    public V getItem(k Key){
        return cache.get(key);
    }
    public void deleteItem(K key){
        cache.remove(key);
    }
    public void clearItem(K key){
        cache.clear();
    }

    public static void main(String[] args){
        SimpleCache cache = new SimpleCache();
        cache.putItem("k1", "v1");
        cache.putItem("k2", "v2");
        cache.putItem("k3", "v3");
        System.out.println(cache.getItem("k1"));
        cache.deleteItem("k3");
        System.out.println("Size of cache is "+cache.size());
        cache.clearItem();
        System.out.println("size after clearing the cache:
"+cache.size());
    }
}

```



```
}  
}
```

Below program demonstrates LRU caching

```
import java.util.*;  
public class LRUCache<K,V> extends LinkedHashMap<K,V> {  
    private final int cp;  
    public LRUCache(int cp){  
        super(cp,0.75f,true);  
        this.cp=cp;  
    }  
    @Override  
    protected boolean removeEldestEntry(Map.Entry<K,V> eldest){  
        return size()>cp;  
    }  
    public static void main(String[] args){  
        LRUCache<String,String> cache=new LRUCache<String,String>(3);  
        cache.put("k1","v1");  
        cache.put("k2","v2");  
        cache.put("k3","v3");  
        System.out.println(cache);  
        System.out.println(cache.get("k1"));  
        cache.put("k4","v4");  
        System.out.println(cache);  
        cache.put("k5","v5");  
        System.out.println(cache);  
    }  
}
```

Below program is a simple demonstration of synchronized methods and threading

```
class BankAcc extends Thread{  
    private int balance=0;  
    BankAcc(){  
        balance=0;  
    }  
    public synchronized void deposit(int amount){
```

```

        balance+=amount;
        System.out.println("Deposited: "+amount+" Balance: "+balance);
    }
    public synchronized void withdraw(int amount){
        if(balance>=amount){
            balance-=amount;
            System.out.println("Withdrawn: "+amount+" Balance: "+balance);
        }else{
            System.out.println("Insufficient balance");
        }
    }
    public int getBalance(){
        return balance;
    }
}

public class BankDemo {
    public static void main(String[] args) throws InterruptedException {
        BankAcc acc = new BankAcc();
        Thread depositThread = new Thread(() -> {
            for(int i=0;i<5;i++){
                acc.deposit(100);
                try{
                    Thread.sleep(100);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        }, "Deposit Thread");
        Thread withdrawThread = new Thread(() -> {
            for(int i=0;i<5;i++){
                acc.withdraw(80);
                try{
                    Thread.sleep(100);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        }, "Withdraw Thread");
        depositThread.start();
        withdrawThread.start();
    }
}

```

```

        System.out.println("Final Balance: "+acc.getBalance());
    }
}

```

Below is another program demonstrating simple caching , and also the concept of **cache hit** and **cache miss**

```

import java.util.*;

public class BookLibrary {
    private final Map <String,Book> books = new HashMap<>();
    private final Map <String,Book> cache = new HashMap<>();
    private final int CACHE_SIZE=5;
    private int cacheHit=0,cacheMiss=0;

    public BookLibrary(){
        books.put("1",new Book("1","b1","a1"));
        books.put("2",new Book("2","b2","a2"));
        books.put("3",new Book("3","b3","a3"));
        books.put("4",new Book("4","b4","a4"));
        books.put("5",new Book("5","b5","a5"));
    }

    public Book getBook(String id){
        Book book = cache.get(id);
        if(book!=null){
            cacheHit++;
            System.out.println("cache hit for book "+id);
            return book;
        }
        else{
            System.out.println("cache miss for book "+id);
            cacheMiss++;
            book = books.get(id);
            if(book!=null){
                cache.put(id,book);
            }
            return book;
        }
    }

    public void addToCache(String id, Book book){
        if(cache.size()>=CACHE_SIZE){

```

```

        String toRemove=cache.keySet().iterator().next();
        cache.remove(toRemove);
        System.out.println("cache is full , removing least recently
used book "+toRemove);
    }
    cache.put(id,book);
    System.out.println("book added to cache "+id);
}
public void printCacheStats(){
    System.out.println("cache Hits: "+cacheHit);
    System.out.println("cache Miss: "+cacheMiss);
    System.out.println("cache size: "+cache.size());
    System.out.println("Hit Miss ratio:
"+((double)cacheHit/(cacheHit+cacheMiss)));
    System.out.println("books in cache: "+cache.keySet());
}
private static class Book{
    String id,bookName,author;
    public Book(String id, String bookName, String author) {
        this.id=id;
        this.bookName=bookName;
        this.author=author;
    }
    @Override
    public String toString(){
        return "Book{id='"+id+"', title='"+bookName+"',
author='"+author+"'}";
    }
}
public static void main(String[] args){
    BookLibrary cache = new BookLibrary();
    String
[]requestedBooks={"1","2","3","4","5","6","1","2","3","4","5","6","5","4",
"3","9"};
    for(String id: requestedBooks){
        Book book = cache.getBook(id);
        if(book!=null){
            System.out.println("Retrieved book: "+book);
        }else{
            System.out.println("Book not found");
        }
    }
}

```

```

    }
    }
    cache.printCacheStats();
}
}

```

Another caching example with reading and writing from and to the documents and the concept of **executor** -> java executor framework is used to run the runnable objects without creating new threads every time and mostly reusing the already existing threads.

```

import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.io.Serializable;
import java.io.ObjectInputStream;
import java.io.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.io.Serializable;
import java.io.ObjectInputStream;
import java.io.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
//Write comment for each line of code auto generated why its is required
public class DetailDocumentCache {
    //ConcurenthashMap is used instead of hashmap because it is thread
    safe and it is used in concurrent environment
    //It allows multiple threads to access and modify the cache
    concurrently without causing any issues

```

```

    //Unlike Collections.synchronizedMap, it does not lock the entire map
    during write operations,
    //better performance and scalability in multi-threaded environments
    private final ConcurrentHashMap<String,Document> cache;

    private final String diskStoragePath;
    private final int maxCacheSize;

    //AtomicInteger is used instead of integer because it provides atomic
    operations on the integer value
    //AtomicInteger ensures that the integer value is updated atomically,
    which means that the value is updated in a single step without any
    interruptions
    //This is useful in concurrent environments where multiple threads may
    be accessing and updating the integer value simultaneously
    //AtomicInteger provides methods like getAndIncrement,
    getAndDecrement, compareAndSet, etc., which are thread-safe and do not
    suffer from the issues of race conditions and inconsistent reads
    private final AtomicInteger cacheHits=new AtomicInteger(0);
    private final AtomicInteger cacheMisses=new AtomicInteger(0);

    public DetailDocumentCache(int maxCacheSize,String diskStoragePath){
        this.maxCacheSize=maxCacheSize;
        this.diskStoragePath=diskStoragePath;
        this.cache=new ConcurrentHashMap<>(maxCacheSize);

        try{
            Files.createDirectories(Paths.get(diskStoragePath));
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    public Document getDocumentIfExistElseSave(String
documentId,DetailDocumentCache cache1) throws
IOException,ClassNotFoundException{
        Document cachedDocument=cache.get(documentId);
        if(cachedDocument!=null){
            cacheHits.incrementAndGet();
            return cachedDocument;

```

```

    }
    else{
        cacheMisses.incrementAndGet();

        Document document=loadDocumentFromDisk(documentId);
        if(document== null){
            Document newDocument=new Document(documentId,"Content for
"+documentId,System.currentTimeMillis(),System.currentTimeMillis());
            cache1.saveDocument(newDocument);
            System.out.println("Saved document: "+documentId);

        }
        //System.out.println("getting from disk , id:" + documentId);

        return newDocument;
    }

}

}

public void saveDocument(Document document) throws IOException{
    cache.put(document.getDocumentId(),document);
    evictCacheIfNecessary();
    saveToDisk(document);
}

private void saveToDisk(Document document) throws IOException{
    Path filePath=Paths.get(diskStoragePath,document.getDocumentId());
    try(ObjectOutputStream out=new
ObjectOutputStream(Files.newOutputStream(filePath))){
        out.writeObject(document);
    }
}

}

public void printCacheStatistics(){
    System.out.println("Cache Hits: "+cacheHits.get());
    System.out.println("Cache Misses: "+cacheMisses.get());
    System.out.println("Cache Size: "+cache.size());
    System.out.println("Cache Capacity: "+maxCacheSize);
    System.out.println("Cache Efficiency:
"+((double)cacheHits.get()/(cacheHits.get()+cacheMisses.get())*100)+"%");
    System.out.println("Cache Hit Ratio:
"+((double)cacheHits.get()/(cacheHits.get()+cacheMisses.get())*100)+"%");
}

```

```

        System.out.println("Cache Miss Ratio:
"+((double) cacheMisses.get() / (cacheHits.get() + cacheMisses.get()) * 100) + "%")
;

    }

    private void addToCache(String documentId, Document document) {
        if (cache.size() > maxCacheSize) {
            evictCacheIfNecessary();
        }
        cache.put(documentId, document);
    }

    private Document loadDocumentFromDisk(String documentId) throws
IOException, ClassNotFoundException {
        Path filePath = Paths.get(diskStoragePath, documentId);
        if (Files.exists(filePath)) {
            try (ObjectInputStream in = new
ObjectInputStream(Files.newInputStream(filePath))) {
                return (Document) in.readObject();
            }
        }
        return null;
    }

    private void evictCacheIfNecessary() {
        while (cache.size() > maxCacheSize) {
            String oldestDocument = cache.keySet().iterator().next();
            cache.remove(oldestDocument);
        }
    }

    //Serializable interface is implemented to allow the Document object
to be converted to a byte stream and stored in the file system
    //This allows the Document object to be saved to the file system and
loaded back into the program
    public static class Document implements Serializable {
        private final String documentId;
        private final String content;
        private final long timestamp;
        private final long lastModifiedTime;

        public Document(String documentId, String content, long
timestamp, long lastModifiedTime) {

```



```

        this.documentId=documentId;
        this.content=content;
        this.timestamp=timestamp;
        this.lastModifiedTime=lastModifiedTime;
    }

    public String getDocumentId(){
        return documentId;
    }

    public String getContent(){
        return content;
    }

    public long getTimestamp(){
        return timestamp;
    }

    public long getLastModifiedTime(){
        return lastModifiedTime;
    }

    @Override
    public String toString(){
        return "Document{"+"documentId='"+documentId+'\''+",
content='"+content+'\''+", timestamp='"+timestamp+",
lastModifiedTime='"+lastModifiedTime+'}';
    }

}

public static void main(String[] args){
    DetailDocumentCache cache=new
DetailDocumentCache(200,"C:\\Users\\ayush\\Desktop\\training_java\\code");
    //It allows tasks to be executed concurrently and provides a way
to control the number of threads in the pool
    //It provides a way to schedule tasks to be executed after a delay
or at regular intervals
    ExecutorService executor=Executors.newFixedThreadPool(2);
    //Simulate multiple threads accessing and modifying the document
    System.err.println("Starting the simulation");

    //Explain these code in detail
    //Simulate multiple threads accessing and modifying the document
    for(int i=0;i<20;i++){

```

```

        final int index=(i%10); //final created to give in lambda function
in executor.

        executor.execute(()->{
            try{

                String id = "Document"+ (index);
                if(index%4==0){
                    if(cache.loadDocumentFromDisk(id)!=null){
                        System.out.println("Document already exists:
"+id);

                        System.out.println("Document:
"+cache.loadDocumentFromDisk(id));
                    }
                    else{
                        Document newDocument=new Document(id,"Content for
"+id,System.currentTimeMillis(),System.currentTimeMillis());
                        cache.saveDocument(newDocument);
                        System.out.println("Saved document: "+id);
                    }
                }else{
                    Document
document=cache.getDocumentIfExistElseSave(id,cache);
                    if(document!=null){
                        System.out.println("Retrieved document: "+id);
                    }else{
                        System.out.println("Document not found: "+id);
                    }
                }
            }catch(IOException | ClassNotFoundException e){
                e.printStackTrace();
            }
            finally{
                System.out.println("Thread "+index+" completed");
                // executor.shutdown();
                try{
                    executor.awaitTermination(1,TimeUnit.MINUTES);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        });
    }
}

```

```

        }

    });

    cache.printCacheStatistics();
}

}

```

Below program demonstrates the concept of executor and serialization

```

import java.util.*;
import java.io.*;
import java.util.concurrent.*;

public class bookLibrarySystem {
    public static void main(String [] args){
        BookLibrary bookLibrary=new BookLibrary();
        bookLibrary.addBook(new Book("123","Harry Potter","J.K.Rowling"));
        bookLibrary.addBook(new Book("456","To Kill a Mockingbird","Harper
Lee"));
        bookLibrary.addBook(new Book("789","1984","George Orwell"));
        bookLibrary.addBook(new Book("101","Pride and Prejudice","Jane
Austen"));
        bookLibrary.addBook(new Book("102","The Great Gatsby","F. Scott
Fitzgerald"));
        bookLibrary.addBook(new Book("103","Moby Dick","Herman
Melville"));
        bookLibrary.addBook(new Book("104","War and Peace","Leo
Tolstoy"));
        bookLibrary.addBook(new Book("105","Hamlet","William
Shakespeare"));
        bookLibrary.addBook(new Book("106","Macbeth","William
Shakespeare"));
        System.out.println(bookLibrary.getAllBooks());
        System.out.println(bookLibrary.getAuthors());
        System.out.println(bookLibrary.getBookCountByAuthor());
        Future<Book>
futureMostPopularBook=bookLibrary.getMostPopularBookAsync();
        try{

```

```

        Book mostPopularBook=futureMostPopularBook.get();
        System.out.println("Most popular book:
"+mostPopularBook.toString());
    }catch(InterruptedException | ExecutionException e){
        e.printStackTrace();
    }finally{
        bookLibrary.executorService.shutdown();
    }
    System.out.println("Program continues while waiting for the most
popular book")
}
static class bookLibrary{
    private List<Book> books = new ArrayList<>();
    private Set<String> authors = new HashSet<>();
    private Map<String,Integer> bookCountByAuthor = new HashMap<>();
    private ExecutorService executorService =
Executors.newSingleThreadExecutor();
    bookLibrary(){
        books=new ArrayList<>();
        authors=new HashSet<>();
        bookCountByAuthor=new HashMap<>();
        executorService=Executors.newSingleThreadExecutor();
    }
    public void addBook(Book book){
        books.add(book);
        authors.add(book.author);

bookCountByAuthor.put(book.author,bookCountByAuthor.getDefault(book.auth
or,0)+1);
    }
    public List<Book> getAllBooks(){
        return new ArrayList<>(books);
    }
    public Set<String> getAuthors(){
        return new HashSet<>(authors);
    }
    public Map<String,Integer> getBookCountByAuthor(){
        return new HashMap<>(bookCountByAuthor);
    }
    public Future<Book> getMostPopularBookAsync(){

```

```

        return executorService.submit(()->{
            Thread.sleep(2000);

            return books.isEmpty()?null:getMostPopularBook();
        });
    }

    private Book getMostPopularBook(){
        return books.get(0);
    }
}

//serializable interface basically allows a class to be serialized ,
it is the process of converting the
//object's state into byte stream which can then be converted back
into object at a later time
static class Book implements Serializable{
    private String isbn;
    private String title;
    private String author;
    public Book(String isbn,String title,String author){
        this.isbn=isbn;
        this.title=title;
        this.author=author;
    }
    transient int curentPage=0;
    @Override
    public String toString(){
        return "Book{"+"isbn='"+isbn+'\''+"", title='"+title+'\''+"",
author='"+author+'\''+"'}";
    }
}
}

```

Below code is that of a library application which demonstrates all of the above mentioned concepts all in one application

```

import java.io.*;
import java.util.*;
import java.util.concurrent.*;

// Custom exception class for handling library-related exceptions
class LibraryException extends Exception {

```

```

    public LibraryException(String message) {
        super(message);
    }
}

// Serializable Book class
class Books implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String title;
    private String author;

    public Books(int id, String title, String author) {
        this.id = id;
        this.title = title;
        this.author = author;
    }

    public int getId() {
        return id;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    @Override
    public String toString() {
        return "Book{" +
            "ID=" + id +
            ", Title='" + title + '\'' +
            ", Author='" + author + '\'' +
            '}';
    }
}

```

```
// Library class simulating book operations with multithreading and
caching
class Library {
    private final Map<Integer, Books> booksCache = new HashMap<>();
    private final ExecutorService executorService =
Executors.newFixedThreadPool(2);

    // Add a book to the library cache
    public void addBookToCache(Books book) {
        booksCache.put(book.getId(), book);
    }

    // Get book details by ID (with caching)
    public Books getBookFromCache(int id) throws LibraryException {
        if (!booksCache.containsKey(id)) {
            throw new LibraryException("Book not found in cache!");
        }
        return booksCache.get(id);
    }

    // Simulate borrowing a book asynchronously
    public Future<String> borrowBook(int bookId) {
        return executorService.submit(() -> {
            Books book = getBookFromCache(bookId);
            return "Borrowed: " + book;
        });
    }

    // Simulate returning a book asynchronously
    public Future<String> returnBook(int bookId) {
        return executorService.submit(() -> {
            Books book = getBookFromCache(bookId);
            return "Returned: " + book;
        });
    }

    // Close the executor service
    public void shutdown() {
        executorService.shutdown();
    }
}
```

```

}

public class LibraryApp {
    public static void main(String[] args) {
        Library library = new Library();

        // Adding books to the library cache
        library.addBookToCache(new Books(1, "The Great Gatsby", "F. Scott Fitzgerald"));
        library.addBookToCache(new Books(2, "1984", "George Orwell"));
        library.addBookToCache(new Books(3, "To Kill a Mockingbird", "Harper Lee"));

        try {
            // Borrow and return books using Future and multithreading
            Future<String> borrowFuture = library.borrowBook(1);
            Future<String> returnFuture = library.returnBook(1);

            // Print results of asynchronous operations
            System.out.println(borrowFuture.get()); // Borrow book
            System.out.println(returnFuture.get()); // Return book

        } catch (InterruptedException | ExecutionException e) {
            System.err.println("Error: " + e.getMessage());
        } finally {
            // Shut down the executor service
            library.shutdown();
        }
    }
}

```

## DSA

### Arrays:

1. Program to find the index of two elements from a given array whose sum will be equal to the given number

```
import java.util.HashMap;
```



```
//pgm to find the index of two elements from an array whose sum should
equal to the given target
public class TwoSumBasic{
    public static void main(String[] args){
        int [] arr ={2,7,11,15};
        int target = 13;
        HashMap <Integer,Integer> map = new HashMap<>();
        int remaining=0;
        for(int i=0; i<arr.length; i++){
            remaining=target-arr[i];
            if(map.containsKey(remaining)){
                System.out.println(map.get(remaining)+" "+i);
                break;
            }
            map.put(arr[i],i);
        }
    }
}
```

## 2. Program to find the subarray from given array that has the largest sum

```
//from the given array find the subarray that has the largest sum
public class largestSum {
    public static void main(String[] args){
        int arr[] = {2,3,-8,7,-1,2,3};
        int n = arr.length;
        int max=arr[0];
        int current=arr[0],start=0,end=0,temp=0;
        for(int i=0; i<n; i++){
            if(arr[i]>arr[i]+current){
                current = arr[i];
                temp=i;
            }else{
                current+=arr[i];
            }
            if(current>max){
                max=current;
                end=i;
                start=temp;
            }
        }
    }
}
```

```

        System.out.println("largest sum is: "+max);
        System.out.println("the subarray with largest sum: ");
        for(int i=start; i<=end; i++){
            System.out.print(arr[i]+" ");
        }
    }
}

```

### 3. Program to find the subarray with largest sum using kadane's algorithm

```

//from the given array find the subarray that has the largest sum using
kadane's algo
public class kadane {
    public static void main(String[] args) {
        int arr[] = {2,3,-8,7,-1,2,3};
        int n = arr.length;
        int maxend=arr[0],res=arr[0];
        for(int i=1;i<n;i++){
            maxend=Math.max(arr[i],maxend+arr[i]);
            System.out.println("maxend "+maxend);
            res=Math.max(res,maxend);
            System.out.println("result "+res);
        }
        System.out.println(res);
    }
}

```

### 4. Program to demonstrate recursion

```

import java.util.Scanner;
public class recursion {
    public static int display(int n){
        if(n==0){
            return 1;
        }else{
            System.out.println("display");
            return display(n-1);
        }
    }
    public static void main(String[] args){

```

```

        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        display(n);
        scan.close();
    }
}

```

## 5. Selection sort of array

```

public class selectionSort {
    public static void main(String[] args){
        int arr[]={4,1,2,5,3,8,6};
        int large=0;
        for(int i=0;i<arr.length-1;i++){
            for(int j=i+1;j<arr.length; j++){
                if(arr[i]>arr[j]){
                    large=arr[i];
                    arr[i]=arr[j];
                    arr[j]=large;
                }
            }
        }
        for(int i=0; i<arr.length; i++){
            System.out.print(arr[i]+" ");
        }
    }
}

```

## 6.Bubble sort

```

public class bubbleSort {
    public static void main(String[] args){
        int[] arr = {64,34,25,12,22,11,90};
        int n=arr.length;
        for(int i=0; i<n; i++){
            for (int j=0; j<n-i-1; j++){
                if (arr[j] > arr[j+1]) {
                    int temp = arr[j];
                    arr[j]=arr[j+1];
                    arr[j+1]=temp;
                }
            }
        }
    }
}

```

```

        }
    }
    for(int i=0; i<n; i++){
        System.out.print(arr[i] + " ");
    }
}
}

```

## 7. Insertion sort

```

public class insertionSort {
    public static void main(String[] args) {
        int[] arr = {5, 2, 8, 1, 9};
        for(int i=1; i<arr.length; i++){
            int key = arr[i];
            int j = i-1;
            while(j>=0 && arr[j]>key){
                arr[j+1]=arr[j];
                j--;
            }
            arr[j+1]=key;
        }
        for(int i=0; i<arr.length; i++){
            System.out.print(arr[i]+" ");
        }
    }
}

```

## 8. Find the second largest element from given array

```

//find the second largest element in array
public class scndLargest {
    public static void main(String[] args){
        int[] arr = {3,12,5,33,33,6};
        int n = arr.length;
        if(n<2){
            System.out.println("Array should have at least two elements");
            //return -1;
        }
        int max = Integer.MIN_VALUE;
    }
}

```

```

        int secondMax = Integer.MIN_VALUE;
        for(int i = 0; i < n; i++){
            if(arr[i] > max){
                secondMax = max;
                max = arr[i];
            }
            else if(arr[i] > secondMax && arr[i] != max){
                secondMax = arr[i];
            }
        }
        System.out.println(secondMax);
    }
}

```

9. Program to find all the unique elements present in the array

```

import java.util.*;
public class uniqueElement {
    public static void main(String[] args) {
        int[] arr = {1,1,2,2,3,3,3,4,4,4,4};
        HashSet <Integer> set = new HashSet<>();
        for (int i = 0; i < arr.length; i++) {
            set.add(arr[i]);
        }
        System.out.println(set);
        System.out.println(set.size());
    }
}

```

10. Program to find all the unique elements present in the array without using set

```

//find the unique elements in array without using set
public class uniqueElementsWithoutSet {
    public static void main(String[] args) {
        int[] arr = {1,1,2,2,3,3,3,4,4,4,4,5,5,5,5,5};
        int i=0,count=0;
        for(int j=1; j<arr.length; j++){
            if(arr[i]!=arr[j]){
                i++;
            }
        }
    }
}

```

```

        count++;
        arr[i]=arr[j];
    }
}
for(int j=0; j<=count; j++){
    System.out.print(arr[j]+" ");
}
}
}

```

11. Program to find the majority elements in the array i.e, array that appears strictly more than  $n/2$  times in the array

```

//find the element from an array that appears strictly more than n/2 times
import java.util.*;
public class majorityElement {
    public static void main(String[] args){
        int[] arr = {2,2,3,2,2,4,2,3,2,2,2,5,5,1};
        HashMap<Integer,Integer>map=new HashMap<>();
        for(int i:arr){
            map.put(i,map.getOrDefault(i,0)+1);
        }
        for(int i:map.keySet()){
            if(map.get(i)>arr.length/2){
                System.out.println("the majority element in array is:
"+i);
            }
        }
    }
}

```

12. Searching an element from the given sorted array using binary search

```

//binary search
public class binarySearch {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int target=3;
        int max=arr.length-1, min=0;
        while (min<=max) {

```

```

        int mid=(min+max)/2;
        if(arr[mid]==target){
            System.out.println("target found at index: "+mid);
            break;
        }
        else if(arr[mid]>target){
            max=mid-1;
        }
        else if(arr[mid]<target){
            min=mid+1;
        }
    }
}
}

```

### 13. Left rotate the array by one place

```

public class leftRotate {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int n = arr.length;
        int temp=arr[0];
        for(int i=0;i<n-1;i++){
            arr[i]=arr[i+1];
        }
        arr[n-1]=temp;
        for(int i=0;i<n;i++){
            System.out.print(arr[i]+" ");
        }
    }
}

```

### 14. Program to move all the zeros present in the array to end of the array

```

import java.util.*;
public class moveZeros {
    public static void main(String[] args){
        int[] arr = {1, 2, 0, 4, 0,3};
        int n = arr.length;
    }
}

```

```

        ArrayList<Integer>result=new ArrayList<>();
        int count=0;
        for(int i=0; i<n; i++){
            if(arr[i]!=0){
                result.add(arr[i]);
            }else{
                count++;
            }
        }
        while(count!=0){
            result.add(0);
            count--;
        }
        System.out.print(result);
    }
}

```

15.program to find the index of a given element

```

import java.util.*;
public class findIndex {
    public static void main(String[] args) {
        int arr[]={2,5,1,7,6,3,8,4,9};
        int num=7;
        HashMap <Integer,Integer>map=new HashMap<>();
        for(int i=0;i<arr.length;i++){
            map.put(arr[i],i);
        }
        System.out.println(map.get(num));
    }
}

```

SQL

We firstly install the my sql then go to my sql workplace create a new connection and enter the password

Then we can create and manage databases

We create a database as

Create database dbName

List databases as



Show databases;  
Switch to a db as  
Use dbName;

We can create a table as  
create table employees (id int auto\_increment primary key,  
firstName varchar(50),  
lastName varchar(50),  
email varchar(100),  
hierDate date,  
salary decimal(10,2));  
We can print all the details in table as  
Select \* from employees;

We can populate the database as  
insert into employees(firstName,lastName,email,hierDate,salary) values  
("ayush","arya","ayush@gmail.com","2024-09-21","10000"),  
("rahul","raj","rahul2@gmail.com","2024-09-21","10000");

We can update a field in our table as  
update employees set salary=20000 where firstName="ayush";  
set sql\_safe\_updates=0;

We can select items from table based on certain conditions as  
select \* from employees where salary>10000;  
We can delete items from table as  
delete from employees where id=2;

Below is the program to connect our **java project with my sql database** using **JDBC** driver and performing **crud operations**.

```
package com.example;
import java.sql.*;
import java.util.Scanner;
public class demo {
    private static final String
url="jdbc:mysql://localhost:3306/employeeManagement";
    private static final String username="root";
    private static final String password="root";
    private static Connection connection;
    private static PreparedStatement preparedStatement;
    private static Statement statement;
    private static ResultSet resultSet;
```

```

public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    try{
        connection=DriverManager.getConnection(url, username,
password);
        int choice=0;
        while(choice!=5) {
            System.out.println("1.createRecord \n2.readRecord
\n3.updateRecord \n4.deleteRecord \n5.exit \n");
            choice=scan.nextInt();
            switch(choice){
                case 1:
                    System.out.println("Enter firstName");
                    String firstName=scan.next();
                    System.out.println("Enter lastName");
                    String lastName=scan.next();
                    System.out.println("Enter email");
                    String email=scan.next();
                    System.out.println("Enter hireDate in format
yyyy-mm-dd");
                    String hireDate=scan.next();
                    System.out.println("Enter salary");
                    Double salary=scan.nextDouble();
                    createRecord(firstName,lastName,email,hireDate,salary);
                    break;
                case 2:
                    readRecord();
                    break;
                case 3:
                    System.out.println("enter the salary");
                    salary=scan.nextDouble();
                    System.out.println("enter the id: ");
                    int id=scan.nextInt();
                    updateRecord(salary,id);
                    break;
                case 4:
                    System.out.println("enter the id");
                    id=scan.nextInt();
                    deleteRecord(id);

```

```

        break;
        default:
            System.out.println("enter valid statement");
    }
}
} catch(SQLException e){
    e.printStackTrace();
} finally{
    try{
        if(connection!=null){
            connection.close();
        }
        if(statement!=null){
            statement.close();
        }
        if(preparedStatement!=null){
            preparedStatement.close();
        }
        if(resultSet!=null){
            resultSet.close();
        }
    } catch(SQLException e){
        e.printStackTrace();
    }
}

private static void createRecord(String firstName, String lastName,
String email, String hireDate, Double salary){
    String query="INSERT INTO EMPLOYEES (firstName, lastName, email,
hierDate, salary) VALUES (?, ?, ?, ?, ?)";
    try{
        preparedStatement=connection.prepareStatement(query);
        preparedStatement.setString(1, firstName);
        preparedStatement.setString(2, lastName);
        preparedStatement.setString(3, email);
        preparedStatement.setString(4, hireDate);
        preparedStatement.setDouble(5, salary);
        preparedStatement.executeUpdate();
        System.out.println("Record created");
    } catch(SQLException e){

```

```

        e.printStackTrace();
    }
}

private static void readRecord() {
    String query="SELECT * FROM EMPLOYEES";
    try{
        statement=connection.createStatement();
        resultSet=statement.executeQuery(query);
        while(resultSet.next()){
            System.out.println("id:
"+resultSet.getInt("id")+"firstName: "+resultSet.getString("firstName")+"
lastName: "+resultSet.getString("lastName")+" email:
"+resultSet.getString("email")+" hireDate:
"+resultSet.getDate("hierDate")+" salary: "+resultSet.getInt("Salary"));
        }
    }catch(SQLException e){
        e.printStackTrace();
    }
}

private static void updateRecord(Double salary, int id){
    String query="UPDATE EMPLOYEES SET salary = ? WHERE id = ?";
    try{
        preparedStatement=connection.prepareStatement(query);
        preparedStatement.setDouble(1, salary);
        preparedStatement.setInt(2, id);
        preparedStatement.executeUpdate();
        System.out.println("salary updated");
        readRecord();
    }catch(SQLException e){
        e.printStackTrace();
    }
}

private static void deleteRecord(int id){
    String query="DELETE FROM EMPLOYEES WHERE id = ?";
    try{
        preparedStatement=connection.prepareStatement(query);
        preparedStatement.setInt(1, id);
        preparedStatement.executeUpdate();
        System.out.println("Record deleted");
        readRecord();
    }
}


```

```

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

**Below is the link to project documentation**

 [project documentation](#)

Below is the code that demonstrates the database insertion and retrieval and also cache insertion and retrieval along with hierarchical caching , and it displays the time taken to perform above mentioned operations, in addition to that I have implemented multithreading to improve the performance and reduce the time taken

```

package com.example;

import java.sql.Connection;
import java.util.Map;
import com.google.common.cache.Cache;
import java.sql.Statement;
import java.sql.SQLException;
import java.sql.DriverManager;
import java.util.concurrent.*;
import java.util.LinkedHashMap;
import com.google.common.cache.CacheBuilder;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.logging.Logger;
import java.util.logging.Level;
import java.util.logging.FileHandler;
import java.util.logging.SimpleFormatter;
import java.util.ArrayList;
import java.util.List;

public class AdvancedDatabaseCachingBenchmark {
    private static final String DB_URL =
"jdbc:mysql://localhost:3306/testdb";
    private static final String DB_USER = "root";
    private static final String DB_PASSWORD = "root";

```

```

private static final int NUM_ELEMENTS = 100000;
private static final int L1_CACHE_SIZE = 10000;
private static final int L2_CACHE_SIZE = 100000;
private static final int L2_CACHE_DURATION_MINUTES = 10;
private static final int PROGRESS_INTERVAL = 10000;
private static final int NUM_THREADS = 10; // Number of threads for
the pool

private static Connection connection;
private static Map<Integer, String> l1Cache;
private static Cache<Integer, String> l2Cache;
private static final Logger LOGGER =
Logger.getLogger(AdvancedDatabaseCachingBenchmark.class.getName());

private static ExecutorService executorService;

public static void main(String[] args) {
    setupLogging();
    executorService = Executors.newFixedThreadPool(NUM_THREADS); //
Create thread pool

    try {
        setupDatabase();
        setupCaches();

        List<BenchmarkResult> results = new ArrayList<>();
        results.add(new BenchmarkResult("Database Insert",
benchmarkDatabaseInsert()));
        results.add(new BenchmarkResult("Database Retrieve",
benchmarkDatabaseRetrieve()));
        results.add(new BenchmarkResult("L1 Cache Insert",
benchmarkL1CacheInsert()));
        results.add(new BenchmarkResult("L1 Cache Retrieve",
benchmarkL1CacheRetrieve()));
        results.add(new BenchmarkResult("L2 Cache Insert",
benchmarkL2CacheInsert()));
        results.add(new BenchmarkResult("L2 Cache Retrieve",
benchmarkL2CacheRetrieve()));
        results.add(new BenchmarkResult("Multilevel Cache Retrieve",
benchmarkMultilevelCacheRetrieve()));
    }
}

```

```

        analyzeAndLogResults(results);
        prepareGraphData(results);

    } catch (Exception e) {
        LOGGER.log(Level.SEVERE, "An error occurred during benchmark
execution", e);
    } finally {
        try {
            if (connection != null) {
                connection.close();
            }
            executorService.shutdown();
        } catch (SQLException e) {
            LOGGER.log(Level.SEVERE, "Error closing database
connection", e);
        }
    }
}

private static void setupLogging() {
    try {
        FileHandler fileHandler = new
FileHandler("benchmark_log.txt");
        SimpleFormatter formatter = new SimpleFormatter();
        fileHandler.setFormatter(formatter);
        LOGGER.addHandler(fileHandler);
    } catch (Exception e) {
        LOGGER.log(Level.SEVERE, "Error setting up logging", e);
    }
}

private static void setupDatabase() {
    try {
        connection = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD);
        Statement statement = connection.createStatement();
        statement.executeUpdate("CREATE TABLE IF NOT EXISTS test_table
(id INT PRIMARY KEY, value VARCHAR(255))");
        statement.close();
    }
}

```

```

        LOGGER.info("Database setup completed successfully.");
    } catch (SQLException e) {
        LOGGER.log(Level.SEVERE, "Error setting up database", e);
    }
}

private static void setupCaches() {
    l1Cache = new LinkedHashMap<Integer, String>(L1_CACHE_SIZE, 0.75f,
true) {
        @Override
        protected boolean removeEldestEntry(Map.Entry<Integer, String>
eldest) {
            return size() > L1_CACHE_SIZE;
        }
    };
    l2Cache = CacheBuilder.newBuilder()
        .maximumSize(L2_CACHE_SIZE)
        .expireAfterAccess(L2_CACHE_DURATION_MINUTES,
TimeUnit.MINUTES)
        .build();
    LOGGER.info("Caches setup completed successfully.");
}

private static long benchmarkDatabaseInsert() throws SQLException {
    LOGGER.info("Starting Database Insert benchmark...");
    long startTime = System.nanoTime();

    List<Callable<Void>> tasks = new ArrayList<>();
    String sql = "INSERT INTO test_table (id, value) VALUES (?, ?)";
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        final int index = i;
        tasks.add(() -> {
            try (PreparedStatement statement =
connection.prepareStatement(sql)) {
                statement.setInt(1, index);
                statement.setString(2, "Value" + index);
                statement.executeUpdate();
            }
            return null;
        });
    }
}

```



```

    }

    executeTasksInParallel(tasks);

    long endTime = System.nanoTime();
    LOGGER.info("Database Insert benchmark completed.");
    return endTime - startTime;
}

private static long benchmarkDatabaseRetrieve() throws SQLException {
    LOGGER.info("Starting Database Retrieve benchmark...");
    long startTime = System.nanoTime();

    List<Callable<Void>> tasks = new ArrayList<>();
    String sql = "SELECT * FROM test_table WHERE id = ?";
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        final int index = i;
        tasks.add(() -> {
            try (PreparedStatement statement =
connection.prepareStatement(sql)) {
                statement.setInt(1, index);
                ResultSet resultSet = statement.executeQuery();
                if (resultSet.next()) {
                    resultSet.getString("value");
                }
                resultSet.close();
            }
            return null;
        });
    }

    executeTasksInParallel(tasks);

    long endTime = System.nanoTime();
    LOGGER.info("Database Retrieve benchmark completed.");
    return endTime - startTime;
}

private static long benchmarkL1CacheInsert() {
    LOGGER.info("Starting L1 Cache Insert benchmark...");
    long startTime = System.nanoTime();

```

```

        List<Callable<Void>> tasks = new ArrayList<>();
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            final int index = i;
            tasks.add(() -> {
                l1Cache.put(index, "Value" + index);
                return null;
            });
        }
        executeTasksInParallel(tasks);

        long endTime = System.nanoTime();
        LOGGER.info("L1 Cache Insert benchmark completed.");
        return endTime - startTime;
    }

    private static long benchmarkL1CacheRetrieve() {
        LOGGER.info("Starting L1 Cache Retrieve benchmark...");
        long startTime = System.nanoTime();

        List<Callable<Void>> tasks = new ArrayList<>();
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            final int index = i;
            tasks.add(() -> {
                l1Cache.get(index);
                return null;
            });
        }
        executeTasksInParallel(tasks);

        long endTime = System.nanoTime();
        LOGGER.info("L1 Cache Retrieve benchmark completed.");
        return endTime - startTime;
    }

    private static long benchmarkL2CacheInsert() {
        LOGGER.info("Starting L2 Cache Insert benchmark...");
        long startTime = System.nanoTime();

        List<Callable<Void>> tasks = new ArrayList<>();
        for (int i = 0; i < NUM_ELEMENTS; i++) {

```

```

        final int index = i;
        tasks.add(() -> {
            l2Cache.put(index, "Value" + index);
            return null;
        });
    }
    executeTasksInParallel(tasks);

    long endTime = System.nanoTime();
    LOGGER.info("L2 Cache Insert benchmark completed.");
    return endTime - startTime;
}

private static long benchmarkL2CacheRetrieve() {
    LOGGER.info("Starting L2 Cache Retrieve benchmark...");
    long startTime = System.nanoTime();

    List<Callable<Void>> tasks = new ArrayList<>();
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        final int index = i;
        tasks.add(() -> {
            l2Cache.getIfPresent(index);
            return null;
        });
    }
    executeTasksInParallel(tasks);

    long endTime = System.nanoTime();
    LOGGER.info("L2 Cache Retrieve benchmark completed.");
    return endTime - startTime;
}

private static long benchmarkMultilevelCacheRetrieve() throws
SQLException {
    LOGGER.info("Starting Multilevel Cache Retrieve benchmark...");
    long startTime = System.nanoTime();

    List<Callable<Void>> tasks = new ArrayList<>();
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        final int index = i;

```

```

        tasks.add(() -> {
            String value = l1Cache.get(index);
            if (value == null) {
                value = l2Cache.getIfPresent(index);
                if (value == null) {
                    try (PreparedStatement statement =
connection.prepareStatement("SELECT * FROM test_table WHERE id = ?")) {
                        statement.setInt(1, index);
                        ResultSet resultSet =
statement.executeQuery();

                        if (resultSet.next()) {
                            value = resultSet.getString("value");
                            l2Cache.put(index, value);
                            l1Cache.put(index, value);
                        }
                        resultSet.close();
                    }
                }
                l1Cache.put(index, value);
            }
            return null;
        });
    }

    executeTasksInParallel(tasks);

    long endTime = System.nanoTime();
    LOGGER.info("Multilevel Cache Retrieve benchmark completed.");
    return endTime - startTime;
}

private static void executeTasksInParallel(List<Callable<Void>> tasks)
{
    try {
        List<Future<Void>> futures = executorService.invokeAll(tasks);
        for (Future<Void> future : futures) {
            future.get(); // Ensure all tasks are completed
        }
    } catch (InterruptedException | ExecutionException e) {
        LOGGER.log(Level.SEVERE, "Error during parallel execution",
e);
    }
}

```

```

    }
}

private static void analyzeAndLogResults(List<BenchmarkResult>
results) {
    LOGGER.info("Performance Analysis:");
    for (BenchmarkResult result : results) {
        logPerformance(result.operation, result.time);
    }

    LOGGER.info("\nPerformance Improvements:");
    BenchmarkResult dbRetrieve = results.stream()
        .filter(r -> r.operation.equals("Database Retrieve"))
        .findFirst()
        .orElseThrow();

    results.stream()
        .filter(r -> !r.operation.equals("Database Retrieve") &&
!r.operation.equals("Database Insert"))
        .forEach(r -> logImprovement(r.operation + " vs Database
Retrieve", dbRetrieve.time, r.time));
}

private static void logPerformance(String operation, long timeNanos) {
    double timeMinutes = timeNanos / (60.0 * 1e9);
    LOGGER.info(String.format("%s: %.4f minutes", operation,
timeMinutes));
}

private static void logImprovement(String comparison, long baseTime,
long improvedTime) {
    double improvement = (baseTime - improvedTime) / (double) baseTime
* 100;
    LOGGER.info(String.format("%s: %.2f%% faster", comparison,
improvement));
}

private static void prepareGraphData(List<BenchmarkResult> results) {
    StringBuilder graphData = new StringBuilder("Operation,Time
(minutes)\n");

```

```

        for (BenchmarkResult result : results) {
            double timeMinutes = result.time / (60.0 * 1e9);
            graphData.append(String.format("%s,%.4f\n", result.operation,
timeMinutes));
        }
        LOGGER.info("Graph Data:\n" + graphData.toString());
    }

    private static class BenchmarkResult {
        String operation;
        long time;

        BenchmarkResult(String operation, long time) {
            this.operation = operation;
            this.time = time;
        }
    }
}

```

Below is the program to create trie map and perform several operations on it

```

import java.util.*;

// TrieNode class represents a single node in the Trie
class TrieNode<T> {
    // Map to store child nodes, where the key is the character and value
    // is the child TrieNode
    Map<Character, TrieNode<T>> childNodes;

    // Value associated with this node (null if this node doesn't
    // represent the end of a word)
    T storedValue;

    // Flag to indicate if this node represents the end of a word
    boolean isWordEnd;

    // Constructor to initialize a new TrieNode
    public TrieNode() {
        this.childNodes = new HashMap<>();
    }
}

```

```

        this.storedValue = null;
        this.isWordEnd = false;
    }
}

// Trie class implements the main functionality of the Trie data structure
class Trie<T> {
    // Root node of the Trie
    private TrieNode<T> mainRoot;

    // Constructor to initialize an empty Trie
    public Trie() {
        this.mainRoot = new TrieNode<>();
    }

    // Method to insert a word and its associated value into the Trie
    public void insert(String text, T val) {
        TrieNode<T> currentNode = mainRoot;

        // Iterate through each character in the text
        for (char character : text.toCharArray()) {
            // If the current character doesn't exist as a child, create a
new node
            currentNode =
currentNode.childNodes.computeIfAbsent(character, k -> new TrieNode<>());
        }

        // Mark the last node as the end of a word and set its value
        currentNode.isWordEnd = true;
        currentNode.storedValue = val;
    }

    // Method to search for a word in the Trie
    public T search(String text) {
        TrieNode<T> nodeFound = findNode(text);

        // Return the value if the word exists and is marked as end of
word, otherwise null
        return (nodeFound != null && nodeFound.isWordEnd) ?
nodeFound.storedValue : null;
    }
}

```

```

    // Method to check if any word in the Trie starts with the given
prefix
    public boolean startsWith(String prefix) {
        // If we can find a node for this prefix, it exists in the Trie
        return findNode(prefix) != null;
    }

    // Helper method to find a node corresponding to a given word or
prefix
    private TrieNode<T> findNode(String input) {
        TrieNode<T> currentNode = mainRoot;

        // Traverse the Trie following the characters in the string
        for (char ch : input.toCharArray()) {
            currentNode = currentNode.childNodes.get(ch);
            // If at any point we can't find a child node, the word/prefix
doesn't exist
            if (currentNode == null) return null;
        }

        // Return the final node we reached
        return currentNode;
    }

    // Method to get all words in the Trie with a given prefix
    public List<String> getWordsWithPrefix(String prefix) {
        List<String> wordList = new ArrayList<>();
        TrieNode<T> prefixNode = findNode(prefix);

        // If the prefix exists in the Trie, collect all words starting
from its last node
        if (prefixNode != null) {
            collectWords(prefixNode, prefix, wordList);
        }

        return wordList;
    }

    // Recursive helper method to collect all words from a given node

```



```

        private void collectWords(TrieNode<T> node, String prefixStr,
List<String> wordList) {
            // If this node is marked as end of word, add the current prefix
to results
            if (node.isWordEnd) {
                wordList.add(prefixStr);
            }

            // Recursively explore all child nodes
            for (Map.Entry<Character, TrieNode<T>> entry :
node.childNodes.entrySet()) {
                collectWords(entry.getValue(), prefixStr + entry.getKey(),
wordList);
            }
        }
    }
}

// Main class to demonstrate the usage of the Trie
public class TrieDemo {
    public static void main(String[] args) {
        Trie<Integer> trieMap = new Trie<>();

        // Insert some words with associated values
        trieMap.insert("apple", 1);
        trieMap.insert("app", 2);
        trieMap.insert("application", 3);
        trieMap.insert("banana", 4);

        // Demonstrate search functionality
        System.out.println("Value for 'apple': " +
trieMap.search("apple")); // Output: 1
        System.out.println("Value for 'app': " + trieMap.search("app"));
// Output: 2
        System.out.println("Value for 'appl': " + trieMap.search("appl"));
// Output: null

        // Demonstrate prefix checking
        System.out.println("Starts with 'app': " +
trieMap.startsWith("app")); // Output: true
    }
}

```

```

        System.out.println("Starts with 'ban': " +
trieMap.startsWith("ban")); // Output: true
        System.out.println("Starts with 'car': " +
trieMap.startsWith("car")); // Output: false

        // Demonstrate getting words with a prefix
        System.out.println("Words with prefix 'app': " +
trieMap.getWordsWithPrefix("app"));
        // Output: [app, apple, application]
    }
}

```

-> Merge sort

```

// MergeSort in Java
public class MergeSort {

    // Method to merge two subarrays L and M into arr
    void merge(int arr[], int left, int middle, int right) {
        // Find sizes of two subarrays to be merged
        int n1 = middle - left + 1;
        int n2 = right - middle;

        // Create temporary arrays
        int L[] = new int[n1];
        int M[] = new int[n2];

        // Copy data to temporary arrays
        for (int i = 0; i < n1; i++)
            L[i] = arr[left + i];
        for (int j = 0; j < n2; j++)
            M[j] = arr[middle + 1 + j];

        // Initial indexes of first and second subarrays
        int i = 0, j = 0;

        // Initial index of merged subarray
        int k = left;

```

```

while (i < n1 && j < n2) {
    if (L[i] <= M[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = M[j];
        j++;
    }
    k++;
}

// Copy remaining elements of L[] if any
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}

// Copy remaining elements of M[] if any
while (j < n2) {
    arr[k] = M[j];
    j++;
    k++;
}

}

// Method to divide the array into two subarrays and sort them
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        // Find the middle point
        int middle = left + (right - left) / 2;

        // Sort the first and second halves
        mergeSort(arr, left, middle);
        mergeSort(arr, middle + 1, right);

        // Merge the sorted halves
        merge(arr, left, middle, right);
    }
}

```

```

// Helper method to print the array
static void printArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Main method to test the program
public static void main(String args[]) {
    int arr[] = {12, 11, 13, 5, 6, 7};

    MergeSort ms = new MergeSort();
    System.out.println("Given Array:");
    printArray(arr);

    ms.mergeSort(arr, 0, arr.length - 1);

    System.out.println("\nSorted array:");
    printArray(arr);
}
}

```

-> Quick sort

```

// QuickSort in Java
public class QuickSort {

    // Method to partition the array on the basis of pivot element
    int partition(int arr[], int low, int high) {
        // Pivot element
        int pivot = arr[high];
        int i = (low - 1); // Index of smaller element

        for (int j = low; j < high; j++) {
            // If current element is smaller than or equal to pivot
            if (arr[j] <= pivot) {
                i++;
            }
        }
    }
}

```

```

        // Swap arr[i] and arr[j]
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

}

// Swap arr[i + 1] and arr[high] (or pivot)
int temp = arr[i + 1];
arr[i + 1] = arr[high];
arr[high] = temp;

return i + 1;
}

// Method to implement QuickSort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Find pivot element such that elements smaller than pivot
are on the left
        // and elements greater than pivot are on the right
        int pi = partition(arr, low, high);

        // Recursively sort elements before and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// Helper method to print the array
static void printArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Main method to test the QuickSort algorithm
public static void main(String args[]) {
    int arr[] = {10, 80, 30, 90, 40, 50, 70};

```

```

        int n = arr.length;

        QuickSort qs = new QuickSort();
        System.out.println("Given Array:");
        printArray(arr);

        qs.quickSort(arr, 0, n - 1);

        System.out.println("\nSorted array:");
        printArray(arr);
    }
}

```

-> Binary tree with in-order , pre-order, post-order traversal and searching , deletion

```

class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {
    Node root;

    // Constructor
    BinaryTree() {
        root = null;
    }

    // Inorder traversal
    void inorder(Node node) {
        if (node == null)
            return;

        inorder(node.left);
        System.out.print(node.data + " ");
    }
}

```

```

        inorder(node.right);
    }

    // Preorder traversal
    void preorder(Node node) {
        if (node == null)
            return;

        System.out.print(node.data + " ");
        preorder(node.left);
        preorder(node.right);
    }

    // Postorder traversal
    void postorder(Node node) {
        if (node == null)
            return;

        postorder(node.left);
        postorder(node.right);
        System.out.print(node.data + " ");
    }

    // Search for a node with a given value
    boolean search(Node node, int value) {
        if (node == null) {
            return false; // Node not found
        }

        if (node.data == value) {
            return true; // Node found
        }

        // Recursively search in left and right subtrees
        boolean foundInLeft = search(node.left, value);
        if (foundInLeft) {
            return true;
        }
        return search(node.right, value);
    }
}

```

```

// Utility functions for calling the traversal methods
void inorder() {
    inorder(root);
}

void preorder() {
    preorder(root);
}

void postorder() {
    postorder(root);
}

// Search utility
boolean search(int value) {
    return search(root, value);
}

// Find the minimum value node in the tree (used to find the in-order
successor)
Node minValueNode(Node node) {
    Node current = node;
    while (current.left != null) {
        current = current.left;
    }
    return current;
}

// Delete a node from the binary tree
Node deleteNode(Node root, int key) {
    // Base case: If the tree is empty
    if (root == null) {
        return root;
    }

    // Recur down the tree
    if (key < root.data) {
        root.left = deleteNode(root.left, key);
    } else if (key > root.data) {

```



```

        root.right = deleteNode(root.right, key);
    } else {
        // Node to be deleted found

        // Case 1: Node with only one child or no child
        if (root.left == null) {
            return root.right;
        } else if (root.right == null) {
            return root.left;
        }

        // Case 2: Node with two children
        // Get the inorder successor (smallest in the right subtree)
        Node temp = minValueNode(root.right);

        // Copy the inorder successor's content to this node
        root.data = temp.data;

        // Delete the inorder successor
        root.right = deleteNode(root.right, temp.data);
    }

    return root;
}

// Utility function to call the deleteNode method
void delete(int key) {
    root = deleteNode(root, key);
}

// Main method
public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();

    // Create a binary tree
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

```

```

        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        // Print traversals
        System.out.println("Inorder traversal before deletion:");
        tree.inorder();

        // Deleting a node
        System.out.println("\n\nDeleting node 3");
        tree.delete(3);

        System.out.println("Inorder traversal after deletion:");
        tree.inorder();

        // Searching for a value in the tree
        int searchValue = 5;
        if (tree.search(searchValue)) {
            System.out.println("\n\nNode " + searchValue + " found in the
tree.");
        } else {
            System.out.println("\n\nNode " + searchValue + " not found in
the tree.");
        }

        searchValue = 8;
        if (tree.search(searchValue)) {
            System.out.println("Node " + searchValue + " found in the
tree.");
        } else {
            System.out.println("Node " + searchValue + " not found in the
tree.");
        }
    }
}

```

-> program to reverse a string

```

package Strings;

public class reverse {

```

```

public static void main(String[] args) {
    String str = "Hello World";
    String result="";
    for(int i=str.length()-1; i>=0; i--){
        result = result + str.charAt(i);
    }
    System.out.println(result);
}
}

```

->program the reverse the sequence of words in a string

```

package Strings;
import java.util.*;
public class reverse_words {
    public static void main(String[] args){
        Stack<String> st = new Stack<>();
        String str="this is a good string";
        StringTokenizer st1=new StringTokenizer(str," ");
        while(st1.hasMoreTokens()){
            st.push(st1.nextToken());
        }
        while(!st.isEmpty()){
            System.out.print(st.pop()+" ");
        }
    }
}

```

-> program to find anagrams

```

package Strings;

import java.util.Arrays;
import java.util.Scanner;

public class anagram {
    public anagram() {
    }

    public static void main(String[] var0) {
        Scanner var1 = new Scanner(System.in);
        System.out.println("enter two strings");
    }
}

```

```

String var2 = var1.next();
String var3 = var1.next();
char[] var4 = var2.toCharArray();
char[] var5 = var3.toCharArray();
Arrays.sort(var4);
Arrays.sort(var5);
String var6 = new String(var4);
String var7 = new String(var5);
if (var6.equals(var7)) {
    System.out.println("Yes anagram");
} else {
    System.out.println("NO not an anagram");
}

var1.close();
}
}

```

->lambda example

```

import java.util.Arrays;
import java.util.List;
import java.util.function.Function;
import java.util.function.Predicate;
public class lambdaExample {
    public static void main(String[] args){
        Runnable runnable = () ->System.out.println("hello lambda");
        runnable.run();

        Predicate<String> isLong=s->s.length()>5;
        System.out.println(isLong.test("lambda"));

        List<Integer>numbers=Arrays.asList(1,2,3,4,5);
        numbers.forEach(n->System.out.println("Number: "+n));
        Function<Integer,Integer> square =x -> x*x;
        numbers.stream().map(square).forEach(System.out::println);
        Function<Integer,Boolean>isEven=getIsEvenLambda();
        System.out.println(isEven.apply(4));
    }
}

```

-> solution to coin change problem

```
package dp;
import java.util.*;
public class coinChange {
    public static void main(String[] args){
        int coins[]={1,2,5,10,20,50,100,500,2000};
        ArrayList<Integer>dp=new ArrayList<>();
        int amount=767;
        int i=coins.length-1;
        while(coins[i]>=0){
            if(amount==0){
                break;
            }
            else if(coins[i]<=amount){
                dp.add(coins[i]);
                amount-=coins[i];
            }else{
                i--;
            }
        }
        System.out.println(dp);
    }
}
```

-> solution to coin change problem using dynamic programming

```
package dp;

import java.util.Arrays;

public class coinChangeDP {
    // Function to find the minimum number of coins needed to make the
    given amount
    public static int coinChange(int[] coins, int amount) {
        // Initialize dp array where dp[i] represents the minimum coins
        needed for amount i
        int[] dp = new int[amount + 1];

        // Set the value for all amounts to a large number (amount + 1),
        as it's an impossible case.
        Arrays.fill(dp, amount + 1);
    }
}
```

```

        // Base case: to make amount 0, we need 0 coins
        dp[0] = 0;

        // Iterate through all amounts from 1 to the target amount
        for (int i = 1; i <= amount; i++) {
            // For each coin, see if it can contribute to the current
amount i
            for (int coin : coins) {
                if (i - coin >= 0) {
                    dp[i] = Math.min(dp[i], dp[i - coin] + 1);
                }
            }
        }

        // If dp[amount] is still amount + 1, then it's not possible to
make that amount
        return dp[amount] > amount ? -1 : dp[amount];
    }

    public static void main(String[] args) {
        int[] coins = {1, 2, 5}; // Example set of coins
        int amount = 11; // Example target amount

        int result = coinChange(coins, amount);
        if (result != -1) {
            System.out.println("Minimum coins needed: " + result);
        } else {
            System.out.println("It's not possible to make the given amount
with the available coins.");
        }
    }
}

```

-> program to find the no's in array which is greater than all the numbers to the right of it

```

import java.util.*;

public class arr_leaders {
    public static void main(String [] args){
        Scanner scan = new Scanner (System.in);
        int n = scan.nextInt();
    }
}

```

```

        int arr[]=new int[n];
        for(int i=0; i<n; i++){
            arr[i]=scan.nextInt();
        }
        ArrayList<Integer> result=new ArrayList<>();
        result = findLeaders(arr, n);
        System.out.println(result);
        scan.close();
    }

    static ArrayList<Integer> findLeaders(int []arr, int n){
        ArrayList<Integer> leaders = new ArrayList<>();
        leaders.add(arr[n-1]);
        int max=0;
        for(int i=n-2; i>=0; i--){
            if(arr[i]>max){
                max=arr[i];
                leaders.add(max);
            }
        }
        Collections.reverse(leaders);
        return leaders;
    }
}

```

-> program to find the longest palindrome substring in given string

```

package Strings;

public class longestPalindrome {
    public static boolean isPalindrome(String s, int left, int right){
        while (left < right) {
            if (s.charAt(left) != s.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    public static String longestSubstring(String s){

```

```

    int n = s.length();
    int start=0,maxlen=1;
    for(int i=0;i<n;i++){
        for(int j=i; j<n; j++){
            if(isPalindrome(s,i,j) && (j-i+1)>maxlen){
                start=i;
                maxlen=j-i+1;
            }
        }
    }
    return s.substring(start,start+maxlen);
}

public static void main(String[] args) {
    String s = "helloracecarworld";
    System.out.println(longestSubstring(s));
}
}

```

-> program to print the elements from given array which is only present once in the array

```

import java.util.*;
public class duplicate {
    public static void main(String[] args){
        int arr[]={1,2,2,3,1,4,5,6,1,4,5};
        int n=arr.length;
        HashMap<Integer,Integer>map=new HashMap<>();
        for(int i=0;i<n;i++){
            if(map.containsKey(arr[i])){
                map.put(arr[i],map.get(arr[i])+1);
            }else{
                map.put(arr[i],1);
            }
        }
        for(int num: map.keySet()){
            if(map.get(num)==1){
                System.out.println(num);
            }
        }
    }
}

```



## SQL

```
create database testdb;
use testdb;
select * from test_table;
select count(*) from test_table;      //count all the elements present in table
drop table test_table;                //delete the table

show databases;
use testdb;

select * from test_table;
rename table test_table to key_value;  //rename an existing table to new name

alter table key_value rename to key_value_pair;  //rename to new name
select * from key_value_pair;

truncate table key_value_pair;         //delete all the elements present in the table
create table test(id int, value varchar(20));

insert into test select * from key_value_pair; //insert all the elements present in a table into
another table

select * from test;
select * from test limit 4;
select min(id) from test;              //find the minimum id value
select max(id) from test;              //find the maximum id value

select count(id) from test;            //count the no id's in table
select avg(id) from test;              //average of id
select sum(id) from test;              //sum of id

select * from test where value like "val%";  //all elements that starts with "val"
select * from test where value like "__l%";  //all elements that has l at third character place

select * from test where value in ("value0","value1");  //where value is value0 or value1
select * from test where id between 10 and 100;  //items with id in between 10 and 100

select * from test as test_table;
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50),
    location_id INT
```

```
);
```

```
CREATE TABLE locations (  
    location_id INT PRIMARY KEY,  
    city VARCHAR(50)
```

```
);
```

```
//creating table that reference to another table using foreign key
```

```
CREATE TABLE employees (  
    employee_id INT PRIMARY KEY,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    email VARCHAR(100),  
    phone_number VARCHAR(20),  
    hire_date DATE,  
    job_id VARCHAR(10),  
    salary DECIMAL(10, 2),  
    department_id INT,  
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
```

```
);
```

```
CREATE TABLE customers (  
    customer_id INT PRIMARY KEY,  
    customer_name VARCHAR(100),  
    email VARCHAR(100)
```

```
);
```

```
CREATE TABLE orders (  
    order_id INT PRIMARY KEY,  
    customer_id INT,  
    order_date DATE,  
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
```

```
);
```

```
CREATE TABLE products (  
    product_id INT PRIMARY KEY,  
    product_name VARCHAR(100),  
    category VARCHAR(50),  
    price DECIMAL(10, 2)
```

```
);
```

```
//inserting data into above created tables
```

```
INSERT INTO departments (department_id, department_name, location_id) VALUES  
(10, 'Administration', 1),
```

```
(20, 'Marketing', 2),  
(30, 'Purchasing', 1),  
(40, 'Human Resources', 2),  
(50, 'Shipping', 3);
```

```
INSERT INTO locations (location_id, city) VALUES  
(1, 'New York'),  
(2, 'Los Angeles'),  
(3, 'Chicago');
```

```
INSERT INTO employees (employee_id, first_name, last_name, email, phone_number,  
hire_date, job_id, salary, department_id) VALUES  
(1, 'John', 'Doe', 'john.doe@example.com', '515.123.4567', '2019-06-17', 'AD_PRES', 24000.00,  
10),  
(2, 'Jane', 'Smith', 'jane.smith@example.com', '515.123.4568', '2020-02-20', 'AD_VP', 17000.00,  
10),  
(3, 'Alice', 'Johnson', 'alice.johnson@example.com', '515.123.4569', '2020-08-11', 'MK_MAN',  
9000.00, 20),  
(4, 'Bob', 'Brown', 'bob.brown@example.com', '515.123.4560', '2021-03-05', 'HR_REP', 6000.00,  
40),  
(5, 'Charlie', 'Davis', 'charlie.davis@example.com', '515.123.4561', '2021-11-30', 'SH_CLERK',  
3000.00, 50);
```

```
INSERT INTO customers (customer_id, customer_name, email) VALUES  
(1, 'Acme Corp', 'contact@acmecorp.com'),  
(2, 'GlobalTech', 'info@globaltech.com'),  
(3, 'Local Shop', 'owner@localshop.com');
```

```
INSERT INTO orders (order_id, customer_id, order_date) VALUES  
(1, 1, '2023-01-15'),  
(2, 1, '2023-02-20'),  
(3, 2, '2023-02-22');
```

```
INSERT INTO products (product_id, product_name, category, price) VALUES  
(1, 'Laptop', 'Electronics', 999.99),  
(2, 'Smartphone', 'Electronics', 699.99),  
(3, 'Desk Chair', 'Furniture', 199.99);
```

```
INSERT INTO discontinued_products (product_id, product_name, category) VALUES  
(101, 'Old Laptop Model', 'Electronics'),  
(102, 'Discontinued Phone', 'Electronics');
```

```
//Inner join
```

```
select e.first_name, e.last_name, d.department_name
from employees e
inner join departments d
on e.department_id= d.department_id;
```

```
//Left join
select c.customer_name , o.order_id
from customers c
left join orders o on c.customer_id =o.customer_id;
```

```
//Right join
select c.customer_name , o.order_id
from customers c
right join orders o on c.customer_id =o.customer_id;
```

```
select first_name,last_name, salary
from employees
where salary >(select avg(salary) from employees);
```

```
//using group by
select department_id , avg(salary) as avgsalary
from employees
group by department_id
having avgsalary >10000;
select * from employees;
select * from departments;
```

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
WHERE e.salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department_id = e.department_id);
```

```
//multiple left joins
select c.customer_name,
count(o.order_id) as order_count,
coalesce(sum(p.price),0) as total_order_value
from
customers c
left join
orders o on c.customer_id =o.customer_id
left join
```

```
products p on p.product_id in (select product_id from orders where order_id = o.order_id )
group by
c.customer_id,c.customer_name
having
count(o.order_id)> 0
order by
total_order_value desc;
```

```
//inner join on same table
select e1.first_name as employee1_first_name,
e1.last_name as employee1_last_name,
e2.first_name as employee2_first_name,
e2.last_name as employee2_last_name,
e1.job_id
from
employees e1
inner join
employees e2
on e1.job_id =e2.job_id and e1.employee_id<e2.employee_id;
```

```
//left outer join
select d.department_name,
e.first_name,
e.last_name,
e.salary,
avg(e.salary) over (partition by d.department_id) as dept_avg_slary,
rank() over (partition by d.department_id order by e.salary desc) as salary_rank
from departments d
left outer join
employees e on
d.department_id =e.department_id
order by d.department_name,e.salary desc;
```

```
//using case (conditional statements) in sql query
select c.customer_name,
o.order_date,
case
when DAYOFWEEK(o.order_date) in (1,7) Then 'Weekend'
else 'Weekday'
end as order_day_type,
p.product_name,
p.price,
case
when p.price<100 then 'Budget'
```

```

    when p.price between 100 and 500 then 'mid range'
    else 'premium'
end as price_category
from customers c
inner join
orders o on c.customer_id =o.customer_id
inner join products p on p.product_id in (select product_id from orders where order_id
=o.order_id)
where
o.order_date <= DATE_SUB(CURDATE(),INTERVAL 1 YEAR)
order by
o.order_date desc;

```

```

CREATE TABLE employee (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    manager_id INT,
    FOREIGN KEY (manager_id) REFERENCES employee(employee_id)
);

```

```

INSERT INTO employee (employee_id, first_name, last_name, manager_id) VALUES
(1, 'John', 'Doe', NULL),
(2, 'Jane', 'Smith', 1),
(3, 'Bob', 'Johnson', 1),
(4, 'Alice', 'Williams', 2),
(5, 'Charlie', 'Brown', 2),
(6, 'David', 'Lee', 3);

```

```

select e.employee_id , concat(e.first_name," ",e.last_name) as employee_name,
m.first_name as manager_name from employee e
left join
employees m on e.manager_id=m.employee_id;

```

```

1  -- Example 2: Product Recommendations
2  • create database product;
3  • use product;
4  • CREATE TABLE products (
5      product_id INT PRIMARY KEY,
6      product_name VARCHAR(100),
7      category VARCHAR(50),
8      price DECIMAL(10, 2)
9  );
10
11 • CREATE TABLE product_recommendations (
12     product_id INT,
13     recommended_product_id INT,
14     strength DECIMAL(3, 2),
15     PRIMARY KEY (product_id, recommended_product_id),
16     FOREIGN KEY (product_id) REFERENCES products(product_id),
17     FOREIGN KEY (recommended_product_id) REFERENCES products(product_id)
18 );
19
20 • INSERT INTO products (product_id, product_name, category, price) VALUES
21     (1, 'Laptop', 'Electronics', 999.99),
22     (2, 'Smartphone', 'Electronics', 699.99),
23     (3, 'Tablet', 'Electronics', 399.99),
24     (4, 'Headphones', 'Electronics', 149.99),
25     (5, 'Smart Watch', 'Electronics', 249.99);
26
27 • INSERT INTO product_recommendations (product_id, recommended_product_id, strength) VALUES

```

```

27 • INSERT INTO product_recommendations (product_id, recommended_product_id, strength) VALUES
28     (1, 2, 0.8),
29     (1, 3, 0.6),
30     (1, 4, 0.7),
31     (2, 1, 0.8),
32     (2, 3, 0.9),
33     (2, 5, 0.7),
34     (3, 1, 0.6),
35     (3, 2, 0.9),
36     (3, 4, 0.5);
37
38 • select p1.product_name,
39     p2.product_name,
40     pr.strength
41 from product_recommendations pr
42 join products p1 on p1.product_id = pr.product_id
43 join products p2 on pr.recommended_product_id = p2.product_id
44 where p1.product_id = 1
45 order by pr.strength desc

```

Result Grid | Filter Rows: | Exports: | Wrap Cell Content: |

	product_name	product_name	strength
▶	Laptop	Smartphone	0.80
	Laptop	Headphones	0.70
	Laptop	Tablet	0.60

Selects product names and the recommendation strength for a specific product (with product\_id = 1 in this case), showing which products are recommended based on the specified strength. The results are ordered by the strength in descending order. This is used to retrieve product recommendations based on the product\_id.



```
50 -- Example 3: Flight Connections
51 • CREATE TABLE airports (
52     airport_code CHAR(3) PRIMARY KEY,
53     airport_name VARCHAR(100),
54     city VARCHAR(50),
55     country VARCHAR(50)
56 );
57
58 • CREATE TABLE flights (
59     flight_id INT PRIMARY KEY,
60     departure_airport CHAR(3),
61     arrival_airport CHAR(3),
62     departure_time TIME,
63     arrival_time TIME,
64     FOREIGN KEY (departure_airport) REFERENCES airports(airport_code),
65     FOREIGN KEY (arrival_airport) REFERENCES airports(airport_code)
66 );
67
68 • INSERT INTO airports (airport_code, airport_name, city, country) VALUES
69 ('JFK', 'John F. Kennedy International Airport', 'New York', 'USA'),
70 ('LAX', 'Los Angeles International Airport', 'Los Angeles', 'USA'),
71 ('LHR', 'London Heathrow Airport', 'London', 'UK'),
72 ('CDG', 'Charles de Gaulle Airport', 'Paris', 'France'),
73 ('NRT', 'Narita International Airport', 'Tokyo', 'Japan');
74
75 • INSERT INTO flights (flight_id, departure_airport, arrival_airport, departure_time, arrival_time) VALUES
76 (1, 'JFK', 'LAX', '08:00', '11:00'),
77 (2, 'LAX', 'NRT', '13:00', '17:00'),
78 (3, 'NRT', 'LHR', '19:00', '23:00');
```

```

84  -- Find all possible one-stop flights from JFK to NRT
85
86  •  select
87      f1.flight_id,
88      a1.city as departure_city,
89      a2.city as layover_city,
90      a3.city as arrival_city,
91      f1.departure_time,
92      f1.arrival_time as layover_arrival,
93      f2.departure_time as layover_departure,
94      f2.arrival_time
95  from flights f1
96  join flights f2 on f1.arrival_airport = f2.departure_airport
97  join airports a1 on f1.departure_airport = a1.airport_code
98  join airports a2 on f1.arrival_airport = a2.airport_code
99  join airports a3 on f2.arrival_airport = a3.airport_code
100 where f1.departure_airport = 'JFK'
101       and f2.arrival_airport = 'NRT'
102       and f2.departure_time > f1.arrival_time;
103

```

Result Grid   Filter Rows:  | Export:  | Wrap Cell Contents: 

	flight_id	departure_city	layover_city	arrival_city	departure_time	layover_arrival	layover_departure	arrival_time
▶	1	New York	Los Angeles	Tokyo	08:00:00	11:00:00	13:00:00	17:00:00

```

106 • CREATE TABLE employees (
107     employee_id int PRIMARY KEY,
108     first_name VARCHAR(100),
109     last_name VARCHAR(50),
110     manager_id int
111 );
112
113 • INSERT INTO employees (employee_id, first_name, last_name, manager_id) VALUES
114 (1, 'John', 'Doe', NULL), -- CEO
115 (2, 'Jane', 'Smith', 1), -- Reports to John
116 (3, 'Bob', 'Johnson', 1), -- Reports to John
117 (4, 'Alice', 'Williams', 2), -- Reports to Jane
118 (5, 'Charlie', 'Brown', 2), -- Reports to Jane
119 (6, 'David', 'Lee', 3), -- Reports to Bob
120 (7, 'Eva', 'Garcia', 3), -- Reports to Bob
121 (8, 'Frank', 'Lopez', 4), -- Reports to Alice
122 (9, 'Grace', 'Kim', 6), -- Reports to David
123 (10, 'Henry', 'Chen', 7);

```

```

125 • with recursive employee_hierarchy as (
126     select employee_id, first_name, last_name, manager_id, 0 as level
127     from employees
128     where manager_id is NULL
129     union all
130     select e.employee_id, e.first_name, e.last_name, e.manager_id, 1
131     from employees e
132     join employee_hierarchy eh on eh.manager_id = e.employee_id
133 )
134 select
135     employee_id,
136     concat(first_name, ' ', last_name) as employee_name,
137     level
138 from employee_hierarchy
139 order by level, employee_id
140

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	employee_id	employee_name	level
▶	1	John Doe	0

```
1 • create database activity;
2
3 • use activity;
4
5 -- Sample Data Setup
6 • CREATE TABLE employees (
7     employee_id INT PRIMARY KEY,
8     first_name VARCHAR(50),
9     last_name VARCHAR(50),
10    department VARCHAR(50),
11    salary DECIMAL(10, 2),
12    hire_date DATE
13 );
14
15 • INSERT INTO employees VALUES
16 (1, 'John', 'Doe', 'IT', 75000, '2020-01-15'),
17 (2, 'Jane', 'Smith', 'HR', 65000, '2019-05-11'),
18 (3, 'Bob', 'Johnson', 'IT', 80000, '2018-03-23'),
19 (4, 'Alice', 'Williams', 'Finance', 72000, '2021-09-30'),
20 (5, 'Charlie', 'Brown', 'IT', 68000, '2022-02-14'),
21 (6, 'Eva', 'Davis', 'HR', 61000, '2020-11-18'),
22 (7, 'Frank', 'Miller', 'Finance', 79000, '2017-07-12'),
23 (8, 'Grace', 'Taylor', 'IT', 77000, '2019-04-22'),
24 (9, 'Henry', 'Anderson', 'Finance', 71000, '2021-01-05'),
25 (10, 'Ivy', 'Thomas', 'HR', 63000, '2022-06-30');
26
27 • CREATE TABLE projects (
28     project_id INT PRIMARY KEY,
29     project_name VARCHAR(100),
```

```
30     start_date DATE,
31     end_date DATE
32 );
33
34 • INSERT INTO projects VALUES
35 (1, 'Database Migration', '2023-01-01', '2023-06-30'),
36 (2, 'New HR System', '2023-03-15', '2023-12-31'),
37 (3, 'Financial Reporting Tool', '2023-02-01', '2023-11-30'),
38 (4, 'IT Infrastructure Upgrade', '2023-05-01', '2024-04-30');
39
40 • CREATE TABLE employee_projects (
41     employee_id INT,
42     project_id INT,
43     role VARCHAR(50),
44     FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
45     FOREIGN KEY (project_id) REFERENCES projects(project_id)
46 );
47
48 • INSERT INTO employee_projects VALUES
49 (1, 1, 'Project Lead'),
50 (2, 2, 'Project Manager'),
51 (3, 1, 'Database Admin'),
52 (4, 3, 'Financial Analyst'),
53 (5, 4, 'Network Engineer'),
54 (6, 2, 'HR Specialist'),
55 (7, 3, 'Data Analyst'),
56 (8, 4, 'Systems Architect'),
57 (1, 4, 'Security Consultant'),
58 (3, 4, 'Software Developer');
```



```

60  -- Questions
61
62  -- 1. Write a query to find the top 3 highest paid employees in each department.
63
64  • SELECT e.department, e.first_name, e.last_name, e.salary
65     FROM employees e
66     WHERE (SELECT COUNT(DISTINCT salary)
67            FROM employees
68            WHERE department = e.department AND salary > e.salary) < 3
69     ORDER BY e.department, e.salary DESC;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	department	first_name	last_name	salary
▶	Finance	Frank	Miller	79000.00
	Finance	Alice	Williams	72000.00
	Finance	Henry	Anderson	71000.00
	HR	Jane	Smith	65000.00
	HR	Ivy	Thomas	63000.00
	HR	Eva	Davis	61000.00
	IT	Bob	Johnson	80000.00
	IT	Grace	Taylor	77000.00
	IT	John	Doe	75000.00

```

72  -- 2. Calculate the running total of salaries in each department, ordered by hire date.
73
74  • SELECT e.department, e.first_name, e.last_name, e.salary,
75         SUM(e.salary) OVER (PARTITION BY e.department ORDER BY e.hire_date) AS running_total
76     FROM employees e
77     ORDER BY e.department, e.hire_date;
78
79

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	department	first_name	last_name	salary	running_total
▶	Finance	Frank	Miller	79000.00	79000.00
	Finance	Henry	Anderson	71000.00	150000.00
	Finance	Alice	Williams	72000.00	222000.00
	HR	Jane	Smith	65000.00	65000.00
	HR	Eva	Davis	61000.00	126000.00
	HR	Ivy	Thomas	63000.00	189000.00
	IT	Bob	Johnson	80000.00	80000.00
	IT	Grace	Taylor	77000.00	157000.00
	IT	John	Doe	75000.00	232000.00
	IT	Charlie	Brown	68000.00	300000.00

```

80  -- 3. Find employees who are working on more than one project, along with the count of projects they're involved in.
81
82  * SELECT e.first_name, e.last_name, COUNT(ep.project_id) AS project_count
83  FROM employees e
84  JOIN employee_projects ep ON e.employee_id = ep.employee_id
85  GROUP BY e.employee_id
86  HAVING COUNT(ep.project_id) > 1;

```

Result Grid	Filter Rows:	Export:	Wrap Cell Contents:
first_name	last_name	project_count	
John	Doe	2	
Bob	Johnson	2	

```

88  -- 4. Identify projects that have team members from all departments.
89
90  * SELECT p.project_name
91  FROM projects p
92  JOIN employee_projects ep ON p.project_id = ep.project_id
93  JOIN employees e ON ep.employee_id = e.employee_id
94  GROUP BY p.project_id
95  HAVING COUNT(DISTINCT e.department) = (SELECT COUNT(DISTINCT department) FROM employees);
96

```

Result Grid	Filter Rows:	Export:	Wrap Cell Contents:
project_name			

```

77
98 -- 5. Calculate the average salary for each department, but only include employees hired in the last 3 years.
99
100 * SELECT department, AVG(salary) AS avg_salary
101 FROM employees
102 WHERE hire_date >= DATE_SUB(CURDATE(), INTERVAL 3 YEAR)
103 GROUP BY department;
104

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	department	avg_salary		
▶	IT	68000.000000		
	HR	63000.000000		

```

106 -- 6. Create a pivot table showing the count of employees in each department, with columns for different salary ranges (e.g., <65000, 65000-75000, >75000)
107
108 * SELECT department,
109         SUM(CASE WHEN salary < 65000 THEN 1 ELSE 0 END) AS "<65000",
110         SUM(CASE WHEN salary BETWEEN 65000 AND 75000 THEN 1 ELSE 0 END) AS "65000-75000",
111         SUM(CASE WHEN salary > 75000 THEN 1 ELSE 0 END) AS ">75000"
112 FROM employees
113 GROUP BY department;
114

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	department	<65000	65000-75000	>75000
▶	IT	0	2	2
	HR	2	1	0
	Finance	0	2	1



```

115
116 -- 7. Find the employee(s) with the highest salary in their respective departments, who are also working on the longest-running project.
117
118 * WITH longest_projects AS (
119     SELECT project_id, DATEDIFF(end_date, start_date) AS project_duration
120     FROM projects
121     ORDER BY project_duration DESC
122     LIMIT 1
123 )
124 SELECT e.first_name, e.last_name, e.salary, e.department
125 FROM employees e
126 JOIN employee_projects ep ON e.employee_id = ep.employee_id
127 JOIN longest_projects lp ON ep.project_id = lp.project_id
128 WHERE e.salary = (SELECT MAX(salary) FROM employees e2 WHERE e2.department = e.department);

```

Result Grid Filter Rows: Exports Wrap Cell Contents:

first_name	last_name	salary	department
Bob	Johnson	80000.00	IT

```

131 -- 8. Calculate the percentage of each department's salary compared to the total salary of the company.
132
133 * SELECT department,
134     SUM(salary) AS department_total,
135     (SUM(salary) / (SELECT SUM(salary) FROM employees) * 100) AS percentage_of_total
136 FROM employees
137 GROUP BY department;
138

```

Result Grid Filter Rows: Exports Wrap Cell Contents:

department	department_total	percentage_of_total
IT	300000.00	42.194093
HR	189000.00	26.582278
Finance	222000.00	31.223629

```

141  -- 9. Identify employees who have a higher salary than their department's average, and show by what percentage their salary exceeds the average
142
143  * WITH department_avg AS (
144      SELECT department, AVG(salary) AS avg_salary
145      FROM employees
146      GROUP BY department
147  )
148  SELECT e.first_name, e.last_name, e.salary, da.avg_salary,
149      ((e.salary - da.avg_salary) / da.avg_salary * 100) AS percentage_above_avg
150  FROM employees e
151  JOIN department_avg da ON e.department = da.department
152  WHERE e.salary > da.avg_salary;

```

first_name	last_name	salary	avg_salary	percentage_above_avg
Jane	Smith	65000.00	63000.000000	3.1746031746
Bob	Johnson	80000.00	75000.000000	6.6666666667
Frank	Miller	79000.00	74000.000000	6.7567567568
Grace	Taylor	77000.00	75000.000000	2.6666666667

```

155  -- 10. Create a query that shows a hierarchical view of employees and their projects, with multiple levels of projects if an employee is in more
156
157  * SELECT e.first_name, e.last_name, ep.project_id, p.project_name, ep.role
158  FROM employees e
159  JOIN employee_projects ep ON e.employee_id = ep.employee_id
160  JOIN projects p ON ep.project_id = p.project_id
161  ORDER BY e.employee_id, ep.project_id;

```

first_name	last_name	project_id	project_name	role
John	Doe	1	Database Migration	Project Lead
John	Doe	4	IT Infrastructure Upgrade	Security Consultant
Jane	Smith	2	New HR System	Project Manager
Bob	Johnson	1	Database Migration	Database Admin
Bob	Johnson	4	IT Infrastructure Upgrade	Software Developer
Alice	Williams	3	Financial Reporting Tool	Financial Analyst
Charlie	Brown	4	IT Infrastructure Upgrade	Network Engineer
Eve	Davis	2	New HR System	HR Specialist
Frank	Miller	3	Financial Reporting Tool	Data Analyst
Grace	Taylor	4	IT Infrastructure Upgrade	Systems Architect

Order of Execution of Queries:

1. From (including join)
2. Where
3. Group By
4. Having
5. Select
6. Order By
7. Top Limit

Understanding and Interpreting EXPLAIN Output for Query Optimization:

1. Id:
  - Identifies each SELECT in the query
  - A sequential number for simple queries
  - Can be the same for JOINed tables or subqueries

2. Select\_type:

- SIMPLE: Simple SELECT (no subqueries or UNIONs)
- PRIMARY: Outermost SELECT
- SUBQUERY: First SELECT in a subquery
- DERIVED: SELECT in a derived table (subquery in FROM clause)
- UNION: Second or later SELECT in a UNION
- UNIONRESULT: Result of a UNION

3. Table:

- The table this row refers to
- Can be a derived table name for subqueries

4. Partitions:

- Partitions read, if the table is partitioned

5. Type:

- Join type, crucial for optimization
- Common values:- system: Table has only one row
- const: Matching one row, very fast
- eq\_ref: One row read per combination of rows from previous tables
- ref: All matching rows from an index read for each combination
- range: Index used to retrieve rows within a range- index: Full index scan
- ALL: Full table scan (slowest)

6. Possible\_keys:

- Indexes that could be used
- NULL if no index could be used

7. Key:

- The index actually chosen
- NULL if no index was used

8. Key\_len:

- Length of the chosen key

9. Ref:

- Columns or constants used with the key

10. Rows:

- Estimated number of rows to examine

11. Filtered:

- Estimated percentage of rows filtered by table condition

12. Extra:

- Additional information like "Using index", "Using temporary", "Using filesort"

Example: Let's say we have this query:

sql

```
EXPLAIN SELECT e.first_name, d.department_name FROM employees e JOIN departments d
ON e.department_id = d.department_id WHERE e.salary > 50000;
```

Possible EXPLAIN output might look like:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
----	-------------	-------	------------	------	---------------	-----	---------	-----	------	----------	-------

1	SIMPLE	e	NULL	ALL	NULL	1000	33.33	Using where	1	SIMPLE	d	NULL	NULL	NULL	NULL	eq_ref	PRIMARY	PRIMARY	4	employees.department_id	1	100.00	NULL
---	--------	---	------	-----	------	------	-------	-------------	---	--------	---	------	------	------	------	--------	---------	---------	---	-------------------------	---	--------	------

Interpreting this:

1. It's a SIMPLE query (no subqueries or UNIONS).
2. It's scanning the entire employees table (type: ALL) which is inefficient.
3. It's using an index lookup on departments (type: eq\_ref).
4. It estimates examining 1000 rows from employees.
5. The WHERE condition is filtering about 33.33% of the rows.
6. It's using a WHERE clause on the employees table.

This output suggests that adding an index on the salary column in the employees table could potentially improve the query's performance. Understanding EXPLAIN output helps in identifying performance bottlenecks and guides index creation and query rewriting for optimization.

Detailed Breakdown of EXPLAIN Output Components for Query Optimization:

1. id:- Purpose: Identifies each SELECT in the query.

- Example: sql

```
EXPLAIN SELECT *FROM employees UNION SELECT *FROM retired_employees;
```

Output:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	employees	...	...	...	...	...	...	...	...	...
2	UNION	retired_employees	...	...	...	...	...	...	...	...	...
3	UNIONRESULT	...	...	...	...	...	...	...	...	...	...

- Explanation: Here, 1 and 2 represent the two SELECT statements, while 3 is the result of the UNION.

## 2. Select\_type:

- Purpose: Indicates the type of SELECT statement.

- Examples:

a) SIMPLE: sql

```
EXPLAIN SELECT *FROMemployees WHEREsalary > 50000;
```

b) SUBQUERY:

sql

```
EXPLAIN SELECT * FROM employees WHERE department_id IN (SELECT id FROM departments WHERE location = 'New York');
```

c) DERIVED:

sql

```
EXPLAIN SELECT * FROM (SELECT id, name FROM employees WHERE salary > 50000) AS high_paid_employees;
```

## 3. Table:

- Purpose: Shows which table the row refers to.

- Example:

sql

```
EXPLAIN SELECT e.name, d.name FROMemployees e JOIN departments d ON e.department_id = d.id;
```

Output might show 'e' and 'd' in the table column.

## 4. Partitions:

- Purpose: Indicates which partitions are being accessed

- Example (assuming 'employees' is partitioned by year):

sql

```
EXPLAIN SELECT * FROM employees WHERE hire_date BETWEEN '2020-01-01' AND '2020-12-31';
```

Might show 'p2020' in the partitions column.

## 5. type:

-Purpose: Shows the join type, crucial for understanding query efficiency.

- Examples:

a) const:

sql

```
EXPLAIN SELECT *FROMemployees WHEREid=1;
```

b) ref:

sql

```
EXPLAIN SELECT *FROMemployees WHEREdepartment_id = 5; (Assuming department_id is indexed but not unique)
```

c) range:

sql

```
EXPLAIN SELECT *FROMemployees WHEREsalary BETWEEN 50000 AND 60000;
```

(Assuming salary is indexed)

6. Possible\_keys:

- Purpose: Lists indexes that could potentially be used.

- Example:

sql

```
EXPLAIN SELECT *FROMemployees WHEREdepartment_id = 5 AND salary > 50000; Might show 'idx_dept_id, idx_salary' if both columns are indexed.
```

7. Key:

- Purpose: Shows the actual index used.

- Example: In the above query, it might show 'idx\_dept\_id' if MySQL decides this is more selective.

8. Key\_len:

- Purpose: Indicates how much of the index is being used.

- Example: If department\_id is an INT (4 bytes), key\_len might show '4'.

9. Ref:

- Purpose: Shows which columns or constants are compared to the index.

- Example:

sql

```
EXPLAIN SELECT * FROM employees e JOIN departments d ON e.department_id = d.id WHEREd.name ='IT';
```

Might show 'const' for the departments table and 'd.id' for the employees table.

10. Rows:

- Purpose: Estimates the number of rows examined.

- Example: In a query on a large table, this might show a high number like 10000, indicating potential for optimization.

11. Filtered:

- Purpose: Estimates the percentage of rows filtered by table conditions.

- Example: In a query with WHERE salary > 50000, if 30% of employees meet this condition, it might show 30.00.

12. Extra:

- Purpose: Provides additional information about how MySQL executes the query.

- Examples:

a) "Using index" (good): sql EXPLAIN SELECT id, name FROM employees WHERE id > 1000;

b) "Using temporary" (potential for optimization): sql EXPLAIN SELECT department\_id,

```
AVG(salary) FROM employees GROUP BY department_id ORDER BY AVG(salary);
```

c) "Using filesort" (potential for optimization):

```
sql EXPLAIN SELECT *FROMemployees ORDER BYlast_name;
```

(Assuming no index on last\_name)

Understanding these elements helps in query optimization. For instance, seeing 'ALL' in the type column or high numbers in the rows column often indicates areas for improvement, potentially by adding indexes or rewriting the query.

## 1. Primary Keys

A primary key is a column or set of columns in a table that uniquely identifies each row.

Key characteristics:

- Must be unique
- Cannot contain NULL values
- Should be immutable (not change over time)

Example:

Let's create a simple "Students" table:

```
sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DateOfBirth DATE
);
```

Here, StudentID is the primary key.

## 2. Foreign Keys

A foreign key is a column or set of columns in one table that refers to the primary key in another table. It establishes a link between two tables.

Key characteristics:

- Creates a relationship between tables
- Ensures referential integrity
- Can contain NULL values (unless specified otherwise)

Example:

Let's create a "Courses" table and an "Enrollments" table:

sql

```
CREATE TABLE Courses (  
    CourseID INT PRIMARY KEY,  
    CourseName VARCHAR(100)  
);
```

```
CREATE TABLE Enrollments (  
    EnrollmentID INT PRIMARY KEY,  
    StudentID INT,  
    CourseID INT,  
    EnrollmentDate DATE,  
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),  
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)  
);
```

Here, StudentID and CourseID in the Enrollments table are foreign keys referencing the Students and Courses tables, respectively.

### 3. Normalization

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. There are several normal forms, but we'll focus on the first three, which are the most commonly used.

First Normal Form (1NF):

- Each column contains atomic (indivisible) values
- No repeating groups

Example:

Consider this unnormalized table:

Students:

StudentID	Name	Courses
1	John Doe	Math, Physics, Chemistry
2	Jane Smith	Biology, Chemistry

To bring it to 1NF:



sql

```
CREATE TABLE Students (  
    StudentID INT PRIMARY KEY,  
    Name VARCHAR(100)  
);
```

```
CREATE TABLE StudentCourses (  
    StudentID INT,  
    Course VARCHAR(50),  
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)  
);
```

Second Normal Form (2NF):

- Must be in 1NF
- All non-key attributes are fully functionally dependent on the primary key

Example:

Consider this table that's in 1NF but not 2NF:

Orders:

OrderID | ProductID | ProductName | Quantity | CustomerID | CustomerName

To bring it to 2NF:

sql

```
CREATE TABLE Orders (  
    OrderID INT,  
    ProductID INT,  
    Quantity INT,  
    CustomerID INT,  
    PRIMARY KEY (OrderID, ProductID)  
);
```

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY,  
    ProductName VARCHAR(100)  
);
```

```
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    CustomerName VARCHAR(100)
```

);

Third Normal Form (3NF):

- Must be in 2NF
- No transitive dependencies (non-key columns depend only on the primary key)

Example:

Consider this table that's in 2NF but not 3NF:

Employees:

EmployeeID | Name | DepartmentID | DepartmentName | ManagerID

To bring it to 3NF:

sql

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    DepartmentID INT,  
    ManagerID INT,  
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)  
);
```

```
CREATE TABLE Departments (  
    DepartmentID INT PRIMARY KEY,  
    DepartmentName VARCHAR(100)  
);
```

These normalized structures reduce data redundancy and improve data integrity. Each piece of information is stored in only one place, making updates and maintenance easier and reducing the risk of data inconsistencies.

Let's start with an unnormalized table for a small bookstore:

BookOrders:

OrderID | CustomerName | CustomerEmail | BookTitle | Author | Price | OrderDate |  
ShippingAddress

## Step 1: First Normal Form (1NF)

To achieve 1NF, we need to ensure that:

1. Each column contains atomic values
2. There are no repeating groups

Our table is almost in 1NF, but let's assume that sometimes multiple books are ordered together. We'll split this into two tables:

sql

```
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY,  
    CustomerName VARCHAR(100),  
    CustomerEmail VARCHAR(100),  
    OrderDate DATE,  
    ShippingAddress VARCHAR(200)  
);
```

```
CREATE TABLE OrderDetails (  
    OrderDetailID INT PRIMARY KEY,  
    OrderID INT,  
    BookTitle VARCHAR(200),  
    Author VARCHAR(100),  
    Price DECIMAL(10, 2),  
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)  
);
```

Now we're in 1NF. Each column contains atomic values, and we've eliminated the repeating group (multiple books per order).

## Step 2: Second Normal Form (2NF)

To achieve 2NF:

1. The table must be in 1NF
2. All non-key attributes must be fully functionally dependent on the primary key

Our Orders table is already in 2NF because all attributes depend on the primary key OrderID.

However, in the OrderDetails table, BookTitle, Author, and Price don't depend on the full primary key (OrderDetailID), but only on part of it (BookTitle). Let's fix this:

sql

```
CREATE TABLE Books (  
    BookTitle VARCHAR(200),  
    Author VARCHAR(100),  
    Price DECIMAL(10, 2)
```

```
BookID INT PRIMARY KEY,  
BookTitle VARCHAR(200),  
Author VARCHAR(100),  
Price DECIMAL(10, 2)  
);  
  
CREATE TABLE OrderDetails (  
    OrderDetailID INT PRIMARY KEY,  
    OrderID INT,  
    BookID INT,  
    Quantity INT,  
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),  
    FOREIGN KEY (BookID) REFERENCES Books(BookID)  
);
```

Now we're in 2NF. All non-key attributes in each table fully depend on their respective primary keys.

### Step 3: Third Normal Form (3NF)

To achieve 3NF:

1. The table must be in 2NF
2. There should be no transitive dependencies (non-key columns should depend only on the primary key)

In our current structure, we have a transitive dependency in the Orders table: CustomerEmail depends on CustomerName, which depends on OrderID. Let's resolve this:

```
sql  
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    CustomerName VARCHAR(100),  
    CustomerEmail VARCHAR(100)  
);  
  
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY,  
    CustomerID INT,  
    OrderDate DATE,  
    ShippingAddress VARCHAR(200),  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);
```

-- Books and OrderDetails tables remain the same

Now we're in 3NF. All non-key attributes in each table depend only on the primary key, and we've eliminated transitive dependencies.

Final 3NF Structure:

sql

```
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    CustomerName VARCHAR(100),  
    CustomerEmail VARCHAR(100)  
);  
  
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY,  
    CustomerID INT,  
    OrderDate DATE,  
    ShippingAddress VARCHAR(200),  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);  
  
CREATE TABLE Books (  
    BookID INT PRIMARY KEY,  
    BookTitle VARCHAR(200),  
    Author VARCHAR(100),  
    Price DECIMAL(10, 2)  
);  
  
CREATE TABLE OrderDetails (  
    OrderDetailID INT PRIMARY KEY,  
    OrderID INT,  
    BookID INT,  
    Quantity INT,  
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),  
    FOREIGN KEY (BookID) REFERENCES Books(BookID)  
);
```

This structure in 3NF offers several benefits:

1. Reduced data redundancy: Customer and book information is stored only once.

2. Improved data integrity: Updating customer or book information only needs to happen in one place.
3. Easier data maintenance: Adding or modifying data is simpler and less error-prone.
4. Flexible querying: It's easier to query and analyze data across different aspects of the business.

Here's a summary of the changes made at each step:

1NF: Split the single table into Orders and OrderDetails to handle multiple books per order.

2NF: Extracted book information into a separate Books table to remove partial dependencies.

3NF:

```
CREATE TABLE customers (  
    customer_id INT PRIMARY KEY AUTO_INCREMENT,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    email VARCHAR(100) UNIQUE,  
    registration_date DATE  
);
```

```
CREATE TABLE categories (  
    category_id INT PRIMARY KEY AUTO_INCREMENT,  
    category_name VARCHAR(50) UNIQUE  
);
```

```
CREATE TABLE products (  
    product_id INT PRIMARY KEY AUTO_INCREMENT,  
    product_name VARCHAR(100),  
    category_id INT,  
    price DECIMAL(10, 2),  
    stock_quantity INT,  
    FOREIGN KEY (category_id) REFERENCES categories(category_id)  
);
```

```
CREATE TABLE orders (  
    order_id INT PRIMARY KEY AUTO_INCREMENT,  
    customer_id INT,  
    order_date DATETIME,  
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)  
);
```

```
CREATE TABLE order_items (  
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,  
    order_id INT,  
    product_id INT,
```

```

    quantity INT,
    price_per_unit DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

CREATE TABLE product_reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    customer_id INT,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    review_text TEXT,
    review_date DATE,
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

```

This example demonstrates how normalization progressively organizes data to reduce redundancy and improve data integrity. Each step builds on the previous one, resulting in a well-structured, efficient database design.

Given table database schema and questions

```

-- Create Tables
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    registration_date DATE
);

CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10, 2),
    stock_quantity INT
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,

```

```
    order_date DATETIME,  
    total_amount DECIMAL(10, 2),  
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)  
);
```

```
CREATE TABLE order_items (  
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,  
    order_id INT,  
    product_id INT,  
    quantity INT,  
    price_per_unit DECIMAL(10, 2),  
    FOREIGN KEY (order_id) REFERENCES orders(order_id),  
    FOREIGN KEY (product_id) REFERENCES products(product_id)  
);
```

```
CREATE TABLE product_reviews (  
    review_id INT PRIMARY KEY AUTO_INCREMENT,  
    product_id INT,  
    customer_id INT,  
    rating INT CHECK (rating BETWEEN 1 AND 5),  
    review_text TEXT,  
    review_date DATE,  
    FOREIGN KEY (product_id) REFERENCES products(product_id),  
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)  
);
```

-- Advanced SQL Practice Questions

-- Question 1: Find the top 3 customers who have spent the most money,  
-- along with their total spend and the number of orders they've made.

-- Question 2: Calculate the average rating for each product category,  
-- but only include categories with at least 2 reviews.

-- Question 3: Find products that have never been ordered.

-- Question 4: For each customer, find the time difference between their  
-- registration date and their first order date.

-- Question 5: Create a report showing the total revenue for each month,  
-- along with a running total of revenue throughout the year.

-- Question 6: Identify customers who have made a purchase but have never left a product  
review.



- Question 7: Find the product that has been ordered the most times (by quantity).
- Question 8: Calculate the percentage of total revenue that each product category contributes.
- Question 9: For each customer, find their most frequently purchased product category.
- Question 10: Create a query to show the distribution of ratings (count of 1-star, 2-star, etc.) for each product.

Ans

1NF-> table should contain only atomic values and each record must be unique , so all the tables already follows 1NF

2NF-> all the tables should be in 1NF form and all non key attributes must fully functionally be dependent on primary keys, all the tables already follows 2NF

3NF-> all the tables should follow 2NF and no non key columns should be dependent on any non key columns, the given schema is already normalized up to 3NF

Sql queries:

1.

```
SELECT c.customer_id, c.first_name, c.last_name,  
       SUM(o.total_amount) AS total_spend, COUNT(o.order_id) AS num_orders  
FROM customers c  
JOIN orders o ON c.customer_id = o.customer_id  
GROUP BY c.customer_id  
ORDER BY total_spend DESC  
LIMIT 3;
```

2.

```
SELECT p.category, AVG(r.rating) AS avg_rating  
FROM products p  
JOIN product_reviews r ON p.product_id = r.product_id  
GROUP BY p.category  
HAVING COUNT(r.review_id) >= 2;
```

3.

```
SELECT p.product_id, p.product_name  
FROM products p  
LEFT JOIN order_items oi ON p.product_id = oi.product_id  
WHERE oi.order_item_id IS NULL;
```

4.

```
SELECT c.customer_id, c.first_name, c.last_name,
```

```
DATEDIFF(MIN(o.order_date), c.registration_date) AS days_to_first_order
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id;
```

5.

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
       SUM(total_amount) AS monthly_revenue,
       SUM(SUM(total_amount)) OVER (ORDER BY DATE_FORMAT(order_date, '%Y-%m')) AS
running_total
FROM orders
GROUP BY DATE_FORMAT(order_date, '%Y-%m');
```

6.

```
SELECT DISTINCT c.customer_id, c.first_name, c.last_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
LEFT JOIN product_reviews r ON c.customer_id = r.customer_id
WHERE r.review_id IS NULL;
```

7.

```
SELECT p.product_id, p.product_name, SUM(oi.quantity) AS total_quantity
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id
ORDER BY total_quantity DESC
LIMIT 1;
```

8.

```
WITH total_revenue AS (
  SELECT SUM(oi.quantity * oi.price_per_unit) AS total
  FROM order_items oi
)
SELECT p.category,
       SUM(oi.quantity * oi.price_per_unit) / (SELECT total FROM total_revenue) * 100 AS
revenue_percentage
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.category;
```

9.

```
SELECT c.customer_id, c.first_name, c.last_name, p.category,
       COUNT(oi.order_item_id) AS order_count
FROM customers c
```

```
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY c.customer_id, p.category
ORDER BY order_count DESC;
```

10.

```
SELECT p.product_id, p.product_name, r.rating, COUNT(r.rating) AS rating_count
FROM products p
JOIN product_reviews r ON p.product_id = r.product_id
GROUP BY p.product_id, r.rating;
```

Specialization:

The table creation of customer can be specialized as:

```
CREATE TABLE corporate_customers (
    corporate_customer_id INT PRIMARY KEY AUTO_INCREMENT,
    company_name VARCHAR(100),
    contact_person VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    registration_date DATE
);
```

```
CREATE TABLE regular_customers (
    regular_customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    registration_date DATE
);
```

```
.....
CREATE TABLE stock_prices (
    date DATE,
    stock_symbol VARCHAR(10),
    closing_price DECIMAL(10, 2)
);
```

-- Insert sample data

```
INSERT INTO stock_prices (date, stock_symbol, closing_price) VALUES
('2023-01-01', 'AAPL', 150.00),
('2023-01-02', 'AAPL', 152.50),
('2023-01-03', 'AAPL', 151.75),
('2023-01-04', 'AAPL', 153.00),
('2023-01-05', 'AAPL', 155.25),
```

```
('2023-01-01', 'GOOGL', 2800.00),  
( '2023-01-02', 'GOOGL', 2825.00),  
( '2023-01-03', 'GOOGL', 2815.50),  
( '2023-01-04', 'GOOGL', 2830.00),  
( '2023-01-05', 'GOOGL', 2850.75);
```

```
select date,  
stock_symbol,  
closing_price,  
lead(closing_price,2) over (partition by stock_symbol order by date) as next_day_price  
from stock_prices  
order by stock_symbol, date;
```

```
select date,  
stock_symbol,  
lag(closing_price,1) over (partition by stock_symbol order by date) as previous_day_price  
,closing_price  
from stock_prices  
order by stock_symbol, date;
```

```
select date,  
stock_symbol,  
closing_price,  
lag(closing_price,1) over (partition by stock_symbol order by date) as previous_day_price  
,closing_price-LAG(closing_price,1) over (partition by stock_symbol order by date) as  
price_change  
from stock_prices  
order by stock_symbol, date;
```

```
-- Create tables  
CREATE TABLE products (  
    product_id INT PRIMARY KEY,  
    product_name VARCHAR(100),  
    category VARCHAR(50),  
    unit_price DECIMAL(10, 2)  
);
```

```
CREATE TABLE sales (  
    sale_id INT PRIMARY KEY,
```

```

    product_id INT,
    sale_date DATE,
    quantity INT,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

-- Insert sample data
INSERT INTO products (product_id, product_name, category, unit_price) VALUES
(1, 'Laptop', 'Electronics', 999.99),
(2, 'Smartphone', 'Electronics', 599.99),
(3, 'Tablet', 'Electronics', 399.99),
(4, 'Desk Chair', 'Furniture', 149.99),
(5, 'Coffee Table', 'Furniture', 199.99),
(6, 'Bookshelf', 'Furniture', 89.99),
(7, 'Running Shoes', 'Apparel', 79.99),
(8, 'T-shirt', 'Apparel', 19.99),
(9, 'Jeans', 'Apparel', 59.99);

INSERT INTO sales (sale_id, product_id, sale_date, quantity) VALUES
(1, 1, '2023-01-15', 2),
(2, 2, '2023-01-16', 3),
(3, 4, '2023-01-17', 1),
(4, 7, '2023-01-18', 4),
(5, 3, '2023-02-01', 2),
(6, 5, '2023-02-02', 1),
(7, 8, '2023-02-03', 5),
(8, 1, '2023-02-15', 1),
(9, 6, '2023-02-16', 2),
(10, 2, '2023-02-17', 2),
(11, 9, '2023-03-01', 3),
(12, 4, '2023-03-02', 2),
(13, 7, '2023-03-03', 3),
(14, 3, '2023-03-15', 1),
(15, 5, '2023-03-16', 1);

-- Problem Statement:
-- Analyze the sales data to find the top-performing product category for each month of 2023.
-- The performance should be based on total revenue (quantity sold * unit price).
-- Your query should return the month, the top-performing category, and its total revenue.
-- In case of a tie, include all categories with the same top performance.

-- Expected output format:
-- | month | top_category | total_revenue |
-- |-----|-----|-----|

```

```
-- | 2023-01 | Electronics | 2799.97 |  
-- | 2023-02 | Furniture   | 399.98   |  
-- | 2023-03 | Electronics | 999.98   |
```

-- Note: The actual values may differ based on the sample data provided.

SQL query:

```
WITH monthly_category_revenue AS (  
    SELECT  
        DATE_FORMAT(s.sale_date, '%Y-%m') AS sale_month,  
        p.category AS category_name,  
        SUM(s.quantity * p.unit_price) AS total_revenue  
    FROM sales s  
    JOIN products p ON s.product_id = p.product_id  
    WHERE YEAR(s.sale_date) = 2023  
    GROUP BY sale_month, p.category  
)  
ranked_categories AS (  
    SELECT  
        sale_month,  
        category_name,  
        total_revenue,  
        RANK() OVER (PARTITION BY sale_month ORDER BY total_revenue DESC) AS  
category_rank  
    FROM monthly_category_revenue  
)  
SELECT  
    sale_month,  
    category_name,  
    total_revenue  
FROM ranked_categories  
WHERE category_rank = 1  
ORDER BY sale_month;
```

Query: SELECT name FROM students WHERE id = 1008;

1. Database Layer:

- Determines student 1008 is in Page 1
- Page 1 = offset 8192, length 8192 bytes in students table file

2. File System Layer:

- Translates to reading two 4KB blocks:

Block 2 (offset 8192-12287)  
Block 3 (offset 12288-16383)

3. LBA Translation:

- Assuming students table starts at LBA 1000000:  
Block 2 → LBA 1000002  
Block 3 → LBA 1000003

4. SSD Controller:

- Receives command: Read LBAs 1000002 and 1000003
- Maps to SSD pages:  
LBA 1000002 → SSD Page 500000, offset 0  
LBA 1000003 → SSD Page 500000, offset 4096

5. Physical Storage:

- Reads entire SSD Page 500000 (16KB)

6. Data Return Path:

- SSD Controller extracts relevant 8KB from Page 500000
- File System reassembles into original 8KB request
- Database receives 8KB page, finds student 1008, returns name

Query: SELECT name FROM students WHERE id = 1008

1. Translate to page

Database Page 1

2. Calculate file offset

File Offset: 8192, Length: 8192 bytes

3. Map to file system blocks

File System Blocks

Block 2

Block 3

Offset: 8192-12287

Offset: 12288-16383

4. Translate to LBA

4. Translate to LBA

LBA 1000002

LBA 1000003

5. Map to SSD page

5. Map to SSD page

SSD Page 500000, Offset 0

SSD Page 500000, Offset 4096

6. Read physical data

6. Read physical data

Read 16KB from NAND Flash

7. Return data

Extract relevant 8KB

8. Reassemble

Reconstruct 8KB database page

9. Process

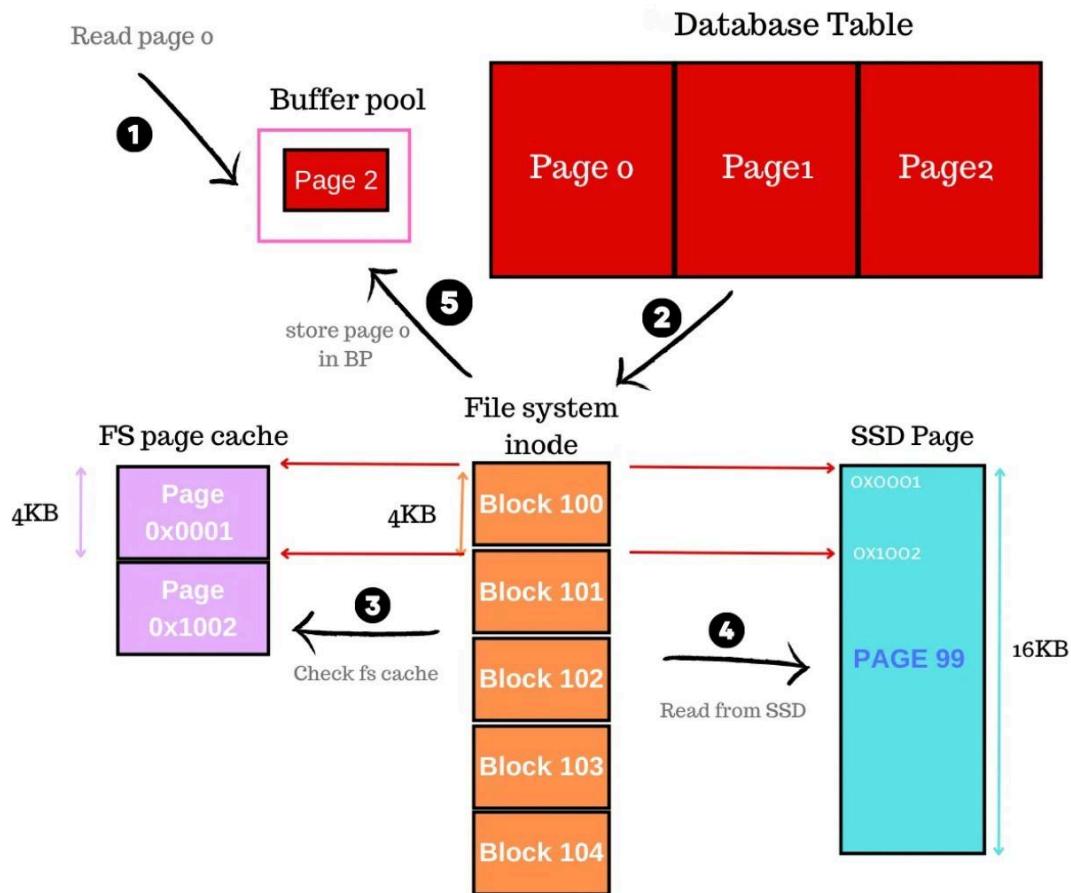
Find student 1008, extract name

10. Return result

Return name to user



# Following a database read to the metal



Let's use a large library as our analogy:

Database (Library Catalog):

Imagine a library catalog system. It doesn't store the actual books; instead, it stores information about where to find each book.

Example:

Book Title: "Database Systems"

Location: "Floor 3, Shelf 24, Position 15"

This is similar to how a database stores offsets instead of actual data. The offset is like saying "Floor 3, Shelf 24, Position 15" instead of storing the entire book in the catalog.

File System (Library Floors and Shelves):

The file system is like the physical organization of the library. It doesn't store the books either, but it manages how the shelves are arranged and numbered.

Example:

Floor 3 has shelves numbered 1-50

Each shelf can hold 100 books

LBA (Logical Book Address):

Now, imagine the library uses a unique numbering system for all books, regardless of their physical location. This is like an LBA.

Example:

"Database Systems" might be assigned LBA 10045

Physical Storage (Actual Book Location):

This is where the book actually sits on the shelf.

Now, let's see how this works in practice:

You ask the librarian (database query) for "Database Systems".

The librarian checks the catalog (database) and finds: "Floor 3, Shelf 24, Position 15".

This translates to a file offset in the computer world.

The librarian then uses a master list (file system) that says: "Floor 3, Shelf 24 = LBA range 10000-10099".

The LBA for this book is calculated as  $10000$  (start of shelf) +  $15$  (position) =  $10015$ .

A robot (SSD controller) that only understands LBAs is sent to fetch book 10015.

The robot has its own map that translates LBA 10015 to an exact physical location.

The book is retrieved and brought back to you.

Why is this system used?

Flexibility: The library can reorganize its shelves without changing the catalog.

Efficiency: It's faster to look up a simple number (LBA) than a complex address.

Abstraction: The librarian doesn't need to know how the robot finds the books.

In computers:

Database offset: Like the catalog entry, it tells where in a file to find data.

File system: Manages how files are organized on the storage device.

LBA: A simple numbering system for all data blocks on a storage device.

Physical storage: The actual location on the SSD or hard drive.

This system allows each layer (database, file system, storage device) to work independently while still communicating effectively.

```
create database test_db_poc;
```

```
use test_db_poc;
create table if not exists large_table(
id int auto_increment primary key,
name varchar(50),
value INT
);
```

```
DELIMITER //
CREATE procedure INSERT_MILLION_RECORDS()
BEGIN
  declare i int default 0;
  while i <100000 do
    insert into large_table(name,value) values(concat('Name',i),floor(1 + RAND()*1000000));
    set i=i+1;
  end while;
END //
```

```
DELIMITER ;
```

```
SHOW procedure status WHERE DB ='test_db_poc';
select count(*) from large_table;
CALL INSERT_MILLION_RECORDS();
```

```
select sql_no_cache sum(value) from large_table
select sum(value) from large_table
```

```
create index idx_value on large_table(value)
alter table large_table drop index idx_value
select * from large_table
```

```
select a.value
from large_table a
```

cross join large\_table b  
on a.value=b.value\*100

```
package com.example;
import java.sql.*;
public class Main {

    private static final String
url="jdbc:mysql://localhost:3306/test_million";
    private static final String username="root";
    private static final String password="root";
    private static Connection connection;
    private static CallableStatement callableStatement;
    private static PreparedStatement preparedStatement;

    public static void main(String[] args) {
        String call_procedure="{CALL INSERT_MILLION_RECORDS}";
        try{
            connection=DriverManager.getConnection(url, username,
password);
            //callableStatement=connection.prepareCall(call_procedure);
            //callableStatement.execute();
            String query="INSERT INTO large_table(name,value)VALUES(?
,?) ";
            preparedStatement=connection.prepareStatement(query);
            for(int i=0; i<1000000; i++){
                preparedStatement.setString(1,"name"+i);
                preparedStatement.setInt(2, i);
                preparedStatement.executeUpdate();
            }
            System.out.println("Procedure executed successfully");
        }catch(SQLException e){
            System.out.println("Error executing procedure:
"+e.getMessage());
        }

    }
}
```

```
create table customers(  
customer_id int primary key,  
name varchar(100),  
email varchar(100),  
registration date);
```

```
select * from customers
```

```
create index idx_customer_reg_Date on customers(registration);  
update customer set registration='2024-0-20'
```

```
set session join_buffer_size=4194304;  
explain select o.orderId,o.orderDate,o.TotalAmount,c.Name  
from orders o  
join customers c use index(idx_customer_reg_Date) on o.customer_id =c.customer_id  
where c.registration>= DATE_SUB(CURDATE(),INTERVAL 30 day);
```

- Populate Customers table

```
INSERT INTO Customers (Customer_ID, Name, Email, registration)  
VALUES  
(1, 'John Doe', 'john.doe@email.com', '2024-09-15'),  
(2, 'Jane Smith', 'jane.smith@email.com', '2024-09-20'),  
(3, 'Bob Johnson', 'bob.johnson@email.com', '2024-09-25'),  
(4, 'Alice Brown', 'alice.brown@email.com', '2024-09-30'),  
(5, 'Charlie Davis', 'charlie.davis@email.com', '2024-10-01'),  
(6, 'Eva Wilson', 'eva.wilson@email.com', '2024-10-02'),  
(7, 'Frank Miller', 'frank.miller@email.com', '2024-10-03'),  
(8, 'Grace Lee', 'grace.lee@email.com', '2024-10-04'),  
(9, 'Henry Taylor', 'henry.taylor@email.com', '2024-10-05'),  
(10, 'Ivy Clark', 'ivy.clark@email.com', '2024-10-06');
```

-- Insert 90 more customers with registration dates spread over the last 60 days

```
INSERT INTO Customers (Customer_ID, Name, Email, registration)  
SELECT  
10 + num,  
CONCAT('Customer', num),  
CONCAT('customer', num, '@email.com'),  
DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND() * 60) DAY)  
FROM (  
SELECT a.N + b.N * 10 + 1 as num  
FROM
```

```
(SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT
4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT
9) a,
```

```
(SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT
4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8) b
```

```
ORDER BY num
```

```
LIMIT 90
```

```
) numbers;
```

```
create table orders(orderid int primary key,
```

```
customer_id int,
```

```
orderdate date,
```

```
totalamount decimal(10,2),
```

```
foreign key(customer_id) references customers(customer_id));
```

```
-- Populate Orders table
```

```
INSERT INTO Orders (OrderID, customer_id, OrderDate, TotalAmount)
```

```
VALUES
```

```
(1, 1, '2024-09-16', 150.00),
```

```
(2, 2, '2024-09-21', 200.50),
```

```
(3, 3, '2024-09-26', 75.25),
```

```
(4, 4, '2024-10-01', 300.75),
```

```
(5, 5, '2024-10-02', 50.00),
```

```
(6, 6, '2024-10-03', 125.50),
```

```
(7, 7, '2024-10-04', 80.00),
```

```
(8, 8, '2024-10-05', 220.25),
```

```
(9, 9, '2024-10-06', 175.00),
```

```
(10, 10, '2024-10-07', 90.50);
```

```
-- Insert 190 more orders with random customers and dates
```

```
INSERT INTO Orders (OrderID, Customer_ID, OrderDate, TotalAmount)
```

```
SELECT
```

```
10 + num,
```

```
1 + FLOOR(RAND() * 100),
```

```
DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND() * 30) DAY),
```

```
ROUND(50 + RAND() * 450, 2)
```

```
FROM (
```

```
SELECT a.N + b.N * 10 + c.N * 100 + 1 AS num
```

```
FROM
```

```
(SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT
4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT
9) a,
```

```
(SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT
4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT
9) b,
(SELECT 0 AS N UNION SELECT 1) c
ORDER BY num
LIMIT 190
) numbers;
```

## # 7 Dimensional Modeling Use Cases: From Normalized to Optimized

### ## Use Case 1: E-commerce Order Analytics

#### ### Problem Statement:

An e-commerce platform with 10 million orders needs to analyze order trends, product performance, and customer behavior.

#### ### Normalized Schema:

sql

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Address VARCHAR(200)
);
```

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    Name VARCHAR(100),
    Category VARCHAR(50),
    Price DECIMAL(10, 2)
);
```

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

```
CREATE TABLE OrderItems (
    OrderItemID INT PRIMARY KEY,
    OrderID INT,
    ProductID INT,
    Quantity INT,
```

```
FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),  
FOREIGN KEY (ProductID) REFERENCES Products(ProductID)  
);
```

### Slow Query:

```
sql  
SELECT p.Category, YEAR(o.OrderDate) AS Year, SUM(p.Price * oi.Quantity) AS TotalSales  
FROM Orders o  
JOIN OrderItems oi ON o.OrderID = oi.OrderID  
JOIN Products p ON oi.ProductID = p.ProductID  
GROUP BY p.Category, YEAR(o.OrderDate)  
ORDER BY Year, TotalSales DESC;
```

### Star Schema Solution:

```
sql  
CREATE TABLE DimDate (  
    DateID INT PRIMARY KEY,  
    Date DATE,  
    Year INT,  
    Month INT  
);
```

```
CREATE TABLE DimProduct (  
    ProductID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Category VARCHAR(50)  
);
```

```
CREATE TABLE DimCustomer (  
    CustomerID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Email VARCHAR(100)  
);
```

```
CREATE TABLE FactSales (  
    SalesID INT PRIMARY KEY,  
    DateID INT,  
    ProductID INT,  
    CustomerID INT,  
    Quantity INT,  
    TotalAmount DECIMAL(10, 2),  
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),
```



```
FOREIGN KEY (ProductID) REFERENCES DimProduct(ProductID),  
FOREIGN KEY (CustomerID) REFERENCES DimCustomer(CustomerID)  
);
```

### Optimized Query:

```
sql  
SELECT dp.Category, dd.Year, SUM(fs.TotalAmount) AS TotalSales  
FROM FactSales fs  
JOIN DimProduct dp ON fs.ProductID = dp.ProductID  
JOIN DimDate dd ON fs.DateID = dd.DateID  
GROUP BY dp.Category, dd.Year  
ORDER BY dd.Year, TotalSales DESC;
```

## Use Case 2: Healthcare Patient Visits

### Problem Statement:

A healthcare system with 7 million patient visits needs to analyze visit patterns, diagnoses, and treatment outcomes.

### Normalized Schema:

```
sql  
CREATE TABLE Patients (  
    PatientID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    DateOfBirth DATE,  
    Gender VARCHAR(10)  
);
```

```
CREATE TABLE Doctors (  
    DoctorID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Specialty VARCHAR(50)  
);
```

```
CREATE TABLE Diagnoses (  
    DiagnosisID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Category VARCHAR(50)  
);
```

```
CREATE TABLE Visits (  
    VisitID INT PRIMARY KEY,
```

```

PatientID INT,
DoctorID INT,
VisitDate DATE,
FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);

```

```

CREATE TABLE VisitDiagnoses (
    VisitDiagnosisID INT PRIMARY KEY,
    VisitID INT,
    DiagnosisID INT,
    FOREIGN KEY (VisitID) REFERENCES Visits(VisitID),
    FOREIGN KEY (DiagnosisID) REFERENCES Diagnoses(DiagnosisID)
);

```

#### Slow Query:

```

sql
SELECT d.Specialty, di.Category, YEAR(v.VisitDate) AS Year, COUNT(*) AS VisitCount
FROM Visits v
JOIN Doctors d ON v.DoctorID = d.DoctorID
JOIN VisitDiagnoses vd ON v.VisitID = vd.VisitID
JOIN Diagnoses di ON vd.DiagnosisID = di.DiagnosisID
GROUP BY d.Specialty, di.Category, YEAR(v.VisitDate)
ORDER BY Year, VisitCount DESC;

```

#### Snowflake Schema Solution:

```

sql
CREATE TABLE DimDate (
    DateID INT PRIMARY KEY,
    Date DATE,
    Year INT,
    Month INT
);

```

```

CREATE TABLE DimSpecialty (
    SpecialtyID INT PRIMARY KEY,
    SpecialtyName VARCHAR(50)
);

```

```

CREATE TABLE DimDoctor (
    DoctorID INT PRIMARY KEY,
    Name VARCHAR(100),

```

```
SpecialtyID INT,  
FOREIGN KEY (SpecialtyID) REFERENCES DimSpecialty(SpecialtyID)  
);
```

```
CREATE TABLE DimDiagnosisCategory (  
    CategoryID INT PRIMARY KEY,  
    CategoryName VARCHAR(50)  
);
```

```
CREATE TABLE DimDiagnosis (  
    DiagnosisID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    CategoryID INT,  
    FOREIGN KEY (CategoryID) REFERENCES DimDiagnosisCategory(CategoryID)  
);
```

```
CREATE TABLE FactVisits (  
    VisitID INT PRIMARY KEY,  
    DateID INT,  
    DoctorID INT,  
    DiagnosisID INT,  
    PatientID INT,  
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),  
    FOREIGN KEY (DoctorID) REFERENCES DimDoctor(DoctorID),  
    FOREIGN KEY (DiagnosisID) REFERENCES DimDiagnosis(DiagnosisID)  
);
```

### Optimized Query:

sql

```
SELECT ds.SpecialtyName, dc.CategoryName, dd.Year, COUNT(*) AS VisitCount  
FROM FactVisits fv  
JOIN DimDate dd ON fv.DateID = dd.DateID  
JOIN DimDoctor doc ON fv.DoctorID = doc.DoctorID  
JOIN DimSpecialty ds ON doc.SpecialtyID = ds.SpecialtyID  
JOIN DimDiagnosis diag ON fv.DiagnosisID = diag.DiagnosisID  
JOIN DimDiagnosisCategory dc ON diag.CategoryID = dc.CategoryID  
GROUP BY ds.SpecialtyName, dc.CategoryName, dd.Year  
ORDER BY dd.Year, VisitCount DESC;
```

## Use Case 3: Financial Transactions Analysis

### Problem Statement:

A bank with 8 million monthly transactions needs to analyze transaction patterns, customer segments, and product performance.

### Normalized Schema:

sql

```
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Segment VARCHAR(50)  
);  
  
CREATE TABLE Accounts (  
    AccountID INT PRIMARY KEY,  
    CustomerID INT,  
    AccountType VARCHAR(50),  
    Balance DECIMAL(15, 2),  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);
```

```
CREATE TABLE Transactions (  
    TransactionID INT PRIMARY KEY,  
    AccountID INT,  
    TransactionDate DATE,  
    Amount DECIMAL(15, 2),  
    TransactionType VARCHAR(50),  
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)  
);
```

### Slow Query:

sql

```
SELECT c.Segment, a.AccountType, YEAR(t.TransactionDate) AS Year,  
       SUM(CASE WHEN t.TransactionType = 'Deposit' THEN t.Amount ELSE 0 END) AS  
TotalDeposits,  
       SUM(CASE WHEN t.TransactionType = 'Withdrawal' THEN t.Amount ELSE 0 END) AS  
TotalWithdrawals  
FROM Transactions t  
JOIN Accounts a ON t.AccountID = a.AccountID  
JOIN Customers c ON a.CustomerID = c.CustomerID  
GROUP BY c.Segment, a.AccountType, YEAR(t.TransactionDate)  
ORDER BY Year, c.Segment, a.AccountType;
```

### Galaxy Schema Solution:

sql

```
CREATE TABLE DimDate (  
    DateID INT PRIMARY KEY,  
    Date DATE,  
    Year INT,  
    Month INT  
);
```

```
CREATE TABLE DimCustomer (  
    CustomerID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Segment VARCHAR(50)  
);
```

```
CREATE TABLE DimAccount (  
    AccountID INT PRIMARY KEY,  
    AccountType VARCHAR(50)  
);
```

```
CREATE TABLE FactTransactions (  
    TransactionID INT PRIMARY KEY,  
    DateID INT,  
    CustomerID INT,  
    AccountID INT,  
    Amount DECIMAL(15, 2),  
    TransactionType VARCHAR(50),  
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),  
    FOREIGN KEY (CustomerID) REFERENCES DimCustomer(CustomerID),  
    FOREIGN KEY (AccountID) REFERENCES DimAccount(AccountID)  
);
```

```
CREATE TABLE FactAccountBalances (  
    BalanceID INT PRIMARY KEY,  
    DateID INT,  
    AccountID INT,  
    Balance DECIMAL(15, 2),  
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),  
    FOREIGN KEY (AccountID) REFERENCES DimAccount(AccountID)  
);
```

### Optimized Query:

```
sql  
SELECT dc.Segment, da.AccountType, dd.Year,
```

```

SUM(CASE WHEN ft.TransactionType = 'Deposit' THEN ft.Amount ELSE 0 END) AS
TotalDeposits,
SUM(CASE WHEN ft.TransactionType = 'Withdrawal' THEN ft.Amount ELSE 0 END) AS
TotalWithdrawals
FROM FactTransactions ft
JOIN DimDate dd ON ft.DateID = dd.DateID
JOIN DimCustomer dc ON ft.CustomerID = dc.CustomerID
JOIN DimAccount da ON ft.AccountID = da.AccountID
GROUP BY dc.Segment, da.AccountType, dd.Year
ORDER BY dd.Year, dc.Segment, da.AccountType;

```

## ## Use Case 4: Telecom Network Performance

### ### Problem Statement:

A telecom company with 6 million daily network events needs to analyze network performance, user behavior, and service quality.

### ### Normalized Schema:

sql

```

CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Name VARCHAR(100),
    PlanType VARCHAR(50)
);

```

```

CREATE TABLE Towers (
    TowerID INT PRIMARY KEY,
    Location VARCHAR(100),
    Capacity INT
);

```

```

CREATE TABLE Services (
    ServiceID INT PRIMARY KEY,
    ServiceName VARCHAR(50),
    ServiceType VARCHAR(50)
);

```

```

CREATE TABLE NetworkEvents (
    EventID INT PRIMARY KEY,
    UserID INT,
    TowerID INT,
    ServiceID INT,
    EventTime DATETIME,

```

```

    Duration INT,
    DataUsed INT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (TowerID) REFERENCES Towers(TowerID),
    FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID)
);

```

### Slow Query:

```

sql
SELECT t.Location, s.ServiceType, DATEPART(HOUR, ne.EventTime) AS Hour,
       AVG(ne.Duration) AS AvgDuration, SUM(ne.DataUsed) AS TotalDataUsed
FROM NetworkEvents ne
JOIN Towers t ON ne.TowerID = t.TowerID
JOIN Services s ON ne.ServiceID = s.ServiceID
WHERE ne.EventTime >= DATEADD(day, -7, GETDATE())
GROUP BY t.Location, s.ServiceType, DATEPART(HOUR, ne.EventTime)
ORDER BY t.Location, s.ServiceType, Hour;

```

### Star Schema Solution:

```

sql
CREATE TABLE DimDate (
    DateID INT PRIMARY KEY,
    Date DATE,
    Year INT,
    Month INT,
    Day INT,
    Hour INT
);

```

```

CREATE TABLE DimTower (
    TowerID INT PRIMARY KEY,
    Location VARCHAR(100),
    Capacity INT
);

```

```

CREATE TABLE DimService (
    ServiceID INT PRIMARY KEY,
    ServiceName VARCHAR(50),
    ServiceType VARCHAR(50)
);

```

```

CREATE TABLE DimUser (

```

```

        UserID INT PRIMARY KEY,
        Name VARCHAR(100),
        PlanType VARCHAR(50)
    );

CREATE TABLE FactNetworkEvents (
    EventID INT PRIMARY KEY,
    DateID INT,
    TowerID INT,
    ServiceID INT,
    UserID INT,
    Duration INT,
    DataUsed INT,
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),
    FOREIGN KEY (TowerID) REFERENCES DimTower(TowerID),
    FOREIGN KEY (ServiceID) REFERENCES DimService(ServiceID),
    FOREIGN KEY (UserID) REFERENCES DimUser(UserID)
);

```

### Optimized Query:

```

sql
SELECT dt.Location, ds.ServiceType, dd.Hour,
       AVG(fne.Duration) AS AvgDuration, SUM(fne.DataUsed) AS TotalDataUsed
FROM FactNetworkEvents fne
JOIN DimDate dd ON fne.DateID = dd.DateID
JOIN DimTower dt ON fne.TowerID = dt.TowerID
JOIN DimService ds ON fne.ServiceID = ds.ServiceID
WHERE dd.Date >= DATEADD(day, -7, GETDATE())
GROUP BY dt.Location, ds.ServiceType, dd.Hour
ORDER BY dt.Location, ds.ServiceType, dd.Hour;

```

## Use Case 5: Retail Sales and Inventory Management

### Problem Statement:

A retail chain with 5 million monthly sales transactions needs to analyze sales trends, inventory levels, and store performance.

### Normalized Schema:

```

sql
CREATE TABLE Stores (
    StoreID INT PRIMARY KEY,
    Name VARCHAR(100),

```



```
    Location VARCHAR(100),  
    Size VARCHAR(20)  
);
```

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Category VARCHAR(50),  
    Supplier VARCHAR(100)  
);
```

```
CREATE TABLE Inventory (  
    InventoryID INT PRIMARY KEY,  
    StoreID INT,  
    ProductID INT,  
    Quantity INT,  
    LastUpdated DATE,  
    FOREIGN KEY (StoreID) REFERENCES Stores(StoreID),  
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)  
);
```

```
CREATE TABLE Sales (  
    SaleID INT PRIMARY KEY,  
    StoreID INT,  
    ProductID INT,  
    SaleDate DATE,  
    Quantity INT,  
    TotalAmount DECIMAL(10, 2),  
    FOREIGN KEY (StoreID) REFERENCES Stores(StoreID),  
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)  
);
```

### Slow Query:

sql

```
SELECT s.Location, p.Category, YEAR(sa.SaleDate) AS Year, MONTH(sa.SaleDate) AS  
Month,  
    SUM(sa.TotalAmount) AS TotalSales,  
    AVG(i.Quantity) AS AvgInventory  
FROM Sales sa  
JOIN Stores s ON sa.StoreID = s.StoreID  
JOIN Products p ON sa.ProductID = p.ProductID  
LEFT JOIN Inventory i ON sa.StoreID = i.StoreID AND sa.ProductID = i.ProductID  
GROUP BY s.Location, p.Category, YEAR(sa.SaleDate), MONTH(sa.SaleDate)
```

ORDER BY Year, Month, TotalSales DESC;

### Galaxy Schema Solution

[Previous content remains unchanged]

### Galaxy Schema Solution for Use Case 5:

sql

```
CREATE TABLE DimDate (  
    DateID INT PRIMARY KEY,  
    Date DATE,  
    Year INT,  
    Month INT  
);
```

```
CREATE TABLE DimStore (  
    StoreID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Location VARCHAR(100),  
    Size VARCHAR(20)  
);
```

```
CREATE TABLE DimProduct (  
    ProductID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Category VARCHAR(50),  
    Supplier VARCHAR(100)  
);
```

```
CREATE TABLE FactSales (  
    SaleID INT PRIMARY KEY,  
    DateID INT,  
    StoreID INT,  
    ProductID INT,  
    Quantity INT,  
    TotalAmount DECIMAL(10, 2),  
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),  
    FOREIGN KEY (StoreID) REFERENCES DimStore(StoreID),  
    FOREIGN KEY (ProductID) REFERENCES DimProduct(ProductID)  
);
```

```
CREATE TABLE FactInventory (  
    InventoryID INT PRIMARY KEY,  
    DateID INT,
```

```

StoreID INT,
ProductID INT,
Quantity INT,
FOREIGN KEY (DateID) REFERENCES DimDate(DateID),
FOREIGN KEY (StoreID) REFERENCES DimStore(StoreID),
FOREIGN KEY (ProductID) REFERENCES DimProduct(ProductID)
);

```

#### Optimized Query:

```

sql
SELECT ds.Location, dp.Category, dd.Year, dd.Month,
       SUM(fs.TotalAmount) AS TotalSales,
       AVG(fi.Quantity) AS AvgInventory
FROM FactSales fs
JOIN DimStore ds ON fs.StoreID = ds.StoreID
JOIN DimProduct dp ON fs.ProductID = dp.ProductID
JOIN DimDate dd ON fs.DateID = dd.DateID
LEFT JOIN FactInventory fi ON fs.StoreID = fi.StoreID
                        AND fs.ProductID = fi.ProductID
                        AND fs.DateID = fi.DateID
GROUP BY ds.Location, dp.Category, dd.Year, dd.Month
ORDER BY dd.Year, dd.Month, TotalSales DESC;

```

## Use Case 6: Social Media Engagement Analysis

#### Problem Statement:

A social media platform with 9 million daily user interactions needs to analyze user engagement, content performance, and advertising effectiveness.

#### Normalized Schema:

```

sql
CREATE TABLE Users (
  UserID INT PRIMARY KEY,
  Username VARCHAR(50),
  JoinDate DATE,
  Country VARCHAR(50)
);

```

```

CREATE TABLE Posts (
  PostID INT PRIMARY KEY,
  UserID INT,
  PostType VARCHAR(20),

```

```
PostDate DATETIME,  
Content TEXT,  
FOREIGN KEY (UserID) REFERENCES Users(UserID)  
);
```

```
CREATE TABLE Interactions (  
InteractionID INT PRIMARY KEY,  
UserID INT,  
PostID INT,  
InteractionType VARCHAR(20),  
InteractionDate DATETIME,  
FOREIGN KEY (UserID) REFERENCES Users(UserID),  
FOREIGN KEY (PostID) REFERENCES Posts(PostID)  
);
```

```
CREATE TABLE Ads (  
AdID INT PRIMARY KEY,  
AdvertiserID INT,  
AdContent TEXT,  
StartDate DATE,  
EndDate DATE  
);
```

```
CREATE TABLE AdImpressions (  
ImpressionID INT PRIMARY KEY,  
AdID INT,  
UserID INT,  
ImpressionDate DATETIME,  
Clicked BIT,  
FOREIGN KEY (AdID) REFERENCES Ads(AdID),  
FOREIGN KEY (UserID) REFERENCES Users(UserID)  
);
```

### Slow Query:

sql

```
SELECT u.Country, p.PostType,  
DATEPART(HOUR, i.InteractionDate) AS Hour,  
COUNT(DISTINCT u.UserID) AS UniqueUsers,  
COUNT(*) AS TotalInteractions,  
SUM(CASE WHEN ai.Clicked = 1 THEN 1 ELSE 0 END) AS AdClicks  
FROM Interactions i  
JOIN Users u ON i.UserID = u.UserID  
JOIN Posts p ON i.PostID = p.PostID
```

```
LEFT JOIN AdImpressions ai ON u.UserID = ai.UserID
    AND CAST(i.InteractionDate AS DATE) = CAST(ai.ImpressionDate AS DATE)
WHERE i.InteractionDate >= DATEADD(day, -30, GETDATE())
GROUP BY u.Country, p.PostType, DATEPART(HOUR, i.InteractionDate)
ORDER BY u.Country, p.PostType, Hour;
```

### Snowflake Schema Solution:

sql

```
CREATE TABLE DimDate (
    DateID INT PRIMARY KEY,
    Date DATE,
    Year INT,
    Month INT,
    Day INT,
    Hour INT
);
```

```
CREATE TABLE DimUser (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50),
    JoinDate DATE
);
```

```
CREATE TABLE DimCountry (
    CountryID INT PRIMARY KEY,
    CountryName VARCHAR(50)
);
```

```
CREATE TABLE DimPost (
    PostID INT PRIMARY KEY,
    PostType VARCHAR(20)
);
```

```
CREATE TABLE DimInteractionType (
    InteractionTypeID INT PRIMARY KEY,
    InteractionTypeName VARCHAR(20)
);
```

```
CREATE TABLE DimAd (
    AdID INT PRIMARY KEY,
    AdvertiserID INT,
    StartDate DATE,
    EndDate DATE
```

);

```
CREATE TABLE FactInteractions (  
    InteractionID INT PRIMARY KEY,  
    DateID INT,  
    UserID INT,  
    CountryID INT,  
    PostID INT,  
    InteractionTypeID INT,  
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),  
    FOREIGN KEY (UserID) REFERENCES DimUser(UserID),  
    FOREIGN KEY (CountryID) REFERENCES DimCountry(CountryID),  
    FOREIGN KEY (PostID) REFERENCES DimPost(PostID),  
    FOREIGN KEY (InteractionTypeID) REFERENCES DimInteractionType(InteractionTypeID)  
);
```

```
CREATE TABLE FactAdImpressions (  
    ImpressionID INT PRIMARY KEY,  
    DateID INT,  
    UserID INT,  
    CountryID INT,  
    AdID INT,  
    Clicked BIT,  
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),  
    FOREIGN KEY (UserID) REFERENCES DimUser(UserID),  
    FOREIGN KEY (CountryID) REFERENCES DimCountry(CountryID),  
    FOREIGN KEY (AdID) REFERENCES DimAd(AdID)  
);
```

### Optimized Query:

sql

```
SELECT dc.CountryName, dp.PostType, dd.Hour,  
       COUNT(DISTINCT fi.UserID) AS UniqueUsers,  
       COUNT(*) AS TotalInteractions,  
       SUM(fai.Clicked) AS AdClicks  
FROM FactInteractions fi  
JOIN DimDate dd ON fi.DateID = dd.DateID  
JOIN DimUser du ON fi.UserID = du.UserID  
JOIN DimCountry dc ON fi.CountryID = dc.CountryID  
JOIN DimPost dp ON fi.PostID = dp.PostID  
LEFT JOIN FactAdImpressions fai ON fi.UserID = fai.UserID AND fi.DateID = fai.DateID  
WHERE dd.Date >= DATEADD(day, -30, GETDATE())  
GROUP BY dc.CountryName, dp.PostType, dd.Hour
```

ORDER BY dc.CountryName, dp.PostType, dd.Hour;

## ## Use Case 7: IoT Sensor Data Analysis

### ### Problem Statement:

An IoT platform collecting data from 10 million sensors needs to analyze sensor performance, environmental conditions, and anomaly detection.

### ### Normalized Schema:

sql

```
CREATE TABLE Devices (  
    DeviceID INT PRIMARY KEY,  
    DeviceType VARCHAR(50),  
    Location VARCHAR(100),  
    InstallationDate DATE  
);
```

```
CREATE TABLE Sensors (  
    SensorID INT PRIMARY KEY,  
    DeviceID INT,  
    SensorType VARCHAR(50),  
    FOREIGN KEY (DeviceID) REFERENCES Devices(DeviceID)  
);
```

```
CREATE TABLE Readings (  
    ReadingID INT PRIMARY KEY,  
    SensorID INT,  
    ReadingTime DATETIME,  
    Value FLOAT,  
    FOREIGN KEY (SensorID) REFERENCES Sensors(SensorID)  
);
```

```
CREATE TABLE Alerts (  
    AlertID INT PRIMARY KEY,  
    SensorID INT,  
    AlertTime DATETIME,  
    AlertType VARCHAR(50),  
    Severity INT,  
    FOREIGN KEY (SensorID) REFERENCES Sensors(SensorID)  
);
```

### ### Slow Query:

```

sql
SELECT d.DeviceType, s.SensorType,
       DATEPART(HOUR, r.ReadingTime) AS Hour,
       AVG(r.Value) AS AvgReading,
       COUNT(DISTINCT a.AlertID) AS AlertCount
FROM Readings r
JOIN Sensors s ON r.SensorID = s.SensorID
JOIN Devices d ON s.DeviceID = d.DeviceID
LEFT JOIN Alerts a ON s.SensorID = a.SensorID
    AND CAST(r.ReadingTime AS DATE) = CAST(a.AlertTime AS DATE)
WHERE r.ReadingTime >= DATEADD(day, -7, GETDATE())
GROUP BY d.DeviceType, s.SensorType, DATEPART(HOUR, r.ReadingTime)
ORDER BY d.DeviceType, s.SensorType, Hour;

```

### Star Schema Solution:

```

sql
CREATE TABLE DimDate (
    DateID INT PRIMARY KEY,
    Date DATE,
    Year INT,
    Month INT,
    Day INT,
    Hour INT
);

CREATE TABLE DimDevice (
    DeviceID INT PRIMARY KEY,
    DeviceType VARCHAR(50),
    Location VARCHAR(100),
    InstallationDate DATE
);

CREATE TABLE DimSensor (
    SensorID INT PRIMARY KEY,
    DeviceID INT,
    SensorType VARCHAR(50),
    FOREIGN KEY (DeviceID) REFERENCES DimDevice(DeviceID)
);

CREATE TABLE FactReadings (
    ReadingID INT PRIMARY KEY,
    DateID INT,
    SensorID INT,

```



```

    Value FLOAT,
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),
    FOREIGN KEY (SensorID) REFERENCES DimSensor(SensorID)
);

```

```

CREATE TABLE FactAlerts (
    AlertID INT PRIMARY KEY,
    DateID INT,
    SensorID INT,
    AlertType VARCHAR(50),
    Severity INT,
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),
    FOREIGN KEY (SensorID) REFERENCES DimSensor(SensorID)
);

```

### Optimized Query:

```

sql
SELECT dd.DeviceType, ds.SensorType, dd2.Hour,
       AVG(fr.Value) AS AvgReading,
       COUNT(DISTINCT fa.AlertID) AS AlertCount
FROM FactReadings fr
JOIN DimSensor ds ON fr.SensorID = ds.SensorID
JOIN DimDevice dd ON ds.DeviceID = dd.DeviceID
JOIN DimDate dd2 ON fr.DateID = dd2.DateID
LEFT JOIN FactAlerts fa ON fr.SensorID = fa.SensorID AND fr.DateID = fa.DateID
WHERE dd2.Date >= DATEADD(day, -7, GETDATE())
GROUP BY dd.DeviceType, ds.SensorType, dd2.Hour
ORDER BY dd.DeviceType, ds.SensorType, dd2.Hour;

```

In each of these use cases, we've demonstrated how transitioning from a normalized schema to a dimensional model (Star, Snowflake, or Galaxy) can significantly improve query performance, especially for analytical workloads on large datasets. The dimensional models reduce the number of joins required, pre-aggregate data where possible, and organize information in a way that's optimized for complex analytical queries.

Key benefits observed across these examples include:

1. Reduced join complexity
2. Improved query performance for analytical workloads
3. Better support for historical analysis through slowly changing dimensions
4. Easier addition of new analytical requirements
5. Improved data consistency and easier maintenance of business rules

Remember that the choice between Star, Snowflake, and Galaxy schemas depends on the specific requirements of each use case, including data volume, query patterns, and the need for flexibility in future expansions of the data model.

// java program

```
package com.example;

import java.sql.*;

public class Main {
    private static final String
url="jdbc:mysql://localhost:3306/dbProject";
    private static final String username="root";
    private static final String password="root";
    private static Connection connection;
    public static void main(String[] args) {
        System.out.println("for normalized schema trying to insert 1
million records in all the present tables");
        try{
            connection=DriverManager.getConnection(url, username,
password);
            PreparedStatement
customer_statement=connection.prepareStatement("INSERT INTO Customers
(CustomerID,Name,Email,Address)VALUES (?, ?, ?, ?) ");
            PreparedStatement
product_statement=connection.prepareStatement("INSERT INTO
Products (ProductID,Name,Category,Price)VALUES (?, ?, ?, ?) ");
            PreparedStatement
orders_statement=connection.prepareStatement("INSERT INTO
Orders (OrderID,CustomerID,OrderDate)VALUES (?, ?, ?) ");
            PreparedStatement
orderItems_statement=connection.prepareStatement("INSERT INTO
orderItems (OrderItemID,OrderID,ProductID,Quantity)VALUES (?, ?, ?, ?) ");

            for(int i=0; i<1000000; i++){
                customer_statement.setInt(1,i);
                customer_statement.setString(2, "name"+i);
                customer_statement.setString(3,"email"+i);
                customer_statement.setString(4,"adress"+i);
```

```

        customer_statement.executeUpdate();
        product_statement.setInt(1,i);
        product_statement.setString(2,"product"+i);
        product_statement.setString(3,"category"+i);
        product_statement.setDouble(4,10.0+i);
        product_statement.executeUpdate();
        orders_statement.setInt(1,i);
        orders_statement.setInt(2,i);
        orders_statement.setDate(3, new
java.sql.Date(System.currentTimeMillis()));
        orders_statement.executeUpdate();
        orderItems_statement.setInt(1,i);
        orderItems_statement.setInt(2,i);
        orderItems_statement.setInt(3,i);
        orderItems_statement.setInt(4,(i+2));
        orderItems_statement.executeUpdate();

    }
    System.out.println("insertion of 1 million records are
successful");

    }catch(SQLException e){
        e.printStackTrace();
    }
    System.out.println("trying to execute slow query");
    long startTimeSlow=System.nanoTime();
    try{
        String query="SELECT p.Category, YEAR(o.OrderDate) AS Year,
SUM(p.Price * oi.Quantity) AS TotalSales\n" + //
        "FROM Orders o\n" + //
        "JOIN OrderItems oi ON o.OrderID =
oi.OrderID\n" + //
        "JOIN Products p ON oi.ProductID =
p.ProductID\n" + //
        "GROUP BY p.Category, YEAR(o.OrderDate)\n"
+ //
        "ORDER BY Year, TotalSales DESC;";

        PreparedStatement
slowQuery=connection.prepareStatement(query);
        ResultSet result=slowQuery.executeQuery();

```

```

        long endTimeSlow=System.nanoTime();
        System.out.println("time taken to execute slow query is
"+(endTimeSlow-startTimeSlow)/100);
    }catch(SQLException e){
        e.printStackTrace();
    }
    System.out.println("for star schema trying to insert 1 million
records in each table");
    try{
        PreparedStatement dimDate=connection.prepareStatement("INSERT
INTO DimDate(DateID,Date,Year,Month)VALUES(?,?,?,?)");
        PreparedStatement
dimProduct=connection.prepareStatement("INSERT INTO
DimProduct(ProductID,Name,Category)VALUES(?,?,?)");
        PreparedStatement
dimCustomer=connection.prepareStatement("INSERT INTO
DimCustomer(CustomerID,Name,Email)VALUES(?,?,?)");
        PreparedStatement
factSales=connection.prepareStatement("INSERT INTO
FactSales(SalesID,DateID,ProductID,CustomerID,Quantity,TotalAmount)VALUES (
?,?,?,?,?,?,?)");

        for(int i=0; i<1000000; i++){
            dimDate.setInt(1,i);
            dimDate.setDate(2,new
java.sql.Date(System.currentTimeMillis()));
            dimDate.setInt(3,2024);
            dimDate.setInt(4,8);
            dimDate.executeUpdate();
            dimProduct.setInt(1,i);
            dimProduct.setString(2,"product"+i);
            dimProduct.setString(3,"category"+i);
            dimProduct.executeUpdate();
            dimCustomer.setInt(1,i);
            dimCustomer.setString(2,"customer"+i);
            dimCustomer.setString(3,"customer"+i+"@gmail.com");
            dimCustomer.executeUpdate();
            factSales.setInt(1,i);
            factSales.setInt(2,i);
            factSales.setInt(3,i);

```

```

        factSales.setInt(4,i);
        factSales.setInt(5,10);
        factSales.setDouble(6,100.0);
        factSales.executeUpdate();
    }
    System.out.println("insertion of 1 million records are
successful");
} catch (SQLException e) {
    e.printStackTrace();
}
System.out.println("for star schema trying to execute fast
query");
long startTimeFast=System.nanoTime();
try{
    String query="SELECT dp.Category, dd.Year, SUM(fs.TotalAmount)
AS TotalSales FROM FactSales fs JOIN DimProduct dp ON fs.ProductID =
dp.ProductID JOIN DimDate dd ON fs.DateID = dd.DateID GROUP BY
dp.Category, dd.Year ORDER BY dd.Year, TotalSales DESC;";
    PreparedStatement
fastQuery=connection.prepareStatement(query);
    ResultSet result=fastQuery.executeQuery();
    long endTimeFast=System.nanoTime();
    System.out.println("time taken to execute fast query is
"+(endTimeFast-startTimeFast)/100);
} catch (SQLException e) {
    e.printStackTrace();
}
System.out.println("Inserting 1 million records in normalized
schema for socia media platform");
try{
    PreparedStatement Users = connection.prepareStatement("INSERT
INTO Users (UserID,Username,JoinDate,Country)VALUES(?,?,?,?)");
    PreparedStatement Posts = connection.prepareStatement("INSERT
INTO Posts (PostID,UserID,PostType,PostDate,Content)VALUES(?,?,?, ?,?)");
    PreparedStatement
Interactions=connection.prepareStatement("INERT INTO
Interactions(InteractionID,UserID,PostID,InteractionType,InteractionDate)V
ALUES(?,?, ?, ?, ?)");
    PreparedStatement Ads=connection.prepareStatement("INSERT INTO
Ads (AdID,AdvertiserID,AdContent,StartDate,EndDate)VALUES(?,?, ?, ?, ?)");

```

```

        PreparedStatement
AdImpressions=connection.prepareStatement("INSERT INTO
AdImpressions (ImpressionID,AdID,UserID,ImpressionDate,Clicked)VALUES (?, ?, ?
, ?, ?)");

        for(int i=0; i<10000000; i++){
            Users.setInt(1,i);
            Users.setString(2,"user"+i);
            Users.setDate(3,new
java.sql.Date(System.currentTimeMillis()));
            Users.setString(4,"country"+i);
            Users.executeUpdate();
            Posts.setInt(1,i);
            Posts.setInt(2,i);
            Posts.setString(3,"post"+i);
            Posts.setDate(4,new
java.sql.Date(System.currentTimeMillis()));
            Posts.setString(5,"content"+i);
            Posts.executeUpdate();
            Interactions.setInt(1,i);
            Interactions.setInt(2,i);
            Interactions.setInt(3,i);
            Interactions.setString(4,"interaction"+i);
            Interactions.setDate(5,new
java.sql.Date(System.currentTimeMillis()));
            Interactions.executeUpdate();
            Ads.setInt(1,i);
            Ads.setInt(2,i);
            Ads.setString(3,"ad"+i);
            Ads.setDate(4,new
java.sql.Date(System.currentTimeMillis()));
            Ads.setDate(5,new
java.sql.Date(System.currentTimeMillis()));
            Ads.executeUpdate();
            AdImpressions.setInt(1,i);
            AdImpressions.setInt(2,i);
            AdImpressions.setInt(3,i);
            AdImpressions.setDate(4,new
java.sql.Date(System.currentTimeMillis()));
            AdImpressions.setInt(5,1);

```

```

        AdImpressions.executeUpdate();
    }
    System.out.println("Inserted 1 million records in normalized
ecommerce schema successfully");
} catch (SQLException e) {
    e.printStackTrace();
}
System.out.println("Trying to execute slow query for above
ecommerce schema");
long eCommerceStartSlow=System.nanoTime();
try{
    PreparedStatement query = connection.prepareStatement("SELECT
u.Country, p.PostType,DATEPART(HOUR, i.InteractionDate) AS
Hour,COUNT(DISTINCT u.UserID) AS UniqueUsers, COUNT(*) AS
TotalInteractions, SUM(CASE WHEN ai.Clicked = 1 THEN 1 ELSE 0 END) AS
AdClicks FROM Interactions i JOIN Users u ON i.UserID = u.UserID JOIN
Posts p ON i.PostID = p.PostID LEFT JOIN AdImpressions ai ON u.UserID =
ai.UserID AND CAST(i.InteractionDate AS DATE) = CAST(ai.ImpressionDate AS
DATE) WHERE i.InteractionDate >= DATEADD(day, -30, GETDATE()) GROUP BY
u.Country, p.PostType, DATEPART(HOUR, i.InteractionDate) ORDER BY
u.Country, p.PostType, Hour");
    ResultSet resultSet = query.executeQuery();
    long ecommerceEndSlow=System.nanoTime();
    System.out.println("Time taken to execute slow query for
ecommerce schema: "+(ecommerceEndSlow-eCommerceStartSlow));
} catch (SQLException e) {
    e.printStackTrace();
}
System.out.println("Trying to insert 1 million records in snowflake
schema of ecommerce ");
try{
    PreparedStatement DimnDate=connection.prepareStatement("INSERT
INTO DimnDate(DateID,Date,Year,Month,Day,Hour)VALUES(?,?,?,?,?,?)");
    PreparedStatement DimnUser=connection.prepareStatement("INSERT
INTO DimnUser(UserID,Username,JoinDate)VALUES(?,?,?,?)");
    PreparedStatement
DimnCountry=connection.prepareStatement("INSERT INTO
DimnCountry(CountryID,CountryName)VALUES(?,?)");
    PreparedStatement DimnPost=connection.prepareStatement("INSERT
INTO DimnPost(PostID,PostType)VALUES(?,?)");

```

```

        PreparedStatement
DimnInteractionType=connection.prepareStatement("INSERT INTO
DimnInteractionType (InteractionTypeID,InteractionTypeName) VALUES (?, ?)");
        PreparedStatement DimnAd=connection.prepareStatement("INSERT
INTO DimnAd(AdID,AdvertiserID,StartDate,EndDate) VALUES (?, ?, ?, ?)");
        PreparedStatement
FactInteractions=connection.prepareStatement("INSERT INTO
FactInteractions (InteractionID,DateID,UserID,CountryID,PostID,InteractionT
ypeID) VALUES (?, ?, ?, ?, ?, ?)");
        PreparedStatement
FactAdImpressions=connection.prepareStatement("INSERT INTO
FactAdImpressions (ImpressionID,DateID,UserID,CountryID,AdID,Clicked) VALUES
(?, ?, ?, ?, ?, ?)");
        for(int i=0; i<10000000; i++){
            DimnDate.setInt(1,i);
            DimnDate.setDate(2,new
java.sql.Date(System.currentTimeMillis()));
            DimnDate.setInt(3,2024);
            DimnDate.setInt(4,(i%12));
            DimnDate.setInt(5,(i%7));
            DimnDate.setInt(6,(i%24));
            DimnDate.executeUpdate();
            DimnUser.setInt(1,i);
            DimnUser.setString(2,"user"+i);
            DimnUser.setDate(3,new
java.sql.Date(System.currentTimeMillis()));
            DimnUser.executeUpdate();
            DimnCountry.setInt(1,i);
            DimnCountry.setString(2,"country"+i);
            DimnCountry.executeUpdate();
            DimnPost.setInt(1,i);
            DimnPost.setString(2,"post"+i);
            DimnPost.executeUpdate();
            DimnInteractionType.setInt(1,i);
            DimnInteractionType.setString(2,"interaction"+i);
            DimnInteractionType.executeUpdate();
            DimnAd.setInt(1,i);
            DimnAd.setInt(2,i);
            DimnAd.setDate(3,new
java.sql.Date(System.currentTimeMillis()));

```



```

        DimnAd.setDate(4,new
java.sql.Date(System.currentTimeMillis()));
        DimnAd.executeUpdate();
        FactInteractions.setInt(1,i);
        FactInteractions.setInt(2,i);
        FactInteractions.setInt(3,i);
        FactInteractions.setInt(4,i);
        FactInteractions.setInt(5,i);
        FactInteractions.setInt(6,i);
        FactInteractions.executeUpdate();
        FactAdImpressions.setInt(1,i);
        FactAdImpressions.setInt(2,i);
        FactAdImpressions.setInt(3,i);
        FactAdImpressions.setInt(4,i);
        FactAdImpressions.setInt(5,i);
        FactAdImpressions.setInt(6,0);
        FactAdImpressions.executeUpdate();
    }
    System.out.println("successfully inserted 1 million records
into snowflake schema of ecommerce application");
    }catch(SQLException e){
        System.out.println(e.getMessage());
    }
    System.out.println("Trying to execute fast query for above
snowflake schema of ecommerce database");
    long ecommerceStartFast=System.nanoTime();
    try{
        String query="SELECT dc.CountryName, dp.PostType, dd.Hour,
COUNT(DISTINCT fi.UserID) AS UniqueUsers,COUNT(*) AS
TotalInteractions,SUM(fai.Clicked) AS AdClicks FROM FactInteractions fi
JOIN DimnDate dd ON fi.DateID = dd.DateID JOIN DimnUser du ON fi.UserID =
du.UserID JOIN DimnCountry dc ON fi.CountryID = dc.CountryID JOIN DimnPost
dp ON fi.PostID = dp.PostID LEFT JOIN FactAdImpressions fai ON fi.UserID =
fai.UserID AND fi.DateID = fai.DateID WHERE dd.Date >= DATEADD(day, -30,
GETDATE()) GROUP BY dc.CountryName, dp.PostType, dd.Hour ORDER BY
dc.CountryName, dp.PostType, dd.Hour;";
        PreparedStatement
fastQuery=connection.prepareStatement(query);
        ResultSet resultSet=fastQuery.executeQuery();
        long ecommerceEndFast=System.nanoTime();

```

```

        long ecommerceFastTime=ecommerceEndFast-ecommerceStartFast;
        System.out.println("Time takes to execute fast query on the
snowflake schema of e commerce database is: "+ecommerceFastTime);
    }catch(SQLException e){
        e.printStackTrace();
    }
}
}

```

// SQL Triggers

```
create database trigger_practise;
```

```
use trigger_practise;
```

```
create table customers(id int auto_increment primary key,
name varchar(100),
email varchar(100));
```

```
create table email_changes_log(
id int auto_increment primary key,
customer_id int,
old_email varchar(100),
new_email varchar(100),
changed_at timestamp default current_timestamp);
```

```
insert into customers(name,email) values('Auahdahd','dqhduiqwh@gmail.com');
```

```
select * from customers;
```

```
DELIMITER //
```

```
CREATE TRIGGER log_email_changes
```

```
before update on customers
```

```
for each row
```

```
begin
```

```
    if old.email!=new.email then
```

```
        insert into email_changes_log(customer_id,old_email,new_email)
```

```
        values(old.id,old.email, new.email);
```

```
    end if;
```

```
end//
```

delimiter ;

```
select * from email_changes_log;  
update customers set email='jdiejweiod@gmail.com' where id =1  
select * from customers
```

## **SPRING**

# Understanding Dependency Injection and Inversion of Control in Spring

## 1. Introduction to Dependency Injection (DI)

Dependency Injection is a design pattern that allows us to develop loosely coupled code. It's a technique for achieving Inversion of Control (IoC) between classes and their dependencies.

### Step 1: Understanding the Problem DI Solves

Let's start with a simple example without DI:

```
java  
public class TextEditor {  
    private SpellChecker spellChecker;  
  
    public TextEditor() {  
        this.spellChecker = new SpellChecker();  
    }  
}
```

In this case, TextEditor is tightly coupled to SpellChecker. If we want to use a different spell checker, we'd have to modify the TextEditor class.

### Step 2: Applying Dependency Injection

Now, let's refactor this using DI:

```
java  
public class TextEditor {  
    private SpellChecker spellChecker;  
  
    public TextEditor(SpellChecker spellChecker) {  
        this.spellChecker = spellChecker;  
    }  
}
```

```
}  
}
```

Now, `TextEditor` is not responsible for creating the `SpellChecker`. It's "injected" from outside.

## ## 2. Types of Dependency Injection

Spring supports three types of dependency injection. Let's explore each one.

### ### Step 3: Constructor Injection

This is the most recommended form of DI in Spring.

```
java  
@Component  
public class TextEditor {  
    private final SpellChecker spellChecker;  
  
    @Autowired  
    public TextEditor(SpellChecker spellChecker) {  
        this.spellChecker = spellChecker;  
    }  
}
```

Spring will automatically inject a `SpellChecker` bean when creating a `TextEditor`.

### ### Step 4: Setter Injection

Setter injection uses, well, setter methods:

```
java  
@Component  
public class TextEditor {  
    private SpellChecker spellChecker;  
  
    @Autowired  
    public void setSpellChecker(SpellChecker spellChecker) {  
        this.spellChecker = spellChecker;  
    }  
}
```

#### ### Step 5: Field Injection

Field injection injects dependencies directly into fields:

```
java
@Component
public class TextEditor {
    @Autowired
    private SpellChecker spellChecker;
}
```

Note: While convenient, field injection is generally discouraged as it makes testing more difficult.

#### ## 3. Inversion of Control (IoC)

IoC is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework.

#### ### Step 6: Understanding IoC

In traditional programming, our custom code makes calls to a library. IoC is the exact opposite - the framework makes calls to our custom code.

#### ### Step 7: Spring IoC Container

Spring's IoC container is responsible for managing object creation, configuring these objects, and managing their lifecycle.

There are two types of IoC containers in Spring:

1. BeanFactory
2. ApplicationContext (a more advanced container and extension of BeanFactory)

#### ### Step 8: Configuring Spring IoC

We can configure the Spring IoC container using XML, Java annotations, or Java code. Let's look at an example using Java annotations:

```
java
@Configuration
public class AppConfig {
    @Bean
    public SpellChecker spellChecker() {
```

```

        return new SpellChecker();
    }

    @Bean
    public TextEditor textEditor(SpellChecker spellChecker) {
        return new TextEditor(spellChecker);
    }
}

```

This configuration tells Spring how to create and wire our objects.

## ## 4. Putting It All Together

Let's see how DI and IoC work together in a Spring application.

### ### Step 9: Creating the Application

```

java
@SpringBootApplication
public class TextEditorApplication {
    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(TextEditorApplication.class, args);
        TextEditor editor = context.getBean(TextEditor.class);
        editor.spellCheck("Hello, wrld!");
    }
}

```

In this example:

1. Spring Boot sets up the ApplicationContext (IoC container).
2. The container creates and configures all the beans.
3. We retrieve the TextEditor bean from the container.
4. We use the TextEditor, which internally uses the SpellChecker that was injected.

## ## 5. Benefits of DI and IoC

- Reduced dependency between classes
- Easier unit testing through mocking
- Greater modularity
- Increased code reusability
- More flexible, configurable applications

By leveraging DI and IoC, Spring allows us to write more modular, testable, and maintainable code.

Design Considerations: @Autowired vs getBean() in Spring

### 1. @Autowired

When to Use

In Spring-managed beans (e.g., @Component, @Service, @Repository, @Controller)

For dependencies that are required throughout the lifecycle of a bean

When you want to leverage Spring's dependency injection fully

### Design Advantages

Loose Coupling: @Autowired promotes loose coupling as the dependent class doesn't need to know about the creation and management of its dependencies.

javaCopy@Service

```
public class OrderService {
    private final PaymentService paymentService;

    @Autowired
    public OrderService(PaymentService paymentService) {
        this.paymentService = paymentService;
    }
}
```

Testability: It's easier to mock dependencies in unit tests.

javaCopy@Test

public v...

<https://leetcode.com/studyplan/top-sql-50/>

@Component

```
public class DependencyValidator {
    @Autowired(required = false)
    private OptionalService optionalService;
    @Autowired
    private RequiredService requiredService;

    @PostConstruct
    public void validateDependencies() {
        if (requiredService == null) {
            throw new IllegalStateException("RequiredService is not injected!");
        }
        if (optionalService == null) {
            System.out.println("OptionalService is not available, some features may be limited.");
        }
    }
}
```

```
}
```

## Key Differences from Constructor Initialization

Timing: `@PostConstruct` methods are called after all dependency injection is complete, whereas constructors are called during bean instantiation.

Access: `@PostConstruct` methods have full access to dependency-injected fields, which might not be initialized in the constructor.

Inheritance: `@PostConstruct` methods in superclasses are called before those in subclasses.

Exception Handling: `@PostConstruct` methods can throw checked exceptions, unlike constructors.

Certainly! I'll provide a detailed, step-by-step example demonstrating the use of both `@Autowired` and `getBean()` in Spring. We'll create a simple application that manages a library system.

### Step 1: Set up the project

1. Go to <https://start.spring.io/>
2. Choose:
  - Project: Maven
  - Language: Java
  - Spring Boot: 3.1.x
  - Group: com.example
  - Artifact: libraryapp
  - Packaging: Jar
  - Java: 17
3. Add Dependencies: Spring Web
4. Generate and download the project
5. Extract and open in your IDE

### Step 2: Create the Book model

Create a new package `com.example.libraryapp.model` and add a `Book` class:

```
java
package com.example.libraryapp.model;

public class Book {
    private String isbn;
    private String title;
    private String author;

    // Constructor, getters, and setters
}
```



### Step 3: Create the LibraryService

Create a package com.example.libraryapp.service and add:

```
java
package com.example.libraryapp.service;

import com.example.libraryapp.model.Book;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;

@Service
public class LibraryService {
    private Map<String, Book> books = new HashMap<>();

    public void addBook(Book book) {
        books.put(book.getIsbn(), book);
    }

    public Book getBook(String isbn) {
        return books.get(isbn);
    }
}
```

### Step 4: Create BookController using @Autowired

Create a package com.example.libraryapp.controller and add:

```
java
package com.example.libraryapp.controller;

import com.example.libraryapp.model.Book;
import com.example.libraryapp.service.LibraryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/books")
public class BookController {
```

```

private final LibraryService libraryService;

@Autowired
public BookController(LibraryService libraryService) {
    this.libraryService = libraryService;
}

@PostMapping
public void addBook(@RequestBody Book book) {
    libraryService.addBook(book);
}

@GetMapping("/{isbn}")
public Book getBook(@PathVariable String isbn) {
    return libraryService.getBook(isbn);
}
}

```

Step 5: Create BeanRetrievalService using getBean()

Add a new service:

```

java
package com.example.libraryapp.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Service;

@Service
public class BeanRetrievalService {

    private final ApplicationContext context;

    @Autowired
    public BeanRetrievalService(ApplicationContext context) {
        this.context = context;
    }

    public <T> T getBean(Class<T> beanClass) {
        return context.getBean(beanClass);
    }
}

```

Step 6: Create AdminController using getBean()

Add another controller:

```
java
package com.example.libraryapp.controller;

import com.example.libraryapp.model.Book;
import com.example.libraryapp.service.BeanRetrievalService;
import com.example.libraryapp.service.LibraryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/admin")
public class AdminController {

    private final BeanRetrievalService beanRetrievalService;

    @Autowired
    public AdminController(BeanRetrievalService beanRetrievalService) {
        this.beanRetrievalService = beanRetrievalService;
    }

    @PostMapping("/books")
    public void addBook(@RequestBody Book book) {
        LibraryService libraryService = beanRetrievalService.getBean(LibraryService.class);
        libraryService.addBook(book);
    }

    @GetMapping("/books/{isbn}")
    public Book getBook(@PathVariable String isbn) {
        LibraryService libraryService = beanRetrievalService.getBean(LibraryService.class);
        return libraryService.getBook(isbn);
    }
}
```

Step 7: Run and Test

1. Run the main application class (LibraryappApplication.java)
2. Use Postman or curl to test:

Add a book (using @Autowired):

POST http://localhost:8080/api/books

Content-Type: application/json

```
{
  "isbn": "1234567890",
  "title": "Spring in Action",
  "author": "Craig Walls"
}
```

Get a book (using @Autowired):

GET http://localhost:8080/api/books/1234567890

Add a book (using getBean()):

POST http://localhost:8080/api/admin/books

Content-Type: application/json

```
{
  "isbn": "0987654321",
  "title": "Pro Spring 5",
  "author": "Iuliana Cosmina"
}
```

Get a book (using getBean()):

GET http://localhost:8080/api/admin/books/0987654321

This example demonstrates:

1. @Autowired: Used in BookController for dependency injection of LibraryService.
2. getBean(): Used in AdminController to retrieve LibraryService at runtime.

The @Autowired approach is simpler and more common, while getBean() offers more flexibility at runtime but is generally used less frequently.

Hello world program

```
J HelloWorldApplication.java U X J HelloWorldController.java U pom.xml U
src > main > java > com > example > J HelloWorldApplication.java > ...
1 package com.example;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class HelloWorldApplication {
8
9     Run | Debug
10    public static void main(String[] args) {
11        SpringApplication.run(HelloWorldApplication.class, args);
12    }
13 }
```

```
src > main > java > com > example > J HelloWorldController.java > ...
1 package com.example;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class HelloWorldController {
8
9     @GetMapping("/api/hello")
10    public String helloWorld() {
11        return "Hello, World!";
12    }
13 }
14
15
```

HTTP <http://localhost:8080/api/hello>

GET

<http://localhost:8080/api/hello>

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

Visualize

Text

⌵

1 Hello, World!

```
src > main > java > com > example > WeatherController.java > WeatherController > getWeatherWithCountry(String, String)
1 package com.example;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.PathVariable;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 public class WeatherController {
9
10     @GetMapping("/api/weather/{city}")
11     public String getWeather(@PathVariable String city) {
12         return "The weather in " + city + " is nice today.";
13     }
14
15     @GetMapping("/api/weather/{city}/{country}")
16     public String getWeatherWithCountry(@PathVariable String city, @PathVariable String country) {
17         return "The weather in " + city + ", " + country + " is 25 degrees Celsius today.";
18     }
19 }
20
```

HTTP <http://localhost:8080/api/weather/London>

GET <http://localhost:8080/api/weather/London>

Params Authorization Headers (7) Body Scripts Settings


Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize Text 

```
1 Weather information not available for London
```

```
src > main > java > com > example >  WeatherApiApplication.java > ...  
1 package com.example;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class WeatherApiApplication {  
8     Run | Debug  
9     public static void main(String[] args) {  
10         SpringApplication.run(WeatherApiApplication.class, args);  
11     }  
12 }  
13
```

```

src > main > java > com > example > J WeatherController.java > ⚡ WeatherController > ⚙ getWeather(String)
1  package com.example;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.web.bind.annotation.*;
5
6  @RestController
7  @RequestMapping("/api")
8  public class WeatherController {
9
10     private final WeatherService weatherService;
11
12     @Autowired
13     public WeatherController(WeatherService weatherService) {
14         this.weatherService = weatherService;
15     }
16
17     @GetMapping("/weather/{city}")
18     public String getWeather(@PathVariable String city) {
19         return weatherService.getWeather(city);
20     }
21
22     @PostMapping("/weather/{city}")
23     public String updateWeather(@PathVariable String city, @RequestParam String condition) {
24         weatherService.updateWeather(city, condition);
25         return "Weather updated for " + city;
26     }
27 }
28

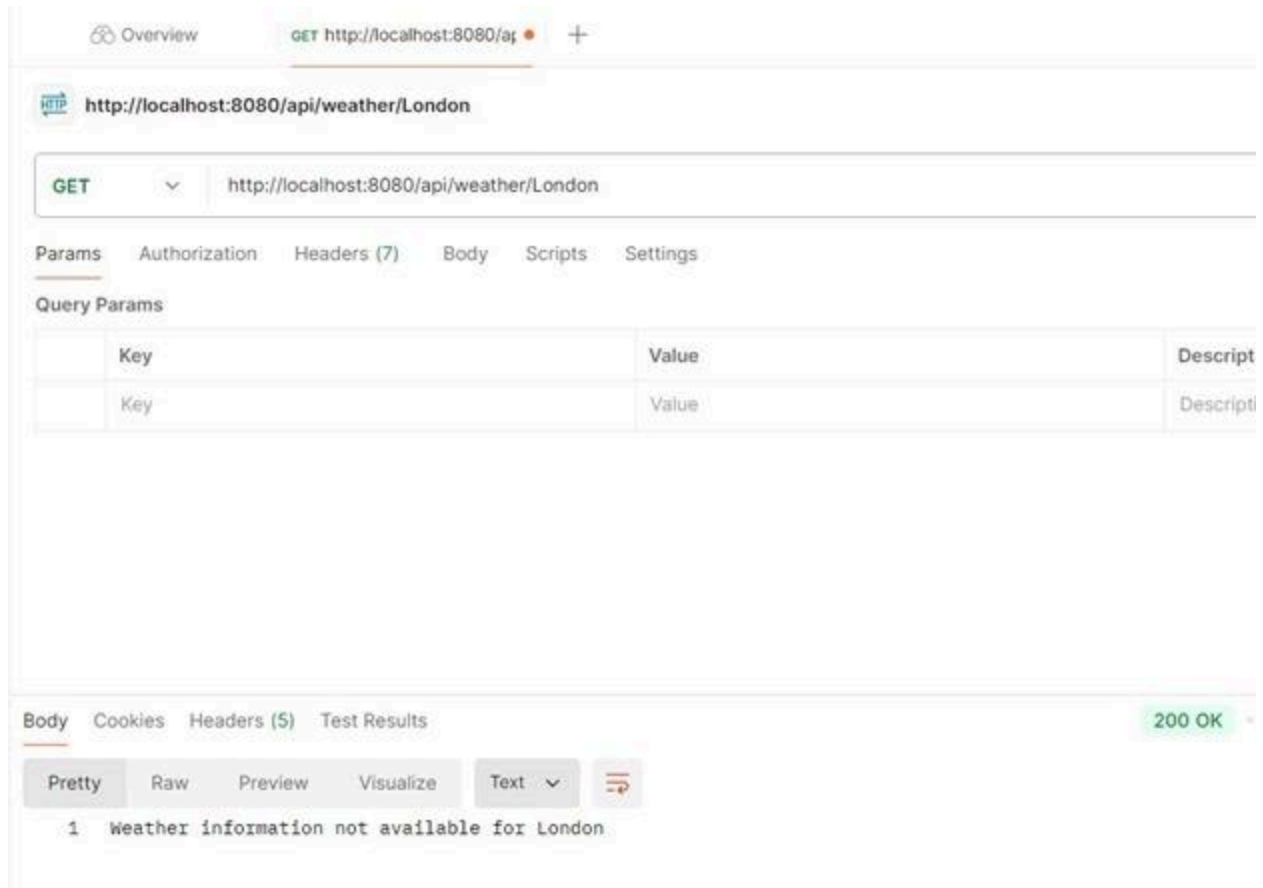
```

```

src > main > java > com > example > J WeatherService.java > ⚡ WeatherService > ⚙ updateWeather(String, String)
1  package com.example;
2
3  import org.springframework.stereotype.Service;
4  import java.util.HashMap;
5  import java.util.Map;
6
7  @Service
8  public class WeatherService {
9
10     private Map<String, String> weatherData = new HashMap<>();
11
12     public String getWeather(String city) {
13         return weatherData.getOrDefault(city, "Weather information not available for " + city);
14     }
15
16     public void updateWeather(String city, String condition) {
17         weatherData.put(city, condition);
18     }
19 }

```





## Weather Update and Retrieval Service


### Flowchart of the Application:

1. Client Interaction:- The process starts with a client sending an HTTP request to our Weather API.
2. WeatherController:- The request is received by the WeatherController.- It has two main endpoints:
  - a. GET /api/weather/{city}: Maps to getWeather method
  - b. POST /api/weather/{city}: Maps to updateWeather method
3. Dependency Injection:- WeatherService is injected into WeatherController.- WeatherConfig is injected into WeatherService.
4. GET Request Flow: When a GET request is received, WeatherController.getWeather calls WeatherService.getWeather.- WeatherService.getWeather checks if the city exists in the weatherData map.- If the city exists, it returns the stored weather condition.- If the city doesn't exist, it returns the default condition from WeatherConfig.
5. POST Request Flow: When a POST request is received, WeatherController.updateWeather calls WeatherService.updateWeather.- WeatherService.updateWeather updates the weatherData map with the new weather condition for the specified city.
6. WeatherConfig:- WeatherConfig is configured by application.properties.- It provides the defaultCondition to WeatherService.
7. In-Memory Data Storage:

- The weatherData map in WeatherService acts as an in-memory database, storing the weather conditions for different cities.

This flow chart now accurately represents the structure and flow of our Java code:- It shows the correct method calls between WeatherController and WeatherService.- It accurately represents how WeatherService uses the weatherData map and WeatherConfig.- It illustrates the configuration flow from application.properties to WeatherConfig. This representation should help students understand:

1. The role of each component (Controller, Service, Config) in the application.
2. How HTTPRequests are handled and processed through the application layers.
3. The use of dependency injection to connect components.
4. How data is stored and retrieved using the in-memory weatherData map.
5. The role of external configuration via application.properties.

src > main > java > com > example >  WeatherApiApplication.java > ...

```
1  package com.example;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class WeatherApiApplication {
8      Run | Debug
9      public static void main(String[] args) {
10         SpringApplication.run(WeatherApiApplication.class, args);
11     }
12 }
13
```

```
src > main > java > com > example > J WeatherConfig.java > ...
```

```
1 package com.example;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4 import org.springframework.context.annotation.Configuration;
5
6 @Configuration
7 @ConfigurationProperties(prefix = "weather")
8 public class WeatherConfig {
9     private String defaultCondition;
10
11     public String getDefaultCondition() {
12         return defaultCondition;
13     }
14
15     public void setDefaultCondition(String defaultCondition) {
16         this.defaultCondition = defaultCondition;
17     }
18 }
19
20
```

```
src > main > java > com > example > J WeatherController.java > ...
```

```
1 package com.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.*;
5 import javax.validation.constraints.NotBlank;
6 import org.springframework.validation.annotation.Validated;
7
8 @RestController
9 @RequestMapping("/api")
10 @Validated
11 public class WeatherController {
12
13     private final WeatherService weatherService;
14
15     @Autowired
16     public WeatherController(WeatherService weatherService) {
17         this.weatherService = weatherService;
18     }
19
20     @GetMapping("/weather/{city}")
21     public String getWeather(@PathVariable @NotBlank String city) {
22         return weatherService.getWeather(city);
23     }
24
25     @PostMapping("/weather/{city}")
26     public String updateWeather(@PathVariable @NotBlank String city,
27                                @RequestParam @NotBlank String condition) {
28         weatherService.updateWeather(city, condition);
29         return "Weather updated for " + city;
30     }
31 }
32
```

src > main > java > com > example > J WeatherService.java > WeatherService > updateWeather(String, String)

```
1 package com.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import java.util.HashMap;
6 import java.util.Map;
7
8 @Service
9 public class WeatherService{
10     private final WeatherConfig weatherConfig;
11     private Map<String, String> weatherData = new HashMap<>();
12     @Autowired
13     public WeatherService(WeatherConfig weatherConfig){
14         this.weatherConfig = weatherConfig;
15     }
16     public String getWeather(String city){
17         return weatherData.getOrDefault(city, weatherConfig.getDefaultCondition());
18     }
19     public void updateWeather(String city, String condition){
20         weatherData.put(city, condition);
21     }
22 }
```

http://localhost:8080/api/weather/London

GET http://localhost:8080/api/weather/London

Params Authorization Headers (7) Body Scripts Settings

Query Params

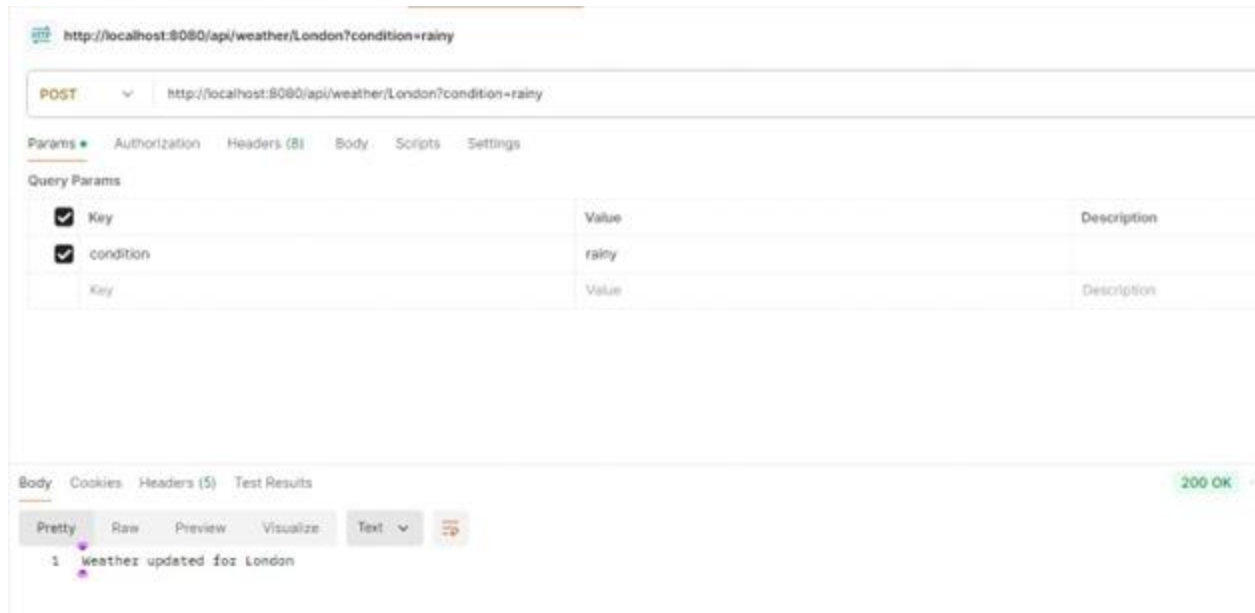
Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

200 OK

Pretty Raw Preview Visualize Text

1 Sunny



## Spring boot weather application

// weather configuration code

```
package com.example;
import java.util.HashMap;
import java.util.Map;

import
org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
@Configuration
@ConfigurationProperties(prefix = "weather")
public class WeatherConfig {
    private String defaultCondition;
    private Map<String, TemperatureRange> cityTempratureRanges=new
HashMap<> ();

    public String getDefaultCondition() {
        return defaultCondition;
    }
    public void setDefaultCondition(String defaultCondition) {
        this.defaultCondition = defaultCondition;
    }
    public Map<String, TemperatureRange> getCityTempratureRanges() {
```

```

        return this.cityTemperatureRanges;
    }

    public void setCityTemperatureRanges (Map<String, TemperatureRange>
cityTemperatureRanges) {
        this.cityTemperatureRanges = cityTemperatureRanges;
    }

    public static class TemperatureRange{
        private double min;
        private double max;
        public double getMin() {
            return min;
        }
        public void setMin(double min) {
            this.min = min;
        }
        public double getMax() {
            return max;
        }
        public void setMax(double max) {
            this.max = max;
        }
    }
}

```

// weather controller code

```

package com.example;

import org.springframework.web.bind.annotation.RestController;

import java.util.Map;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.DeleteMapping;

import javax.validation.constraints.NotBlank;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.validation.annotation.Validated;

```

```

import java.util.List;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.http.HttpStatus;
@RestController
@RequestMapping("/api/weather")
@Validated
public class WeatherController {

    private final WeatherService weatherService;
    @Autowired
    public WeatherController(WeatherService weatherService) {
        this.weatherService = weatherService;
    }

    @GetMapping("/all")
    public List<WeatherRecord> getAllWeather() {
        return weatherService.getAllWeather();
    }

    @GetMapping("/{city}")
    public ResponseEntity<WeatherRecord> getWeatherByCity(@PathVariable
@NotBlank String city) {
        WeatherRecord weather = weatherService.getWeatherByCity(city);
        return weather != null ? ResponseEntity.ok(weather) :
ResponseEntity.notFound().build();
    }

    @PostMapping("/create")
    public WeatherRecord addWeather(@RequestBody WeatherRecord
weatherRecord) {
        return weatherService.addWeather(weatherRecord);
    }

    @PutMapping("/{id}")
    public ResponseEntity<WeatherRecord> updateWeather(@PathVariable Long
id, @RequestBody WeatherRecord weatherRecord) {

```

```

        if(!weatherService.getAllWeather().stream().anyMatch(weather ->
weather.getId().equals(id))){
            return ResponseEntity.notFound().build();
        }
        weatherRecord.setId(id);
        return
ResponseEntity.ok(weatherService.updateWeather(weatherRecord));
    }
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteWeather(@PathVariable Long id){
        if(!weatherService.getAllWeather().stream().anyMatch(weather ->
weather.getId().equals(id))){
            return ResponseEntity.notFound().build();
        }
        weatherService.deleteWeather(id);
        return ResponseEntity.noContent().build();
    }
}

```

// weather record

```

package com.example;

import jakarta.persistence.*;

@Entity
@Table(name="weather_records")
public class WeatherRecord {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false,unique = true)
    private String city;
    @Column(nullable = false)
    private int temperature;
    @Column(nullable = false)
    private String weatherCondition;

    public WeatherRecord(){

```



```

    }

    public void initialize(String city, int temperature){
        this.city = city;
        this.temperature = temperature;
    }

    public Long getId(){
        return id;
    }

    public void setId(Long id){
        this.id = id;
    }

    public String getCity(){
        return city;
    }

    public String getWeatherCondition(){
        return weatherCondition;
    }

    public void setCity(String city){
        this.city = city;
    }

    public void setWeatherCondition(String weatherCondition){
        this.weatherCondition = weatherCondition;
    }

    public void setTemperature(int temperature){
        this.temperature = temperature;
    }

    public int getTemperature(){
        return temperature;
    }
}

```

//weather repository

```

package com.example;

import org.springframework.data.jpa.repository.JpaRepository;

```

```

import org.springframework.stereotype.Repository;
import java.util.Optional;
@Repository
public interface WeatherRepository extends
JpaRepository<WeatherRecord, Long> {

    Optional<WeatherRecord> findByCity(String city);
}

```

// weather service

```

package com.example;

import java.util.HashMap;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class WeatherService {
    private final WeatherRepository weatherRepository;
    @Autowired
    public WeatherService(WeatherRepository weatherRepository) {
        this.weatherRepository = weatherRepository;
    }

    public List<WeatherRecord> getAllWeather() {
        return weatherRepository.findAll();
    }

    public WeatherRecord getWeatherByCity(String city) {
        return weatherRepository.findByCity(city).orElseThrow(() -> new
RuntimeException("City not found"));
    }

    public WeatherRecord addWeather(WeatherRecord weatherRecord) {
        return weatherRepository.save(weatherRecord);
    }

    public void deleteWeather(Long id) {
        weatherRepository.deleteById(id);
    }
}

```

```

    public WeatherRecord updateWeather(WeatherRecord weatherRecord) {
        return weatherRepository.save(weatherRecord);
    }
}

```

// dependencies

```

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>javax.validation</groupId>
        <artifactId>validation-api</artifactId>
        <version>2.0.0.Final</version>
    </dependency>
</dependencies>

```

```

        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.33</version>
        </dependency>
    </dependencies>

```

//Restaurant application

```

package com.example;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import jakarta.persistence.Column;
import jakarta.persistence.OneToMany;
import jakarta.persistence.CascadeType;
import java.util.List;

@Entity
@Table(name = "restaurant")
public class Restaurant {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String name;
    @Column(nullable = false)
    private String cuisine;
    @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL,
orphanRemoval = true)

```

```

private List<Review> reviews;

//getters and setters
public Long getId(){
    return id;
}
public void setId(Long id){
    this.id = id;
}
public String getName(){
    return name;
}
public void setName(String name){
    this.name = name;
}
public String getCusine(){
    return cuisine;
}
public void setCusine(String cuisine){
    this.cuisine = cuisine;
}
public List<Review> getReviews(){
    return reviews;
}
public void setReviews(List<Review> reviews){
    this.reviews = reviews;
}
}

```

```

package com.example;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.Table;
import jakarta.persistence.Column;

```

```
@Entity
@Table(name = "review")
public class Review {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String reviewerName;
    @Column(nullable = false)
    private int rating;
    @ManyToOne
    @JoinColumn(name = "restaurant_id", nullable = false)
    private Restaurant restaurant;
    @Column(nullable = false)
    private String comment;

    //getters and setters
    public Long getId(){
        return id;
    }
    public void setId(Long id){
        this.id = id;
    }
    public String getReviewerName(){
        return reviewerName;
    }
    public void setReviewerName(String reviewerName){
        this.reviewerName = reviewerName;
    }
    public int getRating(){
        return rating;
    }
    public void setRating(int rating){
        this.rating = rating;
    }
    public Restaurant getRestaurant(){
        return restaurant;
    }
}
```

```

    public void setRestaurant(Restaurant restaurant) {
        this.restaurant = restaurant;
    }
    public String getComment() {
        return comment;
    }
    public void setComment(String comment) {
        this.comment = comment;
    }
}

```

```

package com.example;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.example.Restaurant;
import com.example.RestaurantService;

@RestController
@RequestMapping("/api/restaurants")
public class RestaurantController {

    @Autowired
    private RestaurantService restaurantService;

    @GetMapping
    public List<Restaurant> getAllRestaurants() {
        return restaurantService.getAllRestaurants();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Restaurant> getRestaurantById(@PathVariable Long
id) {

```

```

        Optional<Restaurant> restaurant =
restaurantService.getRestaurantById(id);
        return restaurant.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public Restaurant createRestaurant(@RequestBody Restaurant restaurant)
    {
        return restaurantService.saveRestaurant(restaurant);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Restaurant> updateRestaurant(@PathVariable Long
id, @RequestBody Restaurant restaurantDetails) {
        Optional<Restaurant> restaurant =
restaurantService.getRestaurantById(id);
        if (restaurant.isPresent()) {
            restaurantDetails.setId(id);
            return
ResponseEntity.ok(restaurantService.saveRestaurant(restaurantDetails));
        }
        return ResponseEntity.notFound().build();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteRestaurant(@PathVariable Long id) {
        restaurantService.deleteRestaurant(id);
        return ResponseEntity.noContent().build();
    }
}

```

```

package com.example;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;

```



```
import org.springframework.web.bind.annotation.*;

import com.example.Review;
import com.example.ReviewService;

@RestController
@RequestMapping("/api/reviews")
public class ReviewController {

    @Autowired
    private ReviewService reviewService;

    @GetMapping
    public List<Review> getAllReviews() {
        return reviewService.getAllReviews();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Review> getReviewById(@PathVariable Long id) {
        Optional<Review> review = reviewService.getReviewById(id);
        return review.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public Review createReview(@RequestBody Review review) {
        return reviewService.saveReview(review);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Review> updateReview(@PathVariable Long id,
@RequestBody Review reviewDetails) {
        Optional<Review> review = reviewService.getReviewById(id);
        if (review.isPresent()) {
            reviewDetails.setId(id);
            return
ResponseEntity.ok(reviewService.saveReview(reviewDetails));
        }
        return ResponseEntity.notFound().build();
    }
}
```

```

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteReview(@PathVariable Long id) {
        reviewService.deleteReview(id);
        return ResponseEntity.noContent().build();
    }
}

```

```

package com.example;

import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.Review;
import com.example.ReviewRepository;

@Service
public class ReviewService {

    @Autowired
    private ReviewRepository reviewRepository;

    public List<Review> getAllReviews() {
        return reviewRepository.findAll();
    }

    public Optional<Review> getReviewById(Long id) {
        return reviewRepository.findById(id);
    }

    public Review saveReview(Review review) {
        return reviewRepository.save(review);
    }

    public void deleteReview(Long id) {
        reviewRepository.deleteById(id);
    }
}

```

```
package com.example;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.Restaurant;
import com.example.RestaurantRepository;

@Service
public class RestaurantService {

    @Autowired
    private RestaurantRepository restaurantRepository;

    public List<Restaurant> getAllRestaurants() {
        return restaurantRepository.findAll();
    }

    public Optional<Restaurant> getRestaurantById(Long id) {
        return restaurantRepository.findById(id);
    }

    public Restaurant saveRestaurant(Restaurant restaurant) {
        return restaurantRepository.save(restaurant);
    }

    public void deleteRestaurant(Long id) {
        restaurantRepository.deleteById(id);
    }
}
```

```
package com.example;

import org.springframework.data.jpa.repository.JpaRepository;
import com.example.Review;
```

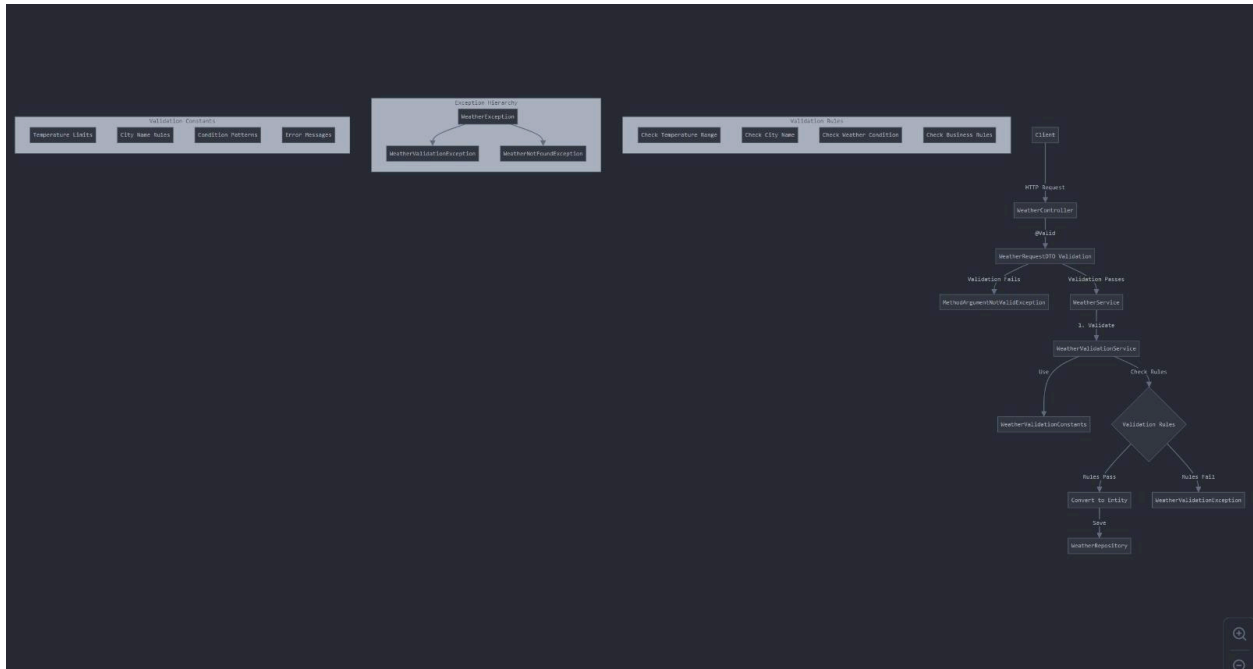
```
public interface ReviewRepository extends JpaRepository<Review, Long> {  
  
}
```

```
package com.example;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import com.example.Restaurant;  
  
public interface RestaurantRepository extends JpaRepository<Restaurant,  
Long> {  
  
}
```

// SQL functionality

```
package com.example;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import java.util.List;  
import org.springframework.data.jpa.repository.Query;  
  
public interface restaurantrepositry extends JpaRepository<Restaurant,  
Long>{  
    List<Restaurant> findByCusine(String cusine);  
  
    @Query("SELECT r FROM Restaurant r WHERE r.name LIKE %?1%")  
    List<Restaurant> searchByName(String name);  
  
    @Query("SELECT r FROM Restaurant r WHERE r.cusine LIKE %?1%")  
    List<Restaurant> searchByCusine(String cusine);  
  
    @Query("select r from Restaurant r join r.reviews rv group by r having  
avg(rv.rating) > ?1")  
    List<Restaurant> searchByRating(double rating);  
}
```

```
@Query("SELECT u FROM User u WHERE u.email LIKE %:email%")  
List<User> findUsersByEmailPattern(@Param("email") String email);
```



// Weather application

```
package com.example;

import java.util.*;

public class WeatherException extends RuntimeException{
    public WeatherException(String message){
        super(message);
    }

    public WeatherException(String message, Throwable cause){
        super(message, cause);
    }
}
```

```
package com.example;

import jakarta.validation.constraints.*;

public class WeatherRequestDTO {
    @NotBlank(message = "City is required and cannot be empty")
    @Size(min = 3, max = 20, message = "City must be between 3 and 20 characters")
```

```

private String city;

@Min(value = -50, message = "Temperature cannot be less than -50")
@Max(value = 50, message = "Temperature cannot be greater than 50")
private int temperature;

@NotBlank(message = "Weather condition is required and cannot be
empty")
@Size(min = 3, max = 20, message = "Weather condition must be between
3 and 20 characters")
@Pattern(regexp = "^(sunny|cloudy|rainy|snowy)$", message = "Weather
condition must be sunny, cloudy, rainy, or snowy")
private String weatherCondition;

//getters and setters
public String getCity(){
    return city;
}
public void setCity(String city){
    this.city = city;
}
public int getTemperature(){
    return temperature;
}
public void setTemperature(int temperature){
    this.temperature = temperature;
}
public String getWeatherCondition(){
    return weatherCondition;
}
public void setWeatherCondition(String weatherCondition){
    this.weatherCondition = weatherCondition;
}
}

```

```

package com.example;

import java.util.HashMap;
import java.util.Map;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import org.springframework.validation.annotation.Validated;
import org.springframework.transaction.annotation.Transactional;
import com.example.WeatherRequestDTO;
@Service
@Validated
public class WeatherService {
    private final WeatherRepository weatherRepository;

    private final WeatherValidator weatherValidator;
    @Autowired
    public WeatherService(WeatherRepository weatherRepository,
WeatherValidator weatherValidator){
        this.weatherRepository = weatherRepository;
        this.weatherValidator = weatherValidator;
    }

    public List<WeatherRecord> getAllWeather(){
        return weatherRepository.findAll();
    }

    public WeatherRecord getWeatherByCity(String city){
        return weatherRepository.findByCity(city).orElseThrow(() -> new
RuntimeException("City not found"));
    }

    private WeatherRecord convertToEntity(WeatherRequestDTO
weatherRecordDTO){
        WeatherRecord weatherRecord = new WeatherRecord();
        weatherRecord.setCity(weatherRecordDTO.getCity());
        weatherRecord.setTemperature(weatherRecordDTO.getTemperature());

weatherRecord.setWeatherCondition(weatherRecordDTO.getWeatherCondition());
        return weatherRecord;
    }
    @Transactional
    public WeatherRecord addWeather(WeatherRequestDTO weatherRecordDTO){
        weatherValidator.validateWeatherCondition(weatherRecordDTO);
    }

```

```

        WeatherRecord weatherRecord = convertToEntity(weatherRecordDTO);
        return weatherRepository.save(weatherRecord);
    }
    public void deleteWeather(Long id){
        weatherRepository.deleteById(id);
    }
    public WeatherRecord updateWeather(WeatherRecord weatherRecord){
        return weatherRepository.save(weatherRecord);
    }
}

```

```

package com.example;

public class WeatherValidationConstants {
    public static final int MIN_TEMPERATURE = -50;
    public static final int MAX_TEMPERATURE = 50;

    public static final String WEATHER_CONDITION_PATTERN =
    "^(sunny|cloudy|rainy|snowy)$";
    public static final int CITY_MIN_LENGTH = 3;
    public static final int CITY_MAX_LENGTH = 20;

    public static final String CITY_BLANK_MESSAGE = "City is required and
cannot be empty";
    public static final String CITY_SIZE_MESSAGE = "City must be between 3
and 20 characters";
    public static final String CITY_PATTERN_MESSAGE = "City must be sunny,
cloudy, rainy, or snowy";

    public static final String TEMPERATURE_MIN_MESSAGE = "Temperature
cannot be less than -50";
    public static final String TEMPERATURE_MAX_MESSAGE = "Temperature
cannot be greater than 50";

    public static final String WEATHER_CONDITION_BLANK_MESSAGE = "Weather
condition is required and cannot be empty";
    public static final String WEATHER_CONDITION_SIZE_MESSAGE = "Weather
condition must be between 3 and 20 characters";
}

```



```
    public static final String WEATHER_CONDITION_PATTERN_MESSAGE =  
    "Weather condition must be sunny, cloudy, rainy, or snowy";  
}
```

```
package com.example;  
  
import java.util.*;  
  
public class WeatherValidationException extends WeatherException{  
    private final List<String> violations;  
  
    public WeatherValidationException(String message, List<String>  
violations){  
        super(message);  
        this.violations = violations.isEmpty()  
            ? List.of()  
            : List.of(String.join(", ", violations));  
    }  
  
    public WeatherValidationException(List<String> violations){  
        super("Weather validation failed: " + String.join("  
",violations));  
        this.violations = new ArrayList<>(violations);  
    }  
  
    public List<String> getViolations(){  
        return violations;  
    }  
}
```

```
package com.example;  
  
import org.springframework.stereotype.Component;  
import java.util.List;  
import java.util.ArrayList;  
  
@Component  
public class WeatherValidator {  
    public void validateWeatherCondition(WeatherRequestDTO  
weatherRequestDTO){  
        List<String> violations = new ArrayList<>();  

```

```

        if(weatherRequestDTO.getTemperature() > 30 &&
"Snowy".equals(weatherRequestDTO.getWeatherCondition())){
            violations.add("Snowy weather is not allowed when temperature
is above 30 degrees");
        }

        if(weatherRequestDTO.getTemperature() < 0 &&
"Sunny".equals(weatherRequestDTO.getWeatherCondition())){
            violations.add("Sunny weather is not allowed when temperature
is below 0 degrees");
        }

        if(violations.isEmpty()){
            return;
        }

        throw new WeatherValidationException(String.join("
",violations),violations);
    }
}

```

```

package com.example;

import org.springframework.web.bind.annotation.RestController;

import java.util.Map;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.DeleteMapping;

import javax.validation.constraints.NotBlank;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.validation.annotation.Validated;
import java.util.List;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.http.HttpStatus;

```

```
import javax.validation.Valid;
@RestController
@RequestMapping("/api/weather")
@Validated
public class WeatherController {

    private final WeatherService weatherService;
    @Autowired
    public WeatherController(WeatherService weatherService) {
        this.weatherService = weatherService;
    }

    @GetMapping("/all")
    public List<WeatherRecord> getAllWeather() {
        return weatherService.getAllWeather();
    }

    @GetMapping("/{city}")
    public ResponseEntity<WeatherRecord> getWeatherByCity(@PathVariable
@NotBlank String city) {
        WeatherRecord weather = weatherService.getWeatherByCity(city);
        return weather != null ? ResponseEntity.ok(weather) :
ResponseEntity.notFound().build();
    }

    @PostMapping("/create")
    public WeatherRecord addWeather(@Valid @RequestBody WeatherRequestDTO
weatherRequestDTO) {
        return weatherService.addWeather(weatherRequestDTO);
    }

    /*
    @PutMapping("/{id}")
    public ResponseEntity<WeatherRecord> updateWeather(@PathVariable Long
id, @RequestBody WeatherRecord weatherRecord) {
        if(!weatherService.getAllWeather().stream().anyMatch(weather ->
weather.getId().equals(id))){
            return ResponseEntity.notFound().build();
        }
    }
    */
}
```

```

        weatherRecord.setId(id);
        return
ResponseEntity.ok(weatherService.updateWeather(weatherRecord));
    }
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteWeather(@PathVariable Long id){
        if(!weatherService.getAllWeather().stream().anyMatch(weather ->
weather.getId().equals(id))){
            return ResponseEntity.notFound().build();
        }
        weatherService.deleteWeather(id);
        return ResponseEntity.noContent().build();
    }
}
*/
}

```

## # Spring Bean Scopes

### ## 1. Singleton Scope (Default)

- Only ONE instance is created for the entire application
- Same instance is shared across all requests
- Default scope in Spring
- Good for: Stateless beans, Services, Repositories

Example:

```

java
@Service
// or @Service(@Scope("singleton")) - not needed as it's default
public class CounterService {
    private int counter = 0; // This value is shared

    public void increment() {
        counter++;
    }

    public int getCount() {
        return counter;
    }
}

```

Usage Pattern:

- All users see the same counter value

- Counter increments globally

### ## 2. Prototype Scope

- New instance created every time requested
- Good for: Stateful beans, Heavy resources
- Each injection gets its own instance

Example:

```
java
@Component
@Scope("prototype")
public class UserRequest {
    private final String requestId = UUID.randomUUID().toString();
    private final LocalDateTime created = LocalDateTime.now();

    public String getRequestId() {
        return requestId;
    }
}
```

Usage Pattern:

- Each time `getBean()` is called, new instance
- Each instance has its own state

### ## 3. Request Scope

- New instance for each HTTP request
- Instance lives only for the duration of HTTP request
- Good for: Request-specific data, Logging

Example:

```
java
@Component
@RequestScope // same as @Scope(value = WebApplicationContext.SCOPE_REQUEST)
public class RequestLogger {
    private final String requestId = UUID.randomUUID().toString();
    private List<String> logs = new ArrayList<>();

    public void addLog(String message) {
        logs.add(message);
    }
}
```

Usage Pattern:

- New instance per HTTP request
- Data dies after request completes

#### ## 4. Session Scope

- New instance for each user session
- Instance lives for entire user session
- Good for: Shopping carts, User preferences

Example:

```
java
@Component
@SessionScope
public class ShoppingCart {
    private List<String> items = new ArrayList<>();

    public void addItem(String item) {
        items.add(item);
    }

    public List<String> getItems() {
        return items;
    }
}
```

Usage Pattern:

- Persists across requests for same user
- Different users get different instances

```
// WeatherRecord.java
package com.example.weather.model;

public class WeatherRecord {
    private String city;
    private double temperature;
    private String condition;

    public WeatherRecord(String city, double temperature, String
condition) {
        this.city = city;
        this.temperature = temperature;
        this.condition = condition;
    }
}
```

```

    }

    // Getters and Setters
    public String getCity() { return city; }
    public void setCity(String city) { this.city = city; }
    public double getTemperature() { return temperature; }
    public void setTemperature(double temperature) { this.temperature =
temperature; }
    public String getCondition() { return condition; }
    public void setCondition(String condition) { this.condition =
condition; }
}

```

```

// WeatherService.java
package com.example.weather.service;

import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;

@Service // Singleton by default
public class WeatherService {
    private Map<String, WeatherRecord> weatherData = new HashMap<>();
    private int totalRequests = 0; // Shared counter for all users

    public void updateWeather(String city, double temperature, String
condition) {
        weatherData.put(city, new WeatherRecord(city, temperature,
condition));
        totalRequests++;
    }

    public WeatherRecord getWeather(String city) {
        totalRequests++;
        return weatherData.get(city);
    }

    public int getTotalRequests() {
        return totalRequests;
    }
}

```

```
}
```

```
// WeatherQuery.java
package com.example.weather.service;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;
import java.time.LocalDateTime;

@Component
@Scope("prototype")
public class WeatherQuery {
    private final String queryId = java.util.UUID.randomUUID().toString();
    private final LocalDateTime queryTime = LocalDateTime.now();

    public String getQueryId() { return queryId; }
    public LocalDateTime getQueryTime() { return queryTime; }
}
```

```
// WeatherRequestLogger.java
package com.example.weather.service;

import org.springframework.web.context.annotation.RequestScope;
import org.springframework.stereotype.Component;
import java.time.LocalDateTime;

@Component
@RequestScope
public class WeatherRequestLogger {
    private final String requestId =
java.util.UUID.randomUUID().toString();
    private final LocalDateTime requestTime = LocalDateTime.now();
    private String cityRequested;

    public void logRequest(String city) {
        this.cityRequested = city;
    }

    public String getRequestInfo() {
```



```
        return String.format("Request ID: %s, Time: %s, City: %s",
                               requestId, requestTime, cityRequested);
    }
}
```

```
// UserWeatherPreferences.java
package com.example.weather.service;

import org.springframework.web.context.annotation.SessionScope;
import org.springframework.stereotype.Component;
import java.util.*;

@Component
@SessionScope
public class UserWeatherPreferences {
    private final String sessionId =
java.util.UUID.randomUUID().toString();
    private final List<String> recentSearches = new ArrayList<>();
    private String temperatureUnit = "Celsius";

    public void addSearch(String city) {
        if (recentSearches.size() >= 5) {
            recentSearches.remove(0);
        }
        recentSearches.add(city);
    }

    public List<String> getRecentSearches() {
        return new ArrayList<>(recentSearches);
    }

    public String getSessionId() { return sessionId; }
    public void setTemperatureUnit(String unit) { this.temperatureUnit =
unit; }
    public String getTemperatureUnit() { return temperatureUnit; }
}
```

```
// WeatherController.java
package com.example.weather.controller;
```

```

import org.springframework.web.bind.annotation.*;
import org.springframework.context.ApplicationContext;
import java.util.HashMap;
import java.util.Map;

@RestController
@RequestMapping("/weather")
public class WeatherController {
    private final WeatherService weatherService;           // Singleton
    private final WeatherRequestLogger requestLogger;       // Request
Scope
    private final UserWeatherPreferences userPreferences;  // Session
Scope
    private final ApplicationContext applicationContext; // For
Prototype

    public WeatherController(
        WeatherService weatherService,
        WeatherRequestLogger requestLogger,
        UserWeatherPreferences userPreferences,
        ApplicationContext applicationContext) {
        this.weatherService = weatherService;
        this.requestLogger = requestLogger;
        this.userPreferences = userPreferences;
        this.applicationContext = applicationContext;
    }

    @GetMapping("/{city}")
    public Map<String, Object> getWeather(@PathVariable String city) {
        // Request scope - log the request
        requestLogger.logRequest(city);

        // Session scope - add to recent searches
        userPreferences.addSearch(city);

        // Prototype scope - create new query
        WeatherQuery query =
applicationContext.getBean(WeatherQuery.class);

```

```

        // Singleton - get weather data
        WeatherRecord weather = weatherService.getWeather(city);

        Map<String, Object> response = new HashMap<>();
        response.put("weather", weather);
        response.put("requestInfo", requestLogger.getRequestInfo());
        response.put("recentSearches",
userPreferences.getRecentSearches());
        response.put("queryInfo", Map.of(
            "queryId", query.getQueryId(),
            "queryTime", query.getQueryTime()
        ));
        response.put("totalRequests", weatherService.getTotalRequests());

        return response;
    }

    @PostMapping("/{city}")
    public Map<String, Object> updateWeather(
        @PathVariable String city,
        @RequestParam double temperature,
        @RequestParam String condition) {
        weatherService.updateWeather(city, temperature, condition);
        return getWeather(city);
    }

    @PostMapping("/preferences/unit")
    public Map<String, String> setUnit(@RequestParam String unit) {
        userPreferences.setTemperatureUnit(unit);
        return Map.of(
            "sessionId", userPreferences.getSessionId(),
            "unit", userPreferences.getTemperatureUnit()
        );
    }
}

```

//Spring security

```
// JwtUtil.java
```

```

@Component
public class JwtUtil {
    @Value("${jwt.secret}")
    private String secret;

    public String generateToken(String username) {
        return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() +
864000000)) // 10 days
            .signWith(SignatureAlgorithm.HS512, secret)
            .compact();
    }

    public String getUsernameFromToken(String token) {
        return Jwts.parser()
            .setSigningKey(secret)
            .parseClaimsJws(token)
            .getBody()
            .getSubject();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
}

```

```

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
}

```

```

@Column(unique = true)
private String username;
private String password;
private String role = "ROLE_USER"; // Simple role management

// Getters and setters
}

```

```

// User.java
package com.example;

import
org.springframework.security.web.authentication.UsernamePasswordAuthentica
tionFilter;
import org.springframework.web.filter.OncePerRequestFilter;
import org.springframework.beans.factory.annotation.Autowired;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import
org.springframework.security.authentication.UsernamePasswordAuthentication
Token;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import java.util.Arrays;
import org.springframework.security.core.context.SecurityContextHolder;

/*
 * Purpose: Intercepts all requests and validates the JWT token for
security
 * Responsibilities:
 * 1. Extracts JWT token from the request
 * 2. Validates the token
 * 3. Sets the authentication in the context, so that downstream filters
can use it
 * Set up security context if token is valid
 * 4. If validation fails, sends 401 Unauthorized response
 * 5. Otherwise, passes the request to the next filter in the chain

```

```

    */
public class JwtFilter extends OncePerRequestFilter {

    @Autowired
    private Jwtutil jwtUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
    HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
        String token = extractToken(request);
        if (token != null && jwtUtil.validateToken(token)) {
            String username = jwtUtil.getUsernameFromToken(token);
            UsernamePasswordAuthenticationToken authenticationToken = new
    UsernamePasswordAuthenticationToken(username, null, Arrays.asList(new
    SimpleGrantedAuthority("ROLE_USER")));

    SecurityContextHolder.getContext().setAuthentication(authenticationToken);

        }
        filterChain.doFilter(request, response);
    }

    private String extractToken(HttpServletRequest request) {
        String authorizationHeader = request.getHeader("Authorization");
        if (authorizationHeader != null &&
    authorizationHeader.startsWith("Bearer ")) {
            return authorizationHeader.substring(7);
        }
        return null;
    }
}

```

```

// AuthController.java
package com.example.security.controller;

import com.example.security.dto.LoginRequest;
import com.example.security.dto.LoginResponse;
import com.example.security.service.UserService;

```

```
import com.example.security.util.JwtUtil;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {
    private final UserService userService;
    private final JwtUtil jwtUtil;

    public AuthController(UserService userService, JwtUtil jwtUtil) {
        this.userService = userService;
        this.jwtUtil = jwtUtil;
    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest request) {
        try {
            // Validate user
            if (userService.validateUser(request.getUsername(),
request.getPassword())) {
                // Get user details
                User user =
userService.findByUsername(request.getUsername());

                // Generate token
                String token = jwtUtil.generateToken(user.getUsername());

                // Create response
                LoginResponse response = new LoginResponse(
                    token,
                    user.getUsername(),
                    user.getRole()
                );

                return ResponseEntity.ok(response);
            }

            return ResponseEntity.badRequest().body("Invalid
credentials");
        }
    }
}
```

```

        } catch (Exception e) {
            return ResponseEntity.badRequest().body("Login failed: " +
e.getMessage());
        }
    }

@PostMapping("/register")
public ResponseEntity<?> register(@RequestBody LoginRequest request) {
    try {
        // Create user
        User user = userService.createUser(
            request.getUsername(),
            request.getPassword()
        );

        return ResponseEntity.ok("User registered successfully");
    } catch (Exception e) {
        return ResponseEntity.badRequest()
            .body("Registration failed: " + e.getMessage());
    }
}
}

```

```

// UserService.java
package com.example.security.service;

import com.example.security.model.User;
import com.example.security.repository.UserRepository;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class UserService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public UserService(UserRepository userRepository, PasswordEncoder
passwordEncoder) {

```



```

        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    public User createUser(String username, String password) {
        // Check if username exists
        if (userRepository.existsByUsername(username)) {
            throw new RuntimeException("Username already exists");
        }

        // Create new user
        User user = new User();
        user.setUsername(username);
        user.setPassword(passwordEncoder.encode(password));

        return userRepository.save(user);
    }

    public User findByUsername(String username) {
        return userRepository.findByUsername(username)
            .orElseThrow(() -> new RuntimeException("User not found"));
    }

    public boolean validateUser(String username, String password) {
        User user = findByUsername(username);
        return passwordEncoder.matches(password, user.getPassword());
    }
}

```

```

// UserRepository.java
package com.example.security.repository;

import com.example.security.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    // Find user by username
}

```

```

Optional<User> findByUsername(String username);

// Check if username exists
boolean existsByUsername(String username);

// Find active users by username
Optional<User> findByUsernameAndActiveTrue(String username);

// Custom query example
@Query("SELECT u FROM User u WHERE u.username = :username AND u.active
= true")
Optional<User> findActiveUser(@Param("username") String username);
}

```

```

// LoginResponse.java
package com.example.security.dto;

public class LoginResponse {
    private String token;
    private String username;
    private String role;

    public LoginResponse(String token, String username, String role) {
        this.token = token;
        this.username = username;
        this.role = role;
    }

    // Getters and setters
    public String getToken() {
        return token;
    }

    public void setToken(String token) {
        this.token = token;
    }

    public String getUsername() {
        return username;
    }
}

```

```
public void setUsername(String username) {
    this.username = username;
}

public String getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}
}
```

```
// LoginRequest.java
package com.example.security.dto;

public class LoginRequest {
    private String username;
    private String password;

    // Default constructor
    public LoginRequest() {}

    // Constructor with fields
    public LoginRequest(String username, String password) {
        this.username = username;
        this.password = password;
    }

    // Getters and setters
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
```

```
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

```
curl -X POST http://localhost:8080/auth/register \
-H "Content-Type: application/json" \
-d '{
  "username": "john.doe",
  "password": "password123"
}'
```